



Tittel: K14: Digitalizing Receptionist Services

Emnekode	IS-304
Emnenavn	Bacheloroppgave i informasjonssystemer
Emneansvarlig:	Hallgeir Nilsen
Veileder	Devendra Bahadur Thapa
Oppdragsgiver:	Knowit Sør

Studenter:

Etternavn	Fornavn
Fredriksen	Sindre-Nicolai Barvik
Glosli	Henrik
Hurum	Jenny
Meisingset	Stian
Solvang	Rikke
Sørensen	Sven

Jeg/vi bekrefter at vi ikke siterer eller på annen måte bruker andres arbeider uten at dette er oppgitt, og at alle referanser er oppgitt i litteraturlisten.	JA	
Kan besvarelsen brukes til undervisningsformål?	JA	
Vi bekrefter at alle i gruppa har bidratt til besvarelsen.	JA	

BACHELOR THESIS IN INFORMATION SYSTEMS

K14: Digitalizing Receptionist Services

Creating the initial module of the K14 mobile application

GROUP 17

Fredriksen, S.
Glosli, H.
Hurum, J.
Meisingset, S.
Solvang, R.
Sørensen, S.

SUPERVISOR

Thapa, D., B.

Preface

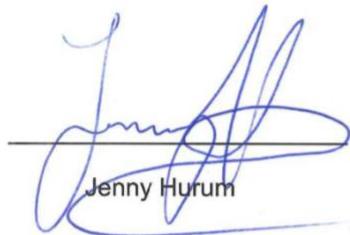
This report contains the project of group 17 in the course *IS-304, Bachelor Thesis in Information Systems*. This report intends to demonstrate the choices we have made and why and take advantage of the knowledge and skills acquired throughout the degree.

We are using relevant research to supplement our report and utilizing methods and previous experiences obtained in other courses to support our choices. This report will demonstrate the use of tools, methods, and technologies for project management and development.

We would like to express our gratitude to Knowlt Sør and the K14 representatives for their confidence in us and for allowing us to gain this experience. First and foremost, thank you to Trym Staurheim for aiding us throughout the process and putting up with our endless questions. Furthermore, we would like to thank our course coordinator, Hallgeir Nilsen, and supervisor, Professor Devendra Bahadur Thapa, for their guidance and assistance. Lastly, we would like to thank each group member for their determination, constant motivation, and outstanding collaboration throughout a challenging semester.

Kristiansand

16.05.22



Jenny Hurum



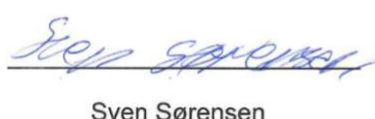
Stian Meisingset




Sindre Fredriksen



Henrik Glosli Gulbrandsen



Sven Sørensen



Rikke Solvang

Abstract

This report documents the process behind the development of a mobile application for iOS and Android. It is written for Knowit Sør, for which we have developed an application for tenants and guests of the future commercial building in Kristiansand, K14. The application seeks to streamline commonly understood and established services offered in a reception environment or by receptionists.

As the project contained few to no requirements, we have used a methodology that facilitates responding to change. The methodology applied is subject to The Agile Manifesto's four basic principles, combined with a selection of Scrum artifacts. We have achieved progress without compromising requirements through short iterations, definite sub-goals, and frequent Sprint Reviews. To ensure product quality, we mapped the end-user's needs by performing substantial data collection. In pursuance of a high-quality system, we have included a risk matrix, conventions and guidelines for code development and project execution, and regular testing.

In the analysis, we covered interviews, analysis of the client's website, and existing off-the-shelf solutions. Our findings resulted in mapping the domain. Thereafter, we applied methods such as sketching and mockups, and David Benyon's design principles, resulting in the current user interface design. The programming language React Native, facilitated an efficient development process, while Azure DevOps increased overall project control.

The team aimed to complete the highest prioritized system requirements. Challenges involving prioritizing functionality versus design and sub-par pull request reviews left us with a somewhat troubling development phase. Regardless of the challenges, the process resulted in a cross-platform compatible application, including functionality like viewing, registering, and updating events and appointments. Even though some requirements were incomplete, we are satisfied with the result and the learning outcomes we have attained.

A walkthrough of the application can be viewed [here](#).

Table of Contents

1	Introduction	7
1.1	<i>Presentation of the Team</i>	8
1.2	<i>Presentation of the Product Owner and Client</i>	8
1.3	<i>Presentation of the Project</i>	9
1.4	<i>Project Issue</i>	10
1.5	<i>Project Goals and Ambitions</i>	10
1.6	<i>Ethical Considerations</i>	10
2	Project Management	12
2.1	<i>Methodology</i>	12
2.2	<i>Definition of Quality</i>	13
2.3	<i>Process Quality</i>	14
2.4	<i>Product Quality</i>	15
3	Analysis	16
3.1	<i>Data Collection</i>	16
3.2	<i>System Requirements</i>	17
3.3	<i>Problem Domain</i>	19
3.4	<i>Application Domain</i>	20
4	Design	21
4.1	<i>Architectural Design</i>	21
4.2	<i>User Interface Design</i>	22
5	Technology and Tools	24
5.1	<i>Azure DevOps</i>	24
5.2	<i>Frontend</i>	25
5.3	<i>Backend</i>	27
6	Project Execution	29

6.1	<i>Project Start</i>	29
6.2	<i>Data Collection and Analysis</i>	32
6.3	<i>Ideation and Design</i>	35
6.4	<i>Development</i>	39
6.5	<i>Project close</i>	41
7	The Final Product	42
7.1	<i>Architecture and Backend</i>	42
7.2	<i>Functionality</i>	43
7.3	<i>Testing</i>	44
7.4	<i>Facilitating Further Development</i>	44
8	Reflection	46
8.1	<i>Process and Methods</i>	46
8.2	<i>Product Decisions</i>	51
9	Team Evaluation	54
10	Conclusion	55
	Bibliography	56
	Appendix	59

List of Tables

Table 1) Overview of the individual team members of the project	8
Table 2) Overview of associates and involved representatives in the project	9

List of Figures

Figure 1) Illustration of the chosen methodology	13
Figure 2) FACTOR model	18
Figure 3) Illustration of the class diagram created	20
Figure 4) The architectural design of the application	22
Figure 5) Revised Gantt Chart	30
Figure 6) Brainstorming session	31
Figure 7) Similarities in the sketches	36
Figure 8) The final sketch	37
Figure 9) Backend architecture	42

List of Appendices

Appendix 1: Potential Development Areas Identified	59
Appendix 2: The Learning Outcomes of The Bachelor Course	60
Appendix 3: Risk Matrix	61
Appendix 4: Rich Picture	61
Appendix 5: Event Table	62
Appendix 6: State chart diagram for Create New Event	62
Appendix 7: State chart diagram for Change Event	63
Appendix 8: State chart diagram for Event Invitation	64
Appendix 9: State chart diagram for Notification of Arrival	65
Appendix 10: UML Class Diagram	66
Appendix 11: David Benyon's 12 Design Principles	67
Appendix 12: Azure Burndown Sheet	68
Appendix 13: Team Contract	69
Appendix 14: Interview Guide	71
Appendix 15: Compiled Interview Guide	72
Appendix 16: Must-Have User Stories	74
Appendix 17: Color Palette for the Design	75
Appendix 18: Aaron Marcus' Color Conventions	75
Appendix 19: Landing Page	76
Appendix 20: K14 building page	76
Appendix 21: Calendar page	77
Appendix 22: Contact personnel page	77
Appendix 23: K14 information page	78

1 Introduction

This report is written as a part of the course IS-304, Bachelor Thesis in Information Systems. The bachelor thesis aims to provide students with an opportunity to demonstrate acquired knowledge and competence by engaging in a project involving information systems in a real setting (University of Agder, 2021). This process also includes planning, estimating, developing, testing, reviewing, improving, and thoroughly documenting the project. The course emphasizes quality management, both in the process and the product, by adopting applicable methods and techniques. Further, it accentuates the importance of iteratively evaluating the consequences and various outcomes of the process. Therefore, the project is of practical interest as it is desirable to measure our knowledge and abilities in a real setting.

The course requires collaboration with an external actor. We have partnered with Knowit Sør for a project concerning the development of an application, intended to aid tenants and visitors of the future commercial building, K14. This building will be a vibrant innovation center and a natural meeting point for the future business community in the entire region (K14, 2021). The project is scoped into a single module, hereby the digital reception of the application. Knowit Sør is the project owner, and representatives from K14 have formally functioned as the client.

While this chapter and its following sections will introduce the team, the parties involved in the process, and the thesis project, chapter two presents how the project has been managed. The methods used to conduct the analysis are then described in chapter three. Chapter four presents the architectural and interface design, followed by chapter five, which presents the technology and tools used. The execution of the project is then described in chapter six before presenting the final product in chapter seven, after which the results will be rigorously reflected upon in chapter eight. Thereafter, a statement from the team regarding the process is presented. Lastly, chapter ten outlines the conclusion and recognizes both limitations to the project and recommendations for further development.

1.1 Presentation of the Team

The team consists of six students in the sixth semester at the bachelor's program IT and Information Systems at the University of Agder and was established at the start of the fifth semester. As we believe a diverse team is crucial to creating ideas that can prosper into valuable solutions, our team embraces a wide range of individual interest areas, knowledge, and backgrounds (Table 1). This has resulted in a collectively broad foundation of knowledge.

Name	Main Responsibility	Other Responsibilities
Jenny	Project management, design, and documentation	Analytics, insight work, communication with product owner
Rikke	Design and frontend development	Analytics, insight work, database design, communication with the client
Sindre	Frontend development and project management	Analytics, database design, insight work
Henrik	Database design and documentation	Analytics, insight work
Sven	Full-stack development, testing	Analytics, insight work, transcribing
Stian	Backend development	Analytics, insight work, database design, testing

Table 1) Overview of the individual team members of the project

1.2 Presentation of the Product Owner and Client

Knowit AB is a Swedish IT consulting company established in 1990 that supports companies and organizations in digital transformation (Knowit, n.d.). Currently, the company is represented in Norway, Sweden, Finland, Denmark, Estonia, and Russia, with Norway being the second-largest expansion. Knowit AS in Norway is a wholly owned subsidiary of Knowit AB and has 15 subsidiaries, including Knowit Objectnet AS, Knowit Decision Oslo AS, and Knowit Reaktor AS.

Knowit Sør was originally our client, and the product owner was K14, represented by Daland Eiendom, BRG Utvikling, Næringsmegleren Sædberg & Hodne, and J. B Ugland Eiendom. However, due to the lack of communication with the representatives from K14, Knowit entered the role as product owner to give us continuous feedback about requirements. A table presenting an overview of associates and involved representatives related to the project and its client is presented below in Table 2.

Name	Professional Role	Association	Project Role
Morten Rosenberg	Marketing and sales manager	Knowit Sør	Project client supervisor
Trym Staurheim	IT-consultant	Knowit Sør	Project mentor and product owner
Geir Morten Hagen	Senior consultant	Knowit Sør	Architect mentor and product owner
Håvard Bjorå	Business and project developer	J. B. Ugland	Client
Unni Mesel	CEO	Næringsmegleren Sædberg & Hodne	Client

Table 2) Overview of associates and involved representatives in the project

1.3 Presentation of the Project

Before starting the project, Knowit Sør had contacted representatives from real estate agent Næringsmegleren Sædberg & Hodne and real estate developer J.B Ugland Eiendom regarding a potential collaboration related to K14. As this commercial building focuses on being contemporary and providing sustainable solutions to its tenants and visitors, they sought a mobile application that would digitalize and facilitate processes in the building. Such processes include booking meeting rooms, coworking areas, canteen arrangement, food ordering, access control, digital receptionist, and more. Please see Appendix 1 for all possible expansions and probable development areas that have been identified.

Our project has been scoped to focus on the initial module of the system, hereby the digital receptionist. It also includes the landing page and access to the application, which involves

registration and login. The main goal of the system is to provide a minimum viable product (MVP), fulfilling the following criteria:

- The application needs a layout that accommodates the various services and amenities available to tenants and their employees and guests registered in the app.
- The application must support both guests and regular user accounts.
- The application needs to support the fundamental functionality of average users, such as booking meetings, notifying arrival, viewing amenities and services, viewing building information, and contacting the receptionist.

This project will work as a foundation for the initial system, as it is planned to be further developed and expanded upon.

1.4 Project Issue

The issue that has been explored and attempted to solve, is the need for a mobile application that accommodates tenants and visitors. The application needs to include information about the facility and enable users to contact the receptionist. Both tenants and visitors need to register as users to benefit from the application; however, the application will provide varying content, depending on user type.

1.5 Project Goals and Ambitions

The overarching goal of the project is to fulfill the learning outcomes defined in the bachelor's course, as these learning outcomes align with both the product owner's and our individual goals. (Appendix 2). The individual goals involve using established methods and techniques together with the knowledge acquired through the degree to prove our ability to carry out an IT/ IS project. The product owner's goal is for the team to focus on the process rather than creating a completed product, enabling us to better understand and apply the acquired knowledge of the system development process into future projects.

1.6 Ethical Considerations

Various ethical considerations have been made as part of the development process. These were mainly regarding privacy and universal design principles. Early in the process, the product owner stated that it would be ideal for the application to track and recognize users, either via facial recognition or geo-tracking. This would enable the application to tailor content to each user. However, such functionality should be carefully considered before

being implemented, as both technologies revolve around gathering and processing large amounts of detailed personal data.

When creating an application intended for public use, regulations and rules regarding universal design need to be complied with. Even though most team members have completed a course in universal design and are familiar with its importance and core concepts, it has not been prioritized. The team has instead decided to focus on designing for user experience. Consequently, the argument can be made that multiple user groups are indeliberately excluded from using the application.

As this project is developed by a group of students still undergoing education, detailed knowledge about, and the handling of, privacy and universal design issues should not be expected. Had this been a project in a non-educational setting, it is possible that an off-the-shelf solution would have been used, possibly fitted to the specific client's needs. Therefore, it is likely that the considerations regarding privacy, design, and legal compliance would have been solved.

2 Project Management

This chapter presents project management in terms of methodology and tools to facilitate various aspects of the management process. Additionally, elaborating on quality assurance in the project. The chapter is divided into three subchapters to best demonstrate the various elements mentioned above. The first subchapter presents our definition of quality in a development project. The second chapter presents process quality, closely tied to product management and methodology, management tools, technical skills as well as test coverage. The third subchapter surrounds product quality, which emphasizes how product management, test coverage, continuous feedback and development, reviews, backlog prioritization, and acceptance criteria have contributed to the quality of the product.

2.1 Methodology

An agile methodology was chosen and adapted to the needs of the project and team to ensure project control and quality. Adopting an agile methodology was also a requirement from the product owner. The chosen methodology is based on The Agile Manifesto's four basic principles, combined with a selection of Scrum artifacts that have proven useful on previous occasions, as shown in Figure 1 (The Agile Manifesto, 2001).

The K14 representatives were unsure of how they wanted to advance with the project; therefore, we considered it appropriate to use an iterative method. The chosen development method contributes to more frequent deliveries to the customer as well as the opportunity to receive feedback in each iteration (Alshamrani & Bahattab, 2015, p. 108). In addition, an iterative approach allows for continuous adaptation of requirements and other project adjustments. Similar to other agile methodologies, Scrum manages most system development as a controlled black box, thereby managing the project as if it is well-defined, even if additional requirements are expected (Sutherland, 2021, p. 55). This fits well with the project as change and expansion throughout the development life cycle are expected. Scrum provides the opportunity to learn from each incremental delivery, resulting in reasonable revisions for the next sprint (Alshamrani & Bahattab, 2015, p. 109). Furthermore, we established that the framework Extreme Programming (XP) may become relevant if time becomes insufficient. Teams operating XP deliver software with fewer errors, faster, and more adaptability (Shrivastava, Jaggi, Katoch, Gupta, & Gupta, 2021, p. 10).

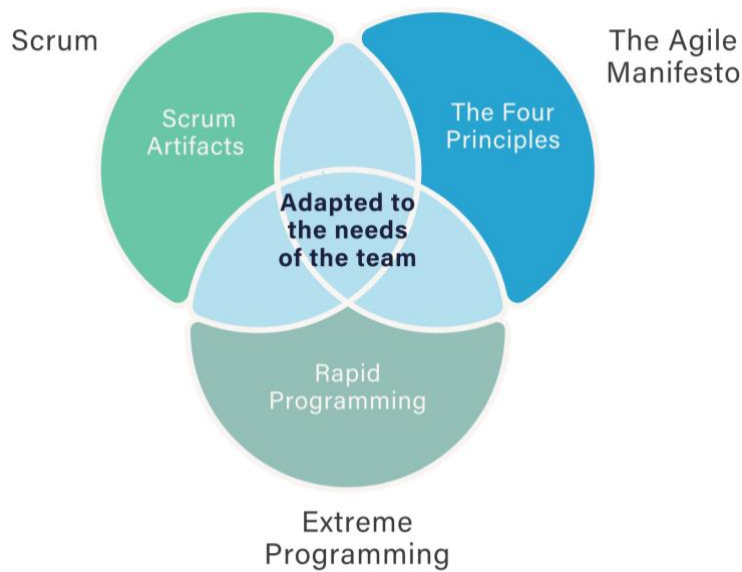


Figure 1) Illustration of the chosen methodology

Supported by the Agile Manifesto for software development, in combination with selected Scrum artifacts that substantiate principles from the Agile Manifesto, we can better adapt to continuous changes if necessary. As a consequence of utilizing Scrum in numerous prior development projects, we have found it appropriate to use selected Scrum artifacts that endorse project control and quality. The artifacts include Sprint Planning, Daily Standup, Internal Review with Code Review, Sprint Review, and Sprint Retrospective. Furthermore, we would like to emphasize the importance of Sprint Planning to minimize future challenges, as several challenges have been a direct or indirect consequence of poor planning based on previous experiences. An internal review will assist the team with a comprehensive overview of completed work and will result in a product increment if it is of usable condition and meets the team's requirements for Definition of Done (DoD). DoD is defined as an agreed set of acceptance criteria for something to be considered accomplished (ProductPlan, 2021). In our case, the acceptance criteria of each user story will signify completion or to which degree the DoD has been fulfilled. Towards the end of the sprints, a Sprint Review with necessary stakeholders and a Sprint Retrospective is held for further review and feedback, with the intent of preventing challenges and improving work for upcoming sprints.

2.2 Definition of Quality

It can be challenging to provide a concrete definition of what quality entails in a development project. Every project is different and consists of teams with diverse attributes, individually applying their skills to achieve a common project goal. The definition of quality may therefore vary reliant on the dependencies of the project. To ensure quality in software development,

we have deemed it appropriate to separate quality into the process- and product quality. We believe an essential aspect is distinguishing them, as high-quality products usually result from high process quality (Münch, Armbrust, Kowalczyk, & Soto, 2012, p. 7).

