
Task Management Software Solution for The Living Room

- Improving Efficiency and Productivity with Java and
Object-Oriented Programming -

Project Report

Group 1

Aalborg University

Electronics and IT



AALBORG UNIVERSITY

STUDENT REPORT

Electronics and IT

Aalborg University

<http://www.aau.dk>

Title:

Task Management Software Solution
for The Living Room

Theme:

Scientific Theme

Project Period:

Fall Semester 2022

Project Group:

1

Participant(s):

Amalie Dilling

Anders Mazen Youssef

Bence Szabo

Freja Lüders Rasmussen

Louise Foldøy Steffens

Magnus Holt

Supervisor(s):

Rikke Hagensby Jensen

Abstract:

This report documents the development of a task management software solution for a cafe called The Living Room. The purpose of the software is to streamline the management of tasks and improve efficiency in a busy work environment.

Java and object-oriented programming were used to create the application, which allows users to create, assign, and track tasks in real-time.

The use of a task management software solution has the potential to benefit the work environment at The Living Room by providing a central location for task tracking and organisation. As a result, our findings highlight the importance of implementing such a solution in order to improve productivity.

Copies: 1

Page Numbers: 136

Date of Completion:

December 19, 2022

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

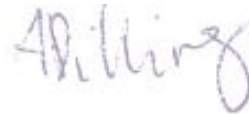
Preface

Aalborg University, December 19, 2022



Anders Mazen Youssef

amyo21@student.aau.dk



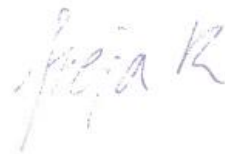
Amalie Pernille Dilling

adilli21@student.aau.dk



Bence Szabo

bszabo21@student.aau.dk



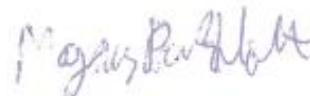
Freja Lüders Rasmussen

flra21@student.aau.dk



Louise Foldøy Steffens

lfst21@student.aau.dk



Magnus Peetz Holt

mph21@student.aau.dk

Contents

Preface	iv
1 Introduction and Motivation	2
1.1 Initial Problem	3
1.2 Report Structure	3
2 Methodology	4
2.1 Object Oriented Programming (OOP)	4
2.1.1 Iterative development model	5
2.1.2 Pair Programming	5
2.2 System Development	6
2.2.1 FACTOR Criterion	6
2.2.2 Class Diagram	7
2.2.3 Event Table	7
2.2.4 Behavioural Patterns	8
2.2.5 Use Cases	8
2.2.6 Functions	8
2.2.7 MoSCoW / Criteria	9
2.2.8 FURPS+	10
2.3 Design and Evaluations of User Interfaces	10
2.3.1 PACT Analysis	10
2.3.2 Qualitative Research	11

2.3.3	Observations	13
2.4	Quality Assurance	13
2.4.1	Unit Testing	13
2.4.2	User Testing	14
2.4.3	Instant Data Analysis (IDA)	15
3	State Of The Art	17
3.1	Digital solutions currently available	17
3.1.1	Creating a task	18
3.1.2	Management of tasks	18
3.1.3	Accounts, roles, and teams	19
3.1.4	Miscellaneous	20
3.2	General Data Protection Regulation (GDPR)	20
3.2.1	Conclusion	21
4	Analysis - pt.1: Current System	23
4.1	First meeting with manager main takeaways	24
4.2	First meeting with employee main takeaway	25
4.2.1	Current tools used by the Living Room	28
4.3	PACT (People, Activities, Context, Technologies)	29
4.4	System Definition and FACTOR	31
4.4.1	System Definition	31
4.4.2	FACTOR	31
4.5	Rich Picture (current system)	32
4.6	Conclusion	33
5	Analysis - pt.2: The new system	34
5.1	System definition and FACTOR (new system)	34
5.2	Rich Picture (new system)	36
5.3	Problem Domain	36

5.3.1	Class Diagram	37
5.3.2	Event Table	38
5.3.3	Behaviour	39
5.4	Application Domain	41
5.4.1	Usage	41
5.4.2	Functions	46
5.4.3	Interfaces	47
5.5	Requirements	48
5.6	Final iteration of the problem statement	52
6	Product	53
6.1	Description of the application	53
6.1.1	Revision of Requirements	55
7	Design	59
7.1	System design	59
7.2	Design concerns	59
7.2.1	Criteria	59
7.2.2	Database Study	61
7.2.3	GUI Framework Study	61
7.3	Architecture	62
7.3.1	Component design	62
7.3.2	Architecture diagram	63
7.4	UI design	64
7.4.1	Sources of inspiration	65
7.4.2	The original UI idea	67
7.4.3	The wireframes	68
7.4.4	The final look	74

Contents	1
8 Implementation	81
8.1 MVC	81
8.2 Code examples	83
8.2.1 Database methods	83
8.2.2 UI methods	86
9 Quality Assurance	90
9.1 Usability Test	90
9.1.1 Exploratory Test	90
9.1.2 Assessment Test	91
9.1.3 Validation Test	92
9.2 Unit Testing	95
10 Discussion	96
10.1 Java as front-end	96
10.2 Watch for changes	97
10.2.1 Listen for update	97
10.2.2 Observer pattern	97
10.3 Future Work	98
10.3.1 Minimizing server interaction	98
10.3.2 Model translation	98
10.3.3 Features	99
10.4 Process	100
10.4.1 Gantt and back-casting	100
10.4.2 Pair coding and code review	102
10.4.3 Communication with client	103
10.4.4 Group contract	103
10.4.5 Conclusion	104
11 Conclusion	105

Chapter 1

Introduction and Motivation

This semester's project is based on the subject, "A well-structured application"[11], in the context of object-oriented programming. The purpose of this report is to document the results of the process, in which the group utilises their competencies in object-oriented software design, evaluation and design of user interfaces, and project management, to develop a functional prototype of a software solution, which attempts to solve a real-life problem of a client. The business partner for this project is the co-owner of The Living Room, Frank Zadi. The cafe is located in the heart of the Latin Quarter neighbourhood in central Copenhagen. They offer a mix of different homemade foods and drinks during the day, as well as a wide range of cocktails in the evenings 11. Other than the bar located on the upper ground floor, they offer a lounge area in the basement filled with couches and a fireplace for the cold winter months. When it is busy at the cafe and the staff are under much greater pressure, good resource management is important. Notably, The Living Room is both a cafe and cocktail bar, which means they have more tasks than a regular cafe or a regular bar. As of today, countless digital solutions are implemented to streamline daily responsibilities, such as arranging work schedules, keeping track of orders, inventory, accounting, etc., but there are still some aspects of a regular workday at The Living Room, where confusion can

slip in, bottle-necking a whole day's work.

1.1 Initial Problem

Although specifying the client company as The Living Room had set some practical limitations for the scope of the project, narrowing down the problem field was still a necessary step to reach an issue specific enough to properly address during a relatively short time frame. Therefore, the initial problem was identified following a meeting with Frank, a walk-through of the information collected, and a subsequent follow-up conversation. Regular consultations with the client were essential for achieving a mutual understanding and ensuring that the client's demands were met. The initial problem field of the project, task management, was selected, as it was deemed to be the root of several other issues, and the solutions currently implemented were the most unsystematic and ineffective.

1.2 Report Structure

The project structure leans heavily on the Software Development Life Cycle (SDLC) models; iterative and waterfall. The creation of the application and the project as a whole have primarily followed a linear process, however, certain sections (design, implementation and evaluation) have been conducted multiple times. The primary motivation behind this hybrid structure was to make up for a tight schedule while applying feedback and improving the product.

Chapter 2

Methodology

2.1 Object Oriented Programming (OOP)

At Aalborg University, the fundamental principles of object-oriented programming are taught using Java programming language as a foundation. During the course, subjects such as classes, interfaces, inheritance and more are taught and explained. These subjects are taught in a way so that they are loosely coupled, with the purpose being, that they can easily be exchanged without impacting the rest of the program. In short, *object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic.*[16] Furthermore, concepts such as problem-handling, exceptions and testing are also explored. Participants of the course are also taught to write clean and correct code, which means that the code should be easy to understand, fix and test. All of these subjects and more are taught through in-person lectures and exercises. The product was developed while leaning heavily on the knowledge gained from this course, while also using tools discovered outside of it.

2.1.1 Iterative development model

The iterative development model is used for developing a software solution in cycles and dividing the problem into smaller parts, often revising specifications, and returning to previous conclusions. The main idea of the iterative model is to learn from previous cycles and use the new knowledge in future iterations. Before the project enters the different cycles, the team must create a plan for the project. When the initial planning is done the team can start to look at requirements and then analyse and design those requirements before implementing them. After the requirements have been implemented the team must test the implementations and thereafter evaluate the entire cycle. When the evaluation is done, a new cycle begins with a planning phase of that specific cycle [20]. Another approach for a development model is the waterfall model, which is mainly focused on planning the entire project's development structure before beginning on the project itself [2]. The iterative model was used for the development of the product because it is a bit more flexible in terms of when to work on different aspects of the project.

2.1.2 Pair Programming

Pair programming is the practice that can be applied during the implementation phase of the project, during coding. With this method, two people work together to plan the logic behind a chunk of code, and while one developer writes the code the other observes or discusses better options with their partner. This way the developers get to offer diverse solutions to problems, and in the process catch errors and write simple and easy-to-understand code [9]. Another style of programming is extreme programming which focuses on the client's requirements and leads to close contact between the developers and the client [4]. Extreme programming was not used due to the client not having enough time.

Pair programming is used in the project to make sure that the whole group has a collective understanding of the program and to ensure understandable code.

2.2 System Development

Developing a new system that works is a complex course of action, in most cases requiring several revisions and discussion of many aspects. The following models and methods used in this project should not be seen as individual pieces, but rather as cogs in a larger machine, working together to produce a well-functioning system.

2.2.1 FACTOR Criterion

The FACTOR criterion is a method used to divide a system description into different criteria or to create a system description. FACTOR breaks down one's system definition into its more crucial elements, and can therefore be a useful tool when creating the system definition or defining the main elements of the system. The FACTOR criterion consists of six different criteria[25]:

Functionality: *The system functions that support the application domain tasks.*

Application domain: *Those parts of an organisation that administrate, monitor, or control a problem domain.*

Conditions: *The conditions under which the system will be developed and used.*

Technology: *Both the technology used to develop the system and the technology on which the system will run.*

Objects: *The main objects in the problem domain.*

Responsibility: *The system's overall responsibility in relation to its context.*

[25].

The FACTOR criterion process is an iterative process, meaning it is to be rewritten until the FACTOR criteria are consistent with the system definition. Small changes in the different elements can lead to significant changes in the system, and discussing variations of the elements can contribute to systematic possibilities and choices. The FACTOR criterion's main goal is to satisfy the customer and users by creating a system that aligns with their expectations. [25].

2.2.2 Class Diagram

A class diagram is a map of a system.[25] It is a model to understand the objects, which were found from the problem domain and their relationships. The class diagram provides an insight into the classes by displaying them in relation to each other and in structures. In short, a class is a template for creating objects and implementing behaviour in a system. An object is an instance of one or multiple classes [25]. The class diagram was used to gain an overview of the classes and how they relate and associate with each other in relation to the problem at hand.

2.2.3 Event Table

An event table plays a big role in modelling the problem domain. Various phenomena involved in the problem domain are abstracted by viewing them as objects and events. The objects are then classified and afterwards an event table is made. The classes can be seen on the horizontal dimension, while the events are placed on the vertical dimension. When a notation is placed within this table, it indicates which objects from a given class are involved in a specific event.

In this case, the event table helped to develop eventual use cases between the user and the program. In turn, this also leads to specifying the functions of the system.[25]

2.2.4 Behavioural Patterns

The behavioural patterns are state charts used to describe and display the behaviour of the classes in the problem domain [25]. The idea of this model is to use the events from the event table to explore the behavioural patterns in each class. The behavioural patterns will produce a number of attributes for each class. The incentive to apply this method was to comprehend the behaviour of each class and figure out what attributes should be applied, in the context of the problem. Also, behavioural patterns should be representative of reality, although some simplifying is most likely needed in order to provide an easily-digestible representation of a class's behaviour.

2.2.5 Use Cases

Use cases are a way to describe the actors' interaction with the target system and are used to analyse the existing application domain. Use cases are used due to their low abstraction level and focus on the interaction between the actors and the system. A use case is a delimited use of a part of the system and can be initiated by an actor or the target system. To find use cases, the developers cooperate with the users and look at their needs. Although a use case comes from an actual need, a use case is an expression of a solution in itself.[25] When all the relevant use cases are found and described in detail, the developers can create an actor table. An actor table is used to visualise which use cases involve which actors [25].

2.2.6 Functions

Functions are used by the system to assist actors of the system in their work. The definition for a function is *"a facility for making a model useful for actors"* [25]. Functions can be divided into different types of functions, the most common listed here:

Update functions are activated by a problem domain event and result in a change in the model's state.

Signal functions are activated by a change in the model's state and result in a reaction in the context; this reaction might be a display to the actors in the application domain, or a direct intervention in the problem domain.

Read functions are activated by a need for information in an actor's work task and result in the system displaying relevant parts of the model.

Compute functions are activated by a need for information in an actor's work task and consist of a computation involving information provided by the actor or the model; the result is a display of the computation's result.

[25].

Functions are used during the analysis to get an overview of the system from a functional point of view.[25]. The functions found in this system were divided into the 4 function types listed above, and they were given a complexity based on the difficulty of implementing them.

2.2.7 MoSCoW / Criteria

The MoSCoW method is used to prioritize the requirements of the system and consists of four different categories:

*"**Must have:** fundamental requirements without which the system will be unworkable and useless, effectively the minimum usable subset.*

***Should have:** would be essential if more time were available, but the system will be useful and usable without them.*

***Could have:** of lesser importance, therefore can more easily be left out of the current development.*

Want to have but won't have this time round: - can wait until later development."

[5].

In this project, the MoSCoW method is used to prioritise the functional requirements. All the must-have requirements form the basis of the minimum viable product (MVP).

2.2.8 FURPS+

FURPS+ is a method to find architectural requirements, whether they are implicit or explicit, by conducting interviews, looking at state-of-the-art and the analysis of the system. It consists of five different categories: Functionality, Usability, Reliability, Performance, and Supportability. The "+" represents further concerns such as design requirements, implementation requirements, interface requirements, and physical requirements. The model is used for this project to categorise the non-functional requirements [15].

2.3 Design and Evaluations of User Interfaces

2.3.1 PACT Analysis

When designing a user experience (UX), the designer must be aware of the people and activities they are undertaking, how their activities play a role in the context, and what type of technology they have at their disposal. The following four elements make up the PACT analysis[5]:

People differ in terms of physical characteristics, psychological differences and in their usage of systems.

Activities differ in terms of temporal aspects, whether they involve cooperation, complexity, whether they are safety-critical and the nature of the content they require.

Contexts differ in terms of physical, social and organizational aspects.

Technologies differ in terms of the input, output, communication and content that they support.

[5]

In the case of this project, a PACT analysis was done to assess the current situation at The Living Room, understanding where possible improvements could be made in terms of the staff and their needs. Finally, all of this was used to envision an improved future situation for both the employees and the manager during an average workday. An analysis method such as this streamlines the process of tailoring a product that resolves the needs of the subject.

2.3.2 Qualitative Research

Qualitative methods have become an increasingly prominent part of fieldwork spanning from humanistic and social sciences, where they are more commonly used, to natural sciences. The qualitative methods seek to understand, describe, and interpret the world and human lives by researching human perspective, experience, and perception. In a way, the research is done from within human life rather than describing what is visible from the outside, making the research more elaborate and adequate. Contrary to quantitative methods, qualitative methods do not use numbers and quantities to describe a particular issue. The qualitative methods take advantage of, among other things, spoken statements and reports from individuals who represent certain views on situations to understand the underlying workings of, for instance, societal relations or other relations in general. One of the commonly used methods for retrieving qualitative data is interviews [37].

Interviews

An interview shares a lot of similarities with an ordinary conversation, with the main difference being that there is a specific purpose for an interview [36]. It is

a method of gaining information with human interaction. In this project, it was beneficial to understand how satisfied a business was with its current workflow. The first step of planning an interview is to choose its structure. Every interview requires preparation, and each structure has its advantages and disadvantages. Each case requires a different structure, and choosing the right one is essential for gathering relevant information. Interviews are often conducted with two people; an interviewer and an interviewee. It is also common to interview subjects in groups. The group approach is often used in social science research, which is why it was not used in this project. Interviews can be either unstructured, semi-structured, or structured. An unstructured approach to an interview would be a good choice for open-ended interviews; for instance, if the interviewer is curious about the interviewees' first impressions of a new interactive design feature. On the other hand, if the interviewee is supposed to share their opinions about a particular design feature, such as a web page, a more streamlined approach, like a structured interview, might be the better choice. The semi-structured approach is a combination of structured and unstructured approaches. It was also one of the two interview approaches that were used during the conversations with the client.

Unstructured interview

Unstructured interviews, also called open-ended interviews, are one method of gathering information through an interview [36]. With a structure like this, there is almost limitless depth to an interview, since questions are not limited to simple "yes" or "no" answers. Answers can be as detailed or brief as one wishes, and the conversations can be steered both by the interviewer and the interviewee. The responsibility of the interviewer is to make an agenda of the main topics that should be covered, so as not to lose structure in the interview if it ever deviates from the intended topic. During the interview itself, there should be a balance between making sure that the questions and answers received are relevant, while

also being able to follow new lines of inquiry that were not necessarily intended.

The unstructured approach was used during the first interview with the manager of The Living Room, as there was a need to be open-minded and exploratory, while also giving a good first impression by having a relaxed structure to the interview.

2.3.3 Observations

Observations are another way to collect qualitative information. When observing, one takes note of anything eye-catching regarding the problem. Observations are made in the context where the problem unfolds and can be helpful when it is difficult for people to put the problem into words. Other times interviewees take their routines for granted and therefore forget to describe them. Again, observations can help cover any information lacking in this aspect.[5] For this project, observations have contributed to better modelling and understanding of the context of the problem.

2.4 Quality Assurance

2.4.1 Unit Testing

Unit testing concentrates on testing "units" of the program. A unit is a piece of code or method that is present in the system. The unit is tested separately from the program, meaning it is ignoring the rest of the system when being tested. When performing the unit tests, the developers ensure that the desired output is returned from the unit often using dedicated testing frameworks such as JUnit and assertions. All the unit tests for this system were based on the return statements of its methods. The unit tests were done before committing to the actual program to verify the newly written code [35].

2.4.2 User Testing

User testing is a subbranch of usability testing and focuses on the end user's experience with the system. The test is performed by participants who are a representative of the end users. The developers must prepare a set of tasks for the participants to complete and observe while taking notes of how they perform. When performing the tasks, the participants should follow the thinking-aloud procedure. This will help the observers of the test to get a better understanding of the logic and way of thought of the participants when they try to solve the given tasks. User testing can be done as a field- or lab test. They both have their advantages and disadvantages e.g. a disadvantage of the lab test is that it can be unnatural. Inspection is another form of user testing where an expert in UX design or usability reviews the design. This was not done for this project as it would be too expensive compared to the tests performed with the test participants. A third method is to gather data on system performance once it is deployed. Since the system was not meant to be deployed officially, this method was not an option.[34] Usability tests can be done with different approaches in mind and at different stages of development. These three approaches are called exploratory, assessment, and validation [34]. The exploratory test is done early in development. It is usually done with low-fidelity wireframes to iterate different concepts and discuss them quickly. An assessment test comes after the exploratory test as it tests the usability of the product with the concepts that were summarized from the exploratory. This test can include slightly more interactivity compared to the previous one. The final usability test is the validation test, which is done late in the development cycle and follows up on the research done earlier in the process with the exploratory and assessment tests. The validation test includes a product that is late in its development cycle, and therefore more complete than the previous designs. The test is slightly faster and more cost-efficient compared to the previous ones. These three approaches were all used in cooperation with the

staff at The Living Room, to ensure a good user experience. Different iterations of the design were presented, tested and discussed, all at different stages in the development cycle.

2.4.3 Instant Data Analysis (IDA)

IDA is a technique for reducing the efforts spent on analyzing data when evaluating a usability test.[23] By using this technique, it is possible to conduct an entire usability evaluation in a day. It is not necessarily the most optimal technique to uncover every potential problem, but it is efficient in considering its speed. It works by quickly identifying the most crucial problems regarding the usability of a software system. IDA is used immediately after the usability test has been conducted, and it is recommended to have 4 to 6 think-aloud sessions, that are structured in the same way as the aforementioned user tests.2.4.2 The host and observer of the usability test articulate and discuss the most critical usability problems, they noticed during the sessions with the participants. Afterwards, it is recommended to rate the severity of these issues. This can range from cosmetic to serious, to critical, and finally catastrophic as seen in figure 2.1. The identified problems can then be inserted into a problem list, which makes it easier to see the number of issues.

	Delay	Irritation	Expectation vs. actual
Cosmetic	< 1 minute	Low	Small diff.
Serious	Several minutes	Medium	Significant diff.
Critical	Total (user stops)	Strong	Critical diff.

Figure 2.1: Problem types. The missing "catastrophic" type stands for a failure from which recovery is not possible.

[25]

Chapter 3

State Of The Art

Considering the specificity and scope of the problem at hand, there are currently no publicly available software solutions, which could be considered applicable. There are, however, similar task-administration tools catered towards project management. Although the focus of these applications differs from this project's, some of the problems they attempt to tackle are quite similar to the ones this project is facing. In this section, the most relevant available products are discussed and sectioned into specific functionalities. The choice of products featured and discussed in this section was primarily based on applicability to this project's specific goal: daily task management. The products were tested as if they had to be used in the context of The Living Room.

3.1 Digital solutions currently available

During the search for relevant digital solutions currently available, a handful of popular project management solutions were found. The most relevant ones were Connecteam [10], Monday.com [27], Trello [41], and Microsoft Azure DevOps [13]. Each product approaches project management in a slightly different way but offers an overall similar set of features with diverse focus points.

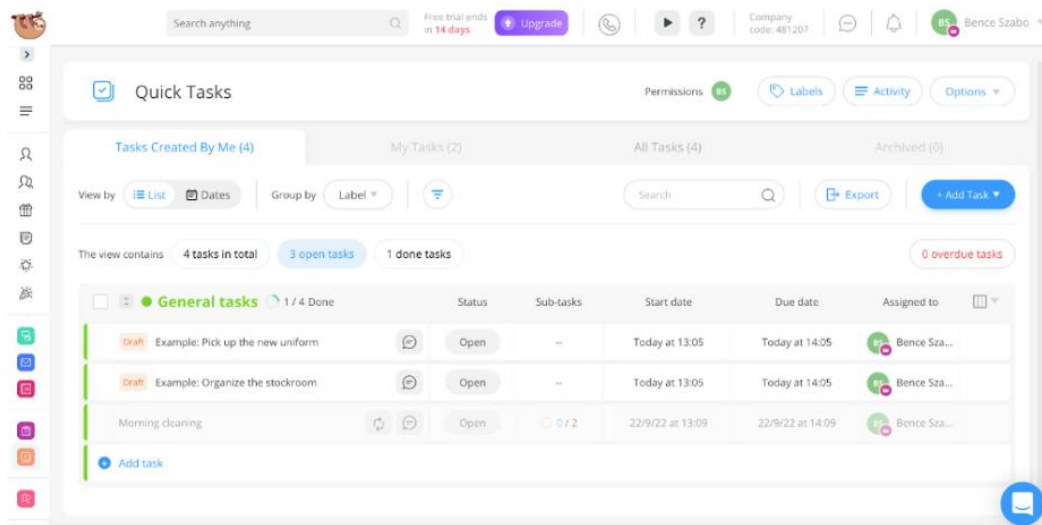


Figure 3.1: Overview of tasks in Connecteam

3.1.1 Creating a task

When creating a task, the following fields were available across the board: Title, description, and assignees. Connecteam and Monday also offer assigning a degree of urgency/priority to the task, as well as a frequency (daily, weekly, monthly), or a deadline. Monday also offers sub-tasks, in case a larger task is to be broken down into smaller assignments.

