Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Architecture and Design
Department of Design

Kristian Wobbes, Jonas Lillebø Haugen, Adrian Nysted Riise

Navigating Preterm Parenthood

A web-based solution for supporting parents of premature infants

Bachelor's thesis in Web Development
Supervisor: Eleftherios Papachristos
May 2023

NTNU
Norwegian University of Science and Technology

Kristian Wobbes, Jonas Lillebø Haugen, Adrian Nysted Riise

Navigating Preterm Parenthood

A web-based solution for supporting parents of premature infants

Bachelor's thesis in Web Development
Supervisor: Eleftherios Papachristos
May 2023

Norwegian University of Science and Technology
Faculty of Architecture and Design
Department of Design

NTNU
Norwegian University of
Science and Technology

NAVIGATING PRETERM PARENTHOOD

A web-based solution for supporting parents of premature infants

Kristian Wobbes, Adrian Nysted Riise, Jonas Lillebø Haugen

NTNU Gjøvik | Institute for Design

Foreword

In our journey towards obtaining our bachelor's degree, we explored the field of web development. This thesis represents our combined efforts, dedication, and enthusiasm for user-focused solutions.

We express our gratitude to our advisor, Eleftherios Papachristos, and extend our appreciation to Oslo Universitetssykehus, Kenneth Strømmen, and Tom Stiris for providing us with this opportunity.

This thesis presents the exploration, design, and development processes behind our web application. Our group aimed to develop a solution meeting the needs and expectations of the target audiences.

Adrian N. Riise
Adrian Nysted Riise

Kristian Wobbes
Kristian Wobbes

Jonas L. Haugen
Jonas Lillebø Haugen

Abstract

Title: Navigating Preterm Parenthood: A Web-Based Solution for Supporting Parents of Premature Infants

Date: 15.05.2023

Participants: Adrian Nysted Riise, Jonas Lillebø Haugen, Kristian Wobbes

Supervisor: Eleftherios Papachristos

Employer: Oslo University Hospital

Subject: Web development

Keywords: Web development, web application, design, development

Number of pages: 80 + 40

Number of words: 14 641

Number of attachments: 17

This bachelor thesis investigates the challenge of providing up-to-date, customized, and accurate information about premature infancy to support parents of premature infants. We developed an online platform using a user-centered design approach, incorporating relevant web development techniques, and creating a custom Content Management System (CMS) for healthcare providers. By incorporating user research and leveraging modern web technologies, we addressed the challenges faced by parents and providers in accessing and managing information about premature infancy. Our user-centered approach aims to ensure a seamless user experience and the custom CMS allows for efficient content management. Upon evaluation, we identified areas for improvement and provided recommendations for further development.

Abstract (Norwegian)

Tittel: Veiledning for foreldre til for tidlig fødte: En nettbasert løsning for støtte til foreldre av premature spedbarn

Dato: 15.05.2023

Deltagere: Adrian Nysted Riise, Jonas Lillebø Haugen, Kristian Wobbes

Veileder: Eleftherios Papachristos

Oppdragsgiver: Oslo Universitetssykehus

Fag: Webutvikling

Stikkord: Webutvikling, webapplikasjon, design, utvikling

Antall sider: 80 + 40

Antall ord: 14 641

Antall vedlegg: 17

Denne bacheloroppgaven undersøker utfordringen med å tilby oppdatert, tilpasset og nøyaktig informasjon om for tidlig fødte barn for å støtte foreldre til premature barn. Vi utviklet en nettbasert plattform ved å bruke brukersentrerte designmetoder, relevante webutviklingsteknikker og et innholdsstyringssystem (CMS) for helsepersonell. Ved å inkludere brukerundersøkelser og utnytte moderne webteknologier, håndterte vi utfordringene foreldre og tilbydere står overfor når det gjelder å få tilgang til og administrere informasjon om for tidlig fødte barn. Vår brukersentrerte tilnærming sikter mot å sikre en sømløs brukeropplevelse, og det tilpassede CMS-et tillot effektiv innholdsadministrasjon. Etter evaluering identifiserte vi områder for forbedring og ga anbefalinger for videreutvikling.

Table of contents

1 Introduction

This thesis is a closing bachelor project in web development at the Institute for Design at NTNU Gjøvik. The main contributions to this thesis are two web applications aiming to provide information about prematurely born children. We developed a user-facing MERN application along with a headless MEVN Content Management System to support medical professionals in maintaining the application.

1.1 Current solutions

The challenges faced by premature infants and their families are addressed by a combination of efforts from healthcare institutions and informational resources. Oslo Universitetssykehus and Norsk Helseinformatikk are two key players in providing information and resources in this domain.

1.1.1 Oslo Universitetssykehus

Oslo Universitetssykehus provides comprehensive information on premature babies on their website, covering aspects such as diagnosis, treatment, follow-up care, and practical guidance for parents. The hospital emphasizes a family-centered care approach and offers details about interdisciplinary collaboration to support premature infants and their families (Oslo Universitetssykehus, 2020).

1.1.2 Norsk helseinformatikk

Norsk helseinformatikk offers general information about premature babies, including causes, treatments, and support organizations. The website discusses potential consequences of prematurity, treatment strategies, long-term complications, and patient organizations like Prematurforeningen (Norsk Helseinformatikk, 2021).

Both Oslo Universitetssykehus and Norsk Helseinformatikk contribute to addressing the challenges faced by premature infants and their families by providing valuable information and resources. Oslo Universitetssykehus focuses on medical care and support, while Norsk helseinformatikk offers a broader overview of prematurity, including causes, treatments, and potential long-term consequences.

1.1.3 Project owner concerns about today's solutions

As per today, the information available to parents of prematurely born children is limited and difficult to find. Information displayed at Oslo Universitetssykehus (Oslo Universitetssykehus, 2020) own webpage lacks content and interactivity. To find information, the user will have to navigate through various pages and dropdown menus, making it difficult to reach information related to the user.

1.2 Problem statement

This thesis addresses the problem of:

> "How can we develop an online platform to provide up to date, customized, and accurate information about premature infancy to support parents of premature infants?",

and presents a solution through the research and development of a web application that leverages modern technologies and practices. To tackle this issue, the research is guided by the following questions:

1. *How can web development techniques be applied to create a user-friendly interface that addresses the specific needs of parents during these critical stages of their parenting journey?*
2. *How can we use web technologies to develop a solution that both answers the client's request and fits the user's needs?*
3. *How can we allow the healthcare providers to efficiently update and disseminate[1] information relevant to parents during the hospital stay and homecare period?*

[1] Spread (information) widely.

1.3 Project scope

1. **Minimum Viable Product:** The final product in this thesis will be developed to meet the criteria as a Minimum Viable Product (MVP).
2. **WCAG:** The Web Content Accessibility Guidelines (WCAG) are a vital part of developing a web application. They were followed during the development of this project; however, this thesis will not cover them in detail.
3. **Deployment:** The product will be deployed only for user testing and to showcase the product to the product owner, further deployment and performance testing will have to be conducted by the product owner.
4. **Provide an application according to product owner wishes:** Our objective is to create an application that effectively addresses the product owner's requirements. To achieve this, we will conduct comprehensive user research and user testing to not only validate but also refine the product owner's suggestions. This approach will guarantee a high level of usability, ultimately resulting in a well-received application.
5. **Content creation:** The thesis will focus on the structure and navigation within the main website rather than the content itself within the articles. Example content will be generated for user testing and showcasing purposes.

2 Exploration

In the Exploration chapter, we investigate the current landscape of the problem domain, focusing on understanding the needs and preferences of the target audience. By conducting user research, such as interviews, affinity mapping, and personas development, we establish a foundation for our design decisions. Additionally, we perform a competitor analysis to obtain inspiration from other websites and uncover opportunities for our solution to stand out. The insights gained in this chapter serve as a crucial starting point for the Design phase.