2.3 Process Quality

To ensure process quality, a combination of established methodologies, technologies, and frameworks needs to be present. However, we believe that there is no such thing as a universal methodology, rather that the methodology should be based on relevant factors and tailored to fit the team and project alike. By incorporating a methodology and artifacts that benefit the team's qualities and skills, we can exploit the benefits of the chosen methodology. Working with agile software development has allowed us to work iteratively and respond to changes and unforeseen problems accordingly. The artifacts selected can supplement the principles behind The Agile Manifesto such as: "Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage." (The Agile Manifesto, 2001). By utilizing Scrum elements such as daily standup and Sprint Review, we are able to communicate progress and problems continuously throughout the project lifecycle. This is especially useful as we were tasked with initiating the development of the application and had little information in the first place. Therefore, a methodology facilitating continuous adaptation to change, and feedback aids the team in ensuring quality in its entirety.

The chosen methodology has further allowed us to adjust sprint lengths iteratively. Initially, the sprints were three weeks long to accommodate the analysis and design phase of the project. When the development phase started, we realized that it would be advantageous to shorten the length of the sprints to two weeks. Consequently, we were able to focus on more frequent deliveries, with the intent of more periodic feedback from the product owner, ensuring the quality of the product. Another way to improve process quality is to do a risk assessment of the project. By establishing potential risks early in the process, the team can identify, minimize, or hopefully eliminate the risks, and help us optimize resources and plan around the potential risk. The risk matrix can be found in Appendix 3.

It is important to utilize tools and technologies that support management control to ensure process quality further. Azure DevOps allows the team to keep track of certain Scrum elements such as product- and Sprint Backlog, ensuring that everyone is up to date with current tasks and progress. Azure DevOps also allows for collaboration on code development, build, and deployment of the product (Microsoft, 2022). In addition to code

development, Azure offers Git-based version control, simplifying the process of reviewing and complying with changes in development. We can review code changes by creating pull requests, making it an important measure to maintain quality in the development process.

Time is often a critical factor in software development projects. It is important not to let insufficient time compromise the quality of the product; therefore, XP can be advantageous in such circumstances. Furthermore, it was deemed appropriate to incorporate pair programming into our development methodology, although it was not utilized in the traditional sense of two developers working on the same workstation. Instead, two developers had continuous communication working on different problem areas, using communication channels such as Discord to share and review code, diminishing the threshold for consecutive communication. Despite the traditional pair programming roles of “driver” and “navigator” not being present, the communication between developers promotes pair programming principles such as facilitating learning, efficiency, sharing, and improved resiliency to interruptions (Agile Alliance, 2022).

2.4 Product Quality

The Agile Manifesto expresses the importance of working software as a primary measure of progress, as well as delivering working software frequently. By having frequent reviews involving the product owner, we can make sure that our vision of the product corresponds. Considering the lack of distinct requirements leading to an unclear definition of how quality was perceived by the client, we concluded that product quality is defined through our data collection and analysis phase, in which the requirements are identified. Performing substantial data collection from the customers ensured that the product was developed to fit the needs of the end-user, consequently, making the product of higher quality. Further, it is desirable to let the customers be a part of several processes in the project life cycle, which is further substantiated by Mantri, Nandi, Kumar & Kumar (2011), stating that software quality is a solution to perform in accordance with the customer specifications. The customer is the end-user; therefore, the processes of software development should revolve around the approval of customer satisfaction.

3 Analysis

This chapter explains the analysis conducted to retrieve relevant information and introduces the methods that have contributed to the project. The upcoming subchapters will present an overarching designation of the methods and techniques; hereby, data collection, system requirements, problem domain, and application domain. Each subchapter consists of subsections elaborating on the methods or techniques relevant to the given subchapter.

3.1 Data Collection

To comply with the project issue, it was necessary to accumulate relevant data to establish a foundation and understanding. Data collection would also contribute to determining the target group, how the system will be used, and the system requirements. It would also be advantageous in identifying fundamental functionality. There are various ways of collecting data, such as surveys and interviews. For this project, both primary and secondary data collection were evaluated as necessary to create an adequate information basis. Primary data is collected by the researcher to provide answers to a clearly defined and current problem and collected directly from primary sources such as interviews. Secondary data is existing data that researchers use to enrich their research projects (Sundbye & Nisted, 2017); (Halvorsen, 2014, p. 114). The various ways in which the data have been collected are presented below:

- Semi-structured interviews (Primary data)
- Examination of K14's website (Secondary data)
- Assessing similar solutions (Secondary data)

3.1.1 Semi-Structured Interviews

A method enabling flexible and extensive information-gathering was required, as we were faced with sourcing most of the data for the project. Therefore, *semi-structured interviews* were chosen, as the method allows us to use prepared questions, enabling the team to collect reliable data, compare the findings, and achieve an in-depth understanding (Benyon, 2019, p. 152). Additionally, encouraging both the interviewee and the interviewer to pose follow-up questions and discuss or explore relevant topics should they arise, possibly leading to the disclosure of otherwise unexpected opinions or feelings. This improves the basis of information to be used when assembling the system requirements, reinforcing the foundation of a user-centered application design.

3.1.2 Examination of K14-Webiste

Besides collecting primary data, it is helpful to examine existing secondary data sources. The website of K14 includes information about the building, the initiators behind the project, a prospect, and a gallery. In other words, it displays their identity and vision. The prospect is comprehensive and provides information regarding K14's environmental and sustainable focus, modern design, and potential users. Examining the website helped determine the color palette for the design, the overall feel, the targeted audience, and the vision of the building.

3.1.3 Similar Existing Applications

In addition to conducting interviews and examining the K14 website, it was also interesting to investigate similar applications, both in the Norwegian and international markets. Looking into similar apps helped the team understand the complexity and scope of such an app and what it entails. The discovery proved highly relevant as it demonstrates how such apps are used in daily life. By comparing available apps, we can exploit existing solutions by using the entirety or parts of the apps and look for ways to accommodate unfulfilled requirements, possibly reducing the total cost of development. Additionally, probing these apps contributed to the ideation process by enabling the team to envisage the application from a user's perspective, thus facilitating the creation of the interview guide.

3.2 System Requirements

The following chapters will describe the methods used to substantiate system requirements, hereunder, an explanation of why we chose to include these methods in the project: user stories, *MoSCoW prioritization*, and a system definition utilizing the *FACTOR criteria*.

3.2.1 User Stories and Prioritization

A *user story* is defined as a description of functionality that will be valuable to either a user or purchaser of a system or software, which was the incentive for developing our user stories (Cohn, 2004, p. 4). We based them on the interview respondents and the client's requirements, as they are the future users of our system. The usage of user stories is beneficial to our project as it facilitates comprehensible communication of system requirements between stakeholders and developers. Further, it fits well with an agile development framework as changes in requirements are expected. The user stories' associated acceptance criteria provide a foundation for determining the completeness of a story, which further facilitates quality control in our project.

To determine the importance of each user story, prioritization is imperative. We chose the MoSCoW method for prioritization, which is a simple way to sort user stories into priority order (Waters, 2009). MoSCoW is an acronym, standing for Must have, Should have, Could have, and Want to have. We chose this method based on past experiences, knowledge about the method, and that it is, just as with user stories, comprehensible to stakeholders. In addition, we can represent a minimum viable product as must-haves, to recognize which user stories are required to be completed before we can launch the system. MoSCoW also fits an agile development framework, as we can easily change a user story's MoSCoW prioritization according to stakeholders' needs.

3.2.2 System Definition Using FACTOR

To provide a simple overview of the system to be developed, a system definition was created by utilizing the *FACTOR criteria*. FACTOR is used to describe the most fundamental decisions involved in creating a sound computerized solution (Mathiassen, Munk-Madsen, Nielsen, & Stage, 2018, p. 24). FACTOR supports us in developing the system definition, by providing criteria to ensure a satisfactory system definition. Figure 2 shows the process of fulfilling the criteria in the system definition.

F	FUNCTIONALITY Expand upon existing receptionist services: visitor- and event-management and information gathering and sharing.
A	APPLICATION DOMAIN View, update and register facility events, tenant information, facility information, appointments, local and adjacent amenities. Administrate appropriate participation and event participation.
C	CONDITIONS Developed based on customer requirements, product owner specifications and interview data from potential tenants by applying our collective domain and technical knowledge.
T	TECHNOLOGY Mobile, cross platform application using React Native for the iOS and Android platform.
O	OBJECTS Employee, Guest, Company, Event, Appointment, Amenity and Facility.
R	RESPONSIBILITY Connect guests, tenants and facility- and amenity-services.

Figure 2) FACTOR model

A system definition should be a concise description of the system expressed in natural language (Mathiassen, Munk-Madsen, Nielsen, & Stage, 2018, p. 24). Our system definition is as follows:

“The initial module for the system to be developed and later expanded upon is a mobile application for iOS and Android using React Native to ensure cross-platform compatibility. It seeks to streamline commonly understood and established services offered in a reception environment or by receptionists. Namely, users of the system will be able to view, register and update events, appointments, and a broad range of information regarding different aspects and entities of the facility and immediate vicinity. Hierarchical roles determine how, or if at all, a user might be permitted to interact with certain functions within the system. An employee can create, delete, and edit meetings or appointments, whereas guests are limited to viewing, attending, or declining said appointments or meetings.”

3.3 Problem Domain

A *problem domain analysis* is about figuring out what information the system manages (Mathiassen, Munk-Madsen, Nielsen, & Stage, 2018, p. 47). More specifically, the information contains the classes, objects, and behavior from reality and how they correlate with each other. The various methods that have contributed to the process of assessing the problem domain are *rich picture*, *event table*, and *class diagram*.

A *rich picture* is an informal drawing that presents the illustrator’s understanding of a situation.” (Mathiassen, Munk-Madsen, Nielsen, & Stage, 2018, p. 26). Based on previous experiences, mapping the actors and classes and how they are related early in the process, is a way of gaining an overview of the system and what it will manage. Therefore, a rich picture was helpful to gain a collective understanding of how the system would interact with each of its components and supports the foundation of an event table (Appendix 4).

We created an *event table* containing selected classes, and their related events, aiding us in determining how actors interact with the system (Appendix 5). An event table contains selected classes presented in the rich picture and related events (Mathiassen, Munk-Madsen, Nielsen, & Stage, 2018, p. 51). Note that this event table does not include classes and events of the whole system, but the most important in correlation to our scope.

The rich picture and event table resulted in a *class diagram*. The diagram contributed to an overview of the various classes in the problem domain, which facilitated the object-oriented system development (Figure 3).

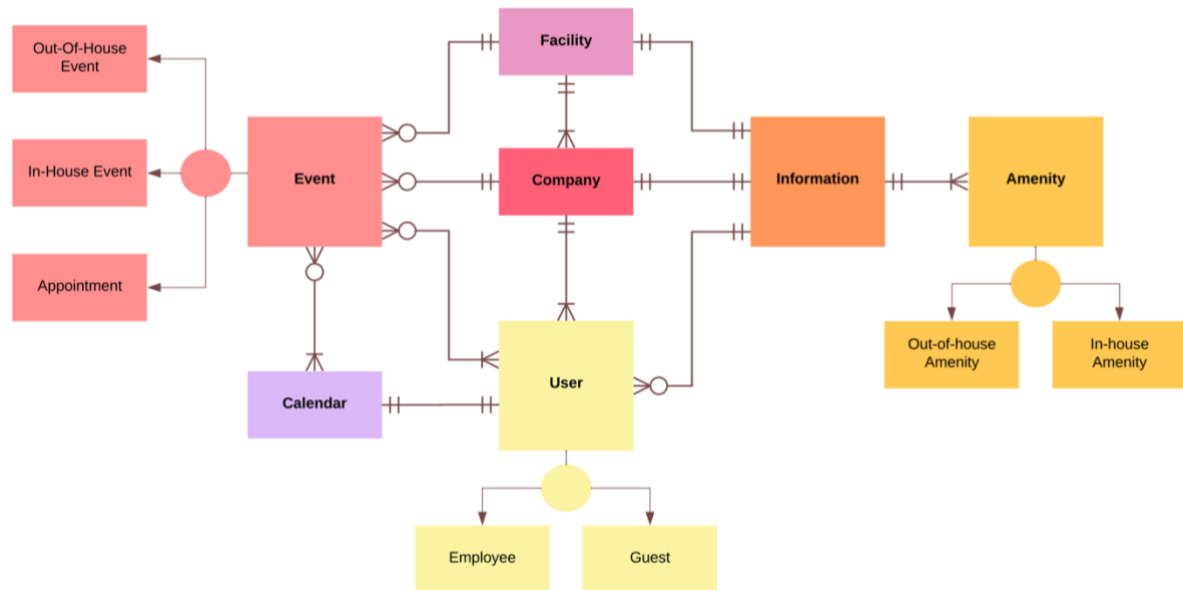


Figure 3) Illustration of the class diagram created

3.4 Application Domain

After exploring the problem domain, we analyzed *the application domain*. An application domain analysis aims to define requirements for functions and the interface (Mathiassen, Munk-Madsen, Nielsen, & Stage, 2018, p. 117). By analyzing the application domain, we can model requirements and how the system interacts with users, enabling us to get a holistic view of the system to be developed. The various methods that have contributed to assessing the application domain are *state chart diagrams*, *UML class diagrams*, and *navigation diagrams*.

To better our understanding and insight into how users interact with some parts of the system, we decided to use *state chart diagrams* as these diagrams define the different states of the interaction and the different ways the system or actor can change that state (Mathiassen, Munk-Madsen, Nielsen, & Stage, 2018, p. 129). Therefore, key events regarding the creation and general interaction with events were mapped through state charts, as we needed additional insight into the flow of this part of the system. By mapping event creation (Appendix 6), change event (Appendix 7), event invitation (Appendix 8), and notifying arrival (Appendix 9), we achieved a better understanding of the system flow and user interactions required to complete each task. Additionally, aiding us in modeling user

stories, as acceptance criteria, and state chart diagrams are comparable since they both define the steps required to complete a user story. However, the acceptance criteria have a technical approach, while the state charts focus on the user journey.

A *UML class diagram* allows modeling the application domain as a static structure (Berardi, Calvanese, & De Giecomo, 2005, p. 73). Therefore, a UML class diagram was created to represent the various classes and their respective functions from a system point of view, to visualize how the system will be working (Appendix 7). The diagram created was based on the event table and the class diagram in chapter 4.3.

To model the user interface, we created a *navigation diagram* that provides an overview of user-interface elements and their transitions (Mathiassen, Munk-Madsen, Nielsen, & Stage, 2018, p. 161). The diagram shows the navigation from an a-to-z perspective, which proved valuable in mapping the navigation flow regarding access and functionality. This established a solid foundation for designing and implementing the system, and can be viewed in its entirety through [this link](#).

4 Design

Simultaneous with the analysis, the team worked with designing the system architecture and user interface. Therefore, this chapter will address processes related to the architectural design, namely what was considered when determining which architecture to use and how this impacts the project. The chapter also addresses processes related to user interface design, such as *sketching*, *design principles*' contribution to the design, and the *mockups* created.

4.1 Architectural Design

Establishing a software architecture is crucial when developing any software system, as it acts as the foundation and provides mechanisms for reasoning about core software quality requirements (Venters, et al., 2018, p. 174). It was necessary to establish data flow and component interaction in the initial design phase. We considered both a monolithic approach, in which all functionality is encapsulated into one single application, and a micro services-oriented approach, in which a single application is developed as a suite of small services

(Ponce, Márquez, & Astudillo, 2019, pp. 1-2). A *micro-service-oriented approach* was considered most suitable due to: limited development time, the customer's desire for further development, and the architectural requirements derived from the analysis, such as scalability and maintainability. Microservices offer benefits appropriate to our project through modularization, maintainability, transferability, and increased availability due to the independence of services. These benefits are additionally enhanced by using the controller-service-repository design pattern. This pattern clearly separates class responsibilities through the enforcement of object-oriented principles, such as high cohesion and low coupling. In combination, this allows us to scope the project into one manageable service, which can be incorporated with additional services at later stages (Figure 4).

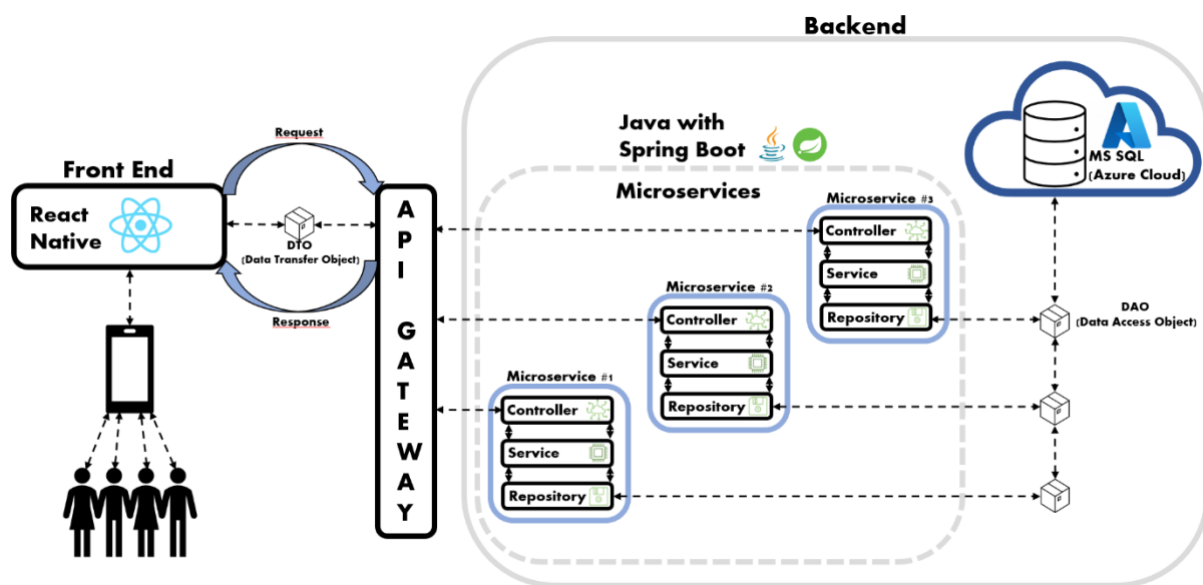


Figure 4) The architectural design of the application

4.2 User Interface Design

User interface design involves anticipating user requirements and ensuring the interface elements are understandable and easy to access (Usability.gov, n.d.). Therefore, the process involves *ideation*, *sketching*, and the evaluation of design and composition principles before arriving at the first draft *mockups*.