3.1.2 Management of tasks

Throughout all the apps, depending on one's role, tasks can be freely edited, or comments can be added. In both Azure and Trello, the status of tasks is also manageable. For instance, a task can be "active" or "resolved". The history of a project and archived tasks are also traceable.

→ Edit task

Task details Sub tasks

Assign to • Bence Szabo × + Add users

Location München, Tyskland

Frequency One-off task Recurring task

Starting from 9/21/2022

Daily Every 1 Days at 13:09

End repeat After 5 tasks

☐ Notify me when this task recurrence starts

Due 60 Minutes After the task's start time

Labels General ta... ×

Publish task Save draft

Figure 3.2: Editing a task using Connecteam

3.1.3 Accounts, roles, and teams

The hierarchy of the users is in essence broken down into administrators and regular users. Administrators are responsible for creating tasks, managing their properties, and assigning users to them, while regular users can update their status and write comments. Connecteam also uses a collection for regular users, called "teams", to give the admin the possibility to hide irrelevant tasks that do not belong to the group of users. In a workplace where the responsibilities of the employees can be broken down into larger partitions, this feature can provide a less overwhelming overview.

3.1.4 Miscellaneous

All of these software solutions are web applications primarily designed for desktop computers. Since they are project-based, these applications are not designed for the day-to-day running of an establishment. Timelines in these apps are designed to depict progress on a monthly basis, rather than give a daily overview. Interacting with these interfaces and updating statuses are also time-consuming activities, as they are designed for an office setting. The user interfaces are (with Trello being an exception) rather complex, considering most of the users would only use a small handful of functionalities.

3.2 General Data Protection Regulation (GDPR)

To preserve and store personal data, one must either have legal authority or a legitimate purpose [6]. Examples of what legal authority is in this scenario are usually one of the following:

1. You have consent from the person whose data you are storing
2. You are required to do so by the law
3. It is needed to uphold a contract or promise

Deciding on a legitimate purpose is difficult, but there is a popular example regarding job applications. If your business receives job applications, saving those applications for a year is considered a legitimate purpose. The reason behind this is that the chances of that applicant still being available after a year is low. Another requirement is that the users need to be able to delete their data from the platform storing their personal data. This also needs to be available if a law authority decides that the business no longer has a legitimate purpose for storing the user's data.

But what exactly is personal data? The Danish Data Protection Agency splits personal information into three different types; non-sensitive, sensitive, and lawful information [12]. Non-sensitive personal data covers information that does not necessarily put the person at risk. Examples of this could be a name, address, age and education. The aforementioned job application would also fall under this category. Sensitive personal data covers information that could put the person at risk. Subjects like race and ethnicity, political views, sexual orientation, and medical information are all important parts of this category. Lawful information covers criminal records et cetera.

The product of this report is only expected to store the user's name, the field of work they are assigned at their workspace, their phone number and email. This should give the manager a legitimate purpose to store the information, as it does not put any of the employees at risk necessarily. As of now, the manager already possesses all of the data mentioned. Last, but not least, there is also a method that allows for the data of an employee to be permanently deleted from, not only the application but also the database where it is stored.

3.2.1 Conclusion

As mentioned before, these are solutions to long-term project management, however, that does not mean there is not resourceful information to be gathered from them. The fields required for creating a task in these solutions would also be beneficial for The Living Room, as fields like these were present, not only in the current solutions of The Living Room but they were also requested by the owner. Both of these topics will be explained in depth later on. Being able to edit and leave comments on tasks like in Azure or Trello would be a viable alternative to calling/texting the assigned person, which is what The Living Room currently does frequently. Having a task status ("active" or "resolved") would be a good way to structure the database storing the tasks, as it increases comprehensibil-

ity. Using a similar hierarchy with an administrator and a regular user would make the product familiar for the users, as it gives them the same solutions as their current system, but in a more streamlined and efficient way. Together with GDPR, all of these features, and more, would be valuable for the product created for The Living Room and will be considered in the requirements later on.

Chapter 4

Analysis - pt.1: Current System

The analysis section of this report is divided into two subsections; the current system, and the new system. The current system section encapsulates an analysis of the currently deployed system, as well as its flaws and challenges. The new system section contains a description and analysis of a new system serving an identical purpose but providing a more optimal solution. This will be the product of this project. The main motivation behind structuring the analysis section in this fashion was to clearly separate old and new systems for the reader. The old system and the observations done on it provided a lot of valuable inspiration for the way the new system was defined.

The project's degree of success may be judged on the level of improvement between the two solutions. The evaluation of both functional and non-functional requirements as well as usability are expected to give a satisfactory conclusion regarding this.

Starting out with an understanding of the fundamentals of the problem domain is essential in order to begin a problem-solving process. Note, that a perfect understanding is arguably impossible to achieve, but it is expected that the degree

of comprehension is refined through good communication with the client. Therefore, an analysis of the situation was conducted, with the help of two interviews, one unstructured and one semi-structured, getting both the manager's and an employee's insight into task management. A field investigation was done by observing the everyday work environment of The Living Room over several days, in order to gain a better understanding of the context.

4.1 First meeting with manager main takeaways

The Living Room is a cafe by day and a cocktail bar by night, which results in an extended list of very different tasks. Their current solutions for task handling are very basic and their tasks are usually given out by word of mouth, SMS, e-mail, and Google Sheets documents. On top of this, they also write tasks down on physical paper and check the tasks off there whenever they are done. Reminders of these tasks are also dependent on memory alone, or if someone reminds others, which would also be in person or over text messages. Some tasks can be done by anyone, but others need to be done by specific people based on their expertise. Specific tasks need to be filled out in a form on a computer, and the application to fill out the form has to be paid for. The level of urgency of a task is only indicated by the tone in which it was given, and there are often descriptions missing from these tasks. All of this leads to a scattered structure since there is no overview for either the manager or the employees.

Ideally, the owner would like to manage tasks digitally from a single platform. They would want the tasks to be given out uniquely and on a given frequency, such as daily, weekly etc. Some tasks could have conditions, such as multiple steps. There should also be a way to assign tasks to a specific employee. A feature involving due dates is important, which implicitly touches on the possibility of handling overdue tasks. Some tasks are also going to need a specific

form in order to be completed. Examples of this are the logging of temperature, which is a requirement by The Danish Veterinary and Food Administration (Fødevarestyrelsen). Although a number of problems and suggestions were made by Frank, only a handful will be addressed, since the scope of the project must be narrowed down to provide an optimal solution. All in all, there is a clear desire for a uniform overview of tasks, in which both manager and employee can see what is going to be done and what has been done during the day.

4.2 First meeting with employee main takeaway

It is essential to gather information from a source different from the manager to establish a more objective comprehension of the problem domain. Therefore, after a basic understanding of the work environment had been acquired, a semi-structured interview was conducted with an employee of The Living Room. Additionally, most employees are also users of the solution, so they are to be considered important stakeholders in this project. The thematic structuring of the interview is the following:

- General information about the interviewee
- Experience with task management and daily life at The Living Room
- Currently applied solutions for task management and issues

The structuring is done this way in order to confirm the relevancy of the person in question, acquire a better understanding of the current work environment and types of assignments, and learn about the tools currently applied and potential problems connected to task management.

All quotes provided in this section are direct transcriptions of the interviewed employees' statements.

An important new discovery was the significance of documenting the completion of daily tasks: *"If we have a really busy day, then we often do not get to prep as early in the day [or before closing time] (...)". "[If someone takes over my task] we would say it verbally."* The quotes from the employee imply that if the cafe is busy, the closing and early morning prep tasks do not get completed and the only way to know about the uncompleted tasks is by word-of-mouth. For that reason, it is not certain that the information about the tasks gets delivered to the right employee. Therefore, communication between employees is not an issue, however, a lack of overview regarding remaining and already active assignments can cause problems - *"[If I did] simple everyday assignments, I would not tell anyone. My colleagues would assume it is solved. But sometimes I would include a little comment."* . As for critical tasks, which require confirmation, an employee would text or call the manager to tell if the task was done or not and simply tell the rest of the staff.

As for the daily tasks, there are clearly a lot of frequent tasks. However, the completion and deadline of the tasks can vary with regard to other spontaneous assignments and the number of customers: *"I feel that there is quite a lot of routine in when the different things are done, and what can make schedule day vary is the number of customers, and how busy we are."* . The role of employees is also very fluid, so everyone should be able to complete any frequent task.

Further observations of The Living Room

Observations were as earlier stated used to get a better understanding of the work environment. The observations added a few new views of how the employees work and under which circumstances. There were about 3 bartenders working at the bar at all times, and one collecting dishes every 30 minutes. There is quite a lot of noise and music, which adds to the busy work environment. Verbal communication is frequent, and a standard iPad (2022) is often passed around for documentation purposes.



Figure 4.1: The bar at The Living Room



Figure 4.2: The inside of The Living Room, with the bar upstairs and seating areas on and below the ground floor

4.2.1 Current tools used by the Living Room

To achieve a better understanding of the current situation at The Living Room, the owner shared documents that the staff currently use for task management. This material was largely helpful for analysing the situation because these are essentially the tools that are expected to be replaced by a more optimal digital solution.

The first document is called *LR Weekly Tasks.xlsx* 11 and it contains static tasks that need to be done on a weekly basis. The document consists of a table, which is created in Google Sheets and is viewed online, not printed out. The table includes the different tasks with the title of the task, which employee has the responsibility for that task, and the phone number of that employee. This is a very basic solution and could be improved. There is no way to let your colleagues know if you have done your task or if there is an issue. It is possible to do this in Google Sheets, but it is done with the help of custom scripts. This is not ideal as it is highly repetitive and poorly maintainable, and should therefore be taken into consideration going forward.

The second document is called *LR Hygiene Forms.xlsx* 11 and is used to record the degrees of different areas of the establishment and various food items. This is a requirement by The Danish Veterinary and Food Administration and should therefore be formulated and formatted following strict guidelines. Possible ways of improving this could be reminders if it has been done, indicating possible mistakes, or possibly incorrect or illegal values. Beyond that, it could also notify the user or administrator if the temperature recorded is abnormal.

The last document is called *LR Bar Routines.docx* 11. It is a sheet that contains all the daily routines, and is printed out and filled out physically throughout the

day. There are various tasks divided into three sections depending on the time of day they must be done. When a task is completed, the staff member that did the task has to sign their initials on the list of tasks. There is no way to tell if another staff member is currently doing a task, which could be beneficial to the overall efficiency of the establishment. Beyond this, there is no way for an administrative role such as a manager, to update this list of tasks dynamically. A staff member would also be completely oblivious to any missing tasks from prior days if the physical piece of paper was not documented well.

4.3 PACT (People, Activities, Context, Technologies)

Now that there has been established a general understanding of the users, their daily routines, and the tools at their current disposal, a PACT analysis can be used to put these elements in context and create a briefer summary in order to gain a better overview and start designing a new solution.

People: The system consists of two main groups; 25 employees and the manager, Frank. Employees can also be broken down into bartenders and cleaners based on their field of expertise. They are all at least partially experienced in the service industry and in the general responsibilities at the cafe. They are comfortable with operating computerised systems. A number of the employees working at The Living Room are students who work on the side. There might be a slight language barrier due to the difference in the manager's and employees' native languages. Some people might have slightly bad eyesight or be colourblind, which could have an effect on the pace at which they interact with digital user interfaces. People can also have different perceptions of the urgency of tasks. There is also a hierarchical difference between the manager, Frank, and his employees.

Activities: The main purpose of the activity is to administrate tasks in The Living Room on a daily basis. The activity is mostly consistent but may include sporadic urgent tasks. Tasks can be everything from cleaning or completing special orders, to fixing a coffee machine. The activity includes several steps done in a specific order; a task is given to the employee by the manager, the task is either completed or failed, and feedback is eventually given to the manager. Some tasks given by the manager are considered routines, which means that they do not require direct feedback. Some activities require multiple employees or a specific employee.

Context: The activity takes place indoors at The Living Room and happens under relatively calm circumstances for Frank; either in his personal office or at any sitting place. The bartenders, however, can be under a lot of stress depending on the number of customers they have to serve, and verbal communication can be difficult due to the general noise and music playing. The employees support each other during their shifts even though they have different roles or responsibilities. The employees responsible for morning shifts clock in at work one hour before the cafe opens to complete regular morning routines and finish any leftover tasks from the day before. The bartenders also partake in cleaning and collecting used dishes from the tables. Meanwhile, other bartenders take care of customers, which is their main concern. When there are no customers present, the employees usually have some tasks to do, which the manager provides a list for.

Technologies: The manager enters the routine-based tasks into a self-made form, and prints it. The employees can mark the tasks on the paper as completed with the use of a pen. The manager communicates infrequent, spontaneous, or especially important tasks with the employees via SMS or email when he is not at work.

4.4 System Definition and FACTOR

4.4.1 System Definition

A system used at The Living Room by the manager to administrate the staff's daily and weekly tasks. The primary purpose of the system is to support the manager in organising daily and weekly tasks for the staff to resolve during their shift. This is done by using a template printed out on a piece of paper, containing the overview of the tasks and a tablet which is implemented to update the tasks within the template if needed. Otherwise, the manager calls or sends an SMS if urgent tasks, or any changes or unexpected events, occur. The staff's only relation to the system is when they are marking off their resolved task(s) by signing the correct field with their initials. The manager makes the system in cooperation with the employees at The Living Room, and the system is accessible without needing any IT skills.

4.4.2 FACTOR

Functionality: The functionality of the system is to provide the staff members with a physical piece of paper that contains a list of daily and weekly tasks for them to complete and fill out.

Application domain: The system is to provide a reliable checklist to assure that the tasks at The Living Room are completed, and in turn, give a better overview for every staff member. The system is used by the manager when adding or removing daily or weekly tasks from the given lists. The employees' relation to the system is that they have to sign the document when the given task has been done.

Conditions: The system is made by the manager at The Living Room in cooperation with the employees of The Living Room. Users use the application

with varying levels of work-related skills.

Technology: The system is used physically at the Living Room in the form of three different documents; two of the documents are on a piece of paper and one document is accessible on a tablet. In case lists need to be updated, the manager would have to print a new piece of paper and inform the employees or update the document on the tablet.

Objects: Task, task list, employee, manager, The Living Room.

Responsibilities: The system supports the administration of daily and weekly tasks at The Living Room by providing an overview of the tasks.

4.5 Rich Picture (current system)

This section contains a rich picture describing the current situation at The Living Room as shown in figure 4.3 below. The format is very simple, but it leads to miscommunication between the employees and the manager. The main takeaway is that the rich picture shows conflicts regarding the lack of a uniform platform and structuring when assigning a task to an employee. More specifically, the manager communicates through too many platforms, which makes the logging of information difficult.

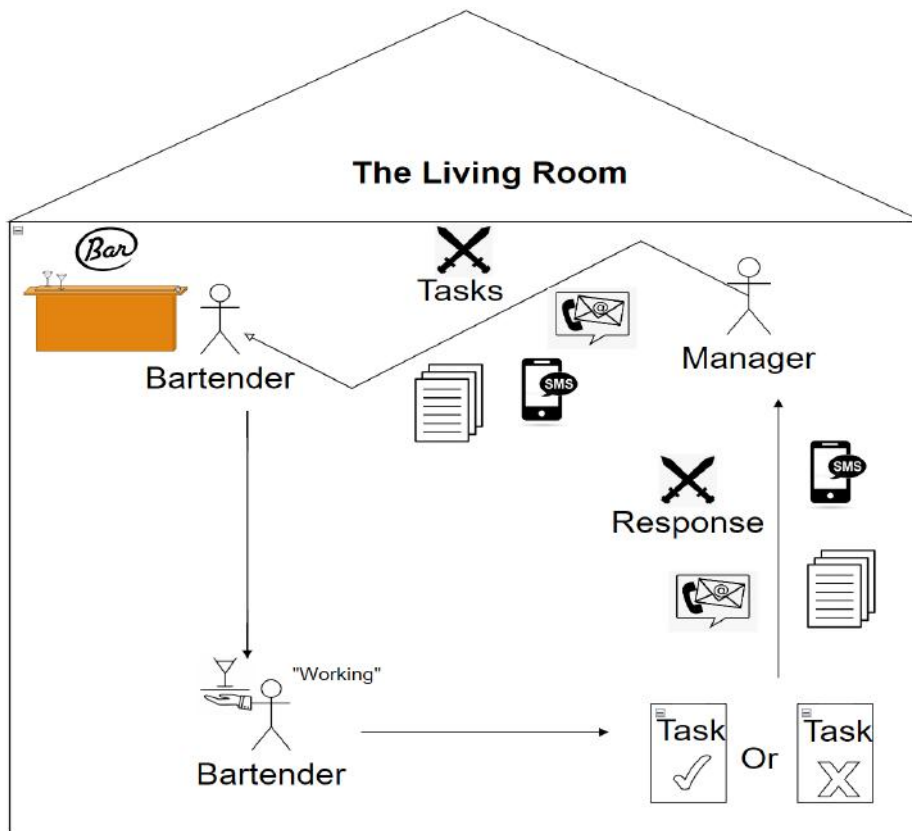


Figure 4.3: Rich Picture For The Current System

4.6 Conclusion

From this analysis, it can be concluded that the current system has both its faults and advantages. The advantages are its simplicity and straightforwardness. The faults belong to the multiple platforms that the system uses, which can easily lead to miscommunication. With this new comprehensive understanding of the problem at hand, there is a clear focus on what an improved system should try to incorporate.

Chapter 5

Analysis - pt.2: The new system

Following the analysis of the currently deployed system, the next step is to attempt to define a new system. The new system's responsibilities and other aspects are likely to be similar. Yet, the solution itself should be more optimal considering the problem field. Beware, that the new system and its details are frequently updated and adjusted, taking knowledge gathered through the project and development into account, as well as feedback and testing with each iteration.

5.1 System definition and FACTOR (new system)

System Definition: A computerised system used to administrate the employees' regular daily routines and weekly tasks defined by the manager at The Living Room. The system should primarily be an administrative tool but secondarily serve as a communication medium between the manager and the employees. The system should be available on both a PC and a tablet and it should also be able to function in a busy work environment.

Functionality: The functionality of the system is to administrate tasks between the manager and the employees at The Living Room and register the information

about the tasks. The system also supports communication between the manager and the employees.

Application domain: The system's main purpose is for the manager to administer the employees and their tasks at The Living Room. The system is used by the manager when defining either regular daily or weekly tasks or communicating work-related issues at The Living Room with the employees. The employees' relation to the system is when they have to report back if the given task has been done or if there is a work-related issue.

Conditions: The system is developed by Group 1's developers in cooperation with the co-owner and manager of The Living Room, Frank Zadi, and with the help of a handful of his employees. It may be necessary to resolve conflicting requirements between Group 1 and Frank. Users of the application may possess slightly varying technological expertise and IT skills.

Technology: The system is to be provided as a desktop application and running on the manager's computer and one or more tablets placed at The Living Room. It communicates through a client-server architecture that handles the tasks and their information using models. If the client-server architecture fails, all the necessary information is available in a database.

Objects: Task, task list, employee, manager, The Living Room (Cafe).

Responsibilities: The system supports the administration of daily and weekly tasks at The Living Room. This is done by providing a complete overview of the tasks and the assigned employees. It also facilitates the manager and the employee's ability to comment on the tasks.

5.2 Rich Picture (new system)

This section contains the rich picture for the system as shown in figure 5.1 below. The main difference between this and the rich picture of the current situation is that the manager no longer has to be at The Living Room to assign tasks or help with issues regarding tasks. The new rich picture also makes the administration and overview of the tasks more manageable.

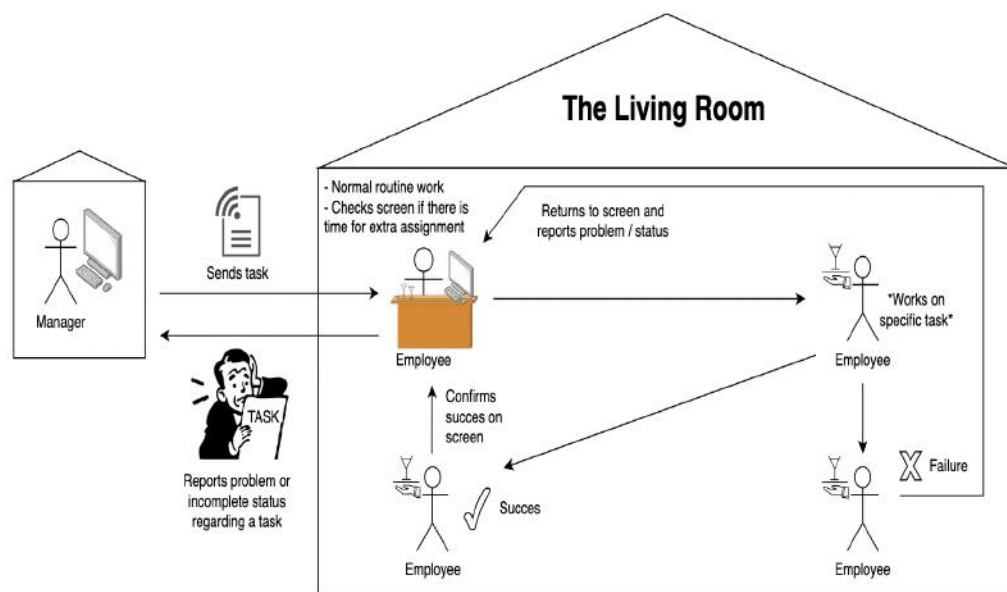


Figure 5.1: Rich Picture of The New System

5.3 Problem Domain

Following the FACTOR analysis and rich picture, an analysis of the problem domain can be made. The problem domain has two principles which should be followed, the first one being: *Model the real world as future users will see it.* [25] The second principle is: *Get an overview first, then supply details.* [25] When following these principles, many details and alternatives to the final system will occur. For readability reasons, only the concrete system and how it was made will be

presented in the report.

5.3.1 Class Diagram

This section contains a description of the structural relationships between the different classes and their different fields. The class diagram can be seen in figure 5.2.

In order for the system to support task management at The Living Room, the system is segmented into four classes: cafe, which is The Living Room; person, an abstract class divided into two classes employee and manager; Task list, containing all tasks; and lastly task, containing all properties of an individual task (class-specific states and behaviours are later defined in the component-design section 7.2).

The classes employee and manager are depicted as a generalisation from the abstract class person, because they are both a person as seen in the figure, but are expected to contain unique attributes.

The system has some aggregations between the classes for example a cafe has a person since the person is working at the cafe. A task also has a person as an aggregation because a person can be assigned to a task and a task cannot be assigned to a person (a person belongs to a task). A task list has a task as an aggregation due to every task belonging to the task list.