2.1 User research

To ensure the development of a quality product, it is crucial to identify users early, enabling the collection of meaningful and relevant information through user research (Baxter, Courage, & Caine, 2015, pp. 32-40).

To gain a thorough understanding of our objectives, we held discussions with the project owner to obtain valuable insights into their vision for our project. During these sessions, we discovered that, over the past two years, multiple teams had been working towards creating content for a website targeting parents of premature children.

At the beginning of this phase, we faced numerous possibilities, prompting us to develop a structured project plan. This plan involved reviewing the existing website content and conducting further user research through semi-structured interviews. Our aim for this phase was to better understand the needs and pain points of parents of premature infants and other stakeholders.

2.1.1 Product owner’s needs

The healthcare project team that the product owner belongs to aims to create a website for expectant parents, current parents, and others seeking information about various aspects related to premature babies. The websites offered by Oslo University Hospital are unsuitable due to their limitations.

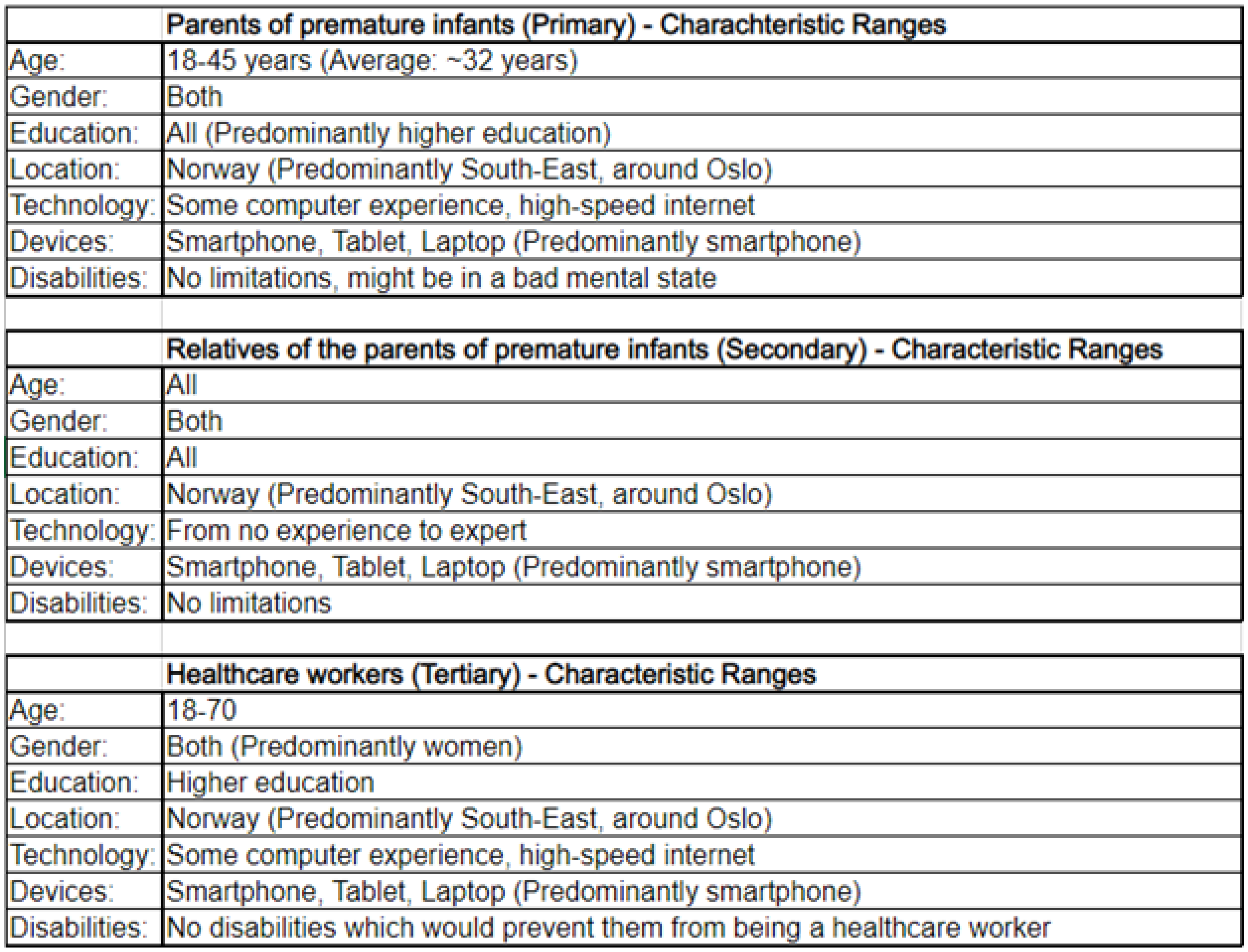
The healthcare project team requires assistance and came with the following suggestions:

1. An X-axis displaying a timeline from before and after birth, as well as before and after admission to the neonatal intensive care unit.
2. A Y-axis with an interactive image of a body, where users can "click" on various body parts, such as the heart, to be directed to a separate page containing information about relevant medical conditions.
3. An interactive service for parents, allowing them to receive tailored information based on their child's progress. For instance, if their child was born three months early and is now in their sixth week of life, the service would provide insights into what parents can expect during the current week and the following week.

2.1.2 User profiles

In our efforts to understand our target audience, we created user profiles that categorize users into primary, secondary, and tertiary segments *(Table 1 - User Profiles* that explain the different features of primary, secondary, and tertiary users in our study. Each profile is described in more detail, helping us better understand the people involved.*).* This process was iterative, meaning that as we gained further insights through user research, we were able to refine and update our user profiles accordingly.

Following discussions with our product owner, the team recognized significant data points including age, location, and potential disabilities. Through our interviews detailed in *section 2.1.3*, we managed to collect valuable data points such as education level, technology proficiency, and the specific devices being utilized.



	Parents of premature infants (Primary) - Charachteristic Ranges
Age:	18-45 years (Average: ~32 years)
Gender:	Both
Education:	All (Predominantly higher education)
Location:	Norway (Predominantly South-East, around Oslo)
Technology:	Some computer experience, high-speed internet
Devices:	Smartphone, Tablet, Laptop (Predominantly smartphone)
Disabilities:	No limitations, might be in a bad mental state

	Relatives of the parents of premature infants (Secondary) - Characteristic Ranges
Age:	All
Gender:	Both
Education:	All
Location:	Norway (Predominantly South-East, around Oslo)
Technology:	From no experience to expert
Devices:	Smartphone, Tablet, Laptop (Predominantly smartphone)
Disabilities:	No limitations

	Healthcare workers (Tertiary) - Characteristic Ranges
Age:	18-70
Gender:	Both (Predominantly women)
Education:	Higher education
Location:	Norway (Predominantly South-East, around Oslo)
Technology:	Some computer experience, high-speed internet
Devices:	Smartphone, Tablet, Laptop (Predominantly smartphone)
Disabilities:	No disabilities which would prevent them from being a healthcare worker

Table 1 - User Profiles that explain the different features of primary, secondary, and tertiary users in our study. Each profile is described in more detail, helping us better understand the people involved.

2.1.3 Interviews

“In the broadest sense, an interview is a guided conversation in which one person seeks information from another” (Baxter, Courage, & Caine, 2015, p. 220). Interviews are a design method used to gather information and insights from various stakeholders involved in a project or problem. They can be conducted in various formats, including structured, semi-structured, or completely unstructured interviews, depending on the goals and needs of the research. Interviews are flexible and can be used in conjunction with other design methods, such as personas, to gain a deeper understanding of users' needs, preferences, and motivations.

Since the nature of the interview was to gain a deeper understanding of what the underlying problems were, the group opted for semi-structured interviews. These interviews were conducted with one couple admitted to the neonatal intensive care unit at Rikshospitalet, a group of parents from Prematurforeningen, and three different nurses from Oslo University Hospital.