4.2.1 Sketching

The team decided to ideate and sketch shortly after meeting the client in the initial phase. *Sketching* is a simple way of exposing abstract ideas to the real world by demonstrating and visualizing simple functionality and was therefore crucial to the design process (Benyon, 2019, p. 184). Due to the method's simplicity, it is easy to distribute the products among the

team members, contributing to further brainstorming. Additionally, the process of creating an interview guide became easier as each member conceptualized and visualized the application, leading to more deliberate questions. The design team used elements from each sketch to create a final sketch as the blueprint for the mockup.

4.2.2 Design Principles

Designing from a human-centered perspective involves numerous aspects. *Design principles* can guide the designer through the process and contribute to complying with the user's needs (Benyon, 2019, p. 116). There are numerous design principles, and David Benyon proposes twelve design principles which can be seen in Appendix 8. All the principles interact in various ways, and it is challenging to design a solution accommodating each principle. Nonetheless, the principles will contribute to key features and elements of a solid design with a higher degree of usability. In this project, seven of Benyon's twelve principles have been used actively in designing the mockup:

- Consistency
- Familiarity
- Affordance
- Navigation
- Feedback
- Style
- Conviviality

4.2.3 Mockups

Mockups were chosen as the second design method to further visualize the ideas proposed through the sketches. This technique is typically a mid-to-high fidelity visualization of a product's appearance and displays basic functionality (Cao, Ellis, & Khachatryan, 2016, p. 9). A mockup distinguishes itself from sketches and techniques such as wireframes, as it adds visual richness through colors, typography, and iconography. It was necessary to adopt this technique to create a visual representation close to the final product. Having a realistic mockup provides the stakeholders with an accurate representation of the final product, enabling meaningful feedback and understanding of the customer's needs.

Additionally, the creators of the mockups can easily showcase the design to the team, which can be advantageous for reviewing the design from a bigger perspective and communicating the design with the developers. The decision was made to make the mockups clickable to better illustrate the intended navigation to each page. By implementing this, the client would gain a better understanding of the intended functionality and navigation, as well as aid the team during the implementation of the application.

5 Technology and Tools

In this chapter, we will detail the system's tools, programming languages, and frameworks that contributed to developing the application and define why they were used.

5.1 Azure DevOps

The product owner required us to utilize Azure DevOps as a project management tool. DevOps makes it easy to keep track of product and Sprint Backlog by allowing group members to place tasks under new, active, resolved, or completed, relative to their progress. By using user stories and task cards, group members were able to write acceptance criteria and descriptions and attach relevant documentation to respective tasks. The use of Azure DevOps increased project control as it provided the group with an overview of remaining work, progress, and both assigned and unassigned tasks.

Two functions of Azure DevOps consistently used throughout this project were the Sprint Retrospective plugin and burndown charts (Appendix 9). The retrospective plugin enabled the group to effectively assess and document the successes and downfalls of each sprint, thereby facilitating bringing changes and improvements into future sprints. Azure Burndown Charts proved to be an effective way of presenting the amount of completed and remaining work at the end of each sprint and indicating the progress throughout each sprint.

In addition to facilitating certain Scrum artifacts, Azure DevOps offers the Git version control system. Version control is the practice of managing and tracking changes to software code over time, aiding development teams in working faster and smarter (Atlassian, n.d.). Using the version control offered by Azure, we could link code to their respective task, thereby increasing the group's control over completed features. To ensure the quality of the code being pushed into production, both a main and development branch were used. The development branch would be used for pushing code throughout sprints, whereas the main branch would be used to push all produced code at the end of each sprint. By doing this, we ensure that the main branch holds only quality, working code, ready for production. Therefore, the use of version control is one of the most important measures taken to maintain quality-assured code throughout the project.

5.2 Frontend

This chapter introduces the tools and technologies applied in the development of the frontend, namely React Native, TypeScript, Expo, Nativebase, OpenAPI, ESLint, and Prettier, briefly explaining why each technology or tool was used and how it impacted the project.

5.2.1 Language and Framework

React Native is a JavaScript framework for designing and creating mobile platform user interfaces and was required by Knowit (O'Reilly, n.d.). React Native streamlines the project development experience by removing the need to learn and write separate languages for the iOS and Android platforms. Other than this, React Native offers the ability to maximize code re-usage through reusable components, thereby enforcing the principle of localizing change. Additionally, as it is a well-established framework, multiple widely used libraries are available, enabling us to take advantage of pre-tested components, thereby increasing the overall quality. Apart from providing us with more time to develop, the application of only one multi-platform language simplifies the process as a whole and substantiates further development.

TypeScript is a strict syntactical language that builds on JavaScript (TypeScript, n.d.). Through strict typing, type annotations, and type inference, TypeScript enhances the quality of the product by enabling us to write more readable, clean, and intuitive code. Said attributes are attained through strictly defining what a variable can hold, making it easier to interpret where individual variables are applied and what value it contains. By increasing data flow clarity through type transparency, Typescript provides better quality and control, ensuring that components and functions receive and pass on the correct data. This makes it easier for future developers to read and understand the code and save time with live exposure of bugs and errors.

Expo is a framework for React Native providing a wide array of tools to get started and maintain the software during development (Expo, n.d.). It aids with and simplifies cross-platform testing through its command-line interface, standardized setup, and ensuring external library compatibility. More specifically, it raises the quality and control of our project by applying automated setup and configuration, deployment, and testing on different platforms through its integrated development server, and automated verification of dependency compatibility.

5.2.2 Tooling

NativeBase is a component library used with React Native to develop UI across iOS, Android, and Web applications (NativeBase, n.d.). The library eases the process of implementing design elements, enabling us to save time and increase quality by using pre-tested, ready-made components. Additionally, these components ensure control through a uniform design by supporting the creation and usage of default properties across pages. The application of a component library reduces the amount of code needed to establish a consistent UI across platforms and operating systems. Using NativeBase also removes the need for learning and writing copious amounts of CSS, possibly saving time, and allowing us to reallocate resources elsewhere.

OpenAPI is the de-facto open-source format used to describe and document API design (SmartBear, n.d.). Combined with its TypeScript generator, OpenAPI provides a tool to automatically generate and maintain resources necessary to communicate with the API. Consequently, this removes the need for manual implementation of API calls, testing of models, and general API configuration. Further, it increases overall product quality and control through automation, thereby reducing the potential for human errors and enabling us to focus on implementing software design and functionality.

ESLint performs continuous static code analysis, flagging errors, bugs, and suspicious constructs (ESLint, n.d.). Upholding a consistent programming style and maintaining code conventions demands less effort when combined with a code formatter like Prettier, which utilizes established configurations and rulesets. These tools enhance the overall readability, ensuring a uniform codebase, and reinforcing both quality and the foundation for further development.

5.2.3 Testing

Testing the frontend ensures that functionality still works as expected after changes to the code, also called regression testing. For this purpose, the tools Selenium and Cucumber have been used (Leung & White, 1989, p. 60).

Selenium is a tool that automates the web browser and can be used for various purposes; however, we used it to automate acceptance tests that communicate with our user interface (Selenium, n.d.). Together with Selenium, the tool Cucumber has been used, which is a tool that supports Behavior Driven Development (BDD). BDD is the process of developing in a

natural language, such as English or Norwegian (SmartBear, n.d.). This has been used as a way of creating automated acceptance tests written in natural language and ensures a common understanding between all parties involved in the project (Furia & Nanz, 2012, p. 279).

Using Cucumber did not only benefit the stakeholders of the project. It also made tests easier to read and understand, as reading extensive test cases written in plain code can be challenging to comprehend. Further, it made writing tests easier, as parts written in natural language could be reused to fit other test cases. Also, it potentially decreased the amount of boilerplate code and the time spent writing tests, giving more room for writing functionality. Additionally, Cucumber enables a 1:1 relationship between a test and a user story's acceptance criteria.

5.3 Backend

This chapter presents the technologies used in backend development, including the main programming language and development framework, different tools used to provide functionality, database and testing frameworks.

5.3.1 Language and Framework

The programming language used for backend development was Java with the Spring Boot web framework. Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can “just run” and is used by 4 in 10 Java developers (Spring, 2022; Maple & Binstock, 2021). Using Spring Boot was a requirement from Knowit, which suited us well considering that several team members have experience with this from previous projects.

For building the project we used Apache Maven which is a software project management and comprehension tool (Porter, van Zyl, & Lamy, n.d.). Maven is also responsible for handling our dependencies to external tools and libraries.

5.3.2 Tooling

We have used several external libraries and tools for developing the backend, such as:

- *JavaMail*, a framework to build mail and messaging applications (Oracle, n.d). This allowed us to send emails to users for authenticating email addresses, resetting passwords and communicating with receptionists.
- *Spring Data JPA* (JPA), Spring's implementation of Java Persistence API which is "a specification that defines an API for object-relational mappings and for managing persistent objects." (Janssen et al., 2021). JPA simplifies querying the database and offers auto generation of Java entities mirroring the database tables. The usage of JPA has enhanced data integrity across the java application and the database, in addition to securing high quality code as the auto generated classes follow a predefined format. In combination this leads to more efficient development, and higher code quality.
- *Lombok*, a java annotation library for reducing boilerplate-code (Project Lombok, n.d). Lombok streamlines the code implementation by allowing annotations to auto-generate code for often used methods, which relieves developers from writing time-consuming boilerplate-code.
- *JWT*, an open, industry standard method for representing claims securely between two parties (auth0, n.d). Used for secure authorization of requests to the API. JWT is an encrypted token based on user credentials. By using this as authorization instead of username and password, it prevents ill-intentioned individuals from fetching this data. This is an important measure to maintain privacy and enforce confidentiality for users of the application.
- *Swagger*, an API-tool created to foster API collaboration and standardization across multiple teams, in our case frontend and backend (SmartBear Software, n.d). Swagger provides API-documentation to applications utilizing the API, enabling standardized format of requests/responses between the API and the frontend.

5.3.3 Database

We have used the Microsoft SQL Server (MSSQL) database by request from Knowit, although, while developing, we used the H2 Database Engine (H2), which is an open-source Java Database Connectivity API (H2 database engine, n.d.). We configured H2 to use the same syntax as MSSQL, relieving us from rewriting the database schema. H2 offers an in-memory database which relieves developers from running a separate instance of the

database while developing. It also allows integration tests to be run in the build pipeline without connecting to an external testing database. Since the database is rebuilt from a shared script each time the application is run, we established a cohesive development environment, which removed the possibility of unsynchronized databases. A visualization of the ER-diagram can be found [here](#).

5.3.4 Testing

For testing, we have used JUnit, a simple framework to write repeatable tests (JUnit, 2021). Knowit required a test coverage of at least 80% in the backend, and to ensure this, we used JaCoCo, a free Java code coverage library (EclEmma, 2022). JaCoCo delivers code coverage reports when tests are run, enabling us to see line-by-line which parts of the code had test coverage, additionally illustrating the test coverage percentage.

6 Project Execution

This chapter presents the execution of the project chronologically through five subchapters, hereby: project start, data collection and analysis, ideation and design, development, and project close. The chapter will give an in-depth review of the process, which involves reasoning for important decisions made.

6.1 Project Start

This chapter presents the starting point of the project, which means the first day of the team meeting and working together to start the first sprint. The chapter will first present the initial phase, a phase initiated prior to meeting Knowit, and the project issue. Further, the chapter presents the project kickoff and meeting the product owners.

6.1.1 Preliminary Phase

It was necessary to initiate the starting phase to establish provisions, clarify expectations, and create a team contract and an early project plan. Among the provisions were:

- Daily documentation of conducted work
- Adopting an agile framework with Scrum artifacts
- Conducting daily standup digitally or physically, without exceptions

Additionally, the provisions emphasized equal workload distribution defining collective goals, ambitions, and guidelines to ensure a project of desired quality. Further, it was necessary to clarify each individual's project expectations to establish a collective understanding and agreement to avoid potential problems, which served as the basis for the contract. Making a contract established an agreement between the team members and sets out the rights and obligations that must be complied with, which assisted in creating a productive workflow. The contract aimed to be realistic and specific by assessing work distribution, conflict management, communication platforms, deadlines, and decision making. The contract was created and agreed upon by each team member (Appendix 10).

An early draft of the project plan was created and later replaced with a Gantt Chart before starting the first sprint (Figure 5). Despite creating the project plan draft prior to starting the project, there were a few discrepancies in the Gantt Chart. Proper planning provided an overview of the project, reducing biased decisions and emphasizing central elements such as the project's scope.

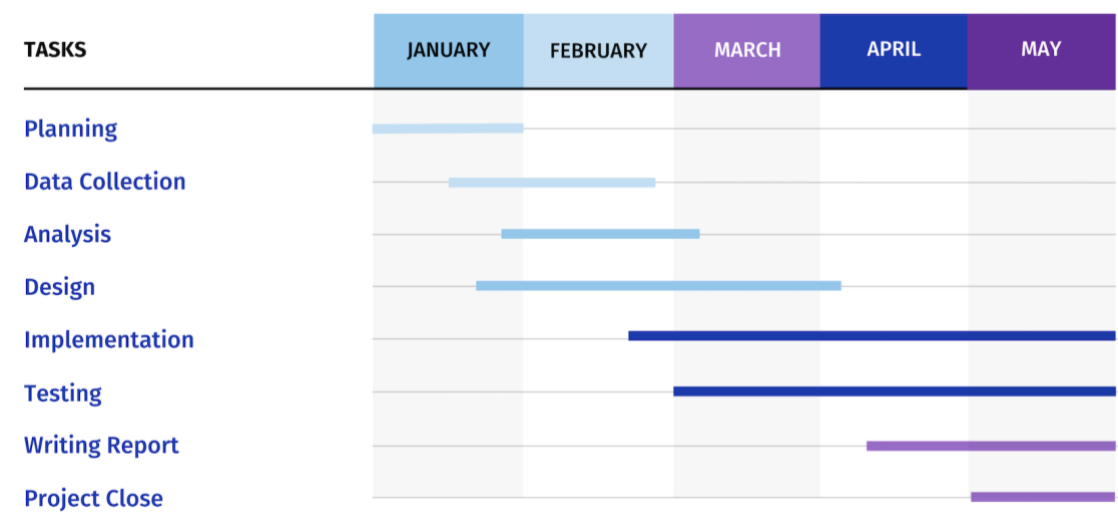


Figure 5) Revised Gantt Chart

6.1.2 Project Kickoff

The project kickoff was set to January 7th. As we met the team from Knowit, they presented us with various cases for our bachelor project. K14 seemed like the most appealing and suitable case, as the team wanted a challenge that required each member's knowledge and abilities to be put to the test. Thereafter, the representatives from Knowit scheduled a meeting with representatives of K14 and prepared us by giving input on how to ask questions and keep the conversation flowing to retain relevant information.

Within a week, the meeting with the client took place. The team had prepared questions and talked through the meeting with representatives from Knowit, encouraging us to act as IT consultants. Despite the client having a vision of the mobile application, there were no predefined requirements. Therefore, it was crucial to engage in the conversation and ask questions to ideate with the client to uncover vital information. However, the team had little experience with such situations, and one of the representatives of Knowit helped guide the meeting and retrieve information. The meeting overwhelmed the team with information and ambitious ideas, resulting in few clear lines. Through the meeting, two of the team members noted down key takeaways. They can be described as follows: The client's vision of the K14 application is to be a one-stop-shop, meaning gathering all services, everyday tasks, and amenities that visitors and employees need in one app. While closing off the meeting, another was scheduled one week afterward for the team to present current understanding regarding the client's vision and needs to determine the scope.

After processing the meeting with the client, it was necessary to review the information to map and define the various categories and elements in the application and their relation to one another. Besides reviewing the information, it was necessary to investigate and brainstorm. The team came across a similar application in the Norwegian market by IZY.AS. This application presented similar functionality to what the client described but appeared to lack the necessary functionality and a user-friendly design. This process was carried out in plenary with the help of a chalkboard, as shown in Figure 6 below.

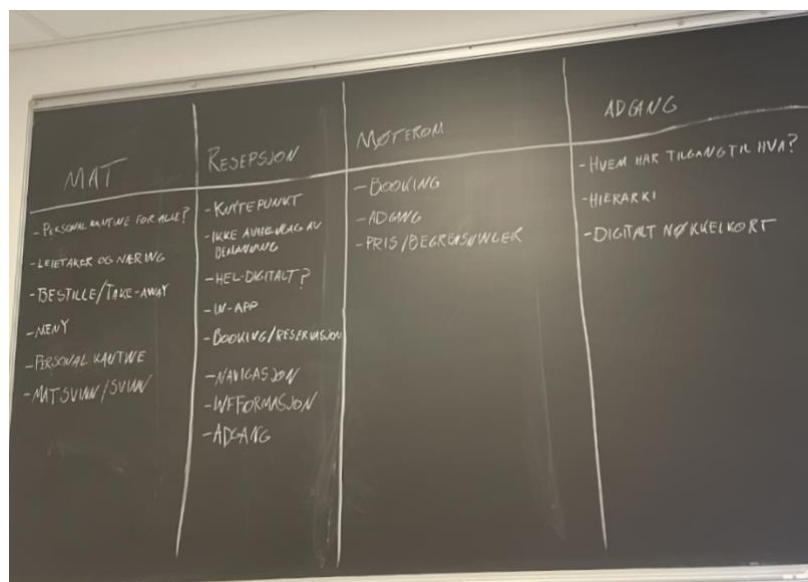


Figure 6) Brainstorming session

The activities mentioned above resulted in a list of questions for the client in the upcoming meeting. It was also decided to present the app IZY.AS, the functionality offered by the app,

and its shortcomings, such as a digital receptionist. This was with the intent of getting feedback regarding application functionality and design.

6.1.3 Determining the Scope

The second meeting proved to be positive in determining the project scope. Initially, the client proposed ideas such as the app receptionist recognizing app users through GPS tracking and verifying access to restricted areas belonging to the various businesses through biometrics. However, such functionality requires extensive knowledge related to both legislation such as GDPR, and complex technology development. As the team does not possess such knowledge, it was not an alternative. The client also expressed a wish for the application to enable food ordering, a carpool and bike rental, functionality for parking, and more (Figure 6). Moreover, the client proposed numerous criteria to be fulfilled in order to produce an MVP, as proposed in chapter 1.3.

Due to the project's time frame of four months, it was critical for the product owner, the client, and the team to establish a realistic scope. It would not be possible to create an app meeting all the visions of the client in four months. Therefore, the final product had to be made in a way that facilitates further development. Thus, the scope was set to create the app's foundation, including working functionality of what was described as the digital receptionist of the app. By defining the scope, it was possible to initiate the next phase of the project: data collection and analysis.

6.2 Data Collection and Analysis

This chapter presents the process related to the interviews conducted, involving the preparation, execution, and analysis of the findings.