Lastly, the system has some associations: cafe and task are connected, as all tasks belong to a specific cafe. Without a cafe existing, tasks would not belong anywhere, therefore an association is required.

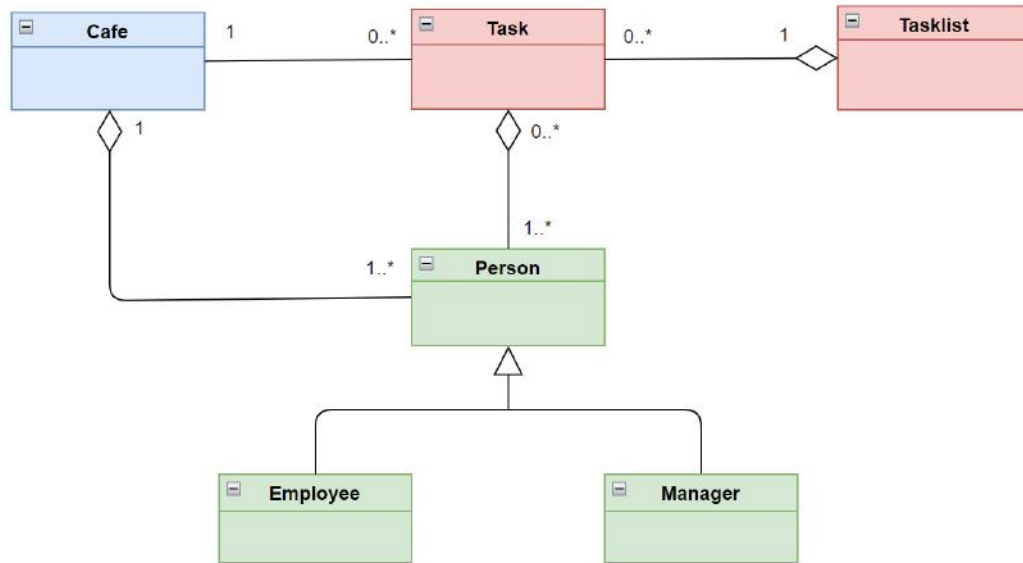


Figure 5.2: Class Diagram

5.3.2 Event Table

The event table is presented in figure 5.3 and shows which events each class in our system is able to participate in. As seen in figure 5.3 the classes employee, manager and task are the main classes of the system due to all the events they are involved in. The task class is involved in all the events except employed and fired/resigned. The task list class is involved in the events involving changing the status of the task or displaying it. The cafe class is only participating in the events concerning employing and resigning employees.

	Employee	Manager	Task	Tasklist	Café
Defined		*	*		
Assigned	*	*	*		
Completed	+	+	+	+	
Cancelled		+	+	+	
Reassigned	*	*	*		
Employed	+	+			+
Fired / Resigned	+	+			+
Reminder sent	*	*	*		
Problem reported	*		*		
Displayed			*	*	

Figure 5.3: Event Table

5.3.3 Behaviour

This section provides state-chart diagrams for the classes in the system along with a description of the states and transitions.

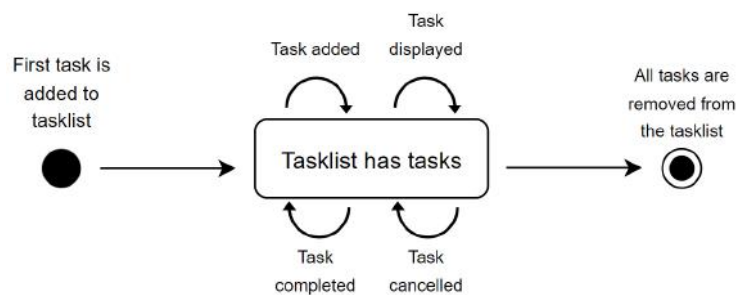


Figure 5.4: Statechart diagram for Tasklist

As seen in figure 5.4 the task list is instantiated when a task is added to it. The task list has four event loops: task added, task displayed, task cancelled and task

completed. All these events can occur while the task list contains a task. The task list reaches its final state when it is empty.

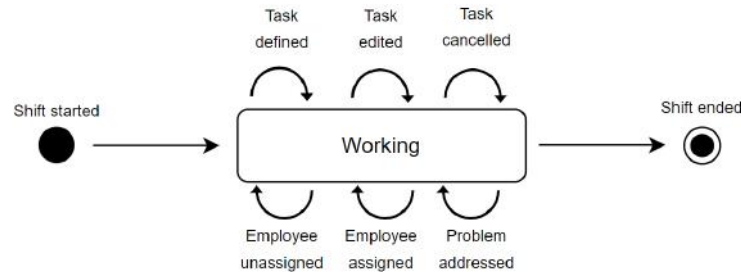


Figure 5.5: State-chart diagram for Manager

As seen in figure 5.5 a manager can have an initial state when their shift starts. Afterwards, they enter the state "Working" which has the following six event loops: Task defined, Task edited, Task cancelled, Employee unassigned, Employee assigned and problem addressed. The manager gets to the final state when their shift ends.

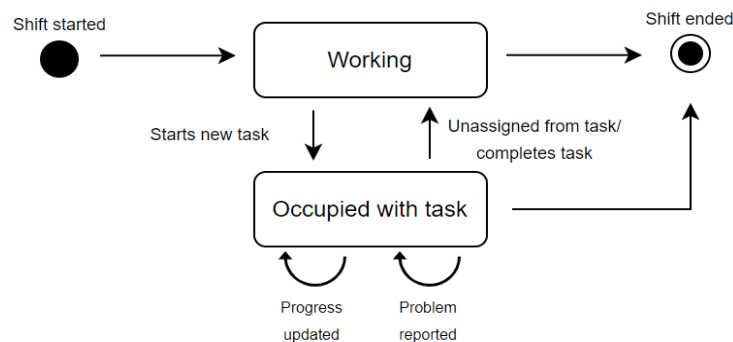


Figure 5.6: State-chart diagram for employee

As seen in figure 5.6 an employee can have an initial state when their shift starts. After the shift has started they move to the state called working, where they can enter the state "Occupied with the task" through the event called "Starts new task". The "Occupied with a task"-state has two event loops called "Progress updated" and "Problem reported". The employee can get back to the state "Working" if they are either unassigned, complete their task, or their shift has ended.

5.4 Application Domain

The following section contains an analysis of the application domain for the system.

5.4.1 Usage

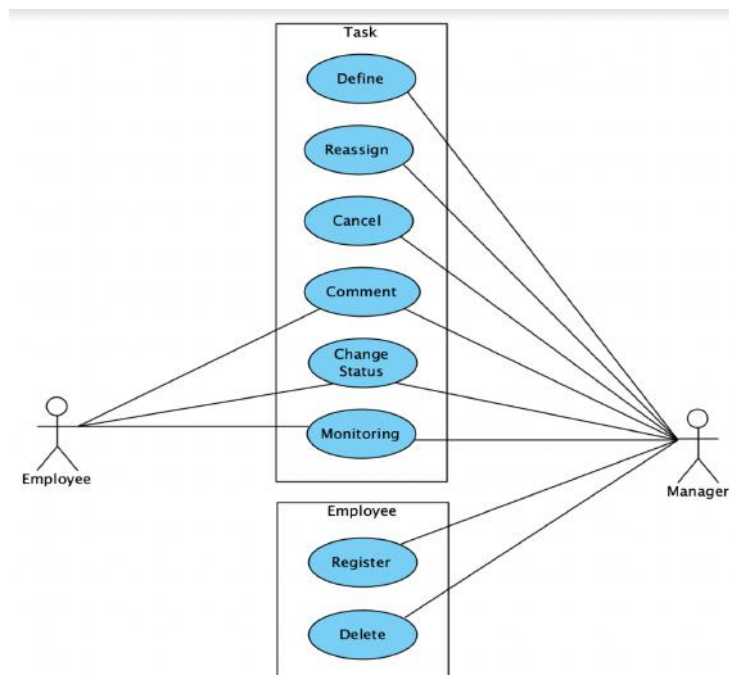


Figure 5.7: Use Cases

This section explains the actors and the use cases they participate in. The actors will be described using an actor specification and the use cases with a use-case specification.

The system has two actors: employees, who have tasks in the system that they must give feedback on, and managers, who use the system to administer the tasks and employees. Below are the actor specifications for the system.

Employee:

Goal: A person who works at The Living Room as a bartender or cleaner. The employee needs to be able to work, which includes doing their assigned daily and weekly tasks and serving customers.

Characteristics: The system has 25 employees who can change the status of tasks and report potential problems.

Manager:

Goal: A person who works at The Living Room as a manager. The manager's primary purpose is to administer the system which includes defining and assigning tasks.

Characteristics: The system has only one manager who can make new tasks. The eight different use cases are listed below.

Define and assign a task:

The manager can define a new task when on the page displaying the overview of the tasks. When the "add Task" button is clicked, a popup window appears. The window contains fields for all the information needed to create a new task. Some fields are optional. When the form is filled the manager clicks the "Submit" button, and the new task will be visible in all task overviews for both manager and employees.

Cancel a task:

Cancelling a task can be initialised by the manager when the task is not relevant anymore or there are no employees that can do the task. The manager is on the "Manager" page and presses the task that shall be cancelled. The information about the task is opened and the manager presses the "Delete" button. The manager is redirected back to the "Manager" page and the task is deleted from the view and database.

Report problems regarding a task:

An employee can report a problem if a question or unexpected issue concerning a task occurs or when a task is not completed yet. If the task is not completed, the employee can add a comment to the task to tell the manager or other employees why it was not completed. The employee opens the system in the employee view and then navigates to the task in question. The task overview page is opened and the employee presses the commentary field. The employee writes the comment and then presses the button “Comment” to add it to the task. The manager then receives a notification about a new comment. When the notification is pressed, the manager is redirected to the given task’s information page. There, the manager can provide a solution, if needed, in the form of an answer to the comment, in the commentary field beneath the original comment. The comment is saved in the same manner as for the employee. The manager can also add a comment to a task by pressing on any given task on the overview page of the application.

Reassign a task:

Reassigning a task (mostly relevant considering tasks with regular occurrence) is initialised by the manager when an employee cannot complete a task assigned to them. The manager presses the task needing a reassignment. The task’s information is opened and the manager can edit the field “Assignee”. There the manager finds a certain employee with the relevant abilities. The manager presses the “Submit” button on the task and is redirected to the “Manager” page.

Monitor a task:

The manager and the employees can monitor the status of the tasks in the different overviews/calendars. Daily and weekly overviews are available to everyone while the all-tasks overview is available exclusively for the manager, which gives an overview of unresolved, new, upcoming, and pending tasks. When a user

navigates to the “History” tab, the user can see the completed tasks displayed in a monthly overview.

Change status of task:

The employees and the manager can change the status of a task by marking it as complete or providing a percentage of completion. When the task is completed the "active"-property of the clicked task is set to "inactive" and is only visible in the "History" tab.

Register a user:

Registering a user is only available for the manager and is started when a new employee is hired. The manager navigates to the “Manage Team” page from the “Manager Overview” page. Then the manager presses the “Add Team Member”-button to fill out a form about the new employee. The form consists of the fields: Name, Role, Phone number, and Email. Next, the manager saves the form and gets redirected to the “Manage Team” page where the added employee appears in the list of employees.

Delete a User:

Like registering a user, deleting a user is only available for the manager and is started when an employee is fired or resigns. The manager navigates to the “Manage Team” page from the “Manager Overview” page. Then the manager presses the “Delete Team Member” button to delete the employee. The employee will disappear from the list of employees and the system’s database.

Actor Table

An actor table of the system can be seen in figure 5.8 and shows the interactions between the actors and the system.

	Use Cases / Actors	Employee	Manager
Task	Define		X
	Cancel		X
	Report problem	X	X
	Reassign		X
	Monitoring	X	X
	Change Status	X	X
User	Registering		X
	Deleting		X

Figure 5.8: Actor Table

The employees participate in 3 use cases: reporting a problem, monitoring, and changing status. Comments are added to a task if the employee reports a problem. Monitoring happens mostly when the employees complete their current tasks and start a new one. The employees change the status of tasks if they complete them.

The manager is involved in all of the use cases. Defining a task is done when a new task occurs and the manager has to define and add it to the system. A task is cancelled if it is not relevant anymore. The manager is involved in reporting a problem because they are the one to provide the solution. Reassigning an employee is done when the manager wants somebody else to do the task, or just make it available to everyone. The manager monitors the tasks to get an overview of which tasks have and have not been done. Changing the status of a task is done if the manager does not think it is necessary anymore, or if they have done it themselves. Registering a new user is done when the manager hires a new employee. Deleting a user happens when an employee resigns or is fired.

5.4.2 Functions

Functions are derived from the use cases described in the previous section. In the functions table 5.9, all the functions are shown in the column on the left by function name. The middle column determines which of the four types of functions, the given function is. The column to the right shows the complexity of each function. For example, the *Create Task* function is an update function as creating a new task will change the state of the model. This function is deemed medium complexity as the function will have to communicate with the database and it creates a new object [25].

Function	Type of Function	Complexity
Create Task	Update	Medium
Edit Task	Update	Simple
Delete Task	Update	Medium
Add Comment	Update	Medium
Delete Comment	Update	Medium
Edit Comment	Update	Simple
Display Tasks	Read	Complex
Create Employee	Update	Medium
Delete Employee	Update	Medium
Query Completed Tasks	Read	Complex

Figure 5.9: Functions Table

The functions vary in complexity from simple to medium to complex. The functions "Edit Task" and "Edit Comment" are of simple complexity because they set a new value to an already existing object in the program. The other medium functions like "Delete Employee" and "Create Employee" are of medium complexity for the same reasons as the function "Create Task" mentioned earlier. Lastly, the complex functions "Display Tasks" and "Query Completed Tasks" are

the only functions, which are read functions. That is because they satisfy the need for information about the problem domain to the application domain. "Display Task" shows the tasks in the problem domain to the user which is the application domain. The same concept applies to "Query Completed Tasks" which looks for all tasks with the status "active = false" in the database and displays them to the user. The functions are complex because they read several objects in the Task class to display the queried tasks in the UI.

5.4.3 Interfaces

This section covers the main ideas behind the interface for the system.

The system follows a direct-manipulation pattern due to a few crucial buttons like completing a task or adding a comment. This makes it easy for the users to create a mental model of the system, because of the few options the user is presented with. For navigating between some sections, a menu-selection pattern is used due to the users not having to navigate around the system's different sections that much. When creating a new user or task, the system follows a form fill-in pattern. This pattern is used as it is the most used pattern for data entry [25].

Navigation Map

This section contains the navigation map for the system. As seen in the navigation map figure 5.10, the system consists of the main page with 3 branches to follow. Each branch is connected, so for example the manager and the employees can go back and forth to the history view. The manager view is different from the other branches as it is password protected. This is done because the manager has a lot of functionalities that the employees should not be able to use. The manager can go to the Manage Team page when on the manager view, thus making the Manage Team view locked behind the password as well.

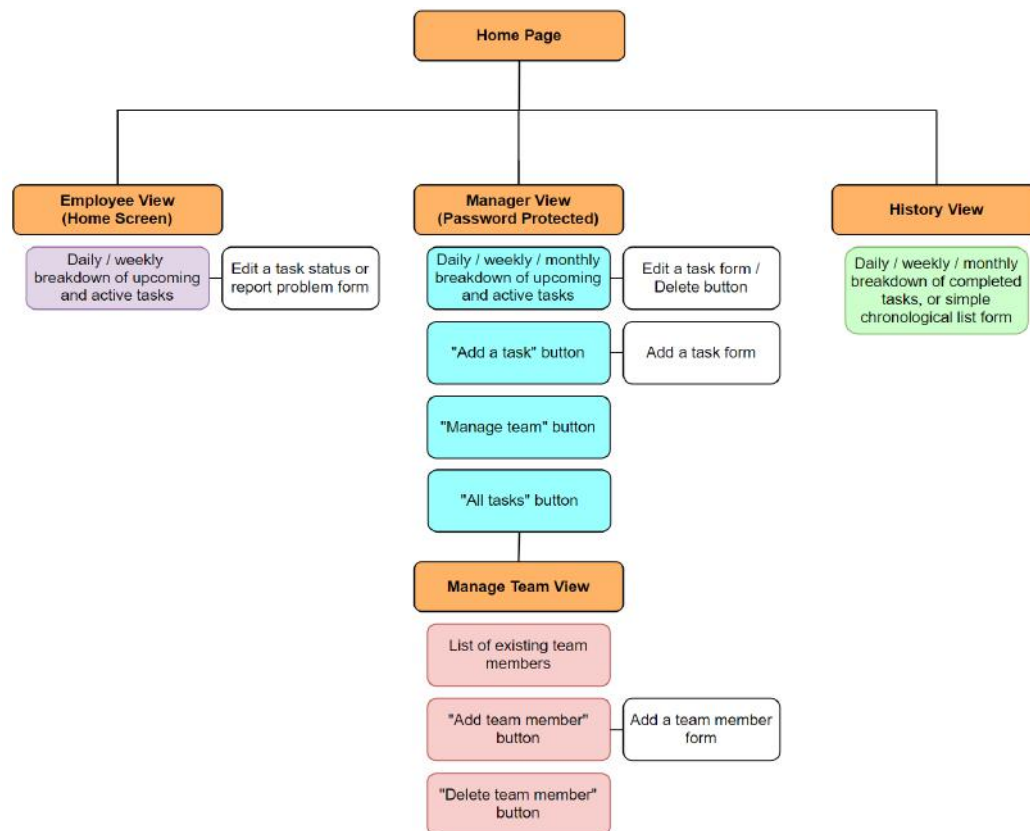


Figure 5.10: Navigation Map

5.5 Requirements

Requirements are concisely formulated criteria that the product, to some extent, should follow to solve the problem fully (keep in mind that such a problem can never be fully solved). The following requirements are divided into functional and non-functional requirements. The functional requirements describe the functionalities of the system and are sorted according to the MoSCoW-method that determines the priority of each requirement. The non-functional requirements describe the qualities of the system and are categorized with the FURPS+ method. In short, functional requirements describe what the product should be able to do, and non-functional requirements define how these things

should be done. The requirements were generated by brainstorming, using the information gathered through research, interviews, meetings, and most importantly the analysis.

Number	Description	Origin	Priority
1	Create tasks with the following properties and save them in a database (a task object is a collection of properties, of which some may be optional): title, assignee(s), description, frequency, deadline, degree of urgency, label, opting into deletion from database, opting into receiving a notification when task is completed.	Interview	Must Have
2	Edit properties of tasks and save them in the database after creating them	Interview	Must have
3	Delete tasks from database	Interview	Must have
4	Display daily tasks separated into two columns, showing active and upcoming tasks for the day	State of the art	Must have
5	Display a daily, weekly, or monthly calendar of tasks, separated into columns for each day	Interview	Must have
6	Create employees and save them in database (employees are objects with two properties: name and phone number, personal ID)	State of the art	Must have
7	Edit employee properties in database (except personal ID)	State of the art	Must have

Figure 5.11: Functional Requirements part 1

8	Delete employees in the database.	State of the art	Must have
9	Assign a status to a task (e.g new, active, completed).	Interview	Must have
10	Different roles have access to different functionalities (manager and employee).	Interview	Must have
11	Append a comment to a task (in case of a problem).	Interview	Should have
12	Sort list of tasks <ul style="list-style-type: none">• Sorted by time• Sorted by priority/urgency• Sorted by frequency (monthly/weekly/daily)• Sorted by assignee	State of the art	Could have
13	Display previously completed tasks in a chronological list-form.	Interview	Could have
14	Display a notification for the manager when a comment has been added to a task.		Won't have

Figure 5.12: Functional Requirements part 2

Number	Description	Requirement type
1	An acceptable response time.	Performance
2	Easy to navigate, and (primarily for employees), easy to use in a busy work environment.	Usability
3	The system must be available at all times - not just during opening hours.	Reliability
4	Must be responsive to a screen's aspect ratio and size - user experience should not be affected by changing screen type (landscape-oriented).	Usability
5	Must be compatible with different types of operating systems.	Supportability
6	A login/logout option.	Functionality
7	The data of the system should be stored using a secure cloud-based database.	Functionality
8	Navigation of the system should easily create a mental model.	Usability
9	User interface needs to be intuitive and comprehensible for both roles.	Usability
10	The system's account creation must require username, password, email and phone number.	Functionality
11	The system must update when new tasks or other events happen automatically.	Functionality
12	A progress bar showing the progress for the current day	Functionality

Figure 5.13: Non-functional Requirements

Disclaimer: See section 6.1.1 for a revision/re-evaluation of which requirements are satisfied by the product.

5.6 Final iteration of the problem statement

Providing an application for small businesses with busy work environments (e.g. The Living Room), how can the group contribute to achieving an improved overview of everyday task management and administration? More specifically, how can a software solution help in the documentation and communication of daily/frequent tasks, as well as the reduction of forgotten assignments and confusion regarding personal responsibilities? To validate the provided solution's degree of success, assess whether the staff of The Living Room would consider the application suitable for everyday use, and be a superior alternative in comparison to currently applied solutions.

Chapter 6

Product

6.1 Description of the application

The product of this project is a software solution, whose primary purpose is the assistance of administration of daily routines and weekly/monthly repeating tasks at The Living Room, or, arguably, any cafe or bar with a similar management structure.

In short, the software provides a platform, with which the manager of the cafe and the employees interact in a parallel fashion. The manager can create tasks with properties such as assignees, frequency, urgency, deadline, etc. If any unexpected issue occurs, they can also communicate with employees using the comment section of each task. In case a task needs changes, or an employee's personal information needs to be updated, both can be edited by the manager. As for logging and tracing previous tasks back, a history view is available for both managers and employees, containing a chronological list of all archived tasks broken down into months. The employee view provides a similar but simplified overview of daily tasks, with efficiency and simplicity being the main focus. Employees can mark progress and completion of a task and if a potential issue or

misunderstanding occurs, they are able to directly contact the manager through the comment section. All in all, the solution provides a clean and uniform way of keeping track of, managing, and planning tasks for both the manager and the whole team.

The software solution is a high-fidelity prototype of a dynamic multi-platform application programmed in Java. The product functions as a desktop application, whose aspect ratio and relative resolution (1024 * 768) matches a 10th-generation iPad - the device on which the real product would run (see previous observations). The product has a user interface designed to suit the busy work environment of The Living Room, and with a landscape-orientation preference. The two types of users of the product are the manager and employees. The previously gained fundamental understanding of the contexts in which these actors would use the program, as well as their demands, were also crucial in the planning and design of the user experience.

In the following paragraphs, the different "views" of the program and its contents are clarified as well as what a "task" exactly covers in the context of the product.

The manager's view is best described as the "control centre" of the program. It is in this view that all information regarding employees and tasks is defined, managed, and edited. During early testing, it was made clear that a manager is not under the same pressure, nor requires the same gentle learning curve, as an employee does. Instead, they would much rather work with a more complex user interface, and be able to control as many aspects of their experience as possible. Therefore, filters, a calendar system for easier date selection and planning, and a non-date-restricted display of all tasks are features worth considering.

The employee view is used by the employees at work, not currently occupied with serving customers. The demands for this view were vastly different. For instance, efficiency, colour association, and instant pattern recognition were higher priorities. Employees also work considerably fewer functions since the system for them should simply be used to communicate tasks and potential problems and report progress and completion.