Interview guides

The group created two different interview guides, one for parents and one for nurses. For the parents, the goal was to identify challenges in the current solutions and better understand the experiences of parents before, during, and after the birth of their premature babies. Questions addressed topics such as information-seeking behavior, the type of information they found, and their thoughts on the quality of information.

The guide aimed at nurses gathered insights from nurses working with premature babies and their parents. The goal was to identify challenges with current solutions and understand how nurses contribute to creating a sense of security among parents of premature infants. Questions addressed the type of information parents lack, what information they ask for, and how nurses provide information. The interview also explored how parents respond to the information provided, the level of detail given, and the most common questions asked.

During the interviews, one group member led the conversation and asked questions, while the two others took notes of the answers given. The notes were written down in Miro as shown in *Appendix 15 - Notes from interviews.*

Results

The interviews revealed that both parents and nurses emphasize the importance of communication, trust, and support. Parents are primarily concerned about the level and quality of information provided and the potential long-term impacts on their child's development and health. They seek detailed information about their baby's condition, treatment, and future prognosis. They also value communication and support from healthcare professionals, highlighting the importance of trust and confidence in these relationships.

On the other hand, nurses focus on providing appropriate, individualized information and support to parents. They acknowledge the importance of a strong nurse-parent relationship for trust and confidence-building. Nurses understand that different parents may require different levels of information and support and prioritize empowering parents through education and involvement in their child's care. They also recognize the need to address parents' mental health and well-being, as well as their child's condition.

2.1.4 Affinity diagram

An affinity diagram is a collection of large amounts of data that is organized into groups or themes based on their relationships. Post-it notes are used, either physical or digital notes, to sort based on similarities. This makes it easier to extract important findings and themes that are vital to the further process of the project (Dam & Siang, 2022).

To begin, the group utilized digital post-it notes containing key findings from interviews and existing content reviews. Initially, the notes were sorted into distinct categories such as hospital stay information, improvement ideas, reassuring knowledge, experiences, and child communication. The purpose of this categorization was to make the information more comprehensible. Next, the group selected the most relevant categories and identified the critical findings within each one.

The three main categories, as shown in *Figure 1 - Affinity diagram priorities*, were: information regarding hospital stay, suggested improvements, and reassuring knowledge. As shown in *Appendix 16 - Affinity Diagram,* the group then determined which priorities were essential for the project within each of these categories, some of those priorities were that it must be easy to update the information and easy to navigate. By using these categories, the group was able to gain a better understanding of the underlying problems, and the most significant issues were used to create the personas.

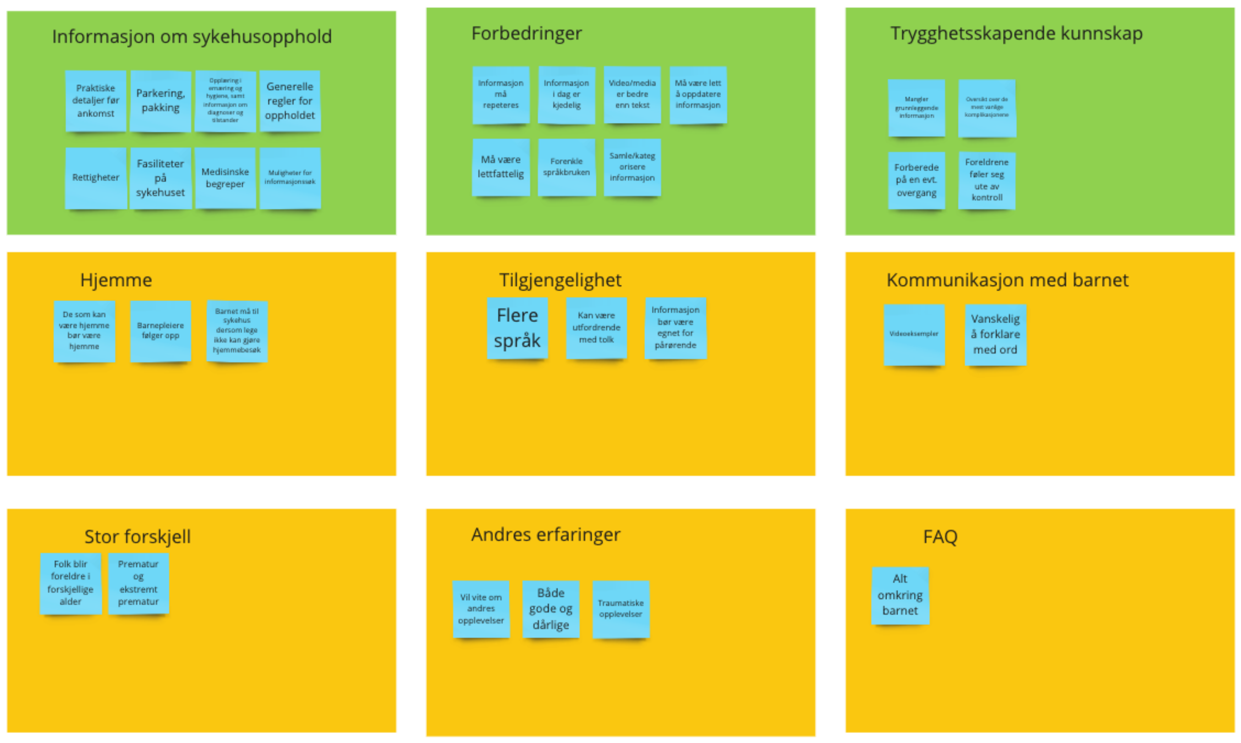

Informasjon om sykehusopphold
Praktiske detaljer før ankomst
Parkering, pakking
Opplæring i ernæring og hygiene, samt informasjon om diagnoser og tilstander
Generelle regler for oppholdet
Rettigheter
Fasiliteter på sykehuset
Medisinske begreper
Muligheter for informasjonssøk
Forbedringer
Informasjon må repeteres
Informasjon i dag er kjedelig
Video/media er bedre enn tekst
Må være lett å oppdatere informasjon
Må være lettfattelig
Forenkle språkbruken
Samle/kategorisere informasjon
Trygghetsskapende kunnskap
Mangler grunnleggende informasjon
Oversikt over de mest vanlige komplikasjonene
Forberede på en evt. overgang
Foreldrene føler seg ute av kontroll
Hjemme
De som kan være hjemme bør være hjemme
Barnepleiere følger opp
Barnet må til sykehus dersom lege ikke kan gjøre hjemmebesøk
Tilgjengelighet
Flere språk
Kan være utfordrende med tolk
Informasjon bør være egnet for pårørende
Kommunikasjon med barnet
Videoeksempler
Vanskelig å forklare med ord
Stor forskjell
Folk blir foreldre i forskjellige alder
Prematur og ekstremt prematur
Andres erfaringer
Vil vite om andres opplevelser
Både gode og dårlige
Traumatiske opplevelser
FAQ
Alt omkring barnet

Figure 1 - Affinity diagram priorities

2.1.5 Personas

Personas is a tool that takes a user profile and then fills in details to create a "typical" user. A persona is simply a fictional individual created to describe a specific user. It can be difficult to relate to an abstract description of a problem or a user. Therefore, it would be easier to identify with a persona that gives life to a user (Baxter, Courage, & Caine, 2015, p. 41).

In a project, it is recommended that multiple personas are developed to represent each user profile, as this approach facilitates the creation of a diverse set of traits for each user category. Focusing solely on a single persona runs the risk of excluding crucial data from end users who do not conform to the parameters of the chosen persona (Baxter, Courage, & Caine, 2015, p. 41). Due to time constraints the group focused primarily on the personas for the primary user group, the parents of premature infants, as shown in *Table 1* - User Profiles that explain the different features of primary, secondary, and tertiary users in our study. Each profile is described in more detail, helping us better understand the people involved.