6.2.1 Preparing the Interview

An important step in conducting a thorough interview is to prepare the interview accordingly, and therefore it was necessary to create an interview guide (Appendix 11). The interview guide was thoroughly assessed by critically evaluating each proposed question to ensure its purpose. Additionally, the team role-played as an informant and interviewer to test whether the interview guide provided sensible answers that would contribute to our analysis. Further on, the interview format was decided to be respondent-to-interviewer rather than a group interview, as group interviews require an additional effort of the informants regarding

planning and scheduling. The interview involved three members of the team functioning as interviewer, observer, and transcriber. In selecting interview objects, the client offered to contact the various companies on behalf of the team. The client presented the situation and asked whether any potential tenants would like to be interviewed. Four companies agreed, and each company was contacted to determine formalities. By interviewing on the informants' premises, comfort is ensured, and disturbances are avoided.

6.2.2 Execution of the Interviews

The interviews were conducted at different locations according to the interviewee's premise. When conducting the interview, the interviewer provided background information regarding the project and the purpose of the interview. Additionally, the interviewer provided information about the interviewees' rights and privacy and assigned each with an ID. The ID would enable the team to track the respondents' answers and remove them if desired.

Several informants expressed difficulty in understanding various questions, which could be due to the complex concept of the project. In such circumstances, the interviewer would elaborate and explain the question differently with examples to clarify the question. However, the general impression indicated that the interviews were comprehensible and served their purpose. After conducting the interview, the respondent was thanked for their participation. Thereafter, the notes from each interview were rewritten accordingly and later compiled into a larger document using the layout of the interview guide (Appendix 12).

In addition to interviewing future tenants, we were also able to interview a receptionist working in a commercial building, similar to K14. The interview was conducted in the same manner as presented above. By interviewing tenants and a receptionist, we were able to get a more nuanced understanding.

6.2.3 Analyzing the Findings

The compiled document was thereafter refined to only contain information relevant to our scope, to categorize and compare answers. Multiple answers could not be directly compared, as semi-structured interviews allow the interviewees and interviewer to explore the conversation beyond the predefined questions. However, some answers proved to have similarities to other interviewees' opinions. Additionally, recurring themes and focus areas provided valuable insight into the users' needs.

6.2.4 Examination of K14's Website

The website of K14 was examined alongside conducting the interviews. Each team member thoroughly examined the website and its pages to retrieve relevant information to support our understanding. The website contained a prospect that proved to be highly relevant as it contained information about the concept of the building, in addition to presenting its target audience. Having a predefined target audience as a basis facilitated the ideation and design process. After examining the website, the team discussed the various elements and impressions in plenary, to collectively agree upon an understanding of K14 and its identity.

6.2.5 Constructing and Prioritizing the User Stories

By compiling data from the interviews, the K14 website analysis, the client's vision, and the product owner's specifications, we initiated the process of user story creation. By utilizing the format: As a "who", I want to "what", so that "why", we ensured that the user stories would be comprehensible and easy to communicate between stakeholders regardless of technical insight. Initially, the team discussed the data from the analysis to further our insights and establish a common understanding of user needs and requirements. Afterward, we worked through the data and requirements in unison, point by point, translating each requirement or user need into a user story. Additionally, every story received a set of acceptance criteria to be used in testing, detailing its conditions to be considered complete. To increase the quality and accuracy of the user stories, each member read through the user stories noting possible improvements or changes that we subsequently discussed and applied where necessary.

The prioritization started with identifying user stories relevant to the project's scope, by using MoSCoW. Using the navigation diagram in combination with the user stories' contents, we identified user story dependencies and pinpointed must-haves (Appendix 13). This resulted in a prioritized overview of the user stories and any technical or functional dependencies necessary to complete the MVP.

6.2.6 Exploring the Domain

Exploring the problem domain was done alongside constructing the user stories, starting with creating a rich picture. The relation between actors and objects was modeled to get a holistic overview of the problem domain. Thereafter, we analyzed which events would occur in the problem domain, visualizing it with an event table. Lastly, after gaining proper domain knowledge, a class diagram was developed to visualize classes and their relationship. We used specialization in some classes to illustrate the different inherited classes.

Following the problem domain analysis, we started creating state chart diagrams to explore the application domain. The problem domain events inspired the events in the diagrams. The diagrams were developed in tandem with defining the acceptance criteria for the user stories, considering the diagrams contribute to visualizing the steps involved in a user story. Furthermore, a UML class diagram was created, which contained all the classes the system needed to consider from a programmatic perspective. Each class contained fields specifying the data types, and class methods specifying parameters and return values. Some classes were aggregated into superclasses due to numerous common attributes. This also applied to the classes Employee and Guest, which were aggregated into User. However, our implemented system did not utilize this aggregation as it was impractical from a database perspective.

Lastly, a navigation diagram was created, supplementing the later developed mockups and creating a solid foundation for implementing the design, more specifically, implementation of page navigation and the authorizations required to access certain pages. The diagram also displays the number of clicks each page has from the index page, which was beneficial to keep track of a page's position in the hierarchy and keep the number of clicks to a minimum.

6.3 Ideation and Design

The ideation and design process was executed in parallel with the analysis, sketches being the first method. While half of the team was conducting the interviews, the design team made mockups for parts of the system that would remain unaffected by the data collection, such as the login and register page. The creation of the mockups continued concurrently with the analysis of the findings to incorporate elements in the compiled document. This resulted in the mockups being based on the must-haves. This subchapter first presents the process of creating the initial and final sketches, followed by a section elaborating on creating the mockups and design choices.

6.3.1 Sketch

Similar to general design practice, it was considered appropriate to outline different ideas, as it acts as a catalyst for the ideation process. Each team member outlined individual drafts based on our understanding of what the application would encompass. The purpose of individually visualizing the ideas was to optimize new ideas without compromising originality. This resulted in some of the contributors sketching for different pages than the rest of the

team. However, this provided new ideas, which later proved helpful in the design process of the final sketch. Then, each sketch was presented to the team while explaining the design and interaction between the pages. Thereafter, the sketches were discussed in plenary to examine parallels, similarities, and unique elements, before deciding which elements to include in the final sketch.

Despite a diverse selection of ideas, there were several similarities in the register, login, and landing page. In each sketch, the register and login page had a minimalistic design with generous space between its fields and buttons, leaving little room for excess text and distractions. The sketches of the landing page all resembled a dashboard in one way or another, with elements, buttons, and widgets presenting what the team perceived as key functionality at the time. Each sketch displayed functionality such as notifying arrival to a meeting, notifications, and an overview of upcoming events in the building. Additionally, most of the sketches presented a toolbar at the bottom with buttons for the profile page, calendar, services and amenities, and booking. Some of the similarities are highlighted below in Figure 7.

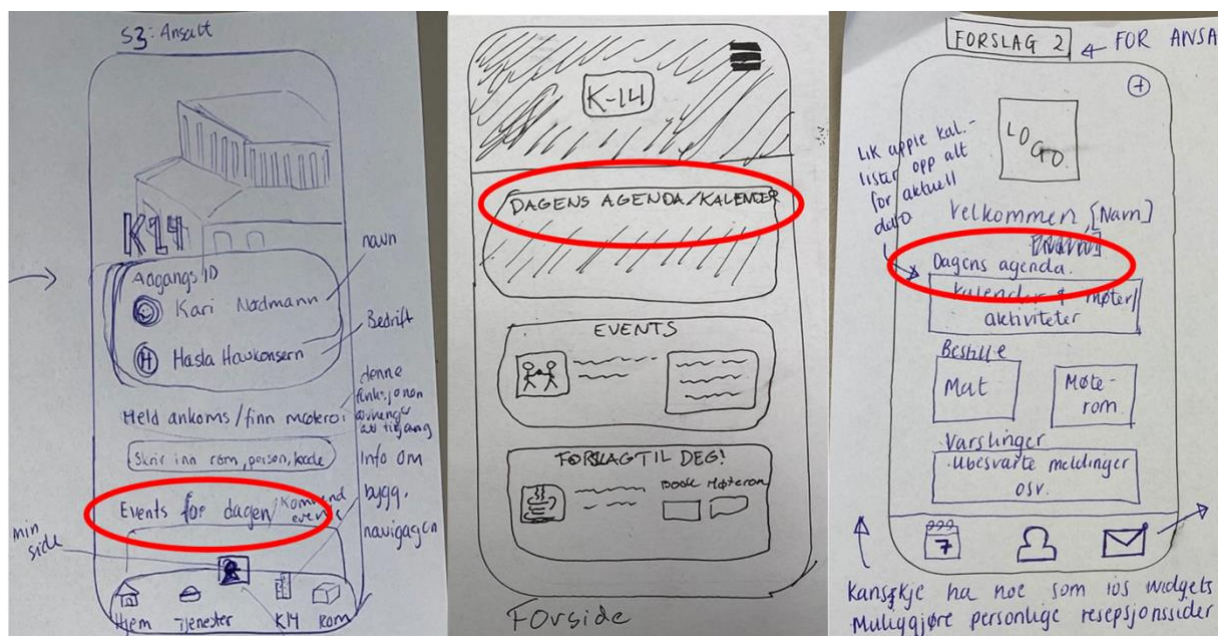


Figure 7) Similarities in the sketches

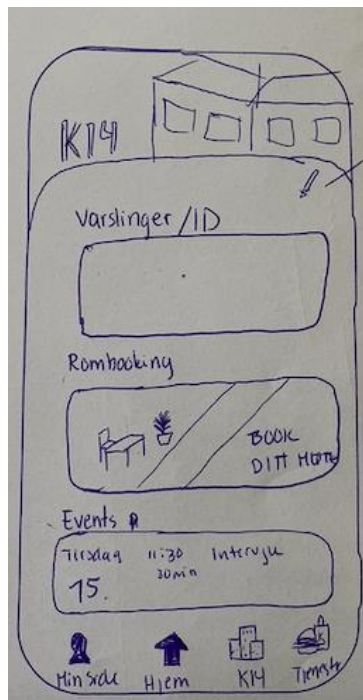


Figure 8) The final sketch

Based on the proposed ideas, the team collectively decided on the main elements to be included in the final sketch. The design team then created a final sketch, compiling the various elements from each sketch and new elements that came to mind during the design process. This was done to create a thorough foundation that would cover the necessary pages. As sketching is a minimalistic method that does not include details, the design team sketched pages for: register and login, the landing page, available services, accepting meeting invitations, and information about the building. As Figure 8 shows, the final sketch included various elements that occurred in the majority of the initial sketches. However, the final sketch also includes a digital ID card. Despite the project's scope being enclosed to one module, emphasizing the digital receptionist, this choice was made as the client expressed a desire to incorporate access into the application seamlessly. This final sketch functioned as the foundation for the next step of the design process: creating mockups.

6.3.2 Mockups

Based on the final sketches and previous experiences with UI/UX design, the mockup was constructed through the product design platform UXPin. The design team was responsible for carrying out the design of the mockup. Despite having a final sketch with the desired design, it was convenient for both designers to create their own take of the final sketch. This is due to sketches being simplistic and merely a draft, leaving room for the design of the final sketch to be explored further. Initially we deemed it necessary to create mockups for key pages within our scope, such as registration, login, the landing page, the profile page, and the page displaying information about the building. However, as the project went by, it became clear that it was necessary to create additional mockups to cover all the must-have user stories. This included pages that enabled the users to actually contact the receptionist, make events and meetings, view the calendar, and view the amenities. Through the design process of the mockups, a set of design principles and precautions were adopted, in addition to a suitable design language.

As mentioned in chapter 4.2.1, seven of Benyon's twelve principles were taken into account when designing the mockup, as shown below. To illustrate each principle, we will refer to the page number in the site map in UXpin, through [this link](#).

- *Consistency (Principle 2)* - concerns consistent use of language and features, and consistent design with similar applications. An example of this is the consistent use of a specific font throughout the design, which will be further elaborated upon, in regard to design language (UXpin page 3.0)
- *Familiarity (Principle 3)* - concerns using language and symbols familiar to the intended audience. An example of this is the icon indicating the profile page (UXpin page 3.0).
- *Affordance (Principle 4)* - involves designing an element in a way that clarifies its purpose, establishing a correspondence between the visual design and its intended properties. An example of this is the buttons to create an event in the calendar by using a consistent shape with shadowing, a descriptive text, together with a logical placement and order at the bottom of the page (UXpin page 11.0).
- *Navigation (Principle 5)* - involves providing feedback that enables users to move around in the system through maps, directional signs, and information. An example of this is the possibility to always use the return-button which is located at the top left in every page except for the landing page (UXpin page 6.0).
- *Feedback (Principle 7)* - concerns rapid feedback from the system to the user to display the effect of their actions. An example of this is the pop-up displayed when a user creates an event, with a descriptive text and a figure symbolizing a check mark (UXpin page 11.2).
- *Style (Principle 11)* - involves a design to be stylish and attractive. This principle is elaborated thoroughly through the latter part of this subchapter regarding design language.
- *Conviviality (Principle 12)* - concerns interactive systems to be polite, friendly and generally pleasant. An example of this is when a person registers a user account, where the last step of this process presents a page with a polite and welcoming message (UXpin page 2.0).

Further, we opted for a suitable design language for the mockups to ensure transparency and help users understand what is going on. A design language consists of the following elements, as expressed by Benyon (2019, p. 227):

- A set of design elements such as the use of color, styles, and types of buttons, sliders, and other widgets
- Some principles of composition (i.e., the rules of putting them together)
- Collections of qualifying situations – contexts and how they affect the rules

To achieve a suitable design language, the color scheme was based on Aaron Marcus's five-color design rules and the colors presented on K14's website and prospect (Benyon, 2019, p. 308). This resulted in a palette of four colors and two variations of black and white (Appendix 14). The four colors had varying opacity according to each design element and page, based on color areas that exhibit a minimum shift in color and or size. Additionally, simultaneous use of high-chrome spectral colors was avoided. The color design is coherent with Marcus' presentation of Western color conventions (Appendix 15). As the mockup's design primarily consists of green and white colors, it corresponds with Marcus' color conversion, with green symbolizing elements such as clearness, vegetation, safety, and okay. These elements confirm K14's vision of being a modern and sustainable facility, with various employees and visitors of numerous generations (K14, 2020, pp. 27-29).

Moreover, the design language's general style is characterized by consistent fonts, including size and color, button types and size, and the shape and radius of elements. K14 has a vision of being modern. Morales (2021) claims that there are four main types of typography. The design language of our project has a Sans Serif font by the name Poppins, which is similar to the font Roboto, one of Morales' recommended font types. Besides being considered a modern font, the chosen font's readability was considered.

The elements and buttons had a rounded rectangular shape with a radius of either twenty-one or ten. This choice is based on rounded corners in both buttons and elements being considered more friendly than sharp edges (Malewicz & Malewicz, 2020, s. 46). Additionally, when using rounded buttons, other on-screen elements in the background have a different ratio than the button, preventing an imbalance in the margins.

6.4 Development

Before development, the group carried out analytical work to lay the foundation for a successful development process. This preparatory work enabled us to start programming and avoid pitfalls.

Each sprint started with Sprint Planning, where user stories would be moved from the Product Backlog into the Sprint Backlog. The choice and abundance of user stories were

based on MoSCoW priorities and the group's total capacity for development in each sprint. As a result of the preparatory work, each user story would have corresponding acceptance criteria. When creating tasks, these criteria would be used as guidelines, and it was ensured that each task was of a manageable size for one group member. Planning poker was the chosen method for estimating time usage. We would clarify each task before placing our initial estimate to avoid confusion. The highest and lowest estimates would discuss their reasoning before placing our second estimate based on the presented arguments. We selected estimates based on the highest number of votes or the median in case of a tie.

Thereafter, two group members were responsible for writing descriptions for each task in DevOps, linking to mockups, diagrams, and other relevant documentation where necessary. Group members were then able to choose a task, assign it to themselves in DevOps to avoid duplication of work, and log the number of hours spent completing respective tasks. After a task was completed, a pull request would be made and marked with the required reviewers. The required reviewer would run the code on their phone through Expo and answer the following questions:

1. Are the acceptance criteria fulfilled?
2. Does the code comply with the code conventions?
3. Does the code run without any bugs?
4. Does the code have test coverage?
5. Does the user interface match the intended design?

Answering no to any of these criteria would result in an unapproved pull request, whereas approved requests would be incorporated into the development branch. For incorporating changes from one branch into another, we had two main options: rebasing or merging. Rebasing was chosen as it rewrites the project history, providing a linear overview (Atlassian, n.d.). We considered this to be more advantageous, as merging preserves the project history, giving a non-linear overview that can make the history cluttered, thus decreasing project control. Pull requests were not thoroughly checked until Sprint 3, which in hindsight compromised the quality assurance process, and will be discussed in chapter 8.2.5. Following Sprint 3, however, pull requests were thoroughly reviewed.

The end of each sprint would then result in a code review and product increment. Each team member would present the code they had produced to the group; thereafter, our build pipeline was executed. The build pipeline builds our application and executes the tests in the cloud while also generating reports for test coverage and an overview of which tests

succeeded or failed. Finally, if the team members accepted the code and the pipeline succeeded, the code would be rebased into their respective main branches. Then followed a Sprint Review with the product owner, before concluding the sprint with a Sprint Retrospective.

From the start of the project, the team was divided into smaller teams, with clear areas of responsibility to secure the most effective use of our capacity. We operated under a four-to-two division. Four group members made up the development team, and two were responsible for documentation and report writing and assisting the development team when needed. Within the development team, the four roles were: frontend developer responsible for the design, frontend developer responsible for the functionality, backend developer, and full-stack developer.

6.5 Project close

This chapter will explain the closing phase of the project, involving changes related to the execution and workload distribution.

Towards the end of the project, the sprints were shortened to two weeks because the focus shifted towards more frequent deliveries and feedback. At the beginning of the development phase, the workload between the six team members primarily focused on developing the product. Two members were committed to developing backend code, two were assigned to develop frontend code, and the last two members were in charge of writing the bachelor thesis. As the project came closer to the deadline, the area of focus shifted more towards finishing the bachelor thesis. Now, five team members were writing the report while the full-stack developer implemented the remaining work, tenacious to finish the must-have user stories to complete the MVP.

Even though the focus shifted, we were determined to adhere to the chosen Scrum artifacts. Daily standup was utilized without exception and assisted us in continuous updates on progress and bottlenecks. The other Scrum artifacts were also present, such as Sprint Review and Retrospective, but the topics rather concerned the bachelor thesis.

7 The Final Product

In this chapter, we will present the solution to our project. We will focus on the architecture, functionality, and further development.

7.1 Architecture and Backend

The backend architecture solution is a Rest API containing necessary endpoints for the frontend to perform requirements stated in the user stories (Figure 9). This included a login-controller to authenticate users through username and password and provide a JWT as authorization. Frontend stores the JWT to gain access to the other endpoints. This is not applicable for the registration-controller and password-controller, as one does not need authorization to create a user or reset a password. Password-reset is authorized through email. The other endpoints is concerned with fetching data such as amenities, businesses, and events and contacting and summoning the receptionist.