As previously mentioned, the history view is accessible to both employees and managers and includes all archived (marked as completed) tasks and their properties. To avoid any confusion, all tasks are sorted chronologically based on their respective deadlines.

"Tasks" in the context of the program represent a collection of properties describing an issue or assignment, which either bartenders or cleaners of The Living Room are responsible to tackle. The mandatory fields are a title, assignee (a task can also be assigned generally to the entire staff) and frequency. The optional fields are the description, date, degree of urgency, and role. Every optional task has a default value, which was mainly determined during tests and consultation with the client. Furthermore, the tasks have non-user-defined properties such as timestamps for the last edit made to a task and its status, which is used for determining if a task is active or not.

6.1.1 Revision of Requirements

This section contains the implemented requirements for the final product and is shown in table 6.1, 6.2 and 6.3. Some of the requirements from chapter 5.5 have been modified to make them clearer for the context of the system. An example could be requirement number 4: "Display daily tasks separated into two columns, showing active and upcoming tasks for the day". This requirement is changed to: "Display daily tasks and tasks that are overdue."

Number	Description	Origin	Priority
1	Create tasks with the following properties and save them in a database (a task object is a collection of properties, of which some may be optional): title, assignee(s), description, frequency, deadline, degree of urgency, role, opting into deletion from database.	Interview	Must Have
2	Edit properties of tasks and update them in the database.	Interview	Must have
3	Delete tasks from the database.	Interview	Must have
4	Display daily tasks and tasks that are overdue.	Test	Must have
5	Create a user with the following attributes: first name, last name, email, phone number, admin, and role.	Interview	Must have
6	Delete users from the database.	State of the art	Must have
7	Update users in the database.	State of the art	Must have
8	Set a task's status to inactive in the database for both employees and managers.	State of the art	Must have
9	Must have two different views according to the users' admin attribute. If the user is an admin, they will see the manager page, and if they are not	Interview	Must have

Figure 6.1: The implemented functional requirements for the system part 1.

	an admin they will see the employee page.		
10	The user should provide a PIN code to get to the manager page.	Interview	Should have
11	See information about a task when clicking the information button.	State of the art	Should have
12	See information about a user when clicking the information button	State of the art	Should have
13	Add a comment to a task.	Interview	Should have
14	Display all tasks in the system, both active and inactive.	Test	Should have
15	Sort list of tasks <ul style="list-style-type: none"> • Sorted by time • Sorted by priority/urgency • Sorted by frequency (monthly/weekly/daily) • Sorted by assignee 	Interview	Could have
16	Display previously completed tasks in a chronological list-form.	Interview	Could have
17	Go back and forth between the different days in the overview for both employees and managers.	Test	Could have
18	Display daily tasks separated into two columns, showing active and upcoming tasks for the day	State of the art	Won't have
19	Display a daily, weekly, or monthly calendar of tasks, separated into columns for each day	Interview / State of the art	Won't have

Figure 6.2: The implemented functional requirements for the system part 2.

Number	Description	Origin	Requirement type
1	Easy to navigate, and (primarily for employees), easy to use in a busy work environment.	State of the art	Usability
2	The system must be available at all times - not just during opening hours.	Interview	Reliability
3	Must be responsive to a screen's aspect ratio and size - user experience should not be affected by changing screen type (landscape-oriented).	Interview	Usability
4	Must be compatible with different types of operating systems.	Interview	Supportability
5	The data of the system should be stored using a secure cloud-based database.	Interview	Functionality
6	Navigation of the system should easily create a mental model.	State of the art	Usability
7	User interface needs to be intuitive and comprehensible for both roles.	State of the art	Usability
8	The system's account creation must require a username, password, email and phone number.	Test	Functionality
9	A progress bar showing the progress for the current day	Test	Functionality

Figure 6.3: The implemented non-functional requirements for the system.

Chapter 7

Design

7.1 System design

In order to create a well-structured application it is important to choose the right design pattern and build the project in a way, in which components such as UI frameworks and databases are kept separate enough, that they are proven to be interchangeable with relative ease. For instance, implementing new CRUD operations, in case it is required to shift to a SQL database, or switch to a web-based UI, should not mean critical changes with regard to the entire code base.

7.2 Design concerns

This section contains the different design concerns which were taken into consideration when creating the system.

7.2.1 Criteria

This subsection contains a table with some classical criteria and how they were prioritised for designing the system [25]. The description of each criterion can be seen in appendix H.

Criterion	Very important	Important	Less important	Irrelevant	Easily fulfilled
Usable	x				
Secure				x	
Efficient	x				
Correct		x			
Reliable			x		
Maintainable		x			
Testable		x			
Flexible				x	
Comprehensible	x				
Reusable		x			
Portable				x	
Interoperable				x	

Figure 7.1: Criteria for the system

As seen in figure 7.1, four criteria in total were deemed irrelevant. This is due to this system only needing to apply to one specific cafe and does not handle any sensitive personal information. The "Portable" and "Interoperable" criteria are also irrelevant as a result of the system not being deployed nor communicating with other systems besides MongoDB services. The "Reliable" criterion is prioritised as less important as a result of the system not having any computational functions. "Correct" and "Testable" were prioritised as important because we created the system specific for The Living Room, and it should fulfil as many of the client's demands as possible. By choosing to create the system using an object-oriented approach together with the MVC model, "Maintainable" and "Reusable" were also deemed important. The "Testable" criterion is important, as the system should always do what the user intends to do. The "Usable" criterion is very important because of the environment at The Living Room e.g., noise, time pres-

sure and handling customers. The system should therefore respond and update as quickly as possible to not delay the employees. Lastly, comprehensibility is deemed very important as well because the environment at The Living Room is noisy and stressful at times.

7.2.2 Database Study

Since the system will be used both at The Living Room by its staff and by the manager wherever he or she is, the database must be available from anywhere. When searching for a suitable database MongoDB, Microsoft Azure SQL Database and Firebase were matching the needs of the system. Microsoft Azure SQL Database was quickly turned down as it costs money to use [26]. Firebase is a database used and developed by google and is mostly for handling a large amount of data. Since our system is not going to use that much data, Firebase was also turned down [18]. This left us with MongoDB as the optional database to use for this system. MongoDB is a document-oriented NoSQL database. It uses JSON-like documents with optional schemas [28]. The method of storing data is almost like using objects in Java which is also a reason for choosing MongoDB, as models for the system look a lot like the stored data. This was also chosen due to the criterion "efficient", which was deemed very important.

7.2.3 GUI Framework Study

The graphical user interface framework chosen for this project was JavaFX. The main reasons behind the selection were the fairly gentle learning curve, simple syntax and thorough documentation, the Oracle Scene-Builder tool support, TestFX test suite, large selection of third-party libraries, and good applicability for desktop applications [40]. Generally, JavaFX was expected to provide a lot of shortcuts, reducing the time consumption of front-end work.

Other potential alternatives, such as AWT [31], Swing [32], SwingX [22], and

JGoodies [21] were explored, but were proven to be largely outdated, and would have required a much larger partition of the project duration to collectively learn and implement. These solutions are also purely code-based (no FXML or HTML support) with no currently available layout-planner tool such as Scene-Builder.

7.3 Architecture

The following section covers the architectural design of the system and includes the component design and an architecture diagram.

7.3.1 Component design

This subsection contains the final iteration of the component design of the system, which highly resembles the earlier presented class diagram. The reason behind this is the use of an iterative development cycle during the making of this system. Each class diagram was revised numerous times in order to make it as precise as possible. Therefore, the component design for the system is the class diagram with attributes and methods assigned to the different components/classes. Another element added to the component design is the specification of some of the attributes such as frequency and urgency. This was done due to the attributes being constants. The specifications are added in the yellow boxes in figure 7.2.

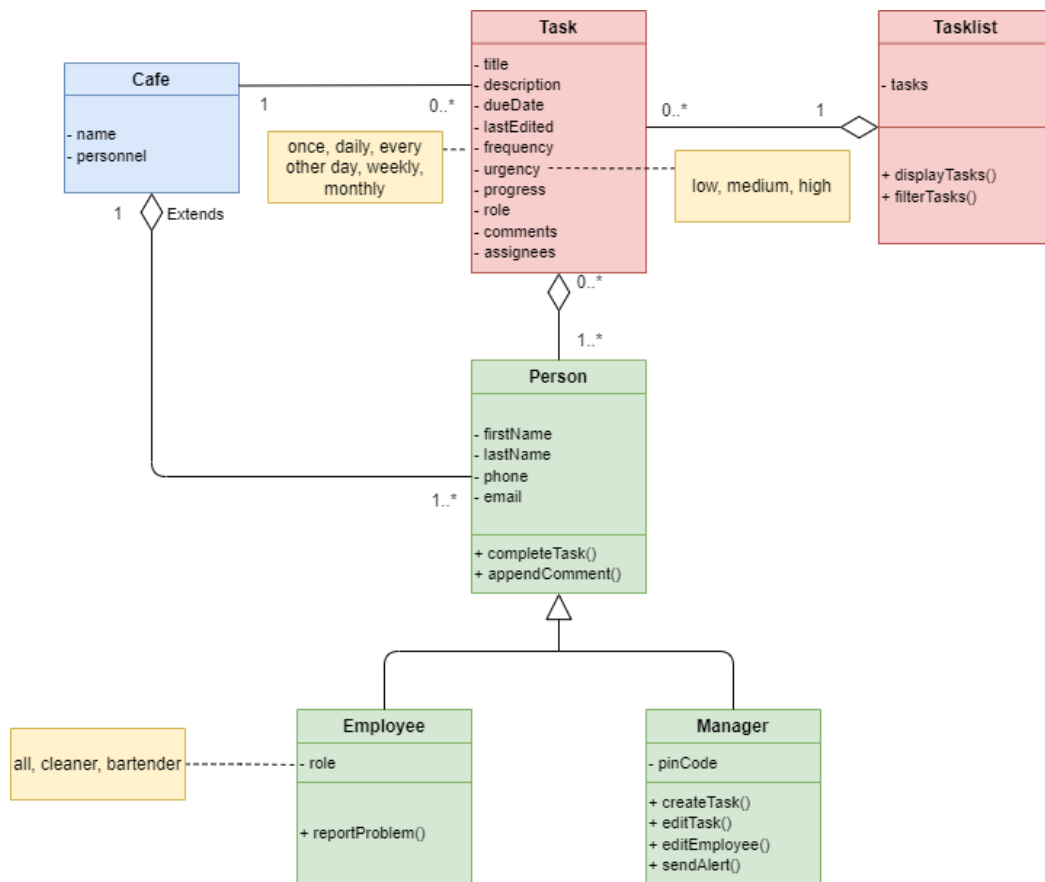


Figure 7.2: Component design

7.3.2 Architecture diagram

As seen in figure 7.3 the system uses both a client-server and layered pattern. The client-server pattern is shown by the link between the client and server through the model and database system components. The direct link between the model and database system is discussed in section 10. The client-server pattern was used because the manager must be able to use the system from any place as long as their device is connected to the internet. The reason for The Living Room not owning a local server is that it would cost much more compared to a cloud-based server and would propose a new set of risks. The layered pattern can be seen in the client component because the three components are divided into three layers:

user interface, functions and model. This was done to keep the components loosely coupled and to make the system as cohesive as possible. Another reason for the system to follow the layered pattern is that the criterions "maintainable" and "reusable", which were deemed important.

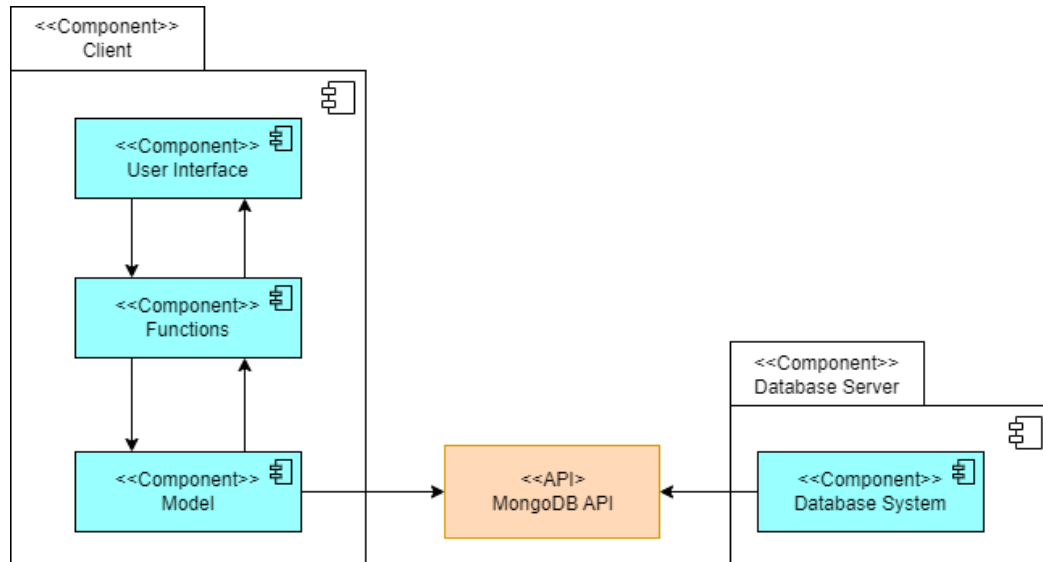


Figure 7.3: Architecture for the system

7.4 UI design

When developing a software solution that will compete with other applications on the market, it is important to create *"an attractive and efficient user interface (UI) that optimizes the user experience (UX)"*[38] since the quality of the UI and UX will distinguish solutions from one another. A good user interface is characterised by its ability to make the user's interaction with a program as effective as possible while also making the user's experience easy and intuitive. The user's expectations are also an important aspect to consider and a UI that meets these expectations is arguably what makes it a great one [38]. This chapter will describe the UI design of this project and how it was created.

7.4.1 Sources of inspiration

Building upon a user's knowledge of other systems, digital design conventions, and general habits is an important action to consider when designing a new user interface. The point of the project is not to invent new design conventions, but to adapt those that work. Therefore, other than looking at competitors and their designs, other relevant applications, with which the target group of the project ought to be familiar, were taken into consideration during the design of the product.

As for how the task lists themselves look, a great deal of inspiration was taken from both competitors explored in the state-of-the-art section, as well as The Living Room's already existing physical solutions and spreadsheets. The idea behind this was that implementing a new system similar to the one already implemented would result in a smoother transition period from the previous to the new system.

For navigating through dates, Google Calendar's system, seen in figure 7.4, was a point of inspiration, to which especially the final product bears a notable resemblance.



Figure 7.4: Google Calendar

To mark urgency (low, medium, high) and a task being overdue (today's date is later than the deadline), Microsoft Outlook's "high importance" exclamation mark icon, seen in figure 7.5 and Apple iOS colour labels, figure 7.6 (often used to mark urgency) were inspiring factors.



Figure 7.5: Microsoft Outlook high-importance exclamation mark

Name		Date Modified
Clean.Code.A.H...raftsmanship.pdf	●	26 Aug 2022 at 14.11
Design_Pattern...ted_Software.pdf	●	25 Oct 2022 at 09.56
designing user experience.pdf	●	26 Aug 2022 at 14.07
Essential C 8.0.pdf		26 Aug 2022 at 14.08

Figure 7.6: IOS colour labels

For the login screen, Netflix's profile selection screen was used as a guiding

principle as seen in figure 7.7.

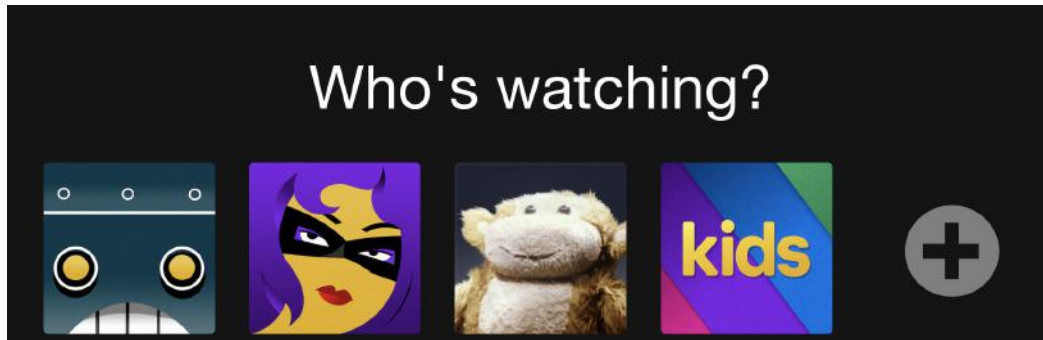


Figure 7.7: Netflix profile selection screen

7.4.2 The original UI idea

The UI design was originally planned out by creating wireframes in the application Balsamiq Wireframes [3]. Before creating the wireframes, potential elements of the UI, and the placement thereof, were discussed, listed and drawn. From that, the wireframes were used to create a mock UI making it easier for the developers to picture what the application could look like. The picture below shows the first idea for the UI of the manager view before the wireframes were created.

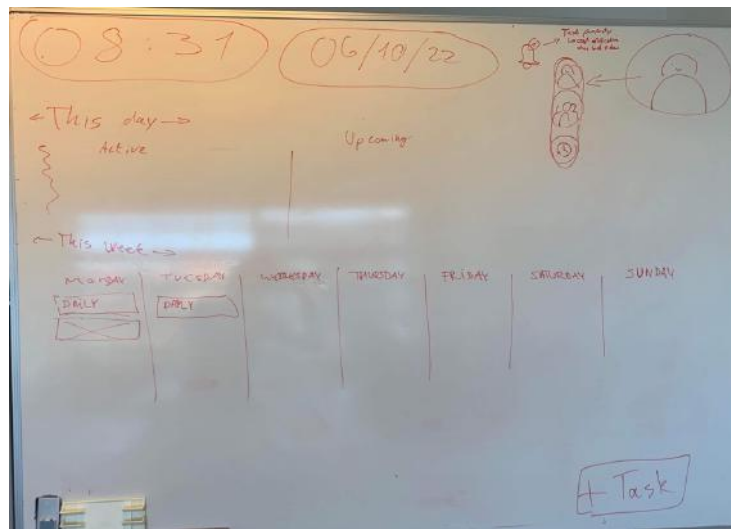


Figure 7.8: Original UI idea

The essential elements here are the navigation, the calendar view, the task overview and the "add task"-button seen in the bottom right corner. The original idea was that the manager should be able to see the tasks for the current day, divided into tasks in progress (active) and tasks yet to be completed (upcoming), for a week at a time and for a month at a time. In these different views, the manager should be able to create tasks and edit/delete tasks. The navigation button seen in the top right corner of the picture is a dropdown button, which makes it possible for the user to navigate between a manager view, an employee view and a history view.

7.4.3 The wireframes

During the process of designing the UI, multiple wireframe iterations were made. The first wireframe iteration was an evolved version of what the picture above shows. Other than the manager view, the first wireframe iteration also included an employee view and a history view with lo-fi interactivity to showcase the proposed interactivity of the buttons and functions. The picture below, figure 7.9, shows the manager's view from the first wireframe iteration. The employee

view is very similar to the manager view with the exception that it has fewer functionalities.

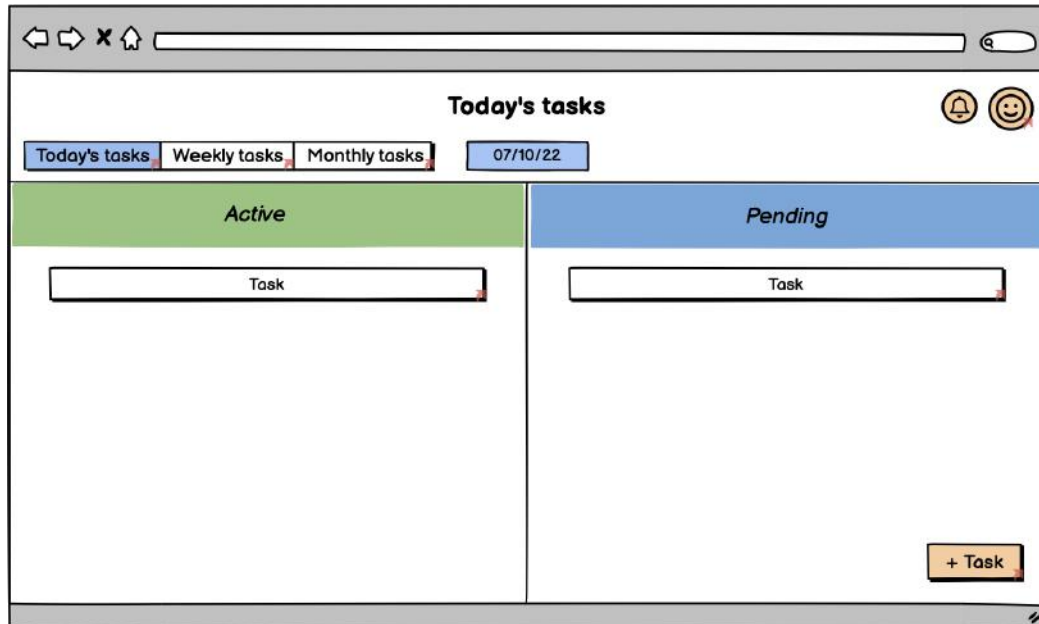


Figure 7.9: Original manager view

Since the scope of the project is narrowed down to tackling issues regarding task management, and the overview of tasks in The Living Room's current solution is not optimal, a better overview of tasks was prioritised. The idea with today's tasks, as seen in the picture 7.9, was to create two states of tasks to make it easier for the employees to distinguish between tasks that are already being worked on and tasks that have not been started yet. In this example, when a task is created, it is automatically stored as a pending task. The task can be activated by clicking on it and an active task can also be moved back to pending by the manager. The following picture, figure 7.10, shows what happens when a pending task is clicked.

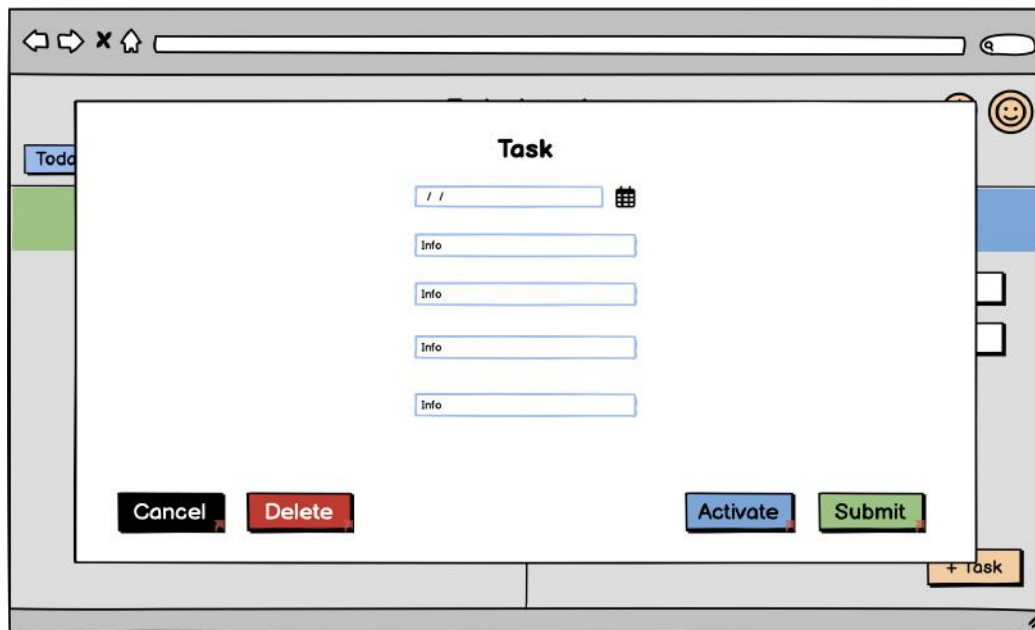


Figure 7.10: The manager view when a pending task is clicked

The idea is to show the information about the task while also providing options for the manager. In this view in figure 7.10, the manager can edit the task information and submit those changes, activate the task, delete the task and go back to the manager view (cancel). The colours of the buttons were chosen based on the state of the art and what a potential user could expect. The colour red is usually used for buttons that terminate or delete something while the colour green is used for buttons that accept or complete something. The colours black and blue were chosen to distinguish the "Cancel"-button and the "Activate"-button from the others.