As part of the project, the group devised three personas to embody the primary user profile, drawing upon insights accumulated from interviews and conversations with the product owner. Among these personas, a single primary persona and two secondary personas were created, shown in *Appendix 1 – Personas*, with the latter primarily shaped by the needs of the former, yet with additional requirements that could be accommodated without impeding the product's capacity to satisfy the primary persona. These personas were vital to the task of safeguarding the user’s genuine needs.

2.1.6 Priority matrix

A priority matrix is a visual representation that organizes tasks according to their impact and effort. The chart is divided into categories, including:

- High effort, high impact
- High effort, low impact
- Low effort, high impact
- Low effort, low impact

The objective is to prioritize tasks based on their significance and manageability (Team Asana, 2022).

The group extracted crucial points from the interviews and arranged them in a priority matrix. The matrix featured a Y-axis representing the user value from low to high and an X-axis illustrating feasibility from low to high.

Key points with the highest user value and feasibility included "preparing parents for changes", "using simple language", and "providing information about hospital stays". Points with high value but low feasibility were "offering multiple languages", "utilizing explanatory animations", and "incorporating video illustrations". This information assisted the group in determining which tasks to prioritize and address first.

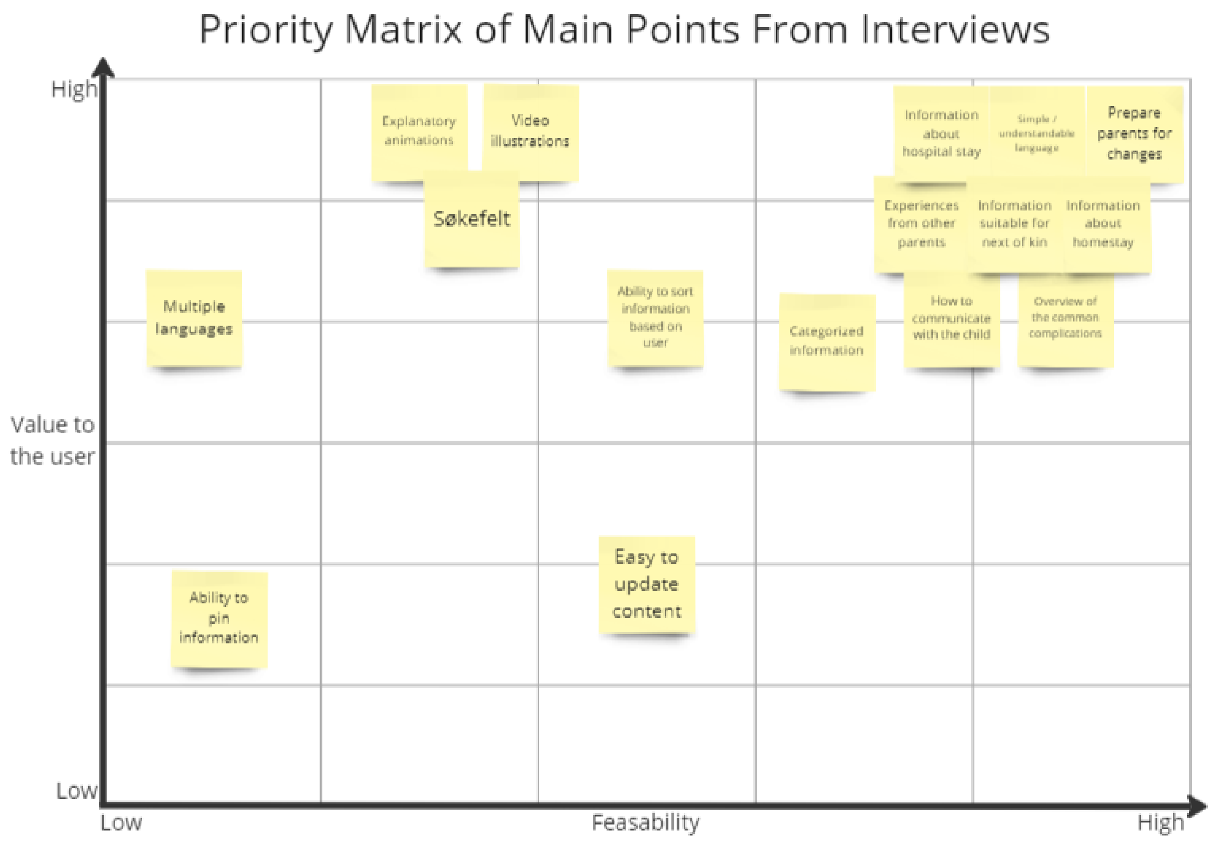
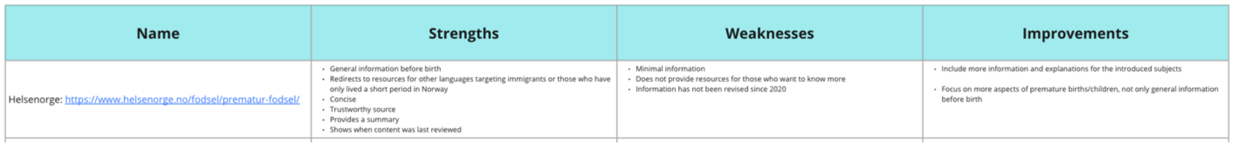


Figure 2 - Priority Matrix created in Figma.

2.2 Competitor analysis

Conducting a competitor analysis can help to gain a better understanding of competitor solutions and differentiate a product in a particular industry. It allows for identifying unique features and characteristics that set a product apart from its competitors, as well as evaluating competitor strengths and weaknesses to enhance and improve a product.

The group analyzed several webpages in the healthcare domain to conduct a competitor analysis on existing solutions. *Figure 3 - Extract from competitive* *analysis* shows an excerpt of our competitive analysis, for a full overview see *Appendix 2 - Competitor analysis.*



Name	Strengths	Weaknesses	Improvements
Helsenorge: https://www.helsenorge.no/fodsel/prematur-fodsel/	• General information before birth • Redirects to resources for other languages targeting immigrants or those who have only lived a short period in Norway • Concise • Trustworthy source • Provides a summary • Shows when content was last reviewed	• Minimal information • Does not provide resources for those who want to know more • Information has not been revised since 2020	• Include more information and explanations for the introduced subjects • Focus on more aspects of premature births/children, not only general information before birth

Figure 3 - Extract from competitive analysis

During the competitor analysis, our primary focus was the following aspects:

- **Structure of content:** By analyzing the content structure of competitor solutions, we aimed to identify industry standards and best practices in organizing and presenting information. This knowledge assisted in creating a user-friendly and easily navigable product, ensuring alignment with user expectations in the target market.
- **Visual profile:** Assessing the visual elements of competitor solutions, such as color schemes, fonts, and imagery, provided insights into what is visually appealing and effective within the industry. This information aided in the development of a distinctive and memorable brand identity and an engaging user experience.
- **Language:** By examining the language and tone used by competitors, we gained insights into the most effective communication style for our target audience. This understanding allowed us to create clear, concise, and persuasive messaging that resonates with users and sets our product apart from the competition.
- **Categories:** Understanding how competitors group information enabled us to design a product that better meets user needs and provides a more comprehensive solution.

To summarize their findings, the group created a table in Miro that outlined the strengths, weaknesses, and potential areas of improvement for each website. Each group member was assigned two to three solutions to analyze and input data into *Appendix 2 - Competitor analysis*.

2.3 Summary

In the Exploration chapter, the team delved into the problem domain, focusing on understanding the target audience's needs and preferences. User research, affinity mapping, and persona development provided a solid foundation for design decisions. Additionally, a competitor analysis identified opportunities for the proposed solution to stand out.