The developed microservice utilizes the controller-service-repository pattern, in which the controller handles requests, the service performs logic such as data validation and calculations, and the repository queries the database, using entities to mirror the database tables as java classes.

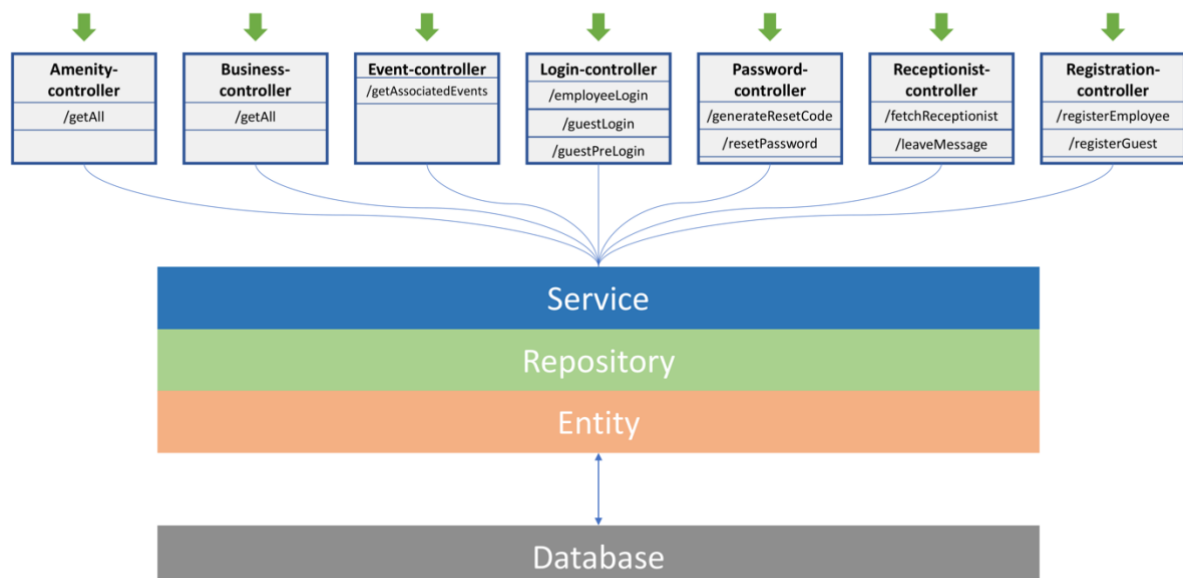


Figure 9) Backend architecture

7.2 Functionality

Appendix 16 displays the application's landing page; the first page users meet after logging in. The purpose is to give users quick and easy access to information and the application's functionality. After conducting the analysis, it was clear that each company had very different needs and desires for functionality. It is therefore intended to make this page customizable, enabling users to choose which features appear on their homepage.

The page shown in Appendix 17 is the K14 building page. The purpose of this page is to easily allow users to navigate to functionality relating to receptionist tasks and services that K14 will offer. The six chosen options are based on our previous communication with K14 representatives and data collected from the future building tenants. Considering that we are creating a receptionist module, we deemed this page as particularly important, thus placing it in the navigation bar to make it easily accessible.

Many interviewees expressed their desire for an easy way to keep track of their daily agenda and meetings. Appendix 18 shows the implemented calendar function. This function allows users to have a clear overview of their daily plan, upcoming events, and create bookings. The K14 building will facilitate room booking for meetings, public conferences, and seminars. The calendar has different colors to help differentiate between private and public events and events spanning multiple days.

Appendix 19 presents the implemented contact personnel page. This page can be accessed from the navigation bar and homepage, as shown in Appendix 16. Having the option of contacting a receptionist was desired by all interviewees in case of urgent need of assistance. This page offers three means of communication, depending on the urgency of the situation:

1. Directly call the receptionist from your phone by pressing the call button.
2. Summon the receptionist to the reception desk.
3. Leave the receptionist a message.

Appendix 20 presents the K14 information page. The purpose of this page is to display static information about the building. Its primary intended use is to give, mainly guests, insight into what K14 offers.

7.3 Testing

To ensure the application's ability to function properly, we have written tests for the majority of the code. For the backend, we used JUnit to create integration tests, meaning that we test the endpoint and verify that it returns the expected HTTP status on a given scenario. JaCoCo reports a test coverage of 87%, exceeding Knowit's requirement of 80%.

One test was written for the frontend, namely a test for ensuring that the login functionality met its acceptance criteria. This was done regardless of it not being one of Knowit's requirements as we wanted to measure and ensure the quality of the user experience.

7.4 Facilitating Further Development

The scope of our project was the first module for the K14 application. The future modules will make up a potential future internship, master's, and bachelor's projects, and it has therefore been essential to facilitate further development.

7.4.1 Analysis

Despite many functions not being within our project scope, we were consistent with including all aspects in our problem and application domain analysis. We reduce the amount of preparatory work needed by future teams and ensure that the application is implemented as intended by addressing these areas. An example is our previously shown navigation diagram. This diagram shows the planned navigation to all the applications pages, additionally showing users with their respective areas of access.

7.4.2 Design

The design team created mockups for some pages outside the scope, including components that facilitate navigation to undesigned pages. For example, pages leading from parts of the navigation bar and the 'edit' and 'alert' icons on the index page do not have any functionality or design. The icons were included to illustrate where this functionality is intended. A consistent color palette was used throughout all designs, along with consistent buttons, backgrounds, and more, to facilitate designing new pages uniform with what we already have produced.

7.4.3 Code Conventions

The application's code is written in English, despite the application's content only being displayed in Norwegian. English is the most widely spoken language, and therefore, we

facilitate developers from all countries in understanding our code (Statista, 2022). We have consistent code documentation and have followed a set of code conventions (Appendix 21). This has ensured clear and logical names for classes, functions, files, and folders to avoid confusion and reduce the time needed to become familiar with the code. By creating default properties for various design components, it is easier to implement our intended design, as well as create a design that is consistent with what we have already implemented.

7.4.4 Database

Numerous changes were made to our database throughout the development phase to facilitate further development. We decided to limit the use of enums in tables and instead create extra tables to hold these values. Enums should be used to hold a static set of values, and the use of this data type would hinder future developers from changing said values. We implemented this change in, for example, the FAQ table by removing the “Category” enum and instead created a table to hold these categories. FAQ pages are often under constant modification; thereby, it is essential that we facilitate the addition and deletion of categories, questions, and answers.

8 Reflection

This final section will reflect on the decisions made throughout the project, the challenges we have faced, and how we have attempted to handle these. We will also discuss how these aspects affected the progress of the project.

8.1 Process and Methods

This subchapter will present the reflection revolving around the most influential processes and methods in this project. Firstly, we will reflect on both the benefits and disadvantages of the chosen methodology, before discussing the exclusion of the Scrum Master role.

Furthermore, we will discuss the importance of sprint planning and associated problems.

Additionally, challenges regarding work capacity and data collection will be reflected upon.

Moreover, we will discuss the transition of moving from one phase of the project to the next, before lastly including adaptation to development challenges and concluding what quality really entails.

8.1.1 Methodology

Despite the overall impression of the chosen methodology being positive, we have faced some challenges in the process. We felt a lack of control concerning the backlog in various sprint iterations, due to modifications in the respective Sprint Backlogs. However, after the meetings with the client, we got the impression that the lack of requirements would demand a more agile approach, therefore, enabling change in the Sprint Backlogs if necessary. Change is inevitable with an ambitious client presenting vague information, few requirements, and limited interactions. We concluded that the chosen methodology befitted this project to a great extent. The reason being that the project was poorly defined, and we had to conduct data collection and a thorough analysis to accommodate the lack of information. Therefore, a methodology that allowed us to respond to ongoing changes was crucial. Moreover, it was established that the XP methodology would be utilized should there be a lack of progress in development. However, rather than adopting a new methodology, we decided to shorten the sprint lengths with the intent of rapid feedback and frequent deliveries. Considering the lack of communication with the K14 representatives, shortening the sprints allowed us to get the sense of direction we needed to advance with the project.

As we decided to use an Agile framework with a selection of Scrum Artifacts, rather than Scrum in a traditional sense, expected roles like Scrum Master were excluded. Early in the process, we established guidelines, individual and collective expectations, and ambitions to develop the team contract further. During the presentation of these topics, a discussion

concerning Scrum Master was brought to light. We quickly concluded that, rather than delegating this role to one individual, no member was officially made Scrum Master. We had a collective understanding of the amount of work needed to reach our goals and our expectations of each other. Therefore, we did not see the value of delegating additional work to one individual but saw it as a collective responsibility. Consequently, making the typical Scrum Master's responsibilities, such as coaching and leading Scrum artifacts, a joint obligation. Looking back at the project, we have the impression that this was the right decision. This resulted in every member stepping into the role when needed and increased ownership of the process.

8.1.2 Planning

Based on previous experiences with Scrum artifacts, we consider Sprint Planning vital, as it lays the foundation for the coming weeks. Each sprint had a rigid scope and schedule, which in our experience, had both advantages and disadvantages.

Sprint Planning increased our control over the project regarding the estimated completion time, which could be presented to the product owner. It also gave us a sense of security that our project was accomplishable within the given time frame. However, having a rigid schedule and scope within our sprints meant that unforeseen tasks would compromise this structure. For example, in the first sprint, we planned for a series of pages to be developed within a given time frame. When the designs for these pages were implemented, we failed to consider the need to implement a framework for calling APIs. Consequently, two problems occurred. Firstly, we did not account for the time such an implementation would infer. To accommodate this, the implementation for some of the pages were moved into the next sprint. Secondly, it weakened our sense of project control because the new task did not fit in any of our user stories in the Sprint Backlog.

Another key factor in Sprint Planning was the importance of communication. Constructing a plan by communicating with the team means that the result is a plan that compliments each member's competence and interest while also tending to the needs and specifications of the client. We also discovered the importance of having people playing *the devil's advocate* within the team. It forced the team to reflect upon multiple possible solutions to various problems, leading the team to pursue what we consider to be the optimal solution.

8.1.3 Misleading Capacity Calculations

As we reached the end of the project, it became apparent that we had room for improvement regarding actively working with and documenting the time spent throughout the project. We have used planning poker for each Sprint Planning and engaged in discussions to provide the most accurate time estimate for each task. Even if we consider the process of Sprint Planning and task estimation a success, it became evident that we needed to increase our time management efforts during the course of each sprint.

While planning the development phase, we estimated that we would have a working capacity of approximately 800 hours across two months of developing the application. By excluding the time devoted to Sprint Planning, Code Review, Sprint Review, and Retrospective, we get approximately 560 hours of capacity dedicated to development tasks. The actual time estimated for all development tasks is, on the other hand, only 304 hours, meaning that a lot of working capacity was left unaccounted for. Further, only 284 hours were recorded, even though the developers spent a number of hours equivalent to the 560-hour capacity. While investigating this deviation, we found that:

- Bugs and additional features added during sprints lacked estimates, and hours spent were not logged due to a misconception that this was not intended.
- Tasks not directly associated with coding were not included in Azure, i.e., checking pull requests.
- We forgot to log the time spent on individual tasks. In contrast to the analysis phase, each team member worked independently during development. This resulted in fewer reminders to log hours spent on tasks.

Consequently, this led to misleading burndown charts not corresponding to the actual work. Considering that we used metrics, such as estimates and hours spent from previous sprints, to determine the number of tasks we could complete in the next sprint, we got an inaccurate conception of how much work we were able to do. Unfortunately, we were not aware of this issue until the final stages of the development, making us unable to counteract it.

8.1.4 Inadequate Communication and Feedback

As there were no explicit product requirements, it required us to gather as much information and requirements as possible to meet the client's vision and the future users' needs. Initially, we sought to interview nine respondents to get the most valuable feedback and not get redundant answers. However, only four of the possible tenants agreed to being interviewed,

consequently not getting as much feedback and insight as first intended. Although we were not able to get the desired number of interview respondents, substantial feedback was provided, contributing to a solid foundation for the design phase.

Furthermore, continuous communication and feedback from the K14 representatives remained absent throughout the project. This resulted in us having to make important decisions, without verification from the client, possibly compromising the integrity of the product by not accommodating user and client expectations and needs.

8.1.5 Transitioning from Analysis to Development

During this project, a significant challenge was establishing when it was sensible to move from one stage of the process to the next, specifically from analysis to development. In our earlier projects, lecturers had given specific guidelines and methods to carry out a thorough system development process. Without these guidelines, we were faced with making these decisions ourselves. It was challenging to find the right balance between having sufficient preparatory work without compromising our limited development time. Before starting the development, we wanted to ensure a solid foundation to secure quality and aid us in being as efficient as possible. We had planned to conduct analysis through sprint one and start development in sprint two while rounding off the remaining analysis tasks. During the first Sprint Review, our mentor expressed his satisfaction with our preparatory work and told us to solely focus on development for the remainder of the project. Along the way, it was made clear that the analysis process was cut short, and the design team was faced with having to produce multiple mockups for pages that had not been designed yet. These mockups had to be rushed, as many development tasks depended on these designs. This was not optimal, as it affected the group's progress, resulting in mockups less thorough than the previous ones.

8.1.6 Adapting to Challenges

During the initial stages of the project, the group intended to incorporate pair programming during the development process to increase the quality of code produced. Many group members had experienced the benefits of using this method in previous projects. It was intended to utilize this method on more extensive and more difficult coding tasks, but more importantly, to pair developers with less experience with those more experienced. As a result of only four group members contributing to the development and limited time, we did not utilize pair programming in its traditional fashion, as mentioned in chapter 2.3. Consequently,

it was clear that group members with less coding experience struggled during the initial stages of programming, resulting in limited code being produced. This issue was voiced during our first Sprint Retrospective, and it was clear we had to initiate measures to help accommodate those struggling. As a group, we decided the best option was to lower the threshold of asking for help. To achieve this, we made sure all developers were available in our Discord during working hours to quickly be able to help when problems occurred or review code when necessary. This proved to be an effective way of aiding progress within the group and was used throughout the project, improving communication and quality.

8.1.7 Understanding Quality

As we see it, the information gathering, and analysis phase is the most critical part of the development process, therefore, we consider this stage the most influential on general product quality. Rushing or failing in parts of this phase could cause the product to not fit its intended domain or fail to satisfy user needs. Based on this, and the fact that we initially received little information, the team decided to retain a significant part of the total timeframe to collect and compile relevant data. This meant that we would have less time for development; therefore, we sacrificed development time to better construct a strong foundation to base the app upon. Interviews with future tenants were conducted to attain a holistic impression of user needs.

Additionally, one group member contacted an acquaintance employed as a receptionist in a commercial building, which aided us in acquiring important domain knowledge. These sources provided both an inside and outside view of the domain. Modeling the domain from different perspectives allows us to better understand, identify, and map user requirements and how to accommodate these. Providing us with key information on how to best develop a user-centered experience. The most important decision we made during the project was to set aside additional time to collect and analyze information. Looking back on how the project started and the preconditions we were faced with, we now realize that product quality and project control were fundamentally dependent on sufficient information being obtained before development began. Additionally, this choice has been imperative in fulfilling project requirements. The time spent and how we have worked with the information gathered have aided in establishing a strong foundation on which to base future development.

Throughout the project, we have realized that the quality of the process is purely based on the context of a project. Even though a set of procedures, rules, and guidelines impact overall project quality, it is essentially contextual. Even though procedures and rules can help improve overall project management, it does not dictate the quality of the project. We believe

that it is more nuanced than some definitions will have us believe; however, understanding how product quality derives from the quality of the process seems to be at the heart of successful projects.

8.2 Product Decisions

This subchapter will present the most prominent challenges that directly or indirectly impact product quality. Firstly, we will discuss the effect of reorganizing the prioritization of user stories before discussing if the chosen design framework was advantageous. Furthermore, the reason for deprioritizing user stories and their consequences will be presented, before reflecting upon the prioritization of functionality or design. Lastly, we will assess if pull requests were sufficiently reviewed before merging branches.

8.2.1 Prioritizing User Stories

One question we often asked while mapping out user stories was, “What is a must-have user story?”. It became clear that each group member had their own definition and ideas of what fulfills must-have criteria. Therefore, it was necessary to discuss each MoSCoW priority to avoid misunderstandings, and unnecessary discussions. In the end, we established that a must-have user story is, most importantly within our scope, and related to functionality that is essential for creating a functioning application. After conducting interviews, collecting relevant data, and creating mockups, we found numerous user stories that, in many aspects, should be considered a top priority. For example, design components essential for navigation, completion of the intended design, and functionality strongly desired by our interviewees. However, these did not meet the previously mentioned criteria and were therefore placed as should-have requirements. Along the way, we found ourselves having to include should-have user stories in our Sprint Backlogs to be able to complete intended functionality and excluding must-have user stories that turned out to be less relevant. This set us back in completing our project goal of completing all must-have user stories.

8.2.2 Selecting a Suitable Design Framework

We faced numerous obstacles throughout the development process, most frequently relating to implementing the design. NativeBase was chosen as the component library to aid us in implementing the design. This decision was based on research in which different design libraries and frameworks were compared. NativeBase appeared to be the best option, claiming that its library enables people to build consistent designs across all platforms. Despite these claims, we encountered significant variation in how the implemented design

appeared depending on the device operating system and screen resolution. As our application was designed to be accessible for all mobile phones, we could not consider many user stories as complete before this problem was fixed. Therefore, implementing the design proved to be a time-consuming process due to the use of NativeBase. It was clear that the process of selecting a design library was rushed, and we should have spent more time not only researching but actively testing suitable options before making a final decision. During sprint five, significant amounts of NativeBase styling were removed to make the design consistent across all platforms. The styling was replaced with CSS, contradicting the point of using NativeBase. This resolved most of the design challenges, therefore making it clear that NativeBase was the wrong choice for this project.

8.2.3 Deprioritizing User Testing

Through previous project experience, we have learned how user testing is central in increasing both quality and customer satisfaction. By conducting user tests regularly throughout the project, we can adjust according to the client's wishes, thereby reducing the risk of having to make significant changes to the product at the project close. Since we started developing mid-semester, we decided not to prioritize user testing to the extent we aimed initially. Instead, we prioritized implementing functionality and completing as many must-have user stories as possible, with the main reason being that we had limited functionality to test on potential users. We concluded that it was inexpedient to test the login and register functionality. This is widely known and commonly used functionality, therefore making it unlikely that we would receive valuable feedback regarding these aspects of the application. Despite this being the most appropriate choice for us, we still believe conducting user testing would have given us more insight into how we could have improved the overall quality of our product. The consequence of not involving the users is that we do not get continuous feedback from the end-user, and thus cannot adapt changes to fit their needs. The lack of feedback regarding design and functionality could result in a reduced user experience, as a high-quality product revolves around customer satisfaction approval.