As mentioned previously the first wireframe iteration also includes a history view. As an employee, you can click a task and get the option to mark it as completed. When the task is marked as completed, it will be visible in the history view under the date on which it is assigned. The picture below, figure 7.11, shows the history view.

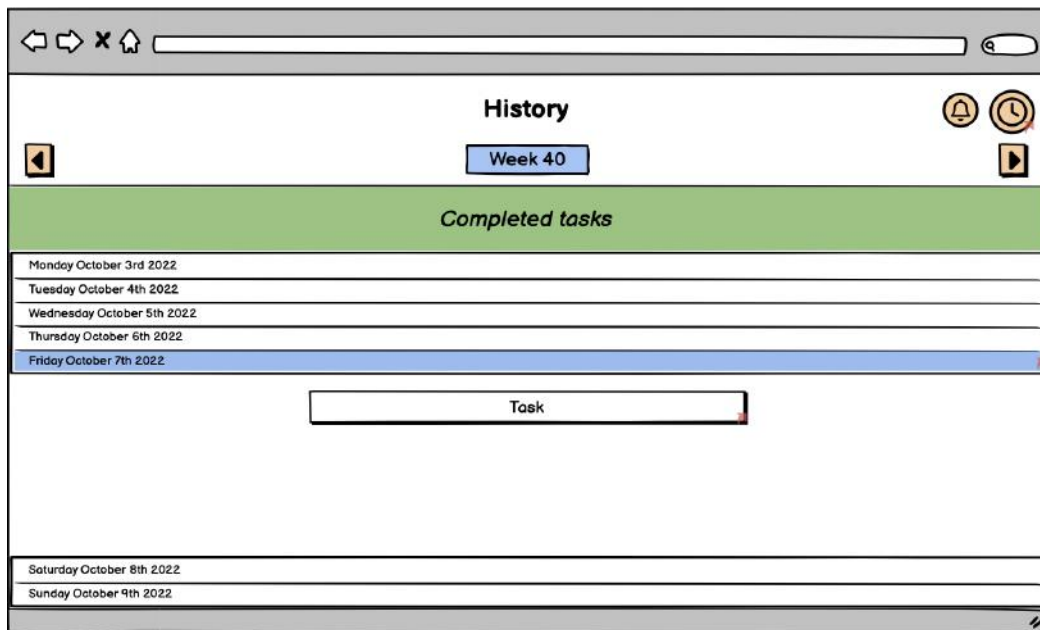


Figure 7.11: The history view

The history view shows a weekly overview of completed tasks. The idea of the history view, besides providing a way to keep track of completed tasks, is to give the user the option to re-activate tasks. For instance, if an employee accidentally marks a task as completed, it should be an option to undo that action and return the task to the "Today's tasks" view.

The second wireframe iteration is similar to the first one but has additional frames and functionality. This version contains a login function where the user can choose whether they're logging in as a manager or an employee. The program is not meant to be specific for each employee but is a commonly used platform. The picture below, figure 7.12, shows the login page.



Figure 7.12: The login page with access to the employee view and the manager view

The first and second wireframe iterations make the first iteration of the UI design and represent the first suggestion for how the program could look. When the second wireframe iteration was finished, a meeting with the client was set up to conduct an exploratory test and to get feedback from the manager (11). After the meeting, a considerable amount of changes had been discussed. The biggest change was for the way tasks are displayed and how the user can access their information and the functionality of the program. Thus, the third and last wireframe was an updated version of the first wireframe where suggestions from the manager had been taken into consideration. The following will describe the third wireframe and the new changes suggested.

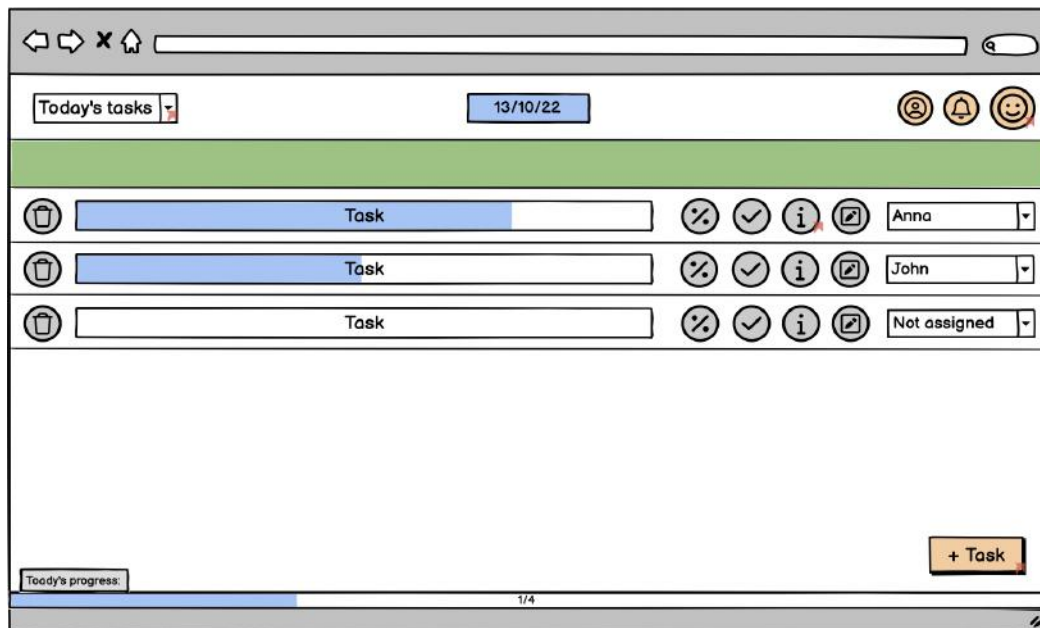


Figure 7.13: Manager's today's tasks view from final wireframe

The picture above, figure 7.13, shows the updated version of the manager's "today's tasks" view. The most noticeable difference is the display of tasks and the number of actions available in this view. The manager made it clear, that he would want the functionality to take as few clicks as possible so as to make it more efficient. Therefore, instead of having to click on the task to access its options, the options are now openly placed on either side of the task. The task itself is now also a progress bar, which can be set by using the "%" -button. Other than updating the progress bar, the buttons also make it possible for the user to delete the task, complete the task, see task information, add a comment to the task and assign employees to the task. The wireframe was also updated to show a general progress bar for the completed tasks of the day. Furthermore, the navigation bar from the last wireframes used to navigate between the displays of tasks (daily, weekly, monthly) has been changed to a dropdown menu, which takes up less space.

As mentioned earlier, the manager view is password protected, because the manager has access to private information about the employees. Therefore, if someone were to try to access the manager view from the employee view (or history view, which is accessible to both managers and employees), the program should prompt for a password. This functionality is added in the last wireframe and an example of this can be seen in the picture below 7.14.



Figure 7.14: The login prompt from trying to access the manager view

7.4.4 The final look

The wireframes were used as a stencil for the program, thus the program's interface was modelled after these as accurately as possible. The following pictures show the final look of the program.



Figure 7.15: The login page

The final login page differs from the last wireframe in that it has an added button which gives access to the history view. The history view access was added to the login page as a shortcut since both managers and employees could find it useful to be able to access the history view directly.

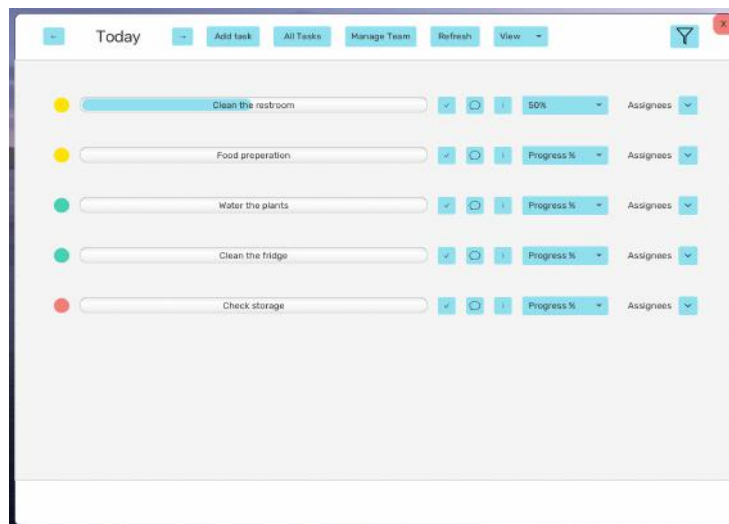


Figure 7.16: The manager view

The task display in the final manager view has been programmed to match

the task functionalities displayed in the wireframes more or less completely. The trashcan buttons and their functionality, however, have been replaced with coloured dots. The dots are meant to represent the urgency of each task. Furthermore, the view has gotten an extended set of functionalities with buttons that make it possible for the user to move between displays of tasks according to different days, add a task, access all tasks, Manage Team, refresh, navigate between views and filter by specific categories. Some of these are new and some of them are the same as the last wireframe.

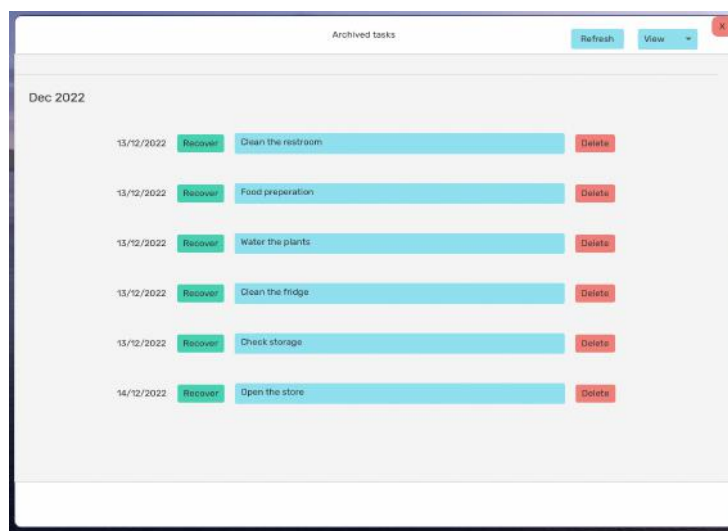


Figure 7.17: The history view

The final history view is very different from the wireframes since it was decided that this similar way of displaying the tasks (as in the task view) was more manageable. The continuity between the scenes also makes it easier for the user to navigate the program.

The wireframes have been very useful tools when creating the UI, but it has also been relevant to apply digital design theory; some of which are presented in the following sections.

Gestalt laws of perception When designing UI, the Gestalt principles of design are useful tools to help improve aesthetics, functionality and user-friendliness. Six of these Gestalt principles, or laws, are listed here[8]:

- Proximity
- Continuity
- Part-whole relationships
- Figure/ground
- Similarity
- Closure

The laws that have been applied in this program are mainly similarity and proximity. The similarity is seen in the task view for both managers and employees and in the history view. The tasks in the task view are displayed similarly to create a recognizable cluster of tasks. The same can be said for the completed tasks in the history view. The law of similarity can, arguably, also be applied to buttons with the same colour. Proximity can also be seen in the before-mentioned views since all the information and options for each task are closely lined up. Thus, distinguishing each task from one another.

WIMP(Windows, Icons, Menus, Pointers)

Windows: The application created in this project runs in a single window, and only changes scenes when navigational buttons are clicked.

Icons: The icons used in this application are primarily seen as buttons. The first icons encountered are the three view-option buttons on the login page. The buttons have labels as a further clarification of the icons, which, respectively, try to depict the idea of a manager, employees and a history feature. All three icons would be categorized as metaphor icons since they are not direct images, but

more of an interpretation of the labels. Icons are also found as buttons on the task page. The icons here would all be categorized as convention icons since they are known from application to application as representing a specific feature. The icons in question are the checkmark, the speech bubble, the "i", the filter button, and the exit button.

Menus: The application does not contain a lot of menus, but the ones that are present are contextual menus in the form of dropdown menus.

Pointers: Since the application is a desktop computer application, the pointer used for functionality will be a computer mouse, however, the pointer will remain static when hovering over clickable elements. If the application was upgraded to work on a tablet, the pointer would be the finger of the user [5].

Affordance Affordance is a term which refers to possible actions that can be performed on an object [1]. In this application, affordance and perceived affordance is seen in the manager and employee task view. The dropdown menu buttons afford to show its options if clicked, while a perceived affordance is found in the arrows next to the date, which afford showing the next or previous date if clicked. Affordance can also be seen in the way the opacity of the buttons changes when the pointer is hovering over them, affording the user to click them.

Colours Another rather important aspect of the design is UI colours. The tone and choice of colours can have a great impact on how an interface is perceived. The main colours of the application are grey, white and blue which, in western colour conventions, are colours of neutrality. The neutral colours were chosen since the program should not take unnecessary focus from employees working in the bar. Other colours used in the application are red and green for various buttons. As mentioned earlier the red and green colours have been chosen due to commonly known applications and expectations of their symbolism. The buttons in question are cancel buttons, exit buttons, and delete buttons, which are red,

and OK buttons, recover buttons and submit buttons, which are green. As previously stated, the coloured dots next to the tasks indicate the urgency of each task. The dots are coloured according to the level of urgency. If the level of urgency is low, the dot is green, if the level of urgency is medium, the dot is yellow and if the level of urgency is high, the dot is red. These colours are commonly known in western colour conventions too, respectively, display danger (red), caution (yellow), and safety (green), which fits the essence of these levels of urgency [24].

Heuristics The User Advocate Jakob Nielsen's revised heuristics is a list of usability heuristics for UI design some of which are [29]:

- Visibility of system status
- Match between system and the real world
- User control and freedom
- Consistency and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use

Heuristics that can be applied to this application are the following:

User control and freedom: The user can, at any given moment, exit the program or cancel a process (go back) by clicking on the visible, red buttons. The user also has the option to undo certain actions. For instance, when completing a task, the task can be recovered from the history view.

Consistency and standards: The application applies knowledge of already existing UI designs to make it intuitive for the user to locate expected options. For instance, an exit button is usually located in an upper corner of an application.

Error prevention: When the manager tries to create a task or a user, they can not submit the form unless all mandatory fields are filled out. The recover feature also falls under this category since the user mistakenly can complete a task, but also undo this action.

Recognition rather than recall: The application uses recognition with button labels and consistent scene designs. The button labels help the user recognise what they do or where they lead to. The consistent scene designs make it easier for the user to navigate the program since they do not have to re-learn how to navigate the scenes every time a new one is displayed.

Flexibility and efficiency of use: When the application is run on a computer it allows some keyboard shortcuts, such as the enter- and escape buttons, to add flexibility.

Chapter 8

Implementation

In this chapter, the implementation of the prototype is covered. The implementation of patterns is going to be explained and they will be shown in the code examples with snapshots of the GUI.

8.1 MVC

Implementing a consistent design pattern ensures the structural segmentation of the code and other miscellaneous files and packages. In this project, a model-view-controller (referred to by the abbreviation MVC) is used. This design pattern was selected because of its good separation of concerns and scalability. Additionally, the product highly resembles a web application, and MVC is widely applicable [40]. This means that a situation in which the entire project requires restructuring is improbable. The group's familiarity with this structure was also worth taking into consideration, since learning a new structure could consume lots of resources.

Briefly, the MVC pattern breaks down code into 3 compartments (all definitions belong to the source [19]):

- *The backend that contains all the data logic.*
- *The frontend or graphical user interface (GUI)*
- *The brains of the application that controls how data is displayed*

In MVC, each component has its own responsibility. The model can send and receive data from the view while the controller receives data from the view and sends it to the model like in figure 8.1. The use of MVC separates the code base into sub-parts, making it more transparent to figure out the origin of the bugs while aiding the readability of the code. It also makes the product more scalable since there is structure to its multiple levels.

MVC Architecture Pattern

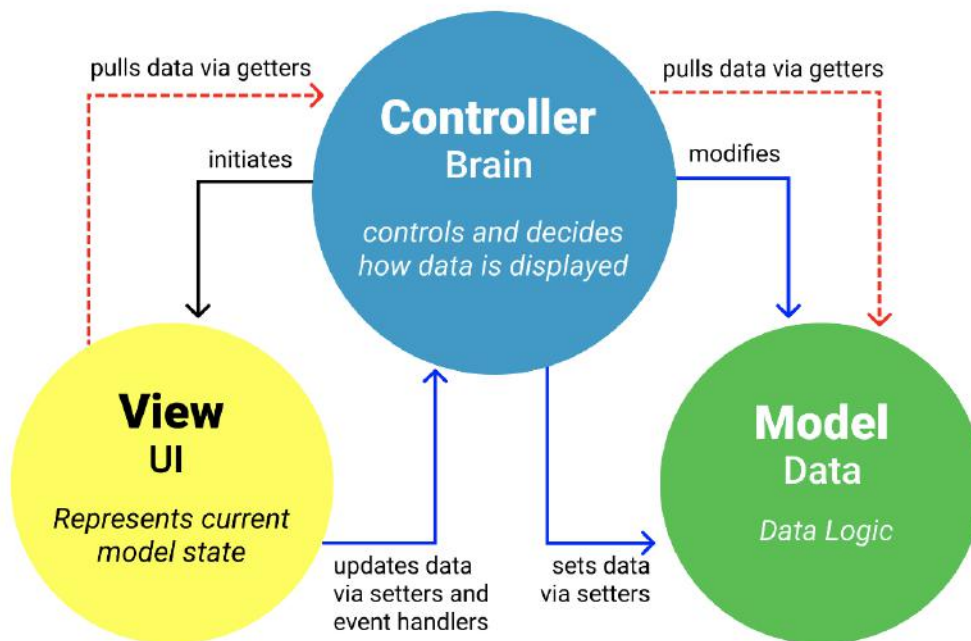


Figure 8.1: MVC Architecture Pattern[19]

8.2 Code examples

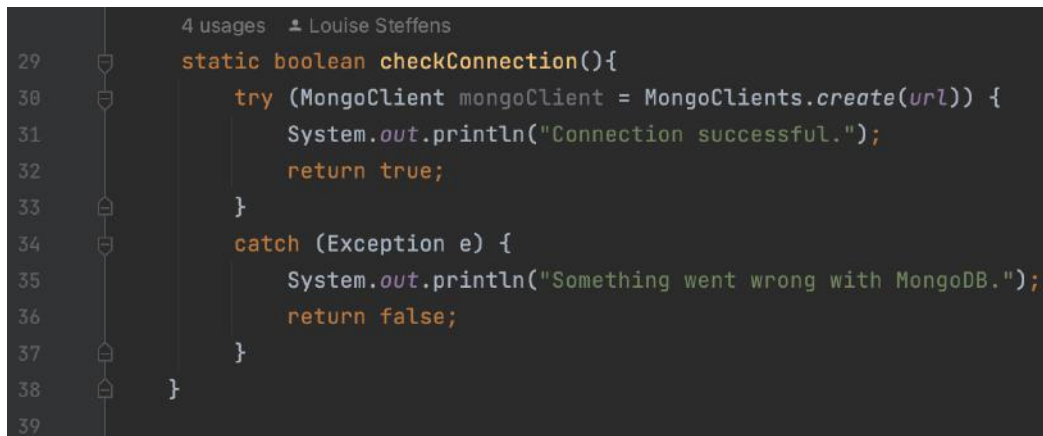
In this section, specific code examples are used to demonstrate the implementation of requirements into functional code. Please note that previously discussed event tables, class diagrams, brainstorming, and pseudocode have all provided assistance in planning and constructing the logic.

8.2.1 Database methods

In the program's separation of concerns, detaching the database-related methods from the rest of the program's logic was a decision made early on in the design phase. The DatabaseMethods interface is responsible for establishing a connection to the database, importing and exporting tasks and users (with the help of models) and generally all CRUD (create, read, update, delete) operations.

checkConnection()

The checkConnection() method is a relatively simple method, figure 8.2, in which a try-catch statement encapsulates an attempt to create a Mongo Client using a method provided by the MongoDB Query API. This method is implemented in other database-related methods since issues with the connection should be accounted for before initiating any other operations.

A screenshot of an IDE window showing a Java method named `checkConnection()`. The code is written in a dark-themed editor. On the left, there is a vertical toolbar with icons for folding, search, and other IDE functions. The code itself is as follows:

```
29      static boolean checkConnection(){
30          try (MongoClient mongoClient = MongoClient.create(url)) {
31              System.out.println("Connection successful.");
32              return true;
33          }
34          catch (Exception e) {
35              System.out.println("Something went wrong with MongoDB.");
36              return false;
37          }
38      }
39  }
```

At the top of the editor, it says "4 usages" and "Louise Steffens".