User research was conducted through interviews with parents and nurses to gather relevant information. Based on these insights, user profiles were established and iteratively refined. A priority matrix was used to identify and prioritize the most impactful and feasible tasks to tackle.

The group also analyzed competitor websites to assess their strengths, weaknesses, and areas that could be improved. The insights gained from the Exploration chapter served as a foundation for the subsequent Design phase.

3 Design

The Design chapter explores the process of creating a user-centric interface for the web application. Various ideation and prototyping techniques, such as Crazy 8s and Lo-Fi/Hi-Fi prototypes, are employed to iterate and refine design concepts. User tests are conducted to ensure the design meets user needs and expectations, providing valuable feedback for further improvements. This chapter emphasizes the importance of a user-centered approach in developing an effective and engaging web application.

3.1 Methods

3.1.1 Crazy 8's

Crazy 8's, a core Design Sprint method, was employed in our design process to generate diverse ideas and encourage creativity (Google, 2023). We performed this exercise by sketching eight distinct concepts in eight minutes, focusing on communicating the ideas rather than artistic perfection. As shown in *Figure 4 - Crazy 8's,* the generated sketches enabled us to explore unconventional solutions and identify potential directions for our web application design. This method was instrumental in overcoming initial design constraints and fostering innovative thinking within the team.

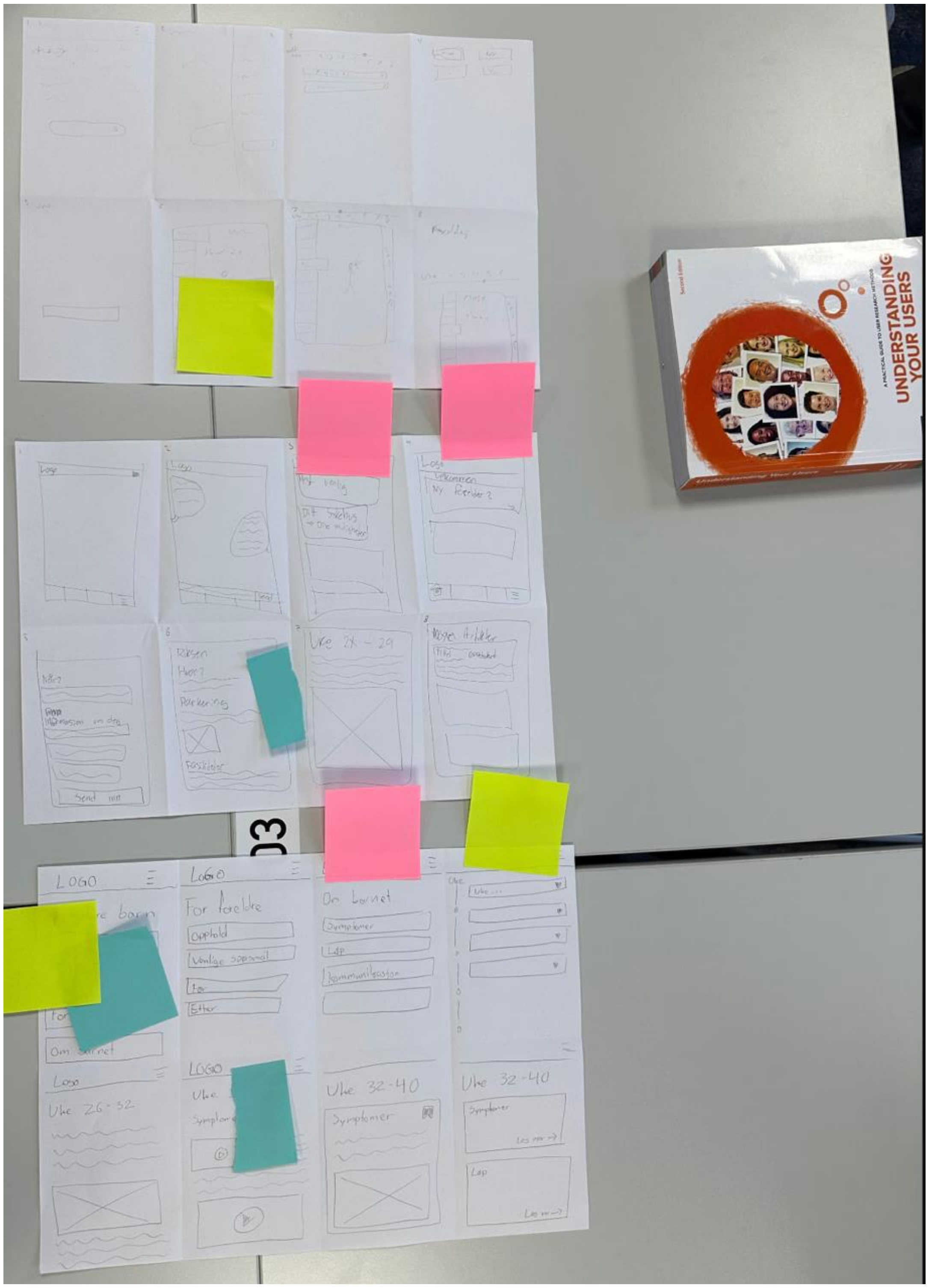


Figure 4 - Crazy 8's

3.1.2 Low-fidelity prototyping

Low-fidelity (Lo-Fi) prototyping was employed in the design process to create simple, interactive representations of the web application. Figma, a collaborative design tool, was utilized to create and iterate on Lo-Fi prototypes.

The initial low-fidelity prototype (Figure 5) was developed based on insights from the Crazy 8's exercise and discoveries mentioned in the exploration phase. This prototype allowed for a focus on the core structure and functionality of the web application, without distraction from visual design elements.

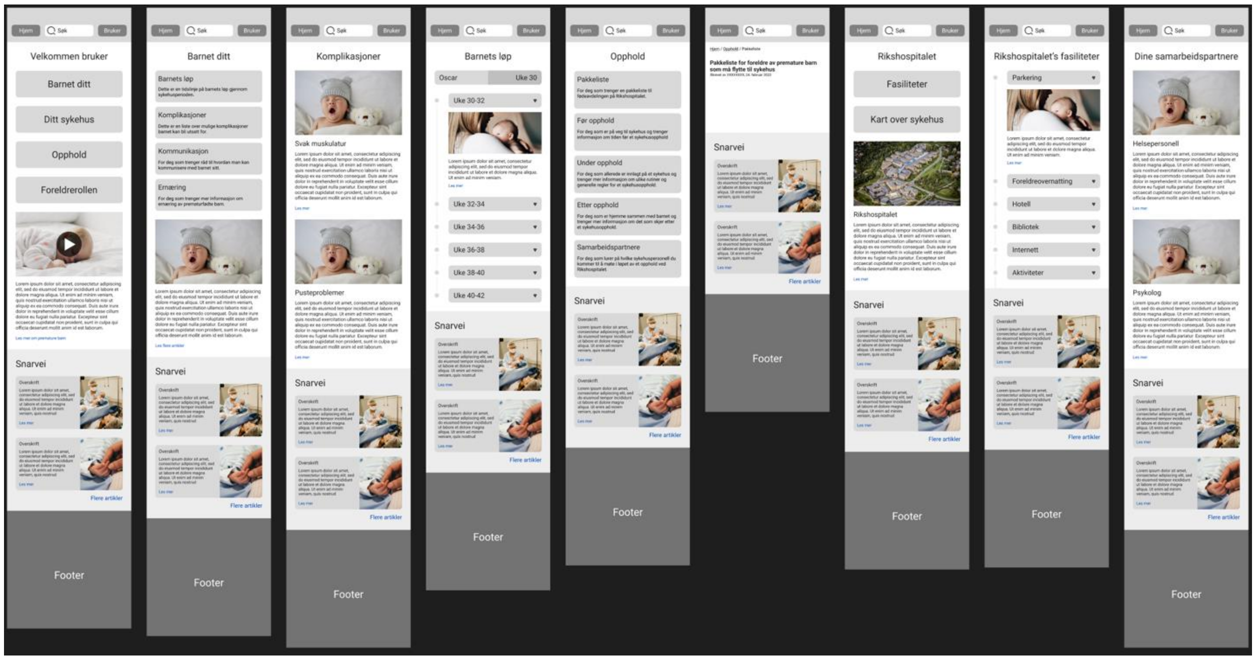


Figure 5 - Initial low-fidelity prototype

3.2 Conducting user tests

Following the initial prototype development, a series of user tests were conducted with eight students on campus to gather feedback and identify improvements. Based on this input, four iterations of the low-fidelity prototype were completed, refining the design to better meet user needs.