8.2.4 Functionality vs. Design

One of the biggest challenges we faced was implementing the intended design to look consistent across all platforms. We were left with multiple must-have user stories with significant design flaws during our fifth and final Sprint Planning. We were therefore faced with the predicament of whether to prioritize the implementation of not yet started user stories or completing previously started but flawed ones, possibly implementing remaining user stories if time allows for it. After debating both sides, the group reached the consensus

of fixing and ensuring the completion of already initiated user stories. Agile methodologies prioritize the delivery of working software; therefore, we agreed that it would be in the best interest of all project stakeholders if the software delivered was fully completed.

8.2.5 Reviewing Pull Requests

Before the third Sprint, the group did not review pull requests thoroughly enough. In practice, a pull request was accepted if it was possible to rebase without conflicts. Following a series of incidents where the application would not function properly after rebasing, we implemented a protocol for reviewing pull requests, as mentioned in 6.4. Following these protocols significantly improved our quality assurance process. Leading up to sprint four, if the design was not consistent across devices, it was considered a minor flaw, and the pull request would be approved. However, in sprint four, we realized that this was a major issue. Therefore, the pull request reviewer would have to check that the design was consistent across devices. This was done by having an emulator, making another member with the required phone review the pull request, or borrowing the required device from another team member.

9 Team Evaluation

Through this project, all group members have gained valuable knowledge and insight into the reality of system development. The group experienced for the first time what it is like to manage and conduct such a project without the guidance of lecturers and teaching assistants. Many challenges and difficult decisions have arisen along the way, which we have, to the best of our ability, handled.

Overall, we have been pleased with the group dynamic, as we have been able to play on each other's strengths. One crucial takeaway from this project is the importance of having clear responsibility areas for each group member. Through this, we were able to ensure efficient use of our time and learn considerably more than if we had to share the responsibility of each area. Each member was also allowed to learn and work on what they desired, which aided in increasing motivation and morale and the amount of work produced. Areas explored for the first time include mobile application development, design frameworks, extensive testing, pipelines, and more. The project has allowed us to both strengthen and expand our knowledge.

Unfortunately, carrying out a project of this size for the first time comes with its challenges. Being a group of six strong-minded individuals, coming to an agreement often proved to be time-consuming. However, more often than not, we managed to have thorough and factual discussions, resulting in well-thought-out decisions and concepts.

To sum up, we are pleased with both the project and final product, and each group member has been able to learn and develop in their own way. The project has left each member with valuable knowledge which can be transferred to similar projects in the future. We are pleased with the partnership with Knowit Sør and their statement can be viewed in Appendix 22.

10 Conclusion

The main objective of this project was to develop an MVP serving as the initial module for the application for K14. The final product fulfills the requirements and needs covered by the completed user stories, based on information from Knowit, future tenants, and the K14 representatives. We consider the overall quality to be high, largely, but not exclusively, as a result of the effort put into the preparatory work, and the initial phases of development, and even though some must-have user stories remain incomplete, we consider the project a success. As we have worked on one module of the application, considerations for further development have been made. One of our main impediments was that our work, in some cases, was overly agile, leading to the incorporation of more changes than we were able to effectively handle, slowing down progress. We have experienced the importance of fitting the methodology to the project, as it can either aid or impair the team in achieving their objective. Additionally, thorough planning, good communication, and well-structured work are indispensable in ensuring project quality and control. As such, we recommend that future developers use the provided foundation for development and consider our experiences when furthering the project, to better their chances of success.

Bibliography

- Agile Alliance. (2022, February 6). *Pair Programming: Does It Really Work?* Retrieved from Agile Alliance: <https://www.agilealliance.org/glossary/pairing/>
- Alshamrani, A., & Bahattab, A. (2015, January). A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model. *IJCSI International Journal of Computer Science Issues*, pp. 106-111. Retrieved from <https://www.ijcsi.org/papers/IJCSI-12-1-1-106-111.pdf>
- Atlassian. (n.d.). *Merging vs. Rebasing*. Retrieved from Atlassian: <https://www.atlassian.com/git/tutorials/merging-vs-rebasing>
- Atlassian. (n.d.). *What is version control: Atlassian Git Tutorial*. Retrieved from Atlassian: <https://www.atlassian.com/git/tutorials/what-is-version-control>
- auth0. (n.d.). *JSON Web Tokens*. Retrieved from jwt.io: <https://jwt.io/>
- Benyon, D. (2019). *Designing User Experience* (Vol. 4). Pearson Education Limited.
- Berardi, D., Calvanese, D., & De Giacomo, G. (2005). Reasoning on UML class diagrams. *Artificial Intelligence*, pp. 70-118. doi:10.1016/j.artint.2005.05.003.
- Cao, J., Ellis, M., & Khachatryan, N. (2016). *The Guide To Mockups: Mockup types, methods and best practices*. Academia. Retrieved from https://www.academia.edu/24729528/The_Guide_to_Mockups
- Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Pearson Education. Retrieved from <https://books.google.no/books?hl=no&lr=&id=SvlwuX4SVigC&oi=fnd&pg=PR13>
- EclEmma. (2022, April 5). *Jacoco Java Code Coverage Library*. Retrieved from EclEmma: <https://www.eclemma.org/jacoco/>
- ESLint. (n.d.). *Pluggable JavaScript linter*. Retrieved from ESLint: <https://eslint.org/>
- Expo. (n.d.). *Expo*. Retrieved from Expo: <https://expo.dev/>
- Furia, C. A., & Nanz, S. (2012). Objects, Models, Components, Patterns. *50th International Conference, TOOLS 2012*. doi:10.1007/978-3-642-30561-0
- H2 database engine. (n.d.). Retrieved from H2 database engine: <https://www.h2database.com/html/main.html>
- Halvorsen, K. (2014). *Å forske på samfunnet - en innføring i samfunnsvitenskapelig metode* (5. utgave ed.). Oslo: Cappelens Forlag AS.
- Janssen, T., Hamed, A., Misha, & Werner, T. (2021, October 31). *What is Spring Data JPA? and why should you use it?* Retrieved from Thorben Janssen: <https://thorben-janssen.com/what-is-spring-data-jpa-and-why-should-you-use-it/>
- JUnit. (2021, February 13). *About JUnit 4*. Retrieved from JUnit: <https://junit.org/junit4/>
- K14. (2020, August). *K14 Prospect*. Retrieved from K14: <https://k14.no/prospekt/>
- K14. (2021, May 27). *K14 blir Kvadraturens urbane storstue*. Retrieved from K14: <https://k14.no/k14-blir-kvadraturens-urbane-storstue/>

- Knowit. (n.d.). *Om Knowit: Digitaliseringskonsulenter*. Retrieved from Knowit: <https://www.knowit.no/om-knowit/>
- Leung, H. K., & White, L. (1989). Insights into Regression Testing. *Proceedings. Conference on Software Maintenance - 1989*, pp. 60-69. doi:doi: 10.1109/ICSM.1989.65194
- Malewicz, M., & Malewicz, D. (2020). *Designing User Interface*. Retrieved from https://designingui.com/designing_interfaces_12_x.pdf
- Mantri, A., Nandi, S., Kumar, G., & Kumar, S. (2011). *High Performance Architecture and Grid Computing*. Heidelberg: Springer. doi:10.1007/978-3-642-22577-2
- Maple, S., & Binstock, A. (2021, November 25). *JVM Ecosystem Report 2018 - About Your Platform and application*. Retrieved from Snyk: <https://snyk.io/blog/jvm-ecosystem-report-2018-platform-application/>
- Mathiassen, L., Munk-Madsen, A., Nielsen, P., & Stage, J. (2018). *Object Oriented Analysis & Design*. Hadsund, Denmark: Metodica ApS.
- Microsoft. (2022, February 9). *What is Azure DevOps?* Retrieved from Azure DevOps | Microsoft Docs: <https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops>
- Morales, J. (2021). *The 7 Best Modern Fonts for Websites: Adobe XD Ideas*. Retrieved from Ideas: <https://xd.adobe.com/ideas/principles/web-design/best-modern-fonts-for-websites/>
- Münch, J., Armbrust, O., Kowalczyk, M., & Soto, M. (2012). *Software Process Definition and Management*. Heidelberg, New York, Dordrecht, London: Springer. doi: 10.1007/978-3-642-24291-5
- NativeBase. (n.d.). *Universal Components for React & React Native*. Retrieved from NativeBase: <https://nativebase.io/>
- O'Reilly. (n.d.). *Learning React Native*. Retrieved from O'Reilly Online Learning: <https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html>
- Oracle. (n.d.). *JavaMail*. Retrieved from JavaMail: <https://javaee.github.io/javamail/>
- Ponce, F., Márquez, G., & Astudillo, H. (2019). Migrating from monolithic architecture to microservices: A Rapid Review. *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, pp. 1-7. doi:10.1109/SCCC49216.2019.8966423.
- Porter, B., van Zyl, J., & Lamy, O. (n.d.). *Welcome to Apache Maven*. Retrieved from Maven: <https://maven.apache.org/>
- ProductPlan. (2021, September 9). *The Definition of Done: What Product Managers Need to Know*. Retrieved from ProductPlan: <https://www.productplan.com/learn/agile-definition-of-done/>
- Project Lombok. (n.d.). *Project Lombok*. Retrieved from Project Lombok: <https://projectlombok.org/>
- Selenium. (n.d.). *Selenium*. Retrieved from Selenium: <https://www.selenium.dev/>
- Shrivastava, A., Jaggi, I., Katoch, N., Gupta, D., & Gupta, S. (2021). A Systematic Review on Extreme Programming. *Journal of Physics: Conference Series*. doi:10.1088/1742-6596/1969/1/012046

- SmartBear. (n.d.). *Behaviour-driven development*. Retrieved from Behaviour-Driven Development - Cucumber Documentation: <https://cucumber.io/docs/bdd/>
- SmartBear. (n.d.). *OpenAPI specification*. Retrieved from Swagger: <https://swagger.io/specification/>
- SmartBear. (n.d.). *The single source of truth for API development*. Retrieved from SwaggerHub | API Design and Documentation with OpenAPI: <https://swagger.io/tools/swaggerhub/>
- Spring. (2022). *Spring Boot*. Retrieved from Spring: <https://spring.io/projects/spring-boot>
- Statista. (2022, April 1). *Most spoken languages in the world*. Retrieved from Statista: <https://www.statista.com/statistics/266808/the-most-spoken-languages-worldwide/>
- Sundbye, L. T., & Nisted, I. (2017, October 11). *Primære og sekundære datakilder*. Retrieved from NDLA: <https://ndla.no/nb/subject:1:433559e2-5bf4-4ba1-a592-24fa4057ec01/topic:2:183191/topic:2:105795/resource:1:93370>
- Sutherland, J. (2021). *The Scrum Papers: Nut, Bolts, and Origins of an Agile Framework*. Lincoln MA. Retrieved from <http://jeffsutherland.org/scrum/scrumpapers.pdf>
- The Agile Manifesto. (2001). *Manifesto for Agile Software Development*. Retrieved from The Agile Manifesto: <https://agilemanifesto.org/iso/no/manifesto.html>
- TypeScript. (n.d.). *JavaScript With Syntax For Types*. Retrieved from TypeScript: <https://www.typescriptlang.org/>
- University of Agder. (2021). *Bachelor Thesis in Information Systems*. Retrieved from University of Agder: <https://www.uia.no/en/studieplaner/topic/IS-304-1>
- Usability.gov. (n.d.). *User Interface Design Basics*. Retrieved from Usability.gov: [https://www.usability.gov/what-and-why/user-interface-design.html#:~:text=User%20Interface%20\(UI\)%20Design%20focuses,visual%20design%2C%20and%20information%20architecture](https://www.usability.gov/what-and-why/user-interface-design.html#:~:text=User%20Interface%20(UI)%20Design%20focuses,visual%20design%2C%20and%20information%20architecture)
- Venters, C. C., Capilla, R., Betz, S., Penzenstadler, B., Crick, T., Crouch, S., . . . Carrillo, C. (2018, April). Software sustainability: Research and practice from a software architecture viewpoint. *Journal of Systems and Software*, 138, pp. 174-188. doi:10.1016/j.jss.2017.12.026.
- Waters, K. (2009). *Prioritization using MoSCoW*. Retrieved from All About Agile: https://comp.anu.edu.au/courses/comp3120/local_docs/readings/Prioritization_using_MoSCoW_AllAboutAgile.pdf

Appendix

Appendix 1: Potential Development Areas Identified

Areas	Description
Canteen and food ordering	The building will include both a canteen for building tenants, and various restaurants for guests. Here we could create a function to be able to order food directly from the application for all these restaurants. The menu for each place would be easily accessible through the app, enabling users to decide where to eat before leaving the office. The function for ordering food could show what the user last ordered or most frequently orders, to save them time and give them a personalized feel. The function would allow for ordering left over from the canteen at the end of the day. This contributes to reducing food waste, further contributing to K14 being a sustainable building.
Meeting room	This function would allow for booking meeting rooms, coworking areas, single desks, seminar rooms, and other areas that K14 will offer. This will be available for both tenants and guests. For guests, there will be limitations for which areas are available to book, as well as a payment function in the application. After guests book, they receive a digital access card which will allow them access relative to the amount of time they have booked. This function can be integrated with the food ordering function, to allow ordering food for meetings.
Access and authorization	Though implementing authorization, we are able to map who has access to what in the building and set up a hierarchy to distribute functionality in the app. After logging in, the front page will include a digital ID card, which allows access into the building, as well as restricting areas of the building. By integrating access areas, we can separate tenant and guest functionality in the application. By using a digital ID card, we are contributing to the sustainability of the building, by removing the need for plastic ID cards.
Carpool and electric scooter rental	The K14 building will offer rental of both electric cars and scooters. It is intended that the application should allow for a simple rental process. The application would present the number of cars and

	scooters available, to avoid wasting user's time. The application should also include a payment function, to make rental an easy process.
Parking	The building will include its own parking space, and therefore it would be convenient to integrate a parking function in the application. This function should allow for both registering parking and payment. This functionality could be integrated with the reception app/index page, to easily display parking information to the user, such as remaining parking time.

Appendix 2: The Learning Outcomes of The Bachelor Course

Learning outcomes

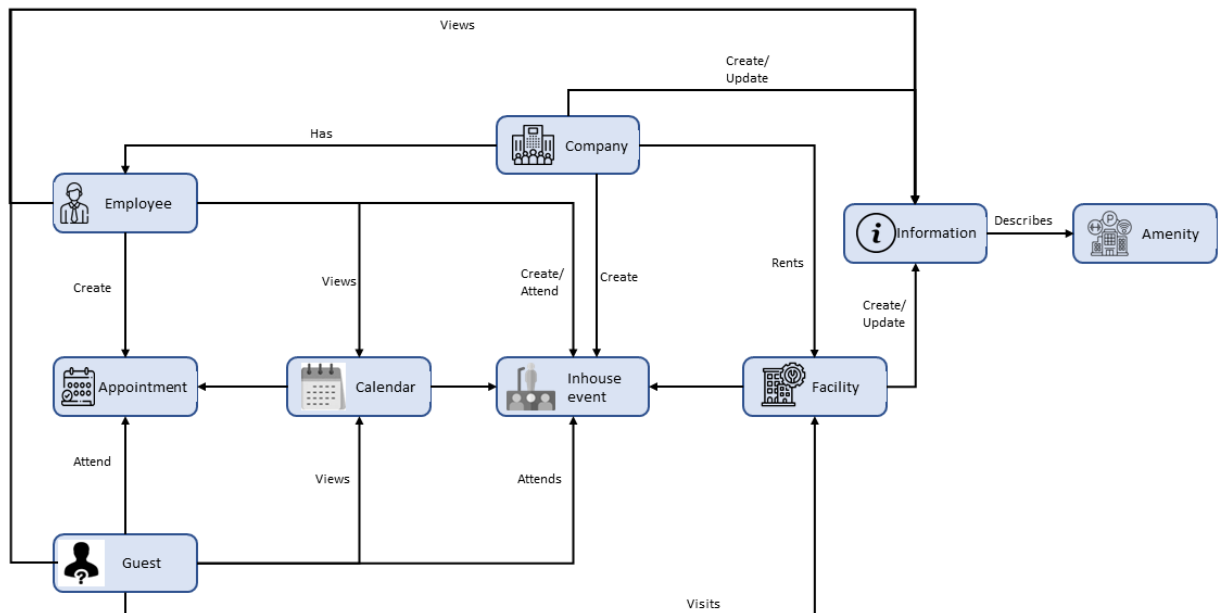
Upon completion of the course, students will be able to:

- carry out an IT/IS-related project using established methods and techniques, including the following elements: plan, estimate, perform, test, follow up and document the project
- define quality in the actual project and implement quality assurance actions
- control progress and quality throughout the project, with focus on delivering value to the client
- assess the opportunities and risks associated with creating value from the project/product
- Work with user participation, before and during work with the product, as well as be able to evaluate the effects of the product, and suggest actions to adapt the product or its intended use and value.