Figure 8.2: checkConnection method

getTasksFromDB()

The `getTasksFromDB` method, seen in figure 8.3, returns an `ArrayList` containing either all active or all inactive tasks from a specific collection depending on the arguments passed. Following an assertion in line 64, a for-each loop iterates through all documents of the provided collection, finds all active/inactive tasks, sorts them based on date, and adds them to the locally stored `ArrayList` with the help of the `createTaskToDisplay()` method. The `createTaskToDisplay()` method acts as a translator from MongoDB database values to model values, so task objects can be created with the values stored in the cloud. At last, the function returns the `ArrayList` tasklist.



```

59 @
60 static ArrayList<Task> getTasksFromDB(Bson filter, boolean isActive, String collName) {
61     ArrayList<Task> taskList = new ArrayList<>();
62
63     MongoClient<Document> coll = getDBColl(collName);
64
65     assert coll != null;
66     for (Document doc : coll.find(Filters.and(filter,
67         eq( fieldName: "active", isActive))).sort(ascending( ...fieldNames: "date")))) {
68         ArrayList<Object> values = new ArrayList<>(doc.values());
69
70         taskList.add(createTaskToDisplay(values));
71     }
72     return taskList;
73 }

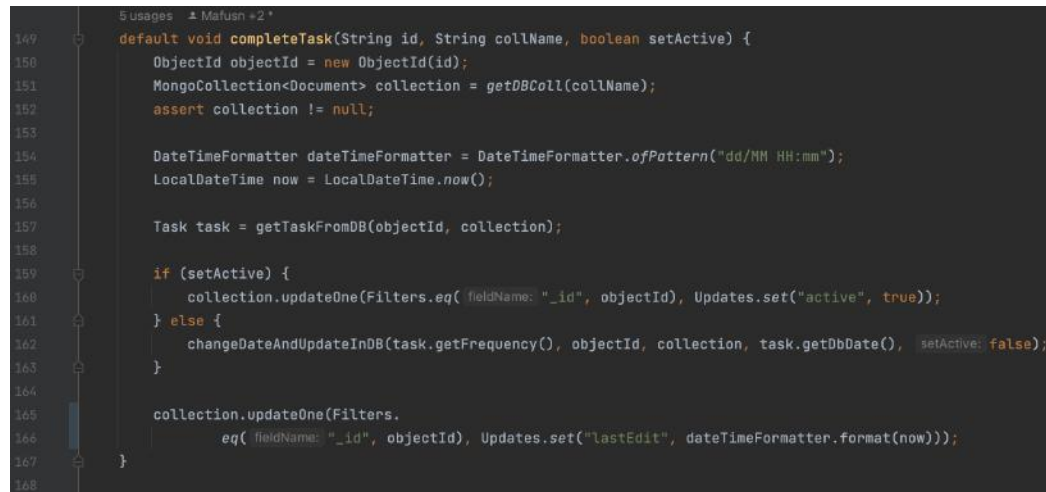
```

Figure 8.3: getTasksFromDB method

completeTask()

The method `completeTask()`, seen in figure 8.4, handles the process of completing a task by updating the attribute "active" from true to false, or false to true, depending on the context. It has three parameters: `String ID`, which is the ID of the task fetched from the database, `String collName`, which is the collection name to fetch the task from the database, and `boolean setActive`, which indicates if the task is active. The `objectID` created on line 150 converts the given ID from the parameters to an `ObjectID` type, which can not be done before calling the function. Line 151 then uses another function from the interface to get the actual collection and store it in the collection variable. Lines 154 and 155 make it possible to convert the variable "now" to the specified date format on line 154. This makes it possible to change and search via the date attribute in the database. Line 157 gets the task with the ID from the database and the function `getTaskFromDB()` is a function from the `DatabaseMethods` interface. The if-statement starting on line 159 handles the "frequency" attribute and it manages how the task is updated in the database. If `setActive` is true, the task attribute "active" is simply set to true. The function `updateOne()` is from MongoDB's API. The value of the

attribute "frequency" for the task is handled in the function `changeDateAndUpdateInDB()`. If, for example, the "frequency" value of the task is monthly, the date of the task is set one month forward instead of changing the "active" attribute. Lastly, on lines 165-166 the task's attribute "lastEdit" is changed to the current day and time when the `completeTask()` function is called.



```

149  default void completeTask(String id, String collName, boolean setActive) {
150      ObjectId objectId = new ObjectId(id);
151      MongoCollection<Document> collection = getDBColl(collName);
152      assert collection != null;
153
154      DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("dd/MM HH:mm");
155      LocalDateTime now = LocalDateTime.now();
156
157      Task task = getTaskFromDB(objectId, collection);
158
159      if (setActive) {
160          collection.updateOne(Filters.eq("fieldName": "_id", objectId), Updates.set("active", true));
161      } else {
162          changeDateAndUpdateInDB(task.getFrequency(), objectId, collection, task.getDbDate(), setActive: false);
163      }
164
165      collection.updateOne(Filters.
166          eq("fieldName": "_id", objectId), Updates.set("lastEdit", dateTimeFormatter.format(now)));
167  }
168
  
```

Figure 8.4: completeTask method

8.2.2 UI methods

The following section explains the main functions used in the user interface layer.

switchScene()

The function `switchScene()`, seen in figure 8.5, switches the scene displayed in the application with a new scene based on the path given. The function takes two input parameters: `BorderPane` pane and `String` path. The `BorderPane` pane is the current JavaFX `BorderPane` which the function is called from. For example, if the function is called from the login screen with `loginBorderPane` as its `BorderPane`, the `loginBorderPane` variable will be the `BorderPane` parameter used in the function. The `String` path is the `.fxml` file which is loaded and switched to.

The function is one big exception statement and catches an input-output exception if anything goes wrong on lines 40 and 41. The code on line 40 creates a brand new `BorderPane` from the `.fxml` file taken as a parameter. On line 41, the `BorderPane` taken as a parameter loads all its children/components and sets them to the new `BorderPane` created on line 40. If an exception was caught, the application will throw an exception instead of crashing.

```
38      default void switchScene(BorderPane pane, String path) {  
39          try {  
40              BorderPane borderPane = FXMLLoader.load(getClass().getResource(path));  
41              pane.getChildren().setAll(borderPane);  
42          } catch (IOException e) {  
43              e.printStackTrace();  
44          }  
45      }  
46  }
```

Figure 8.5: switchScene method

displayComments()

This function, displayed in 8.6, is called when the user is either viewing the description of a task or tries to comment on a task. The function displays comments made under that specific task. A string array list is created, which stores the comments from the given task by using the function `getCommentsFromDB` from `DatabaseMethods`. The next step is to populate the page with comments, which is done with the exception of the try-catch. The code within the try statement tries to load FXML files and populate them with comments while also checking for errors, and if any errors happen the catch statement signals that an I/O exception of some sort has occurred.

```

85 public void displayComments() {
86     ArrayList<String> comments = new ArrayList<>(DatabaseMethods.getCommentsFromDB(id, collName: "tasks"));
87
88     int columns = 1;
89     int rows = 1;
90
91     try {
92         for (String comment : comments) {
93             FXMLLoader loader = new FXMLLoader();
94             loader.setLocation(getClass().getResource("comment-box-page.fxml"));
95
96             HBox hBox = loader.load();
97             hBox.setPrefWidth(170);
98
99             CommentBoxController commentBoxController = loader.getController();
100             commentBoxController.setCommentToUI(comment, labelWidth: 165);
101
102             commentGridPane.add(hBox, columns, rows);
103
104             rows++;
105         }
106     } catch (IOException e) {
107         e.printStackTrace();
108     }
109 }

```

Figure 8.6: Display comments method

populateManageTeamViewWithUserBoxes()

This method first calls the DatabaseMethod `getEmployeesFromDB()`, seen in figure 8.7, to fetch all employees from the database. Subsequently, it iterates through each user and creates an "employee-box-page.fxml" element on the user interface. Handling task elements as widgets that can be added to the page helps with the dynamicity and re-usability of code and is also used for displaying and interacting with tasks. The method sets the ID of the vertical box element containing the displayed user's information to the user's ID. This is done on line 52. On line 54 a new `UserBoxController` is created so one of its methods can be called on Line 55. The method `setUserBoxToUI()` simply sets the different fields and labels corresponding to the user in a vertical box. The vertical box is then added to the grid and the "rows" variable is incremented by one so that the next potential box can be added underneath. If any input-output exception is caught during this iteration through the users, it will be caught and displayed in the console.

```
1 usage: ▲ Bence +1
38 public void populateManageTeamViewWithUserBoxes() {
39     ArrayList<User> users = new ArrayList<>(DatabaseMethods.getEmployeesFromDB( isAdmin: false, collName: "users"));
40
41     int columns = 1;
42     int rows = 1;
43
44     try {
45         for (User user : users) {
46
47             FXMLLoader loader = new FXMLLoader();
48
49             loader.setLocation(getClass().getResource( name: "employee-box-page.fxml"));
50
51             VBox vbox = loader.load();
52             vbox.setId(user.getId().toString()); // Store user id as vbox id
53
54             UserBoxController userBoxController = loader.getController();
55             userBoxController.setUserBoxToUI(user);
56
57             userGrid.add(vbox, columns, rows);
58             rows++;
59         }
60     } catch (IOException e) {
61         e.printStackTrace();
62     }
63 }
64
```

Figure 8.7: Populate manage team with user boxes method

Chapter 9

Quality Assurance

Quality assurance (QA) is defined as "any systematic process of determining whether a product or service meets specified requirements." [17]. The group has gone through several methods within QA throughout the product's life-cycle: a usability test which includes an exploratory test, an assessment test, a validation test and a think-aloud test, and unit testing to assure the product meets the requirements.

9.1 Usability Test

The purpose of the usability test is to identify problems concerning the system. It created a starting point for the refinements of the design during the product life-cycle [34]. This resulted in a ranked list of the problems and comprehension of the elements that work well in the system.

9.1.1 Exploratory Test

This test's main purpose is to examine the effectiveness of the product's preliminary design concepts [34]. The test was conducted after the specification of requirements and preliminary design was completed. The test was based on the

wireframes displayed in section 7.4.3, which consisted of the fundamental elements of the interface. To collect feedback on the product and see if the UI meet the client's expectations, the client was chosen as the participant for the test. The main points from the feedback were:

- Fewer clicks for changing status and assigning tasks
- Notifications are not first priority
- Inline buttons of the task in a row
- The use of progress bars for the tasks

All the notes from the exploratory test can be seen in appendix G.

9.1.2 Assessment Test

The assessment test is still in the "Prototyping, Design and Testing"-phase in the product's life-cycle like the exploratory test [34]. The test was conducted after the "Detailed Design"-phase was completed. The test builds upon the findings of the exploratory test by evaluating the usability of the product. This is done by getting a user, the client, to perform various tasks. After the tasks had been performed, the main takeaways were:

- The types of a tasks are cleaner and bartender
- The screen size of what the UI should fit
- The overdue tasks should be placed at the top of the list of tasks with a warning next to it
- Implementing a feature to filter tasks by date, assignees, urgency, frequency and type
- Logging the timestamp when editing, progressing and completing a task

9.1.3 Validation Test

To assure that the UI of the product is user-friendly, a usability test, in the form of a validation test, was conducted in cooperation with staff members of The Living Room and 4 students from Aalborg University Copenhagen. The test was conducted in the "Final Testing and Product Launch"-phase of the product life-cycle and after the "Product Build"-phase [34].

The purpose of the test was to identify any problems in the system that were related to usability. When planning a usability test, it is important to keep in mind that the amount of new usability problems the observer is able to discover per participant often decreases rapidly after 5 to 6 participants [30]. Therefore, only five participants conducted the validation test.

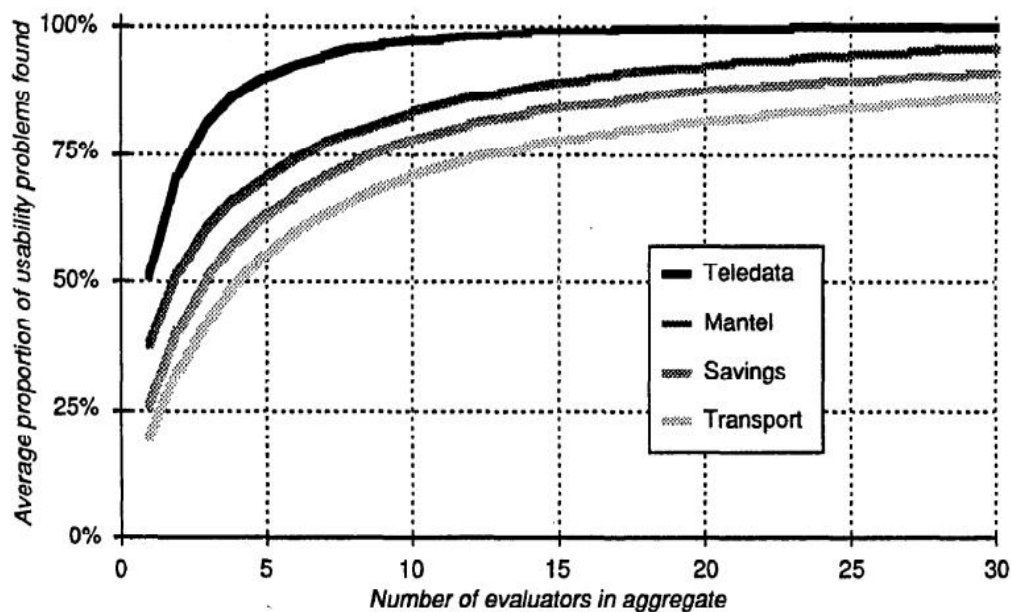


Figure 9.1: Proportion of usability problems found by aggregates of size 1 to 30

[30]

University students were asked to join the test, as it was not possible to have additional staff members of The Living Room participate in the test due to time

constraints. The students were chosen as a viable alternative, as the employees of The Living Room were also young adults who most likely attend some form of higher education. This could also be seen as a validation test, as there was a need to ensure that the product satisfied the user's needs and the established requirements, with the currently implemented features. The structure of the test could be referenced as a lab test, as opposed to a field test, as the focus was to give the participants a specific set of assignments which would reflect the same tasks they would complete on an average workday. To combat the weaknesses of a lab test, the assignments given would be realistic so the user could validate whether or not the product would be a viable alternative to their current solutions at The Living Room.

Before the test was conducted, a plan for how the test should be structured was made (see appendix A). This includes a planning process, with the purpose mentioned previously, a task list, and a set of instructions to prepare for the test. The task list was designed to fit the user's average workday tasks and was therefore heavily inspired by the tasks provided by The Living Room in chapter 4.2.1.

While the participant would interact with the product, they were asked to think aloud while trying to complete the assignments. In addition to the participant, there was an observer, who would notify any usability problems that arose. After the problems were identified, a precise description of the cause of the issue was created and ranked into a problem list which can be seen in figure 9.2.

Assignment	Problem description	Category	Experienced by					
			Manager	Employee	Person 1	Person 2	Person 3	Person 4
"There's an upcoming task called "Event preparation", but we don't know the date. Can you find it without knowing the date?"	Was not able to complete the assignment.	Critical High delay						
	Could not find the "All tasks" button. This was because the text on the button was cropped and therefore they were not able to read it.		x					
"There are multiple tasks due today, all with different amounts of progress, try to sort them by their progress in order to complete them efficiently."	Closed the program.	Critical Significant diff. in expectation vs actual						
	Since the filter button is right next to the exit program button, the user interpreted the exit program button as a button that would close the filter options.		x					
"You need to call your employee called Freja, but you have forgotten her number, try to find it."	Took a long time to find the correct menu.	Serious Medium delay						
	The button "manage team" was hard to find because the user tried to find the information within the tasks.		x					
"There are multiple tasks due today, try to sort them by their urgency in order to complete them efficiently"	Took a long time to find the correct menu.	catastrophic High amount of delay						
	The filter button was difficult to find and took a long time to find in comparison to how fast they solved the previous assignments.			x	x		x	

Figure 9.2: Problem list formed from the validation test

As seen in the problem list, a number of problems did arise during the validation test. They were all observed by the designated observer and assessed afterwards. The assessment of the issues was made to give a clear description of what the issue was, what the cause of the issue was, and how big of an obstacle the issue could potentially be for the user. There was a similarity between the cause of the problems, which was that most of them were caused by the user interface being unclear. This was expected, but by having the problem list, the exact solution to

the problem becomes clearer. The evaluation method used for this usability test was the instant data analysis (IDA) method 2.4.3, whose purpose is to analyze data quickly. There was not enough time to create another revision of the product, but the information gathered from the IDA technique was valuable for the future works section 10.3 as it includes the issues of the usability problems and their possible solutions. The document used for the IDA technique is available in appendix A.

9.2 Unit Testing

The Java testing framework, which was used to assure reliability and maintainability via unit testing, was JUnit 5. The tests targeted the models and the DatabaseMethods interface. The code coverage in the product is not near 100%, because the controllers and views are using FXML files and JavaFX GUI in their methods. The FXML files are non-testable by JUnit 5 and the JavaFX GUI has a test library TestFX, which the group has decided, is too time-consuming to implement into the product within the given time. The main focus of the tests was to test that fetching and sending the data to and from the database works as expected. Furthermore, additional tests were implemented to control that the model classes Task and User do not accept invalid input from the user when creating tasks and users.

It is essential that while developing the product, the group assures that the functions return the expected outputs. It saves the group time when developing the features further because the existing code can be checked that it has not been broken by running the automated tests instead of manually testing the whole product multiple times. Automatic testing also makes the code base easier to maintain throughout the creation of the product. When a unit test is created and a bug occurs in the product, it is effortless and fast to check where in the code base the function and methods are outdated or failing.

Chapter 10

Discussion

In this section, the solution of the project, and the process leading up to the solution, are discussed. As for the product, different (likely better) approaches to designing the solution, the degree of success at which current features are implemented, and yet-to-be-implemented features from the MoSCoW-model and user feedback are discussed. Considering the project itself, the course of action and cooperation with the client are discussed in the process section 10.4.

10.1 Java as front-end

In hindsight, it is clear that using Java for purely back-end, server-side development in the context of a web application would have resulted in a more maintainable and up-to-date solution. Because, *when the browser matured and became a clear difference between backend and frontend programming models, Java shifted towards server-side work.* [33] A viable option for a new front-end could be a combination of React (UI library) and Bootstrap (front-end toolkit), providing a component-based, (almost) drag-and-drop level simplicity, and solid looks.

10.2 Watch for changes

10.2.1 Listen for update

From a usability standpoint, listening for changes in the database and automatically updating the user interface was considered a relatively high priority, but due to time limitations, this crucial functionality was left out. Certainly, refreshing a page rather than listening for changes is not a favourable option. Even though a full implementation was not possible, the technology itself was experimented with, and a working, small-scale example was written, implementing the guide provided by the MongoDB team [39].

The key concept to understanding real-time updates in a MongoDB setting is *change streams*. In basic terms, *a change stream allows applications to watch for changes to data and react to them* [39]. A change stream returns change events if changes occur, which are documents containing information about the updated data. Change streams in Java are not asynchronous, which means that no parallel operation can be run while a change stream is open. There are some potential workarounds, such as closing and opening the change streams with every action or, preferably, running a parallel application in the background.

10.2.2 Observer pattern

Another solution to updating the UI, when the models get updated, is to make use of the observer pattern. The observer pattern is a design pattern in which an object has a list of observers who are notified by the object when a change has occurred. This is usually done by calling one of their methods, for instance, to update the UI [14]. This was not done due to time constraints.

10.3 Future Work

10.3.1 Minimizing server interaction

Another aspect of the prototype, which could be improved to better align with the needs of the client, is the loading time of certain pages. Following brief research, it was found that working with MongoDB, *a single complex query can bring one's code grinding to a halt* [7]. Fetching numerous tasks and passing them to the rather outdated JavaFX-based UI is understandably an expensive operation, but the constant query could be streamlined in combination with the implementation of the previously described change streams. Fetching all data and working with them locally, and only updating new entries, could dramatically reduce loading times as no new connection should be established when doing simple read operations. Minimizing server interaction was not achieved due to time limitations.

10.3.2 Model translation

The prototype did not implement a translator for the database model. This is a disadvantage if the group decides to change the database and database API at some point in the future. A model translator could prevent this issue because the database functions and the methods in the UI would not be dependent on each other. The translator could translate the data from the database into JSON, which is very lightweight in the transportation back and forth in HTTP requests and it is easily read. In the prototype's case, the datatype from MongoDB is a Document type which should be translated into JSON and then the UI methods should always handle JSON data. With the database functions and the UI methods separated and independent, the database is easily interchangeable. This is obtainable by only editing the database functions to adapt the new database and convert the new data into JSON and the UI methods would stay the same.

10.3.3 Features

The prototype has a few problems and missing functionality which could not be fixed, nor implemented, due to the short time frame of the project and limited resources allocated for each iteration. The prototype has some critical issues (stated in the problem list 9.2), which would be a priority to fix. The formerly mentioned problem list contains all issues discovered during instant data analysis (IDA) and would be a main source of inspiration for future work. The problems found are: it took a long time to find the correct menu when looking for information on an employee, the user closed the program by mistake when trying to reset the sorting feature, and the "All Tasks" button was not visible enough.

It was brought up that resetting the date selection to today's date since manually navigating can be tedious. Also, removing all filters when closing the filter-selection dropdown should be a fairly easy fix to add, since there is already logic coded that resets filters. During testing, being able to permanently delete tasks from not only the history view but from the manager view as well, was also a desired feature. This should be fairly simple to implement, creating a new fxml file for tasks containing a delete button should be a fine solution - the functionality already exists, it just needs to be applied. Lastly, both in history and all-tasks views, rather than grouping tasks in a monthly manner, an additional indicator for weeks was also a desired feature. The logic behind this is considerably more difficult, since identifying a new year and month is a lot simpler than knowing exactly which week of a month a task's deadline is placed - generally throughout the project, working with dates was difficult and time-consuming.

Having notifications was a feature, which was regarded as a relatively high priority at the start of the project, but was unfortunately left out because of time constraints and initial client feedback. Also during testing, it was mentioned

by a participant, that new comments and edits should result in some kind of a signal function sending a notification to the users as seen in appendix G. From a usability standpoint, notifications could provide a substantial improvement, since a manager is not likely to manually check comments for every task, nor always keep track of progress.

At last, the UI could be more context-specific with larger buttons and text, since a busy bartender might not be able to get a close view of the screen

10.4 Process

This section goes in-depth about how the process of this project went and in what aspects it could have been improved. The topics discussed in this section cover how the process was planned and how the team worked together - both on the product and with the client.

10.4.1 Gantt and back-casting

During the early stages of the development of the project, backcasting was used to establish deadlines, and assign them a date they should be reached. The method works by starting at the end of a timeline and then defining the steps needed to reach it by going backwards until the starting date for the project is reached.

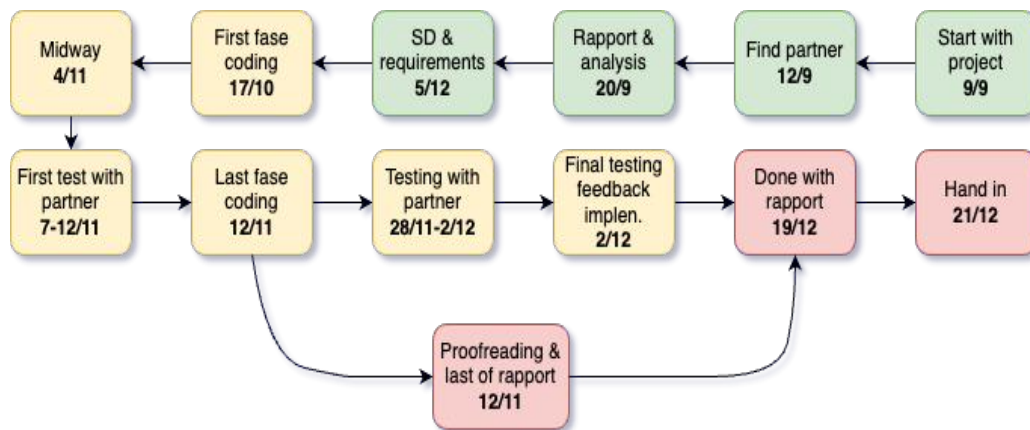


Figure 10.1: Backcasting made during the early stages of the project

In the context of this project, backcasting was used as an outline for the eventual Gantt diagram as seen in figure 10.1 that was created afterwards. The reason for this was that the Gantt diagram was easier to read and much easier to edit.