3.2.1 Creating the tasks

Four tasks were assigned to each participant using the low-fidelity prototype:

1. Find information about the most common complications for your child.
2. Find the packing list for your stay.
3. From the packing list, navigate to the facilities the hospital has.
4. From the facilities, navigate to your partners.

Task data, including participant number, task number, time, mistakes, and notes, was documented in an Excel table as shown in *Appendix 17 - Low Fidelity User Testing*.

3.2.2 Feedback and results

For each test iteration, mistakes and feedback were used to improve the prototype. Feedback included:

- Thumbnail description should have a length limit.
- Navigation buttons should have better descriptions.
- Breadcrumbs should be implemented.
- Unsure about results in search bar.

Based on feedback from participants, a last version of the low fidelity prototype was created.

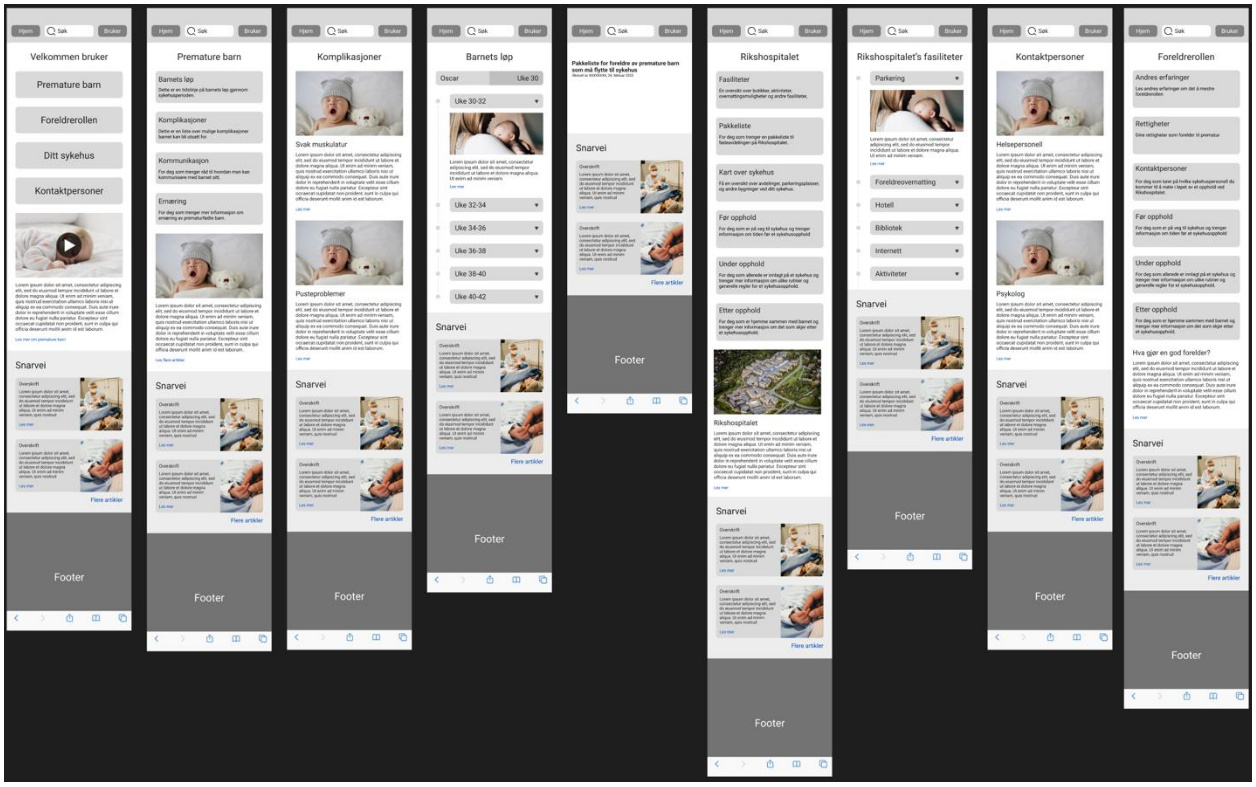


Figure 6 - Fourth iteration of low-fidelity prototype

3.3 Summary

This chapter focuses on creating a user-centric web application interface using ideation techniques like Crazy 8s and Lo-Fi/Hi-Fi prototypes. User tests validate the design and offer valuable feedback for improvements, emphasizing a user-centered approach. The design process involved Crazy 8's exercises, low-fidelity prototyping, and user tests, leading to four prototype iterations based on participant feedback. A final low-fidelity prototype was created accordingly. The group was ready to move on to developing the solution.

4 Development

The Develop chapter focuses on the technical aspects of building web applications. We begin by discussing the selection of appropriate technologies, considering factors such as client-side rendering (CSR) versus server-side rendering (SSR) and the choice of backend and frontend frameworks. We then outline the process of setting up the backend and creating a custom content management system (CMS) tailored to the project's requirements. The chapter also covers frontend development using React and SASS[2], as well as the deployment of the application. This section demonstrates the practical application of web development technologies in bringing the design to life.

4.1 Choosing technologies

Choosing the appropriate technologies for an application is a critical decision in its development process. Making the wrong choice could lead to user dissatisfaction and low performance scores.

4.1.1 CSR vs. SSR

When considering the appropriate technologies for a website, it is crucial to introduce the two main approaches for modern website rendering: client-side and server-side.

Client-Side Rendering

In Client-Side Rendering (CSR), web pages are rendered directly in the user's browser using JavaScript, where all the processing of logic, data retrieval, template rendering, and routing is done on the client-side instead of the server (Miller & Osmani, 2022).

[2] Syntactically Awesome Style Sheets: https://sass-lang.com/

A Single-page Application (SPA) is a web application design that initially loads a single web document and dynamically updates its body content using JavaScript APIs such as XMLHttpRequest and Fetch when displaying different content. This approach offers users a more dynamic experience and potential performance improvements by not reloading entire pages from the server. However, it also presents drawbacks, including SEO challenges, increased effort required to manage state and navigation, and complexities in performance monitoring (mdn web docs, 2023).

The advantage of utilizing Client-Side Rendering (CSR) to render web pages is that it allows for faster initial loading times, as the server only sends the necessary HTML, CSS, and JavaScript files required for rendering the page. CSR is an excellent option for Single-Page Applications (SPAs) that load a considerable amount of dynamic content as it can enhance the overall user experience and application performance. However, a drawback of using CSR with SPAs is that subsequent loading of data and dynamic content may be slower, as the client-side needs to fetch data and render the content after the initial page load.

Server-Side Rendering

Server-Side Rendering (SSR) involves the generation of HTML pages from the server in response to webpage navigation, eliminating the need for extra roundtrips for data retrieval and template rendering on the client-side as it is processed before the browser receives a response (Miller & Osmani, 2022).

Using Server-Side Rendering (SSR) for a webpage offers advantages, including improved Search Engine Optimization (SEO) as the fully rendered HTML file is sent from the server. This enables search engines to efficiently index the content, potentially enhancing the page's search engine ranking. However, this approach also has some drawbacks, including increased server load and slower initial loading times as the server needs to process data retrieval and template rendering before sending the fully rendered page to the client.