Appendix 3: Risk Matrix

		Severity				
		Negligible 1	Marginal 2	Moderate 3	Critical 4	Catastrophic 5
Probability	Certain 5		Scope/time estimation error			
	Likely 4		Lack of competence		Working inefficient	
	Possible 3	Programming fault Stuck in project			Disproportionate workload	
	Unlikely 2			Covid-19 or illness / Miscommunication with customer	Team conflict	
	Rare 1				Damaged or malfunctioning equipment	

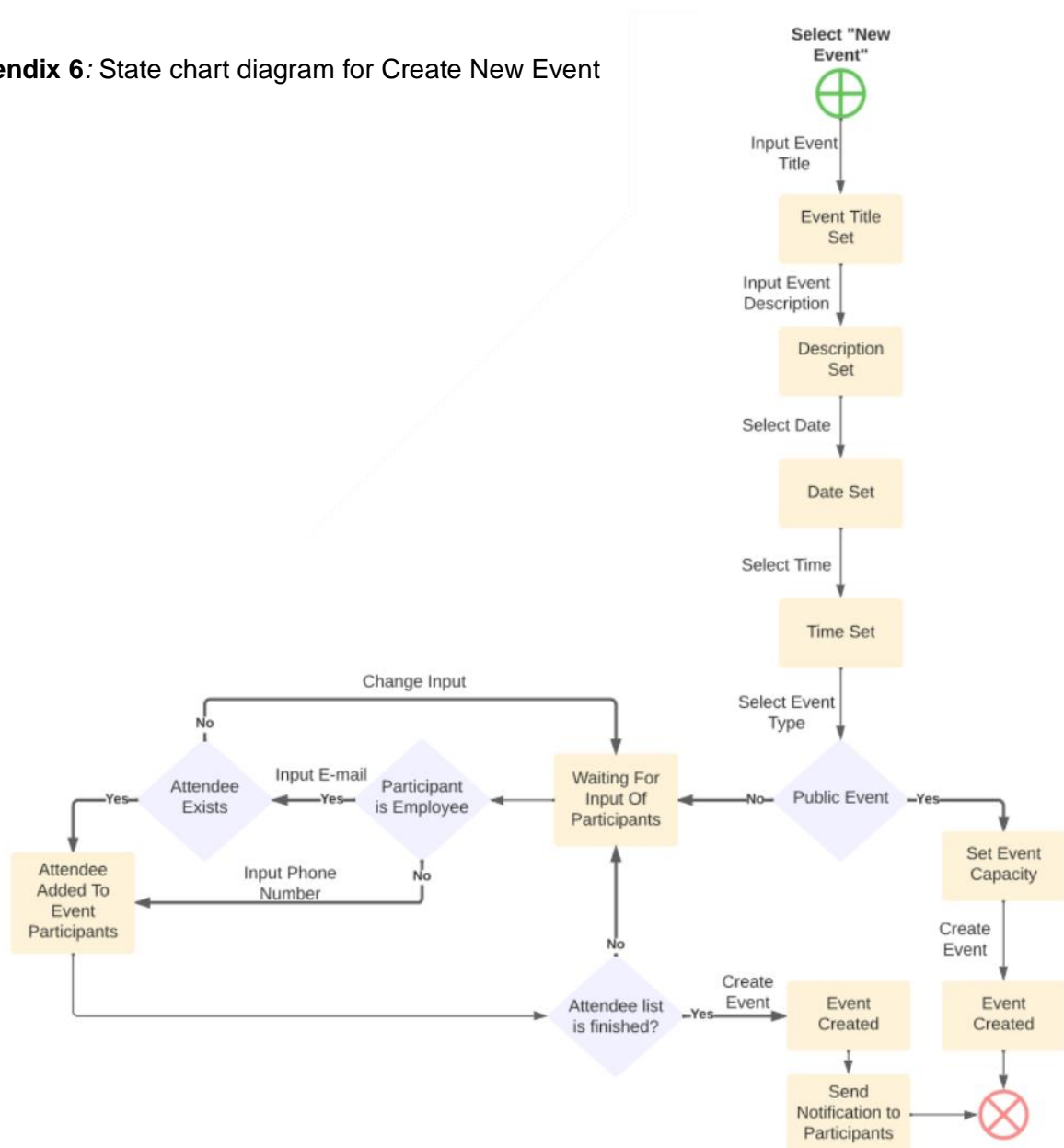
Appendix 4: Rich Picture



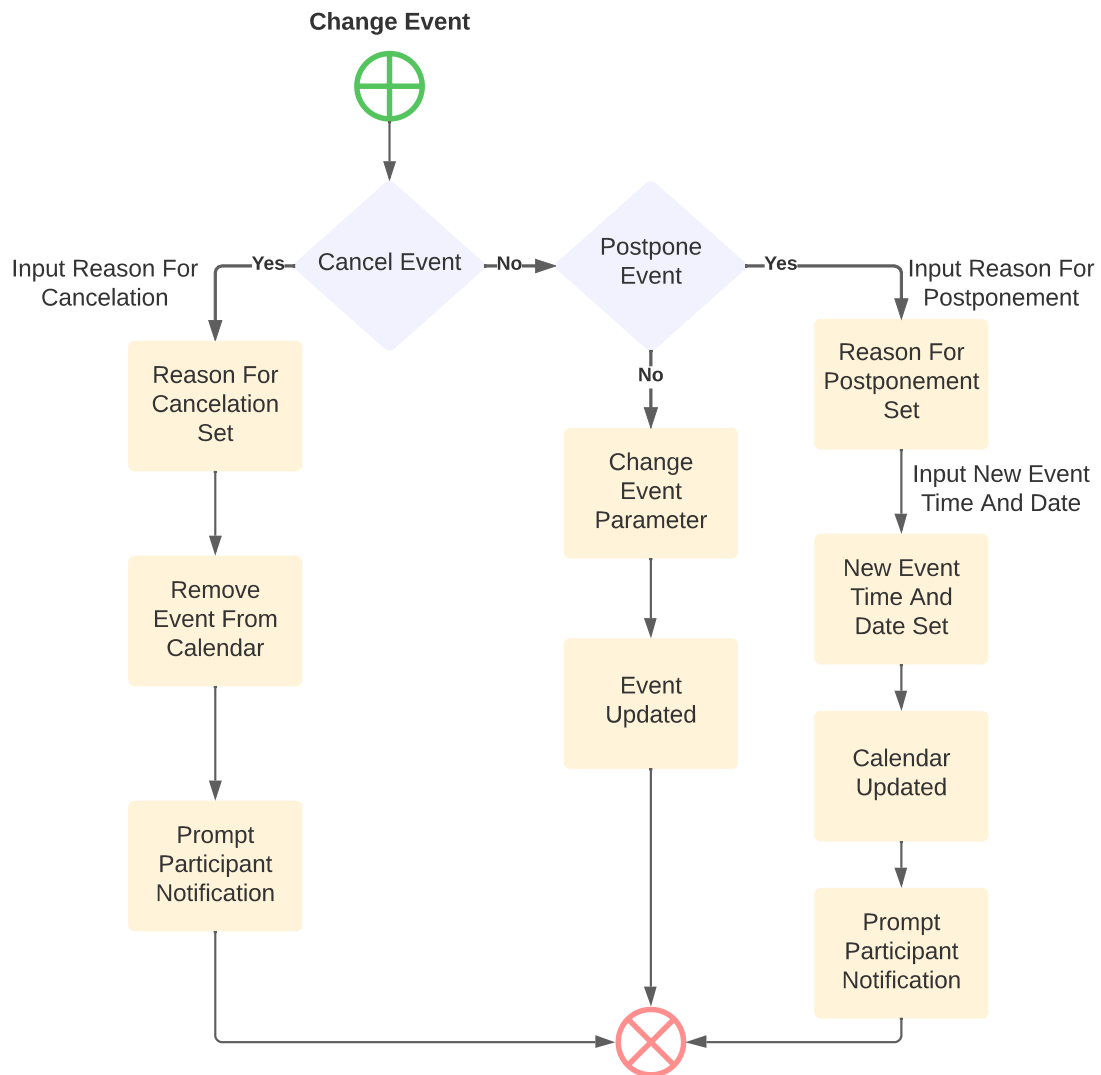
Appendix 5: Event Table

Events	Facility	Employee	Calendar	Company	Inhouse event	Guest	Appointment	Information	Amenity
Create event	x	x	x	x	x				
Cancel event	x	x	x	x	x				
View event	x	x	x	x	x	x			
Accept event	x	x	x	x	x	x			
Edit event	x	x	x	x	x				
Create appointment	x	x	x				x		
Edit appointment	x	x	x				x		
Accept appointment	x	x	x			x	x		
Cancel appointment	x	x	x				x		
View appointment	x	x	x			x	x		
Decline appointment	x	x	x			x	x		
Edit information	x			x	x			x	x
Remove information	x			x	x			x	x
View information	x	x		x	x	x		x	x
Visit facility	x	x		x		x			

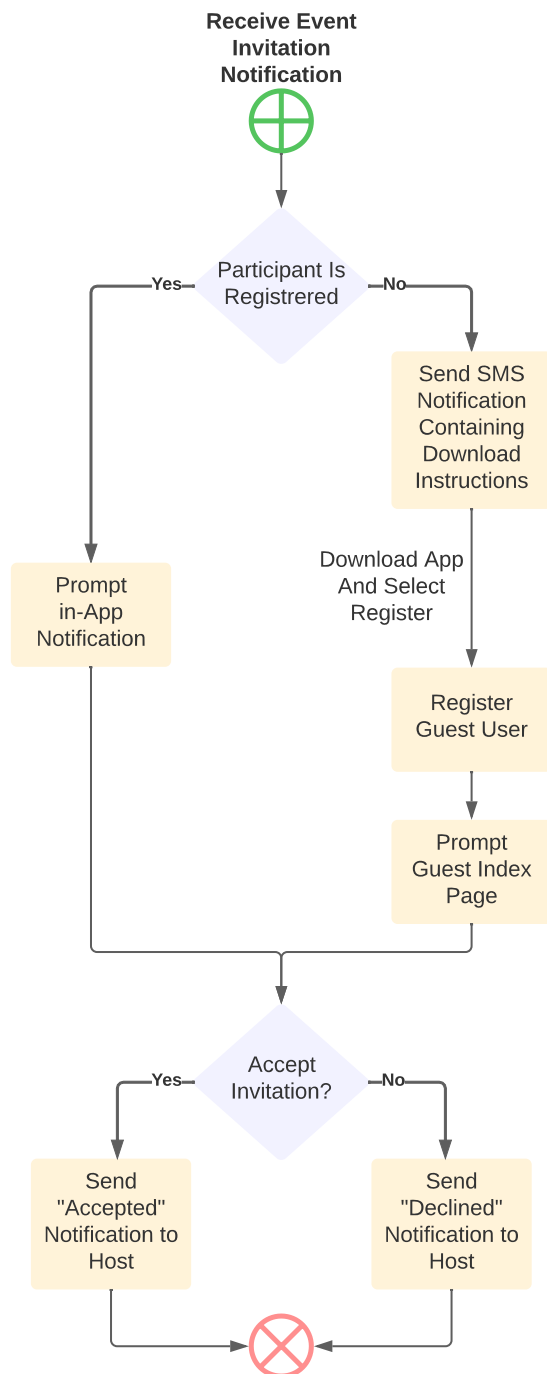
Appendix 6: State chart diagram for Create New Event



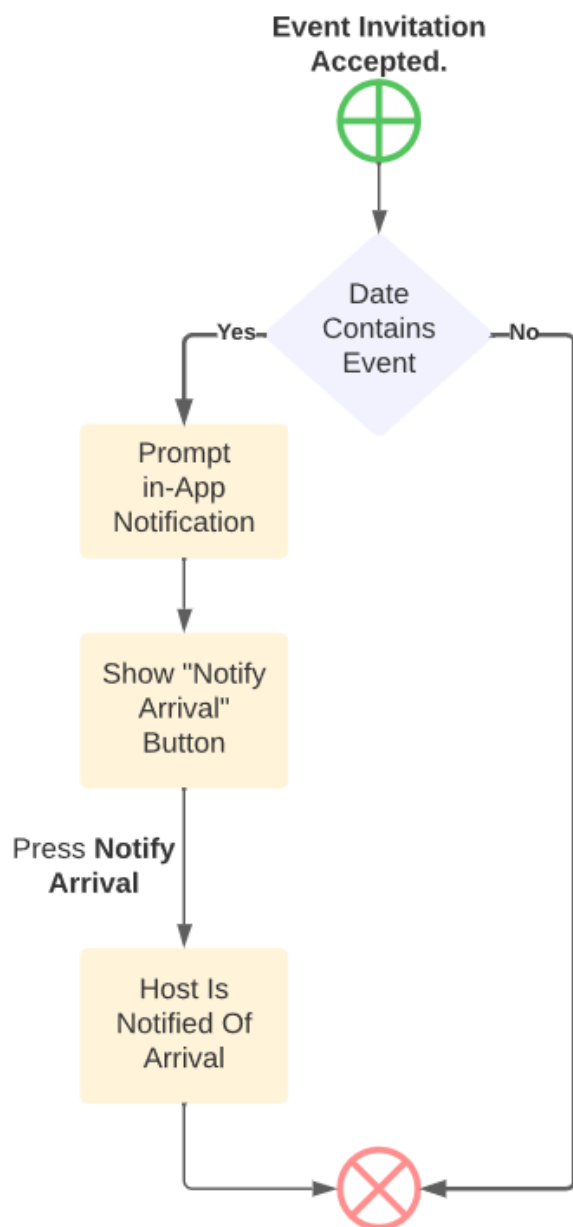
Appendix 7: State chart diagram for Change Event



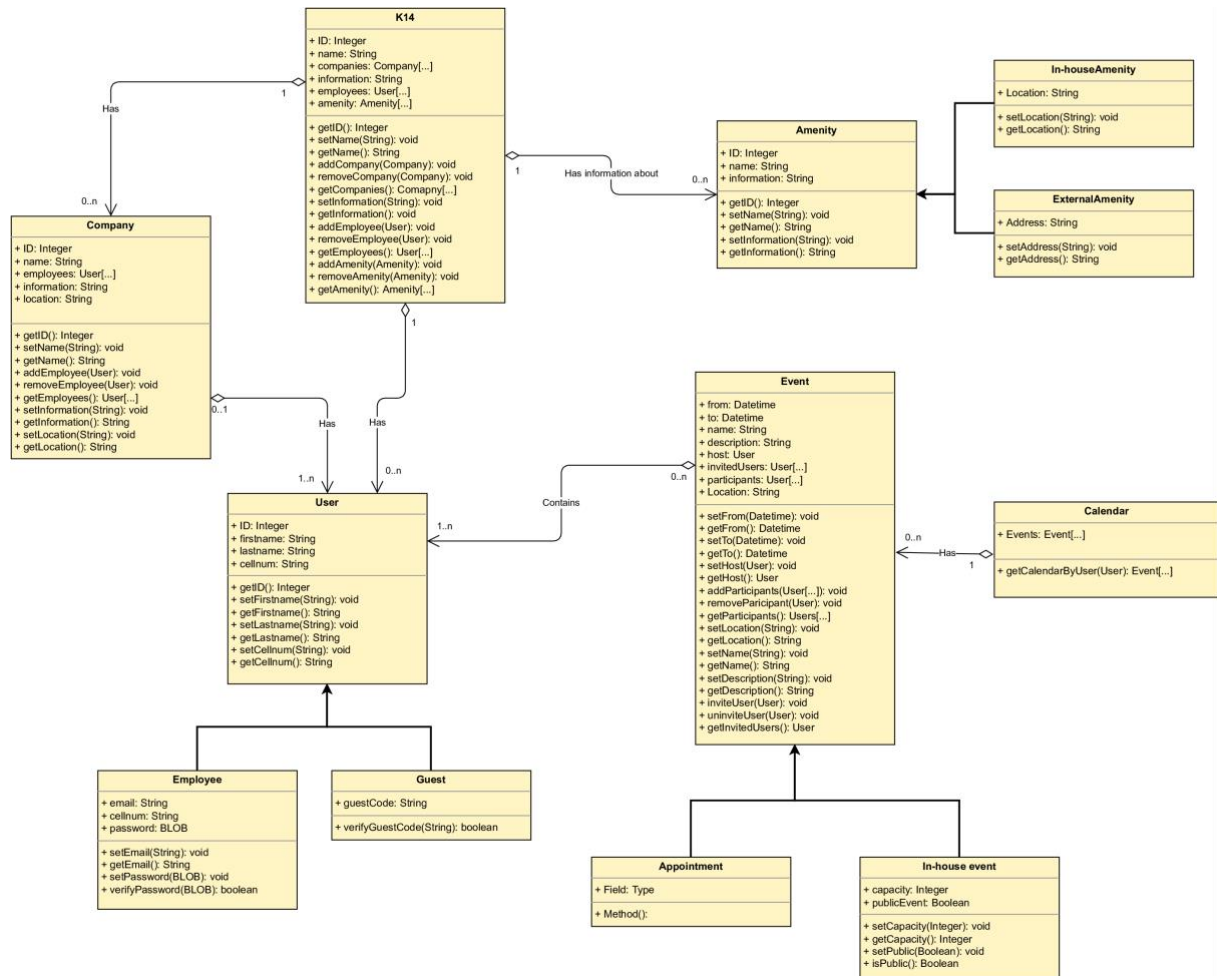
Appendix 8: State chart diagram for Event Invitation



Appendix 9: State chart diagram for Notification of Arrival



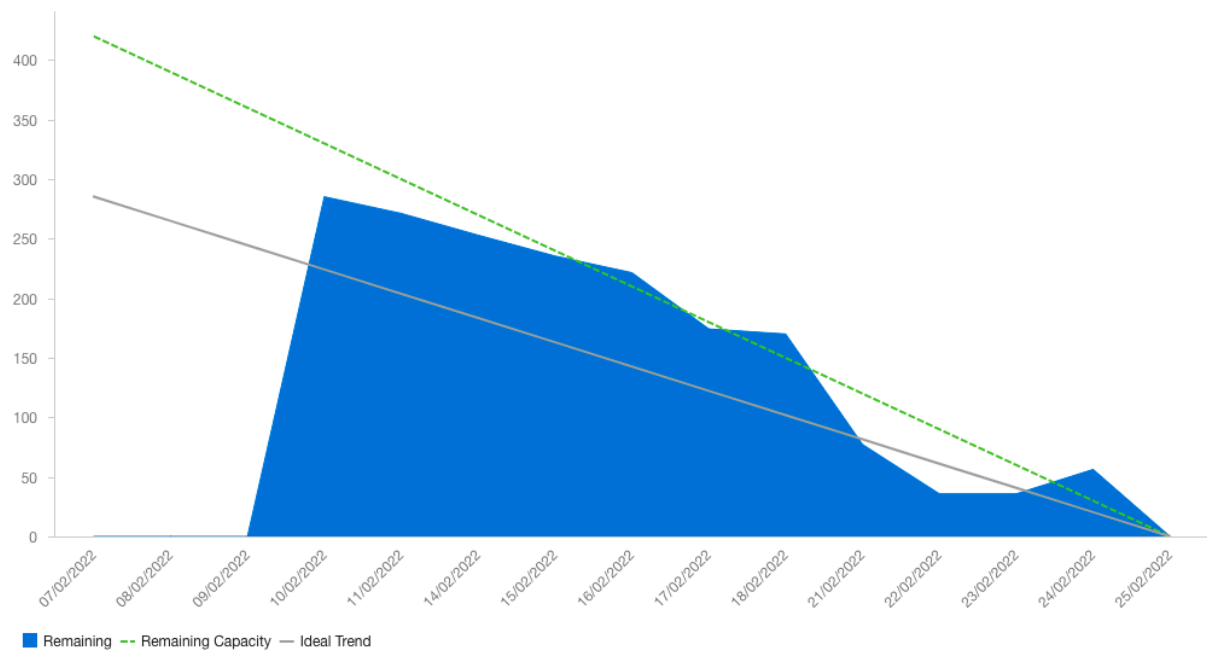
Appendix 10: UML Class Diagram



Appendix 11: David Benyon's 12 Design Principles

1	Visibility	Visibility is about ensuring that things are visible for the user, this includes available functions and what the system is doing.
2	Consistency	Consistency is both about consistency with the usage of design features and consistency to similar systems.
3	Familiarity	Familiarity is using symbols, language, and design features that the users are familiar with.
4	Affordance	Affordance is designing things so that the users know what their purpose is.
5	Navigation	Navigation is about providing the users with support to move around the system.
6	Control	Control is about the clarity of who and what is in control, allowing the user to take control.
7	Feedback	Feedback is about letting the user know what is happening and what their action results in.
8	Recovery	Recovery is about providing the user with ways to recover from their actions.
9	Constraints	Constraints are about preventing users from making serious errors.
10	Flexibility	Flexibility is about providing the user with multiple ways of doing things, to ensure usability regardless of experience and disabilities.
11	Style	Style is about designing a system that is stylish and attractive.
12	Conviviality	Conviviality is about creating a system which is polite, friendly, and pleasant.