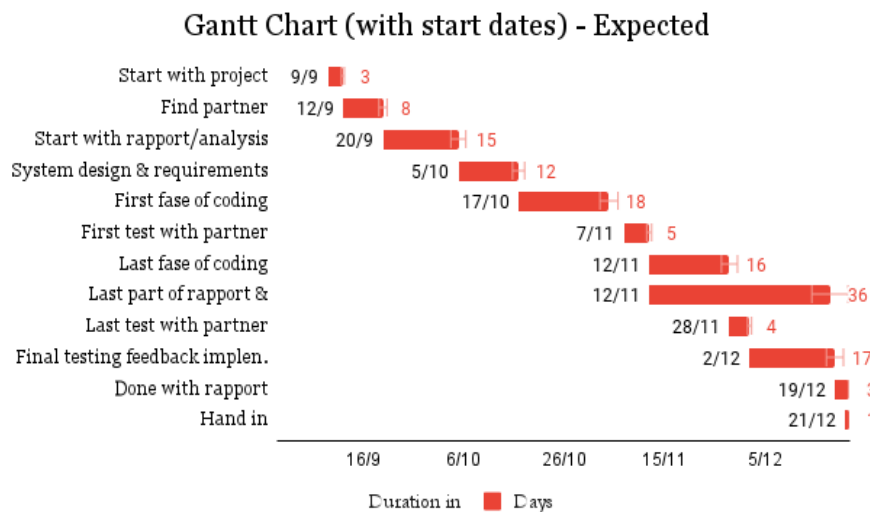


Figure 10.2: Gantt diagram with milestones and the dates we expected to reach them

The first iteration of the Gantt diagram can be seen in figure 10.2, where the dates displayed represent when we planned to reach them. In order to ensure proper documentation, another Gantt diagram was made, which has the

actual dates of the milestones were reached. This iteration was updated regularly throughout the project and can be seen in figure 10.3.

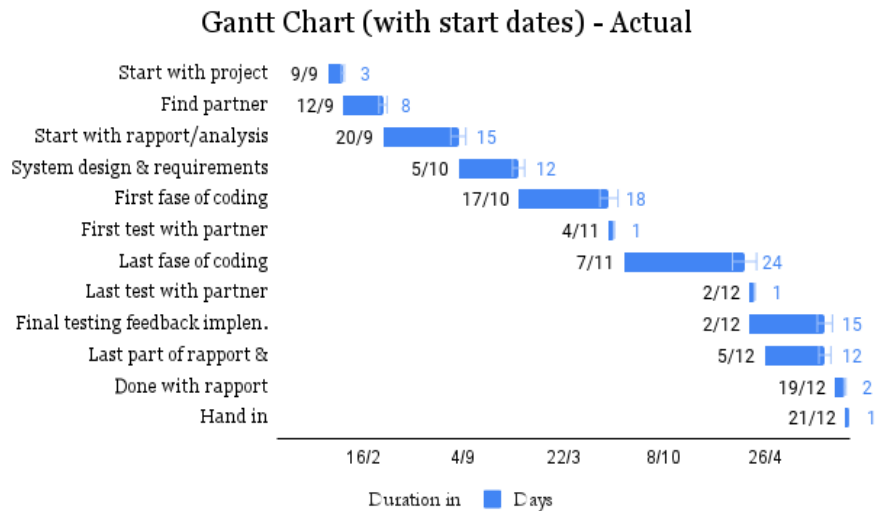


Figure 10.3: Gantt diagram with milestones and the dates we actually reached them

Whenever there was confusion about time management or whether there was time to do certain tasks, the Gantt diagram was used to clear that confusion. Overall, there was a consensus that this method of time management was much better than what members have previously experienced.

10.4.2 Pair coding and code review

Throughout the coding process, pair coding was applied. The main motivating factors were the following; when several people brainstorm and sketch a solution, it is often more optimal, and nobody is left out of the coding process. Since the logic has to be explained before a person can begin its implementation, a number of flaws should come to light and the solution can be optimised. In other cases, two people may consider providing vastly different solutions and they must therefore vouch for their case, which can resemble an exam-like situation, further justifying the validity of the code base.

Before merging new code into the main branch, a code review by two people other than the pair responsible for the code must be done. This helps both in further validating the code, as well as making sure that everyone understands everything in the code.

10.4.3 Communication with client

Communication with the client was done primarily through email and pre-arranged meetings at the cafe. During the very first meeting, the frequency of meetings and communication, as well as expectations from both sides, were established. Both parties agreed that each Friday, a status update should be sent by the group, containing all noteworthy progression, and if possible, a suggestion for the next physical meeting's date and time. During meetings, both regarding the problem at hand and the solution, information was often validated, and material relating to the project, such as time schedules were provided (a consent form for using this information was also signed). Frequent communication was also helpful during testing, as the product was successfully tested with both a manager and an employee despite the busy schedule of the workplace.

10.4.4 Group contract

In order to establish an alignment of expectations, a group contract was made and signed by each member of the group (see appendix B). This was done during a group meeting where each topic was discussed together as a group. When something was agreed on, it was written down. To give examples, some of these topics were "Meeting times", "Roles" and "Break times". The contract was more or less forgotten, as many of these points were dismissed unofficially. There were not any major conflicts either, so there was not much reason to refer back to the group contract anyways.

10.4.5 Conclusion

By analyzing the process of this project, it was clear to see in what ways the choices made influenced the group dynamic, and in turn, the quality of the product. Having proper planning and time management was incredibly beneficial, as it could have been easy to lose focus, both because of the size of the project and the group itself. Establishing a fixed method to develop the product ensured that its quality would remain the same, no matter which member worked on it. Some aspects of the project were not as beneficial for the process, like the group contract, which was not used much during the project, but evaluating and documenting it is important for future work.

Chapter 11

Conclusion

The software solution developed throughout this project proposes an answer to the problem statement 5.6. The problem statement itself was formulated in consultation with the client of this project. During this project, the currently deployed system was explored, alternative solutions were researched, and a new system was designed and refined through an iterative work process. This process included requirement specification, design, implementation, testing, and evaluation. One of the main points of the problem statement was to contribute to achieving an improved overview of everyday task management and administration - including documentation and communication of tasks. Based on the tests conducted, it can be concluded that the solution provides a prototype for a uniform platform, which in theory is a vast improvement in comparison to the current situation. However, due to the relatively short time span of the project, not all requirements specified in section 5.5 were satisfied. As a result, the application did not reach its full potential and was unfortunately never deployed and used at The Living Room. Another crucial point of the problem statement was to reduce the confusion regarding personal responsibilities. Based on testing results, especially focusing on employees, the solution provides solid and easily comprehensible information and provides a platform for communication in case

there is any uncertainty regarding an assignment.

Bibliography

- [1] Nick Babich. *What Is Affordance and How Does it Impact Design?* URL: <https://xd.adobe.com/ideas/principles/web-design/what-is-affordance-design/>. (accessed: 15.12.2022).
- [2] S Balaji and M Dr. Murugauyan. *WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC*. URL: <https://mediaweb.saintleo.edu/Courses/COM430/M2Readings/WATEERFALLVs%20V-MODEL%20Vs%20AGILE%20A%20COMPARATIVE%20STUDY%20ON%20SDLC.pdf>. (accessed: 07.11.2022).
- [3] Balsamiq. *Balsamiq Wireframes*. URL: <https://balsamiq.com/wireframes/>. (accessed: 19.12.2022).
- [4] Kent Beck. *extreme programming eXplained: embrace change*. Addison-Wesley, 2000. ISBN: 9780201616415.
- [5] David Benyon. *Designing User Experience*. 4th ed. Pearson Education, Jan. 2019. ISBN: 9781292155517.
- [6] Alexander Bojsen. *Hvor længe må man opbevare persondata? Få svaret HER!* URL: <https://persondatakonsulenterne.dk/blog-om-gdpr/hvor-laenge-maa-man-opbevare-persondata-faa-svaret-her/>. (accessed: 14.12.2022).
- [7] Craig Buckler. *Solutions for MongoDB*. URL: <https://www.sitepoint.com/7-simple-speed-solutions-mongodb/>. (accessed: 05.12.2022).

- [8] Cameron Chapman. *Exploring the Gestalt Principles of Design*. URL: <https://www.toptal.com/designers/ui/gestalt-principles-of-design>. (accessed: 14.12.2022).
- [9] Alistair Cockburn and Laurie Williams. *The Costs and Benefits of Pair Programming*. URL: <https://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF>. (accessed: 04.11.2022).
- [10] Connecteam. *Homepage (Logged In)*. URL: <https://connecteam.com/>. (accessed: 2.10.2022).
- [11] AAU Institut for Datalogi. *En Velstruktureret Applikation*. URL: <https://moduler.aau.dk/course/2022-2023/DSNSWCB310?lang=da-DK>. (accessed: 14.12.2022).
- [12] Datatilsynet. *Hvad er personoplysninger?* URL: <https://www.datatilsynet.dk/hvad-siger-reglerne/grundlaeggende-begreber-/hvad-er-personoplysninger#:~:text=Personoplysninger%20kan%20for%20eksempel%20v%C3%A6re,at%20oplysningen%20er%20%22personhenf%C3%B8rbar%22..> (accessed: 14.12.2022).
- [13] Azure DevOps. *Azure DevOps Services | Microsoft Azure*. URL: <https://azure.microsoft.com/da-dk/products/devops/#overview>. (accessed: 30.09.2022).
- [14] Andrew Eales. *The Observer Pattern Revisited*. URL: https://www.citrenz.ac.nz/conferences/2005/concise/eales_observer.pdf. (accessed: 13.12.2022).
- [15] Peter Eeles. *Capturing Architectural Requirements*. 2005. URL: <https://web.archive.org/web/20210315183751/http://www.ibm.com/developerworks/rational/library/4706-pdf.pdf>. (accessed: 14.10.2022).
- [16] A. S. Gillis. *What is object-oriented programming?* URL: <https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-00P>. (accessed: 14.12.2022).

- [17] Alexander S. Gillis. *DEFINITION quality assurance (QA)*. URL: <https://www.techtarget.com/searchsoftwarequality/definition/quality-assurance>. (accessed: 02.12.2022).
- [18] Google. *Firebase*. URL: <https://firebase.google.com/>. (accessed: 02.12.2022).
- [19] R. D. Hernandez. *The Model View Controller Pattern – MVC Architecture and Frameworks Explained*. URL: <https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/>. (accessed: 08.12.2022).
- [20] Pankaj Jalote et al. “Timeboxing: a process model for iterative software development”. In: 70.1 (2004), pp. 117–127. URL: <https://www.sciencedirect.com/science/article/pii/S0164121203000104>.
- [21] JGoodies. *rofessional Java Desktop*. URL: <https://www.jgoodies.com/>. (accessed: 16.12.2022).
- [22] Jyoti Jha. *Java Swingx Example*. URL: <https://examples.javacodegeeks.com/desktop-java/swing/java-swingx-example/>. (accessed: 16.12.2022).
- [23] Jesper Kjeldskov, Mikael B. Skov, and Jan Stage. “Instant Data Analysis: Conducting Usability Evaluations in a Day”. In: *Proceedings of the Third Nordic Conference on Human-Computer Interaction*. NordiCHI '04. Tampere, Finland: Association for Computing Machinery, 2004, 233–240. ISBN: 1581138571. DOI: 10.1145/1028014.1028050. URL: <https://doi.org/10.1145/1028014.1028050>.
- [24] Aaron Marcus. *Graphic Design for Electronic Documents and User Interfaces*. 1. ed. Pearson Education, Inc., 1992. ISBN: 9780201543643.
- [25] Lars Mathiassen et al. *Object-oriented analysis & design*. 2. ed. Metodica, 2018. ISBN: 9788797069301.
- [26] Microsoft. *Azure SQL Database*. URL: <https://azure.microsoft.com/en-us/products/azure-sql/database/>. (accessed: 02.12.2022).

- [27] Monday. *monday.com | A new way of working*. URL: <https://monday.com/>. (accessed: 30.09.2022).
- [28] MongoDB. *What Is MongoDB Atlas*. URL: <https://www.mongodb.com/docs/atlas/>. (accessed: 02.12.2022).
- [29] Jakob Nielsen. *10 Usability Heuristics for User Interface Design*. URL: <https://www.nngroup.com/articles/ten-usability-heuristics/>. (accessed: 14.12.2022).
- [30] Jakob Nielsen and Rolf Molich. "Heuristic evaluation of user interfaces". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1990, pp. 249–256.
- [31] Java T Point. *Java AWT Tutorial*. URL: <https://www.javatpoint.com/java-awt>. (accessed: 16.12.2022).
- [32] Java T Point. *Java Swing Tutorial*. URL: <https://www.javatpoint.com/java-swing>. (accessed: 16.12.2022).
- [33] M. Rahul. *Is Java Used In Backend Or Frontend?* URL: <https://www.intervue.io/blog/is-java-used-in-backend-or-frontend#>. (accessed: 08.12.2022).
- [34] Jeffrey Rubin and Dana Chisnell. *Handbook of usability testing: how to plan, design, and conduct effective tests*. 2nd ed. Wiley Pub, 2008. ISBN: 9780470185483.
- [35] P. Runeson. "A survey of unit testing practices". In: 23.4 (2006-07), pp. 22–29. URL: <http://ieeexplore.ieee.org/document/1657935/>.
- [36] Helen Sharp, Yvonne Rogers, and Jenny Preece. *Interaction Design*. 2 ed. John Wiley & Sons, 2007. ISBN: 9780470018668.
- [37] Svend Brinkmann Lene Tanggaard. *Kvalitative metoder: En grundbog*. 3. ed. Hans Reitzels Forlag, 2020. ISBN: 9788741277264.
- [38] Indeed Editorial Team. *What Is User Interface (UI)?* URL: <https://www.indeed.com/career-advice/career-development/user-interface>. (accessed: 02.12.2022).

- [39] MongoDB Editorial Team. *Watch for Changes*. URL: <https://www.mongodb.com/docs/drivers/java/sync/current/usage-examples/watch/>. (accessed: 05.12.2022).
- [40] OpenJFX Team. *JavaFX*. URL: <https://openjfx.io/>. (accessed: 05.12.2022).
- [41] Trello. *Boards | Trello*. URL: <https://trello.com>. (accessed: 30.09.2022).

Appendices

Appendix A

Usability Testing

Things to do before the test

- Manager
 - Make sure there is 1 overdue task for the date, and 2 normal tasks
 - Make a task called “Clean bathroom”, and leave this comment: “We don’t have any more cleaning supplies and the toilet brush is missing!”
 - Create an employee called Freja with all info filled out
 - Create an employee called Erik with all info filled out
 - Create an employee called Thomas with all info filled out
 - Create a task with the date 03-12-22 and set the percentage to 50%
 - Create a task called “Event preparation” with the date being sometime in January
 - Create 5 tasks with the date 02-12-22 and set them to different percentages
- Employee
 - Reset changes made during the test of the manager
 - Create a task called “Open the store”, and set the progress to 0%
 - Create a task called “Clean the restroom”, and set the progress to 0%
 - Create a task called “Food preparation”, and include a basic description about making a sandwich
 - Create 4 different tasks with different degrees of urgency

Tasks

- Tasks for the test subject
 - Manager
 1. Go to the manager page
 2. Try to create a new task called “clean syrup pumps” that repeats once a week
 3. Actually, the task needs to repeat every day, try to edit the task to fix it
 4. An employee is unable to clean the toilet because supplies are missing, they have left a comment on the task to let you know, respond with a comment to let them know you are aware of the issue.

5. You need to call your employee called Freja, but you have forgotten her number, try to find it.
 6. Another employee Erik has changed his phone number to 28335358, try to update it
 7. Thomas has been fired, try to remove them from the team
 8. There is a task for tomorrow that already is set to 50% progress, try to fix it by setting it back to 0%
 9. There's an upcoming task called "Event preparation", but we don't know the date. Can you find it without knowing the date?
 10. There are multiple tasks due today, all with different amounts of progress, try to sort them by their progress in order to complete them efficiently
- Employee
 1. Go to the employee page
 2. There is a task called "Open the store", assuming that has already been done today, try to check that task off as completed
 3. The task called "Clean the restroom" can not be done because the cleaning supplies are missing! Create a comment within that task to let Frank and the other staff members know
 4. You were able to clean the restroom somewhat despite the missing supplies, try to set the progress for that task to 50%
 5. The next task "Food preparation" includes a step-by-step guide that describes how the food should be prepared, try to find it
 6. There are multiple tasks due today, try to sort them by their urgency in order to complete them efficiently

Follow-up questions

- Manager
 - "Did you notice the colored indicator on the left side of a task? If so, what do you think it means?"
 - "Did you notice the exclamation mark on the left side of a task? If so, what do you think it means?"
 - Is there anything that you felt worked well with the program?
 - Is there anything that did not work well?

- Anything you would like to change about it?
- Employee
 - “Did you notice the colored indicator on the left side of a task? If so, what do you think it means?”
 - “Did you notice the exclamation mark on the left side of a task? If so, what do you think it means?”
 - Is there anything that you felt worked well with the program?
 - Is there anything that did not work well?
 - Anything you would like to change about it?

User testing - results

Manager:

- Log in på manager: ingen problemer
- Virker rimeligt intuitivt for ham at tilføje task
- When type selected - vis kun employees fra den kategori
- Typer should be role
- Spørger hvad all er
- Manager view - arrows ved dato
- Today button - brings you back to today
- Update fungerer fint
- Add comment fungerer fint
- Bigger buttons - maybe all the buttons/elements should be bigger
- Find number - not intuitive that it's under manage team, but he found it eventually and said it was fine
- Update number fungerede fint
- Delete employee fungerede fint
- Update progress fungerede fint
- Find task without knowing the date var ikke intuitiv og kunne ikke gennemføres. Anders blev nødt til at hjælpe. Knappen “All tasks” var cropped og kunne derfor ikke ses ordentligt. Bigger buttons...
- Sort urgency fungerer fint
- Nulstil filtre når man trykker på filterknappen (ud ad filterfunktionen)
- Sort by dropdown - all the categories in the filter function - sort not filter
- The trashcan is missing - mangler vi ikke at kunne slette tasks?
- Checked off task says the wrong date in history
- Ask if you're deleting series of tasks or just one instance
- Genovervej hvordan vi sletter tasks
- Red dot (notification dot) når der bliver tilføjet kommentarer
- Mark new tasks? Until someone interacts with it
- Color codes aren't intuitive - leave the dots - Frank, color the bars instead? Or maybe the text

- Make the progress button smaller (just show %), add urgency next to progress button
- Remove the color codes - keep the red circles for overdue tasks - remove !
- Remove the exit button next to the filter button when entering that function
- Weekly view
- Diskussion omkring hvordan employees assignes til tasks

Employee:

- Filterknappen er ikke nem at finde
- Urgency farver kunne være omvendt
- Sortering kunne være nice

Student 1:

- Complete task fungerer fint
- Man kunne også trykke på progress og se om den har en der hedder "done"
- Add comment fungerer fint
- Set progress fungerer fint
- Find description fungerer fint
- Filter fungerer fint - vi burde dog få den til at sortere frem for filtrere
- Forstod godt prikkernes betydning
- Programmet virkede 👍
- Det var langsomt - ingen flydende overgang 🙄
- Ville ikke fjerne farvede prikker. Ville gerne kunne se færdige tasks

Student 2:

- Complete task fungerer fint
- Overvejede om progress havde en 100%
- Add comment fungerer fint - blev lidt forvirret over den tidligere comment
- Set progress fungerer fint - gjorde det før vi bad hende om det
- Find description fungerer fint
- Filter fungerer fint
- Forstod godt prikkernes betydning
- Det er overskueligt. Fine valg af ikoner - det er let at gætte hvad der skal ske 👍
- Ingen kritik - det er meget intuitivt og giver god mening
- Kunne evt. skrive urgency over prikkerne (eller symbol)

Student 3:

- Complete task fungerer fint
- Add comment fungerer fint
- Set progress fungerer fint
- Find description fungerer fint
- Kunne ikke finde filterknappen med det samme - fandt den og så fungerede det fint
- Forstod godt prikkernes betydning
- Meget nemt at finde rundt i
- Filter kunne være mere obvi

Student 4:

- Complete task fungerer fint - lægger mærke til tooltip

- Add comment fungerer fint
- Set progress fungerer fint
- Prøvede at klikke på Assignees, men opdagede så i'et - fungerede fint
- Filter knappen var lidt svær at finde - ellers fungerer den fint
- Forstod godt prikkernes betydning
- Synes det var meget godt. Kunne godt lide tooltips
- Vil gerne kunne se de tasks, der er færdige

Problem list

Assignment	Problem description	Category	Experienced by					
			Manager	Employee	Student 1	Student 2	Student 3	Student 4
“There’s an upcoming task called “Event preparation”, but we don’t know the date. Can you find it without knowing the date?”	<p>Was not able to complete the assignment.</p> <p>Could not find the “All tasks” button. This was because the text on the button was cropped and therefore they were not able to read it.</p>	<p>Critical</p> <p>High delay</p>	x					
“There are multiple tasks due today, all with different amounts of progress, try to sort them by their progress in order to complete them efficiently.”	<p>Closed the program.</p> <p>Since the filter button is right next to the exit program button, the user interpreted the exit program button as a button that would close the filter options.</p>	<p>Critical</p> <p>Significant diff. in expectation vs actual</p>	x					
“You need to call your employee called Freja, but you have forgotten her number, try to find it.”	<p>Took a long time to find the correct menu.</p> <p>The button “manage team” was hard to find because the user tried to find the information within the tasks.</p>	<p>Serious</p> <p>Medium delay</p>	x					
“There are multiple tasks due today, try to sort them by their	<p>Took a long time to find the correct menu.</p>	<p>catastrophic</p> <p>High</p>		x	x		x	

urgency in order to complete them efficiently”	The filter button was difficult to find and took a long time to find in comparison to how fast they solved the previous assignments.	amount of delay	
--	--	-----------------	--

Appendix B

Samarbejdsaftale til P3

1. Mødetid

- Hvis vi ikke har forelæsninger mødes vi i et planlagt tidsrum, som er 9-16, medmindre andet er aftalt.
- Mødetider eller ændringer hertil skal aftales senest kl. 20 dagen før.

2. Mødeform og indhold

- Som udgangspunkt mødes vi på skolen i grupperummene.
- Hvis vi har aftalt, hvad der skal skrives/laves, kan vi mødes på teams.
- Møder over teams aftales ved fysiske møder.
- Indhold af møderne planlægges fra gang til gang.
- Der holdes morgenmøde om onsdagen.
- Ugentlig kode-opsamling om fredagen.

3. Referater

- Der er altid mindst 1 person til møderne, som noterer, hvad der bliver sagt / snakket om.
- Der skrives logbog hver arbejdsdag.

4. Roller

- Sekretær, kontaktperson til vejleder.
- Ordstyrer til samtlige møder.
- Referent, Logbog/Gantt.
- Korrektur-person..
- Coding buddies.

5. Pauser

- Vi holder pause når alle er enige om at det er okay at holde en pause. Individuelle pauser er også okay, så længe resten af gruppen godkender det.
- Hvis man forlader gruppen, må man gerne sige, hvor man går hen.

6. Specielle behov

- Arbejdstider regnes med i gruppens mødeform.
- Fritidsaktiviteter regnes med i gruppens mødeform.
- Hvis en deadline ikke kan overholdes, skal dette diskuteres og revurderes i gruppen.

7. Ferie

- I officielle ferier fra uni (juleferie, sommerferie): Det forventes ikke, at man arbejder på projektet, men forventningen kan ændres, hvis gruppen føler for, at rapporten skal arbejdes på i ferieperioden.
- Hvis der er tale om privat, individuel ferie, så skal man oplyse gruppen så tidligt som muligt.
- Man fritages ikke for ansvar til gruppen ved privat ferie.
- Ferie for gruppen kan aftales.

8. Weekend

- Weekenden er fri medmindre andet er aftalt.