Summary

To summarize, Client-Side Rendering (CSR) involves rendering webpages directly in the user’s browser using JavaScript, allowing for faster initial loading times and enhanced user experience for Single-Page Applications (SPA). However, it presents some drawbacks, including SEO challenges and slower subsequent loading of data and dynamic content.

On the other hand, Server-Side Rendering (SSR) generates HTML pages from the server in response to webpage navigation, eliminating the need for extra roundtrips for data retrieval and template rendering on the client. This approach offers improved SEO and potentially better search engine rankings but can also result in slower initial loading times and increased server load.

4.1.2 Deciding on an approach

As described in section 4.1.1 CSR vs. SSR, there are two primary approaches when it comes to rendering webpages. To choose the right approach for our project, the group needs to consider factors such as the project requirements, user experience, and development resources.

Factors when choosing technologies

When it comes to selecting a technology for rendering webpages, the group identified factors to consider. These include:

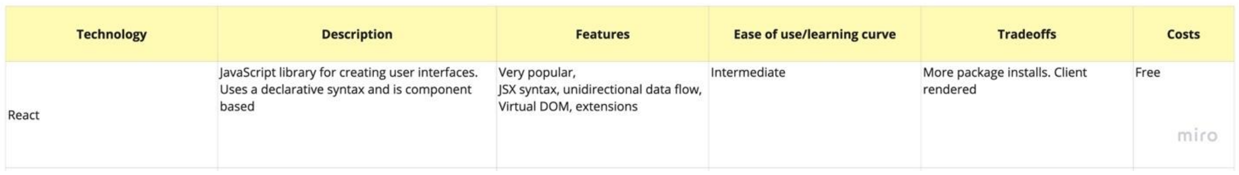
- Features
- Ease of use/learning curve
- Tradeoffs
- Costs

Considering these factors can help the group make an informed decision when selecting the appropriate technology for the project.

Evaluating Technology Stacks

To evaluate different technology stacks, it's important to consider the factors mentioned in the previous section. The group created a spreadsheet that compares the options based on these factors.

Figure 7 shows an excerpt of the spreadsheet used to evaluate the different technologies. See, *Appendix 13 - Technology evaluation* for the complete version of the evaluation.



Technology	Description	Features	Ease of use/learning curve	Tradeoffs	Costs
React	JavaScript library for creating user interfaces. Uses a declarative syntax and is component based	Very popular, JSX syntax, unidirectional data flow, Virtual DOM, extensions	Intermediate	More package installs. Client rendered	Free

miro

Figure 7 - Extract from the technology evaluation

Choosing the MERN / MEVN stack

Based on the factors discussed above, the group decided that a Client-Side Rendered approach was the optimal choice for the project due to its fast initial render, user-friendliness, and decreased server load.

The MERN stack is a full-stack JavaScript framework for developing web applications. MERN is an acronym for MongoDB, Express, React.js and Node.js. These are the four technologies that make up the layers of the stack. The MEVN stack is a variation of MERN, that uses Vue.js instead of React.js for the frontend (MongoDB, Inc, 2023).

The group opted for a mix of MERN and MEVN stacks to meet the project's demands for efficiency, scalability, and time constraints. Vue 3's exceptional performance and Quasar's extensive component library allowed for a rapid setup and swift application development. This enabled the group to swiftly develop the CMS with a familiar user interface and customize the frontend of the application with *React.js*.

In the upcoming sections, we will explore these technologies in greater depth, including their usage in the frontend, backend and CMS of the application.

4.2 Frontend

The frontend application is designed to be used by the end-users, primarily parents of prematurely born babies. It serves as the software system's visual and interactive layer, rendering and displaying content, and processing user inputs.

The frontend application was developed using React as the main JavaScript library. It was styled using SCSS and communicates with the backend using Axios[3]. React Router (React Router, 2023) allowed the application to operate with client-side routing, enabling our application to update the URL upon clicking a link without the need to request an additional document from the server.

4.2.1 Frontend technologies

React

React is an open-source JavaScript library developed by Facebook for building user interfaces and has gained widespread popularity due to its emphasis on component-based architecture, which promotes reusability and modularity in web applications (Arancio, 2021). React's main concepts include components, state, and props.

Components are self-contained, reusable pieces of UI that can manage their own state, while props are used to pass data between components. React applications have improved performance through a virtual DOM, which optimizes the updating of the actual DOM, resulting in faster rendering (Arancio, 2021). The library also integrates seamlessly with other tools, libraries, and frameworks, making it an ideal choice for modern web application development.

[3] Axios: https://axios-http.com/docs/intro

SCSS/SASS

SCSS (Sass) is a CSS preprocessor that extends the capabilities of CSS, making it more maintainable, modular, and scalable than normal CSS (Richards, 2020). SCSS introduces features such as variables, nesting, mixins, and inheritance, which streamline the development process and help manage large-scale projects more effectively. By utilizing SCSS, developers can create organized, reusable, and easily maintainable stylesheets, improving the overall development experience (Fileformat, 2023).

4.2.2 Setting up the frontend

The frontend was set up using Vite with the command “yarn create vite client”. “client” was the name of the folder which was located inside the main project folder alongside the backend and the CMS.

Vite

Vite was used to create our application because it significantly improves development experience by addressing performance bottlenecks commonly encountered with traditional JavaScript tooling. Vite leverages native ES modules in the browser and benefits from the rise of JavaScript tools written in compile-to-native languages. It optimizes the dev server start time by categorizing modules into dependencies and source code (Vite, 2023).

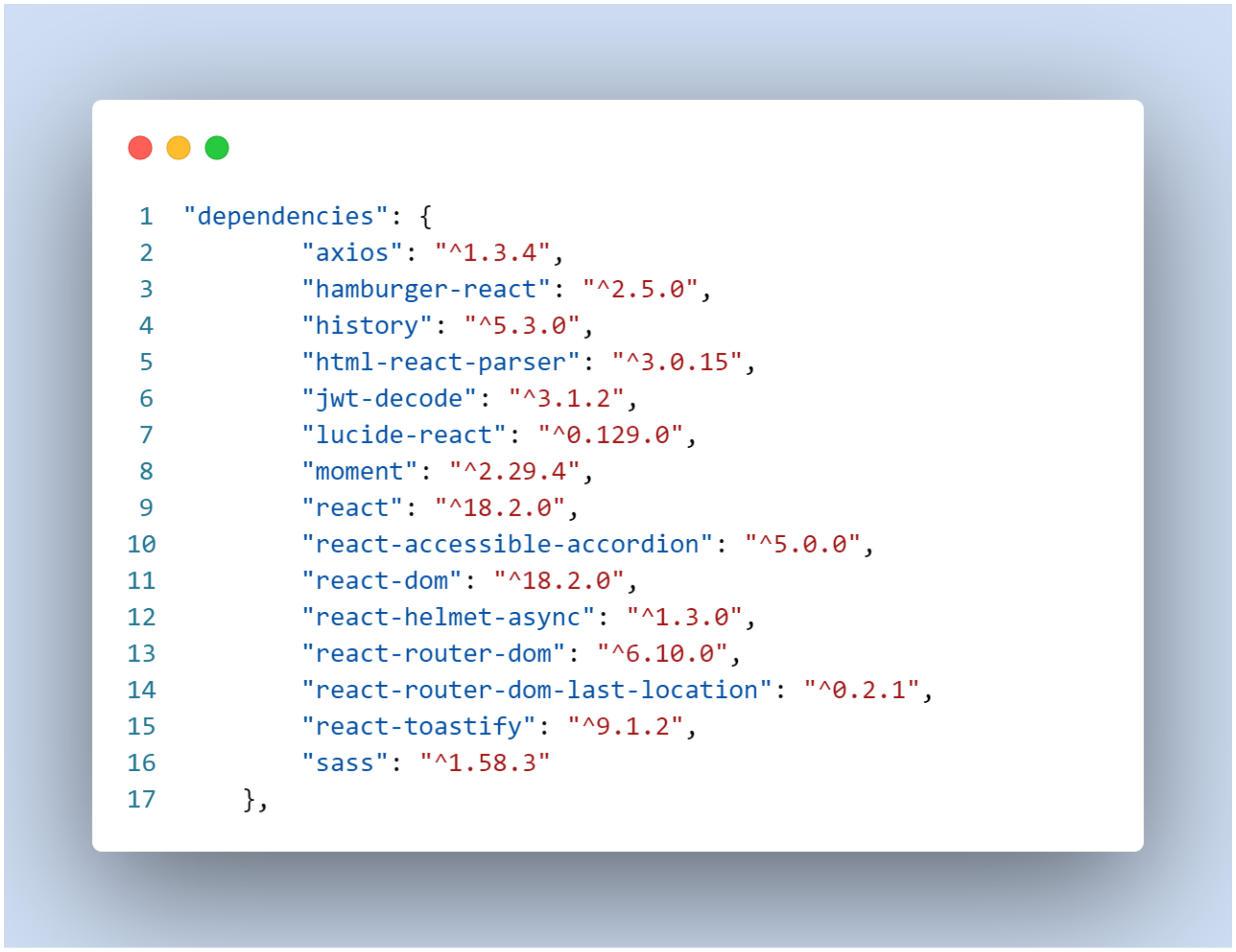
Dependencies are pre-bundled using esbuild, a faster tool written in Go, while source code is served over native ESM, allowing the browser to manage part of the bundling process. This results in quicker server starts, more efficient file processing, and an overall better development experience, ultimately boosting developers' productivity and satisfaction (Vite, 2023).