Appendix 12: Azure Burndown Sheet



Gruppekontrakt

Følgende medlemmer:

Fredriksen, S-N, B., Gulbrandsen, H., Hurum, J., Meisingset, S., Solvang, R., & Sørensen, S.

er herved samstemt og godtar kriteriene, reglene og forventningene beskrevet nedenfor i gjeldende fag med tilhørende prosjekter:

- **IS-304 Bacheloroppgave i informasjonssystemer**
- **IS-305 Aktuelle IT-relaterte tema, bærekraft og digitalisering**

I. Bakgrunn og motivasjon

Vi ønsker å lære om, og mestre fagets læremål, slik de er oppgitt på Canvas.

II. Bestemmelser om fremmøte og samarbeid

Jeg forplikter meg overfor gruppen min, til at jeg vil:

- Følge forelesninger, og delta i gruppearbeid, samlinger og øvelser, be om/ta imot veiledning og selvstendig studere relevant faglitteratur.
- Arbeid som ikke møter gruppens krav til kvalitet og innhold skal under bred enighet i gruppen endres eller gjennomføres på nytt.
- Overholde kravet om 80% frammøte ved forelesning og også oppgi grunn hvis jeg er forhindret.
- Møte opp innen 5. minutt fra planlagt tid. Dersom dette unnviker uten gyldig grunn må en bot på 50 kroner betales til felleskasse. Denne felleskassen besittes av Stian Meisingset, og skal føres i et regneark som finnes i driven. Summen av bøtene skal benyttes til en hyggelig aktivitet der hele gruppen deltar i mai/ juni. Dersom aktivitet ikke blir gjennomført, distribueres pengene likt på alle i gruppen. Gruppen avgjør i fellesskap i det enkelte tilfellet om årsak til forsentkomming er gyldig.
- Overholde frister og levere oppgaver som publiseres på Canvas, innen tiden.
- Overholde frister og levere oppgaver i henhold til prosjekteiers krav.
- Sørge for at arbeidet fordeles jevnt mellom gruppemedlemmer, men samtidig utnytte hver enkelt students spesielle ferdigheter og bakgrunn.
- Spørre om hjelp eller tilby å arbeide med oppgaver dersom ferdig med egne oppgaver, eller ser andre står fast.

- Ta ansvar for eget arbeid, ved aktiviteter som krever hele gruppens oppmerksomhet avgjør fellesskapet hvordan tiden styres.
- Forplikte meg til å holde meg oppdatert på gjeldende informasjon på samtlige kommunikasjonsplattformer, heretter spesielt *Messenger* og *Teams*.
- Dersom mangelfullt arbeid og innsats kan gruppen fatte enighet om gi en advarsel til vedkommende. Ved mer enn 1 advarsel, blir man kastet ut av gruppen og mister rettighetene til arbeidet som er utført.
- Dersom man forlater gruppen, mister vedkommende rettighetene til arbeidet som er utført.
- Alle i gruppen er forbeholdt taushetsplikt ved både arbeidsmessig og privat informasjon som måtte komme frem under samarbeidet i gruppa

Evt. konflikter rundt samarbeid og innsats søkes løst gjennom diskusjon i gruppen. Fører ikke dette frem, kontaktes veileder, så snart som mulig, og senest før innleveringsfrister.

III. Andre bestemmelser

Ved uenighet eller diskusjon av en lengde som passer til temaets viktighet, avveid mot behovet for å ta en beslutning og komme videre i prosjektet, tas det en omforent beslutning.

IV. Korona

Man skal følge lovpålagte regler og anbefalinger fra helsemyndighetene ved korona, av hensyn til gruppens beste velgående og gjennomføring av arbeidet.

V. Konvensjoner

Det stilles krav om minimum 25 timers arbeid i uken til hver av medlemmene. Arbeidet skal dokumenteres gjennom Azure DevOps.

Gruppen innehar bred enighet om at det skal være fritak fra arbeid i helger hvis mulig. Ved motstridende enighet bortfaller dette.

Appendix 14: Interview Guide

GRUPPE 17 – Knowit BACHELOR VÅR 2022

INTERVJUGUIDE FOR AKTUELLE BRUKERE

Intervjudetaljer

Navn på kandidat		Kandidat-ID	
Dato		Tid start - slutt	
Rolle		Bedrift	

Spørsmål

Svar

Har du vært et sted eller i et bygg som hadde resepsjon? (Hvor?)	
Hva gjorde resepsjonen som du var fornøyd med?	
Var det noe med resepsjonen du var misfornøyd med, eller som resepsjonisten kunne gjort bedre?	
Tror du en resepsjon på arbeidsplassen kunne gjort arbeidsdagen enklere?	
	Hvis ja: Hvorfor?
	Hvis nei: Hvorfor ikke? Hva skal til for at en resepsjon skal gjøre arbeidsdagen enklere).
Er du / dere kjent med fasilitetene K14 tilbyr?	
Hvis / Når dere flytter til K14 bygget hvilke tjenester ønsker du at resepsjonen skal tilby / kunne hjelpe deg med?	
Er det noen av disse tjenestene du kunne tenke deg vært i en app?	
Er det noen arbeidsoppgaver du skulle helst ha en resepsjonist for, altså fysisk person?	

I en perfekt verden, hvilke funksjoner / muligheter bør en resepsjonsapplikasjon på arbeidsplassen ha?	
Er det andre funksjonaliteter som du mener ville vært hensiktsmessig å ha med i en slik app?	
Er det noe du umiddelbart tenker kunne vært irriterende eller distraherende med en slik applikasjon?	
KOMMENTARER Eventuelle tilleggskommentarer for å støtte intervjufunnene	
INTERVJUANSVARLIG	OBSERVATØR
REFERENT	TOTAL TID

Appendix 15: Compiled Interview Guide

GROUP 17 – Knowit BACHELOR SPRING 2022

Assembled interview answers

Question	Answer
Have you visited a place or building that has had a reception?	<ul style="list-style-type: none"> - Yes <ul style="list-style-type: none"> o Hotels o Workplaces o Commercial buildings
What aspects of this reception were you happy with?	<ul style="list-style-type: none"> - Meeting a smile when you walk into the building - Good service - Help with navigation, booking meeting rooms - All necessary information is available when you need it - Immediate help - Easy registering when guests arrive or when you arrive as a guest - Increases the feeling of security in the building as the receptionist helps filter the people who should have access to the building and who shouldn't
Were there any aspects you were unhappy with, or that the receptionist could've done better?	<ul style="list-style-type: none"> - When you arrive at a reception, and they aren't able to help you with all services i.e., registering / paying for parking - When the receptionist isn't service-minded - Wasn't able to get an answer on something they needed - An empty reception, when the receptionist is in the back room or not around

		<ul style="list-style-type: none"> - Renting of electric cars, bikes, and scooters - Coworking areas
If/when you company moves in to K14, what services would you like the reception to offer / help you with?		<ul style="list-style-type: none"> - Calendar which includes events, seminars, presentations that different companies will be holding - Functionality for ordering food for meetings - General information about the building for guests - Information about the different companies and projects in the building to allow for networking opportunities - Renting bikes, scooters, carpool and seeing their availability - Booking meeting rooms - Cleaning after meetings - Notify tenants when a guest has arrived - Able to request maintenance, both immediately and for non-essential/critical problems
Are there any of these services you could imagine being in an app?		<ul style="list-style-type: none"> - Meeting room booking - Renting electric bikes - Deviation service - Contact maintenance, caretaker/janitor, technical help - Ordering food for meetings - Ordering extra office if needed one day - Cleaning after meetings - Calendar - Information about who is renting in the building and where are they located - Information about what is happening that day - Ordering
If your place of work offered a reception/receptionist, would it make your work life easier?		
	If yes: Why?	<ul style="list-style-type: none"> - Navigating guests to where they need to be, can be time consuming - Allow guests to book appointments - Cleaning after meetings, make sure the coffee machine is working, welcoming guests - Notify tenants when their guests have arrived - Not pleasant entering a building or office space without being welcomed - Better experience for guests
	If no: Why not? What would the receptionist need to offer to make your work life easier?	<ul style="list-style-type: none"> - Don't have guests often - Worked in a building without a receptionist until now, a receptionist isn't necessary for their day-to-day work life
Are you familiar with the facilities K14 will offer?		<ul style="list-style-type: none"> - One reception for all building tenants - Conference possibilities - Meeting rooms - Cantine

	<ul style="list-style-type: none"> - Prepare the meeting rooms and clean after - Booking meeting rooms - Booking taxi, trains, bus tickets from the application - Calendar - Notifications for what is happening in the building - Internal chat - Register access for guests, so they don't need to book an appointment
Is there any other functionality that could be convenient in a reception app?	<ul style="list-style-type: none"> - List of phone numbers for people in the building - Payment for food in the app
Are there any elements in this application you could imagine being annoying or distracting?	<ul style="list-style-type: none"> - Bad design - If you're dependent on the application for everything - Too many steps for completing a process - Shouldn't be necessary for guests to download the app to visit someone in the building - Software updates all the time - If it's difficult to use, or not get any use out of it - Technical problems - Overflow of information - If it's not reliable - Too many notifications

COMMENTS or additional comments

- Updates and newsletters in the application
- Cantine menu
- Environmental reports, for example for recycling, CO2 emissions, food waste. Make people more aware of their actions
- Controlling air conditioning, lights, blinds

Appendix 16: Must-Have User Stories

Title	Description
📖 Overview of companies in the building	As a guest or tenant, I want an overview of which companies are located in the building, so that I can locate the company I'm lookin...
📖 Information about K14	As a visitor, guest or employee I want to be able to access general information about the K14 project and building, so that I can bec...
📖 See available services	As a guest, I want to see the services available to me in the K14 building such as restaurants, stores etc, so that I know which facilitie...
📖 Contact personnel	As a user of the building, I want to be able to ask the personnel questions the application cannot answer, so that they can help me s...
📖 Calendar customization	As a building tenant, I would like to add or delete things in my calendar, so that I can keep track of my daily and upcoming schedule
📖 Create an event	As a building tenant, I want to create a new event, so that other people can see it in the list/calendar of events
📖 View calendar	As a K14 tenant, I want a calendar overview of upcoming events hosted by the different companies of the building, so that I know w...
📖 Log out	As a building tenant, I want to be able to logout of the system, so that I don't have access to the system services.
📖 Log in via username and password	As a K14 tenant, I want to login to the system, so that I get access to the systems services.
📖 Change password	As a K14 tenant, I want to be able to change my password, so that I can access to systems services even though I forget my passwor...
📖 Log in as guest	As a guest, I want to be able to log in as a guest, so that I do not need to create a user for visiting the building.
📖 Register a user	As a building tenant, I want to be able to register an account, to be able to gain access to the systems services
📖 Test coverage	As a system developer, I want at least 80% test coverage for my code so that I can improve the quality of the project
📖 View index page	
📖 Application Routing	...

Appendix 17: Color Palette for the Design

K14 Color Palette

Color palette based on Aaron Marcus' five-color design rules



French Rose
#f5498e
rgb (245, 73, 142)



Carnation
#f26216
rgb (242, 97, 97)



Yellow Green
#b8e18
rgb (184, 25, 128)



Asparagus
#f5498e
rgb (131, 161, 88)



Cornflower Blue
#6162f2
rgb (97, 98, 242)

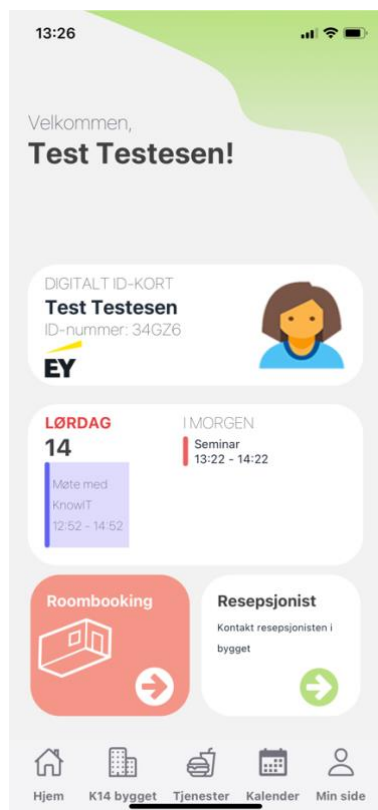


Ebony Claw
#262c36
rgb (38, 44, 54)

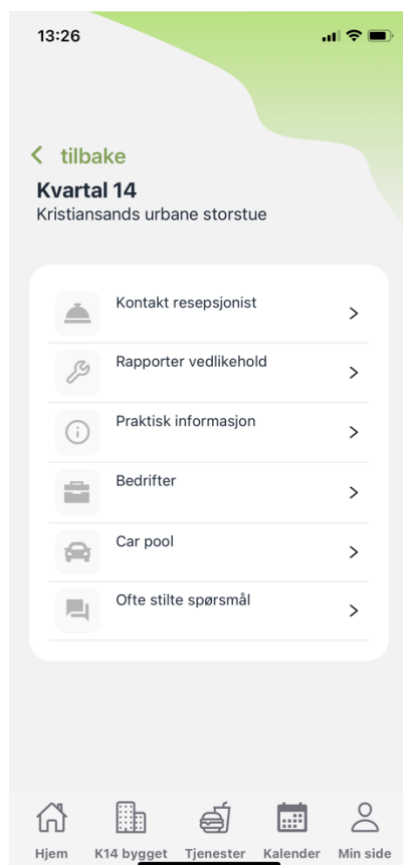
Appendix 18: Aaron Marcus' Color Conventions

Red	Danger, hot, fire
Yellow	Caution, slow, test
Green	Go, okay, clear, vegetation, safety
Blue	Cold, water, calm, sky
Warm colors	Action, response required, proximity
Cool colors	Status, background information, distance
Greys, white and blue	Neutrality

Appendix 19: Landing Page



Appendix 20: K14 building page



Appendix 21: Calendar page



Appendix 22: Contact personnel page



Appendix 23: K14 information page



Appendix 21: The Team's Code Conventions

Code Conventions

This document contains instructions and code conventions to be applied in the K14 project.

Why use code conventions?

40% - 80% of the total cost of software is spent on its maintenance

Software is almost never fully supported by its original author

Coding standards improve software readability by allowing developers to understand new code faster and better

Like any other product, the software must be "well packaged" and clean

When selecting tasks

The Sprint will have defined tasks that can be found in the backlog as cards. Most cards or tasks are based on user stories with associated acceptance criterias. When selecting tasks, pick them from the most left column "not started".

When selecting a task, move the chosen task over to the column displaying active tasks to ensure that everyone knows it has been started and by whom.

Use the comment field in your chosen task actively by noting the amount of time spent working on it, and update the "Effort" which includes time spent on completing it and remaining time.

When the task is resolved (and fulfilled Definition of Done) and presents actual functionality or production, move the task over to the column to the far right, "closed".

Definition of Done

A task can be considered done when the following requirements are met:

- Produced code solves the needs of the user story
- The source code have been documented accordingly to the defined criterias found in the code convention
- Backend compiles and frontend runs
- Existing features have not been altered i a way that ruins its functionality
- Tasks are RESOLVED until the agreed upon test coverage has been reached and the merge request has been approved.
- Tasks are CLOSED when all criteria of the task/userstory/feature has been fulfilled and test coverage is satisfactory.

Conventions

Backend

- Controller-Service-Repository pattern
- Controller - Should exclusively manage the REST interface (endpoints) to the business logic.
- Service - Business logic implementation
- Repository - Storing and retrieving sets of data

Frontend

- One folder per feature - src/pages/companies/details
- General / re-usable components in root/components folder
- Files should be placed where it belongs
- Styling for a feature should be located in the feature folder.
- Filestructure should fit routing.
- Assets folder should contain fonts and images only.

Naming

- A class, method or function name should be descriptive of what it does, handles or relates to.
- Readability in mind.
- Camelcase is to be considered best practice.
- Class: CompanyService
- Method: getCompanyName

Version Control

- Rebasing should be used as the main method of merging branches.
- New branches should be named as shown below:
k14_[task number]_[task name]
- Main should at all times contain tested, runnable code.
- DEV branch should be created during sprint planning and should be merged with MAIN during internal reviews.
- All new features during a sprint should branch out of DEV and merged back via rebasing during a sprint.
- Merge requests are to be reviewed and approved by a party not involved in the development of the feature of said MR.

Testing

- Project requirement: Test coverage of at least 80%
- Code merged with main should at all times be in compliance with the above mentioned criteria.
- If possible features should be fully tested when requesting approval of merge requests.

Statement from Knowit Sør

Knowit Sør assigned the bachelor project team consisting of Rikke Solvang, Henrik Glosli Gulbrandsen, Sven Sørensen, Jenny Hurum, Sindre-Nicolai Barvik Fredriksen and Stian Meisingset. As we have experienced it, they have delivered in a timely and agile manner.

Trym Staurheim and Geir Morten Hagen followed the team though out the project with a minimum of bi-weekly sprint demos and conversations of the application.

The project

The assignment was to come up with a prototype solution for our customer K14. K14 needs an app to fully automate their upcoming company housing. It involved a phase of data collection and interviews with both tenants and the owners of K14, and a phase of developing the receptionist module for mobile. They were to use our inhouse stack throughout the toolchain, entailing Azure DevOps for project management, Azure Cloud for deployment and our inhouse standard development stack consisting of Spring/Java, React/TypeScript, and MSSQL.

As we see it, these are most prominent areas where they excelled.

- The team uncovered already existing applications that could solve the customers problem.
- They collected valuable data from tenants and owners of K14 and were eager to share ideas and prospects for a prototype in the earlier phases of the project.

- The team acted autonomous and self-organized, they continuously improved their agile/scrum method and divided the team into appropriate roles to take advantage of their broad skillsets.
- They were to use our inhouse stack which contained languages, frameworks and systems which was new to them. This was something they quickly took advantage of rather than stumbled on.
- They have delivered a solution of acceptable quality, focusing on delivering sub-modules of the system completely, rather than delivering half-finished sub-modules.