9. Forventninger

- I skriveperioden forventes det, at man læser rapporten igennem mindst én gang om ugen.
- Der skal være plads til at alle kan dele sine tanker. Gruppen lytter aktivt.
- Der tjekkes op på hjemmearbejde, når der arbejdes på dette.
- Lav klare aftaler om, hvad der skal ske/laves og udspecificer arbejdsopgaver.
- Der skal være plads til, at alle kan få hjælp til sine opgaver, hvis der er behov for dette.
- Alle skal have kodet på programmet på en eller anden måde.
- Løbende kommunikation er vigtig!
- Følg guidelines for kodning, se "Guidelines" (Navngivning af diverse variabler og funktioner).
- Alle skal kunne stå inde for produkt inden deadline.
- Der skal være mulighed for at være sociale som gruppe uden for skolen.

10. Kurser

- Det er forventet, at hvis man er bagud i et fag, så skal man aktivt prøve at komme op på et acceptabelt niveau. Andre gruppemedlemmer skal også være opmærksomme på at holde faglighedens generelle niveau ved at hjælpe hinanden.

11. Arbejdsformer

- Coding buddies/pair programming.
- Opgaver i mindre grupper.
- Individuelt arbejde med opgaver.

12. Deadline

- Gruppen er opmærksom på at specificere deadlines og hvad de omfatter undervejs, da en deadlines indhold kan variere gennem forløbet.
- Der udarbejdes deadlines internt i gruppen løbende gennem projektet, så projektet føles så overskueligt som muligt.

13. IT-Værktøjer

- LaTeX, Google Drive, Github, Visual Studio Code/Intellij IDEA, Discord, Teams, Draw.io.

14. Kommunikation

- Respektivt. Hvis der er noget man ikke kan nå eller deltage til, notificeres gruppen A\$AP over messenger. Hvis det er aftalt at gruppen arbejder er det ligegyldigt hvor i verden man er, man skal være tilgængelig.
- En aftale er vedtaget, når alle i gruppen har reageret på beskeden omkring den givne aftale over messenger.

15. Rettelser

- Hvis man vil lave markante rettelser i andres tekst i Overleaf, skal man lave en kommentar på teksten, før man laver rettelsen. Rettelser af stavefejl eller lette omformuleringer er ellers i orden.
- Rettelser skal være konstruktive og må ikke være rettet mod en person.

- Man skal kunne acceptere eventuelle rettelser i sin egen tekst uden at blive fornærmet.

16. Konfliktbehandling

- Cool seat: man kan kalde cool seat på ethvert vilkårligt tidspunkt når gruppen er samlet. Personen har fri taletid i maks 5 min, uden afbrydelser fra resten af gruppen.
- Hvis gruppe-kontrakten ikke overholdes af et medlem, laves der en intervention

Samarbejdskontrakt med vejleder

Samarbejdskontrakt mellem gruppe 1 (Louise, Anders, Amalie, Freja, Magnus og Ben) og vejleder med følgende krav:

- Der mødes til tiden.
- Der skrives, hvis der er en, som bliver forsinket eller hvis mødet skal aflyses.
- Der sendes agenda for det kommende møde mindst 24 timer før mødet.
- Mødet skal tage udgangspunkt i det sendte agenda.

Kontrakten kan opdateres løbende, hvis begge parter er enige om det.

Appendix C

Date	Cake (5 Co)	Milk (5 Co)	Food (5 Co)	Kitchen Left (5 Co)	Kitchen right (5 Co)	Bar reezer (-18 Co)	Dish- washer (80 Co)	Outside Fridge - Left (5 Co)	Outside Fridge - Right (5 Co)	Inside Storage reezer (-18 Co)
03 January (Monday)										
06 January (Thursday)										
09 January (Sunday)										
12 January (Wednesday)										
15 January (Saturday)										
18 January (Tuesday)										
21 January (Friday)										
24 January (Monday)	4.8	4.2	4.3	4.3	4.3	-20.0	85.0	4.3	4.2	-21.0
27 January (Thursday)	4.2	4.3	4.3	4.5	4.3	-20.0	84.0	4.3	4.3	-20.0
30 January (Sunday)	4.3	4.3	4.5	4.5	4.3	-20.0	85.0	4.3	4.3	-20.0
02 February (Wednesday)	4,2	4,3	4,5	4,9	4,3	-20.0	84.0	4,3	4,3	-21.0
05 February (Saturday)	4,2	4,2	4.2	4.3	4.4	-21.0	85.0	4.5	4.4	21.0
08 February (Tuesday)	4.2	4,9	4.7	4.5	4.3	-20.0	84.0	4.3	4.5	-20.0
11 February (Friday)	4.3	4,2	4.1	4.3	4.3	-22.0	84.0	4.3	4.5	-21.0
14 February (Monday)	4.2	4.3	4.3	4.7	4.6	-21.0	84.0	4.3	4.5	-22.0
17 February (Thursday)	4.5	4.2	4.3	4.3	4.3	-21.0	85.0	4.2	4.4	-22.0
20 February (Sunday)	4.3	4.3	4.5	4.5	4.3	-20.0	85.0	4.3	4.3	-20.0
23 February (Wednesday)	4.3	4,2	4.1	4.3	4.3	-21.0	84.0	4.3	4.5	-22.0
26 February (Saturday)	4.2	4.3	4.3	4,9	4,3	-20.0	84.0	4.3	4.5	-20.0
01 March (Tuesday)	4.6	4.5	4.2	4.3	4.5	-20.0	83.0	4.8	4.6	-22.0
04 March (Friday)	4.2	4.5	4.2	4.5	4.4	-22.0	84.0	4.1	4.6	-20.0
07 March (Monday)	4.1	4.5	4.3	4.8	4.1	-22.0	84.0	4.2	4.3	-22.0
10 March (Thursday)	4.2	4.5	4.3	4.3	4.2	-21.0	84.0	4.3	4.4	-20.0
13 March (Sunday)	4.2	4,9	4.7	4.5	4.3	-20.0	84.0	4.3	4.5	-20.0
16 March (Wednesday)	4.3	4.3	4.5	4.5	4.3	-20.0	85.0	4.3	4.3	-20.0
19 March (Saturday)	4.5	4.2	4.3	4.3	4.3	-21.0	85.0	4.2	4.4	-22.0
22 March (Tuesday)	4.1	4.5	4.3	4.8	4.1	-22.0	84.0	4.2	4.3	-22.0
25 March (Friday)	4.2	4,9	4.7	4.5	4.3	-20.0	84.0	4.3	4.5	-20.0
28 March (Monday)	4,2	4,2	4.2	4.3	4.4	-21.0	85.0	4.5	4.4	21.0
31 March (Thursday)	4.8	4.2	4.3	4.3	4.3	-20.0	85.0	4.3	4.2	-21.0
03 April (Sunday)	4.3	4.3	4.5	4.4	4.2	-21.0	85.0	4.3	4.4	-21.0
06 April (Wednesday)	4.6	4.5	4.2	4.3	4.5	-20.0	83.0	4.8	4.6	-22.0
09 April (Saturday)	4.2	4.5	4.2	4.5	4.4	-22.0	84.0	4.1	4.6	-20.0
12 April (Tuesday)	4.3	4.3	4.5	4.5	4.3	-20.0	85.0	4.3	4.3	-20.0
15 April (Friday)	4.5	4.2	4.3	4.3	4.3	-21.0	85.0	4.2	4.4	-22.0
18 April (Monday)	4.1	4.5	4.3	4.8	4.1	-22.0	84.0	4.2	4.3	-22.0
21 April (Thursday)	4.2	4,9	4.7	4.5	4.3	-20.0	84.0	4.3	4.5	-20.0
24 April (Sunday)	4.3	4.7	4.4	4.3	4.2	-20.0	85.0	4.3	4.4	-20.0
27 April (Wednesday)	4.6	4.5	4.2	4.3	4.5	-20.0	83.0	4.8	4.6	-22.0
30 April (Saturday)	4.2	4.5	4.2	4.5	4.4	-22.0	84.0	4.1	4.6	-20.0
03 May (Tuesday)	4.1	4.5	4.3	4.8	4.1	-22.0	84.0	4.2	4.3	-22.0
06 May (Friday)	4.3	4.3	4.5	4.5	4.3	-20.0	85.0	4.3	4.3	-20.0
09 May (Monday)	4.5	4.2	4.3	4.3	4.3	-21.0	85.0	4.2	4.4	-22.0
12 May (Thursday)	4.1	4.5	4.3	4.8	4.1	-22.0	84.0	4.2	4.3	-22.0
15 May (Sunday)	4.2	4,9	4.7	4.5	4.3	-20.0	84.0	4.3	4.5	-20.0
18 May (Wednesday)	4.3	4.7	4.4	4.3	4.2	-20.0	85.0	4.3	4.4	-20.0
21 May (Saturday)	4.6	4.5	4.2	4.3	4.5	-20.0	83.0	4.8	4.6	-22.0
24 May (Tuesday)	4.5	4.5	4.2	4.6	4.7	-21.0	84.0	4.4	4.3	-21.0
27 May (Friday)	4.6	4.5	4.2	4.3	4.5	-20.0	83.0	4.8	4.6	-22.0
30 May (Monday)	4.2	4.5	4.2	4.5	4.4	-22.0	84.0	4.1	4.6	-20.0
02 June (Thursday)	4.3	4.4	4.2	4.3	4.4	-21.0	85.0	4.3	4.2	-20.0
05 June (Sunday)	4.5	4.5	4.2	4.5	4.3	-20.0	84.0	4.3	4.5	-20.0
08 June (Wednesday)	4.6	4.5	4.2	4.3	4.2	-20.0	85.0	4.3	4.4	-20.0
11 June (Saturday)	4.3	4.3	4.5	4.3	4.5	-20.0	83.0	4.8	4.6	-20.0
14 June (Tuesday)	4.5	4.2	4.3	4.6	4.7	-21.0	84.0	4.4	4.3	-20.0
17 June (Friday)	4.3	4.4	4.2	4.3	4.4	-21.0	85.0	4.3	4.4	-20.0
20 June (Monday)	4.2	4.5	4.2	4.5	4.3	-20.0	84.0	4.4	4.3	-21.0
23 June (Thursday)	4.3	4.3	4.5	4.3	4.4	-21.0	85.0	4.3	4.2	-20.0
26 June (Sunday)	4.5	4.2	4.3	4.6	4.7	-20.0	83.0	4.8	4.6	-22.0
29 June (Wednesday)	4.5	4.2	4.3	4.5	4.6	-20.0	84.0	4.8	4.5	-21.0
02 July (Saturday)	4.5	4.2	4.3	4.2	4.4	-21.0	83.0	4.6	4.4	-21.0
05 July (Tuesday)	4.3	4.3	4.5	4.3	4.4	-21.0	85.0	4.3	4.2	-20.0
08 July (Friday)	4.6	4.5	4.2	4.3	4.5	-20.0	83.0	4.8	4.6	-22.0
11 July (Monday)	4.3	4.3	4.5	4.3	4.5	-20.0	83.0	4.8	4.6	-20.0
14 July (Thursday)	4.5	4.2	4.3	4.6	4.7	-21.0	84.0	4.4	4.3	-20.0

17 July (Sunday)	4.5	4.2	4.3	4.6	4.7	-20.0	83.0	4.8	4.6	-22.0
20 July (Wednesday)	4.5	4.2	4.3	4.5	4.6	-20.0	84.0	4.8	4.5	-21.0
23 July (Saturday)	4.5	4.2	4.3	4.2	4.4	-21.0	83.0	4.6	4.4	-21.0
26 July (Tuesday)	4.6	4.5	4.2	4.3	4.5	-20.0	83.0	4.8	4.6	-22.0
29 July (Friday)	4.3	4.3	4.5	4.3	4.5	-20.0	83.0	4.8	4.6	-20.0
01 August (Monday)	4.2	4.5	4.2	4.5	4.3	-20.0	84.0	4.4	4.3	-21.0
04 August (Thursday)	4.3	4.3	4.5	4.3	4.4	-21.0	85.0	4.3	4.2	-20.0
07 August (Sunday)										
10 August (Wednesday)										
13 August (Saturday)										
16 August (Tuesday)										
19 August (Friday)										
22 August (Monday)										
25 August (Thursday)										
28 August (Sunday)										
31 August (Wednesday)										
03 September (Saturday)										
06 September (Tuesday)										
09 September (Friday)										
12 September (Monday)										
15 September (Thursday)										
18 September (Sunday)										
21 September (Wednesday)										
24 September (Saturday)										
27 September (Tuesday)										
30 September (Friday)										
03 October (Monday)										
06 October (Thursday)										
09 October (Sunday)										
12 October (Wednesday)										
15 October (Saturday)										
18 October (Tuesday)										
21 October (Friday)										
24 October (Monday)										
27 October (Thursday)										
30 October (Sunday)										
02 November (Wednesday)										
05 November (Saturday)										
08 November (Tuesday)										
11 November (Friday)										
14 November (Monday)										
17 November (Thursday)										
20 November (Sunday)										
23 November (Wednesday)										
26 November (Saturday)										
29 November (Tuesday)										
02 December (Friday)										
05 December (Monday)										
08 December (Thursday)										
11 December (Sunday)										
14 December (Wednesday)										
17 December (Saturday)										
20 December (Tuesday)										
23 December (Friday)										
26 December (Monday)										
29 December (Thursday)										
01 January (Sunday)										
04 January (Wednesday)										

NB! If there is a problem with the temperature of any of the machines then use [the Problem / Solution Form](#)

Appendix D

LR Bartender – Daily Routines


Date: _____

Opening Shift: _____ Afternoon Shift: _____ / _____


'After Opening' Bartender Routines:

(Sign each item when completed)

Sign

- Bake **Cookies, Muffins & Banana Cake**
- Check that music is at correct volume levels (all knobs should be at **RED** lines)
- Write products on the glass façade (cookie & muffin display)
- Slice extra tomatoes in metal container (**NB!** drain extra water from container)
- Fill 1 square metal container with bananas (cut into 1/3 pieces)
- Prepare 2 bottles of Mango / Passionfruit mix (6 dl Mango + 3 dl Passionfruit)
- Prepare Rooibos & Hibiscus teas (NB! write name and date on containers)
- Prepare Ginger/Lime juice
- Make Bread Roll mix for next day
- Check Tuna mix – if necessary to make single or double portion
- Cut 1 cm from bottom of fresh Dill and change water (only 1 cm of water)
- Write today's date on bottom of delivered breads from bakery
- Check: milk draw is full and check expiration dates of milk, yoghurt & cream
- Check: 1 Sugar Syrup bottle by the food prep area + 1 Sugar bottle on shelf
- Peel ginger and place in plastic container (**NB!** change water everyday)
- Check Calendar book for customer reservations and prepare reservation signs
- Refill the sugar bowl, straws and napkins by the service station
- Refill and clean the Salt and Pepper grinders by the service station
- Clean Monin **syrup pumps** (Cleaning Instructions: pump syrup back into bottles and then clean pump with hot water from tea brewer)
- Clean **Monin metal stand** next to the espresso machine in the dishing machine
- Bake cakes: Tiramisu, Muffins, Cheesecake, Brownie Cake (*if necessary*)
- Wash food stains from the 'Grey' wall behind the toaster and espresso machine
-  **Clean brick wall behind the juice grinder**
- Straighten liquor bottles on the shelves & turn bottle labels facing forward
- Change glasses by cash register (credit card receipts & bottle caps)
- Clean all the menu display signs above the cookie display with window cleaner
- Wash snack bowls and glasses by the food station in the dishwasher
- Restock cakes from outside storage room (if less than ½ cake in fridge)
NB! Remember to write expiration date on the back of the cake display sign

 **Check salad ingredients / dressings to see if they need to be prepared / refilled**

● Restock bar from outside storage room	
 Evening Shift: To bake / prepare cakes if less than 3/4 in the cake fridge	
● <u>Monday:</u> Clean the 'take away cover' & 'sleeve' containers in dishwasher	
● <u>Monday:</u> Clean the 'plastic containers' in the draws under toaster in dishwasher	
● <u>Tuesday:</u> Wash the 'sugar stick' glass jar in dishwasher	
● <u>Tuesday:</u> Wash area under the bar sink and the 'black box' in dishwasher	
● <u>Wednesday:</u> Wash sugar bowl, cinnamon & choco shakers in dishwasher	
● <u>Thursday:</u> Clean the fan near the toaster (take it apart and clean in dishwasher)	
● <u>Thursday:</u> Make sugar syrup for weekend shifts (4 bottles)	

Standard Bartender Shift Tasks:

● Prepare coffee beans and add to the coffee grinder (NB! Grind just before brewing coffee)
● Clean fresh juice container & empty used fruit bucket at the back of the machine
● Check fruit containers (for smoothies) and prepare extra fruit if necessary
● Rinse black rags
● When restocking fresh limes rinse them first in the sink (inside the metal basket)
● Clean food station area
● Refill napkins, sugar and straws by the service station
● Wipe espresso grinders clean (grinder tray and around machine)
● Make an extra chocolate sauce
● When making a new whipped cream write the date on the container in colored marker
● When opening a new bottle of Red or White wine to write the date on the bottle NB! Red wine is OK for 3 days from opening / White wine is ok for 4 days from opening
● Place delivered bakery products out on plates or into plastic storage boxes or take to the outside storage room (cakes, carrot cake, etc) NB! Write date on the box
● Restock any missing cakes from the storage room if necessary
● @ 1900: Change the 'Cake / Cocktail' sign to the 'Cocktail' side
Before Afternoon Restocking:
● Check milk draw
● Check fresh fruit
● Check frozen fruit draw (strawberries, raspberries, blueberry, pineapple)

● Check sliced tomatoes		
● Check Tuna mix		
<u>Things ‘To Do’ when not busy:</u> <i>(Write the time and sign your initials when each item is completed)</i>	Time	Sign
● Clean inside cake refrigerator (wipe glass shelves and any food on bottom)		
● Sort and organize glasses under the cash register NB! ‘Small’ glasses on the left / ‘Large’ glasses on the right		
● Clean white cutting board next to sandwich toaster		
● Clean the bottom of the milk draw		
● Clean inside the metal milk containers with a metal sponge		
● Clean Imported Beer Frig shelves and window (<i>remove beer first</i>)		
● Change the water in flower vase by the cash register (<i>if we have flowers</i>)		
● Clean underneath Espresso Machine (as far underneath machine as possible)		
● Clean & organize the liquor shelves above the bar sink NB! 3 bottles of each liquor / 6 bottles of Havana Club white Rum		
● Refill coffee beans in coffee grinders (at the front of the bar)		
● Restock Take Away cups, Napkins, etc from the downstairs storage room		
● Clean Syrup <u>Rack</u> (next to the espresso machine) in the dishwasher		
● Make a new Carrot Cake (if there are 4 pieces or less in the cake frig)		

Appendix E

LR Weekly Task List

	Employees	Mobile	Responsibility Area
Bartenders	Amalie	27507730	Clean & Organize Shelves Under Cash Register
	Amanda	40271038	Clean Food Fridge Gaskets and metal draws inside dishwasher
	Cari	29714166	Clean & Organize Draws Under Toaster
	Flora	53542969	Clean Baking Oven Area (inside oven, front façade, and on top of oven)
	Frida	42227942	Clean & Organize Shelves under Cookie Display area (NB! Remember to rotate milk and juices on dates)
	Karla	51427404	Clean all facades behind bar + Wash underneath Main Bar Sink
	Luna	31215078	Clean and organize draws under baking oven
	Mikkel	93864513	Blackboards / Social Media
	Mikkeline	60546098	Clean Inside Milk Refrigerator (Pull out draws and wash inside and draw gaskets)
	Sara J	29264403	Clean & Organize Shelves under 4 Coffee Grinders
	Sara S	28184909	Clean & Organize Corner Shelves by Food Refrigerator
	Vencel	53809750	Clean inside Ice Machine
	?		Clean Cake Fridge (Bottom shelf and inside Door track)
	Lasse	30424890	Backup Bartender
Cleaners	Anna Egholm	42615291	Clean 2 Glass Shelves at the end of the bar (above toaster)
	Vera	53809750	Clean 4 Glass shelves behind bar
	Kira	28876555	Clean shelves in Dishwashing Room
	Anna Ohlsen	52477804	
	Frida	42227942	
	Helena	27140025	Wash Kitchen Fridges (inside shelves and front door)
	Marie	28304756	Clean Baking Oven Area (on top and inside shelves)

Appendix F

Re: Progress report!

Frank Zadi <frank@thelivingroom.dk>

ti 08-11-2022 09:54

Indbakke

Til: Anders Mazen Youssef <amyo21@student.aau.dk>;

Hey Anders,

Here's a quick overview of our cafe as requested by your student colleagues. I hope this helps...

The Living Room café is owned by Frank Zadi and Tiril Haaland. The café is located in the heart of the Latin Quarter neighborhood in Copenhagen. We offer a mix of different homemade food and drinks as well as a wide range of cocktails in the evenings. Our style is a mix of modern and Danish 70's retro to create a cozy 'hygge' vibe. We have a lounge area in the basement filled with couches and a fireplace for those cold winter months as well as a Moroccan room with a more middle eastern touch.

Regards,
Frank Zadi
The Living Room
Larsbjørnsstræde 17
1454 Copenhagen - Denmark
Cafe: +45 33326610
Mobile: +45 26291000

----- Original Message -----

Subject: Progress report!

Date: 27-10-2022 14:16

From: Anders Mazen Youssef <amyo21@student.aau.dk>

To: Frank Zadi <frank@thelivingroom.dk>

Hi Frank,

We are currently in full development of our first prototype, where our milestones are getting the basic functions down.

[We would like to meet with you again soon, in order to get your thoughts on our current version.](#)
[Are you available sometime next week?](#)

In the meantime, we have attached some screenshots for you to see :)

The screenshots are not mockups. The buttons have functionality and communicate with our database.

Kind regards,

AAU Group 1

Appendix G

Living room Exploratory & assessment notes

Wednesday, 12 October 2022 13.06

Exploratory test notes

- Look to google calander for inspiration on the layout of how a task should be created
- Confusion in what pending tasks mean
- When an employee moves a task to active, they should maybe be able to tell who they are
- Less clicks of the task assigning
 - Frank said to maybe just have a table of tasks like the sheets document
 - Rework the whole page into a table format
- Notifications in general are not important
- Email notifications would be a good idea when changes are made
- Maybe use progress bars even more, like for each task

Assessment test notes

- Cleaner or bartender types
- Frequency fine
- Log timestamp for progression, other edits and completion
- Filters for showing tasks (overview)
- Filter based on roles, percentages, assignees
- Always show overdue tasks first
- Clearly mark overdue tasks (an exclamation mark or other icon could be a good idea)
- Status bar colour code based on urgency
- Mark if the manager writes a comment. Maybe mark with the assignee's name if the employee comments
- Note internet dependency in the report

Appendix H

<i>Criterion</i>	<i>Measure of</i>
Usable	The system's adaptability to the organizational, work-related, and technical contexts.
Secure	The precautions against unauthorized access to data and facilities.
Efficient	The economical exploitation of the technical platform's facilities.
Correct	The fulfillment of requirements.
Reliable	The fulfillment of the required precision in function execution.
Maintainable	The cost of locating and fixing system defects.
Testable	The cost of ensuring that the deployed system performs its intended function.
Flexible	The cost of modifying the deployed system.
Comprehensible	The effort needed to obtain a coherent understanding of the system.
Reusable	The potential for using system parts in other related systems.
Portable	The cost of moving the system to another technical platform.
Interoperable	The cost of coupling the system to other systems.

Figure 9.1: Classical criteria for software quality

Figure 1: Description for each criterion [25]