Installing Dependencies

Key dependencies include:

- axios (v1.3.4): Enables browser XMLHttpRequests.
- jwt-decode (v3.1.2): Decodes JWT tokens.

- moment (v2.29.4): Parses JavaScript time formats.
- react-router-dom (v6.10.1): Provides React Router bindings.
- sass (v1.58.3): JavaScript-compiled Dart Sass distribution.



```
"dependencies": {
        "axios": "^1.3.4",
        "hamburger-react": "^2.5.0",
        "history": "^5.3.0",
        "html-react-parser": "^3.0.15",
        "jwt-decode": "^3.1.2",
        "lucide-react": "^0.129.0",
        "moment": "^2.29.4",
        "react": "^18.2.0",
        "react-accessible-accordion": "^5.0.0",
        "react-dom": "^18.2.0",
        "react-helmet-async": "^1.3.0",
        "react-router-dom": "^6.10.0",
        "react-router-dom-last-location": "^0.2.1",
        "react-toastify": "^9.1.2",
        "sass": "^1.58.3"
    },
```

Figure 8 - Frontend dependencies in "package.json".

Folder Structure *(Appendix 12 - Client folder structure)*

The client's folder structure is built upon Vite's default structure. Most of the folders and code lives within the "src" folder, and the group tried to separate content into meaningful folders to better organize code and maintain separation of concerns.

- **public**: Favicon
- **src**: Contains the primary source code for the application.
    - **api**: Files such as APIcalls using axios.

- **assets**: Client-side assets such as icons.
- **components**: Custom built React components.
- **pages**: React components used as elements in React Router.
- **routes**: Private and public routes based on AuthContext.
- **scss**: Color and spacing variables, and a global “*index.scss*” file.
- **utils**: AuthContext and helper functions.

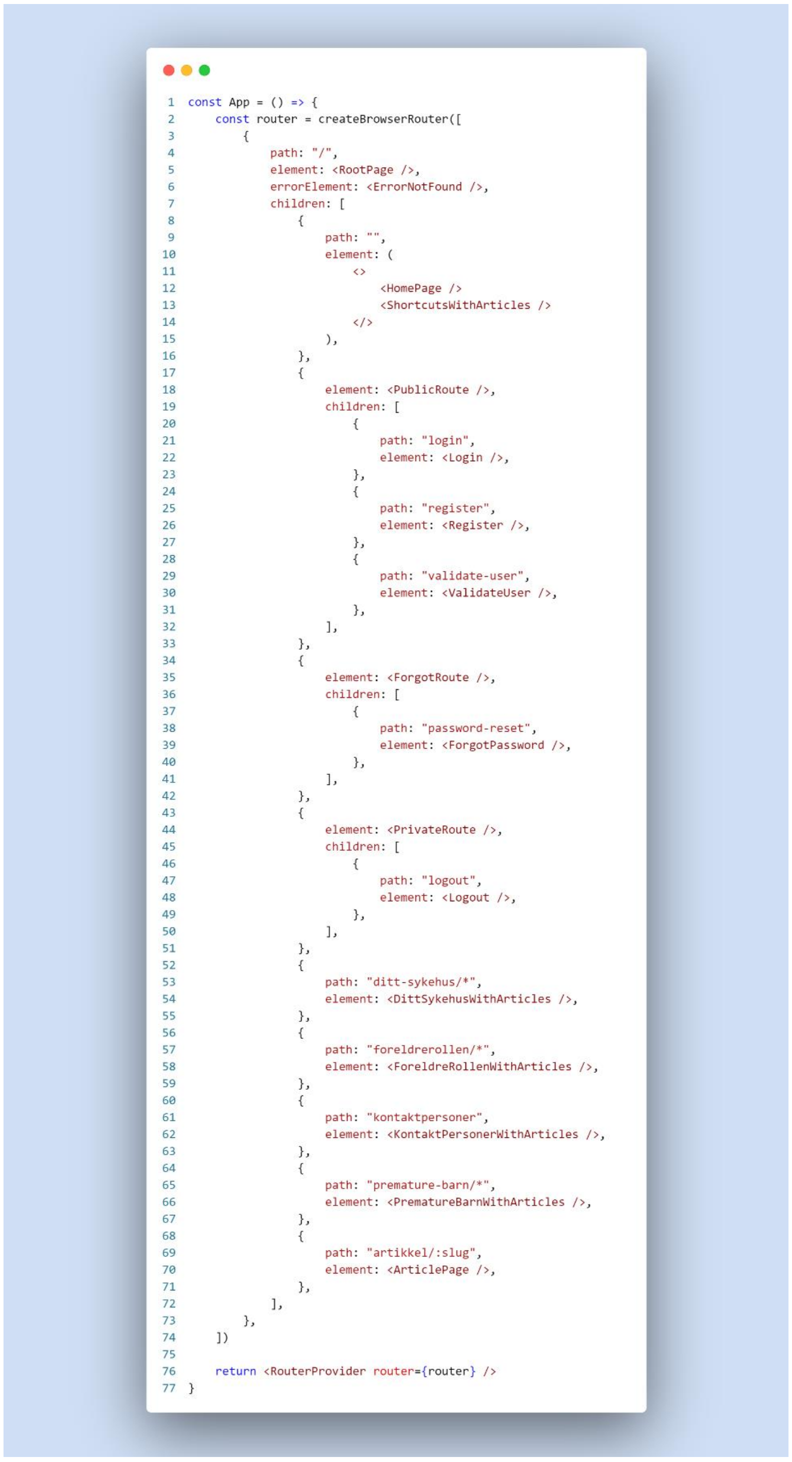
Routing

React Router was utilized to implement client-side routing. In contrast to conventional websites that require server requests for each page load, client-side routing updates the URL without fetching new documents. This approach allows for immediate rendering of new UI elements and data retrieval, leading to quicker and more engaging user experiences, including animations (React Router, 2023).

The App component is a functional component that sets up the application's routing. It uses createBrowserRouter to define an array of route objects, each containing a path and an associated element. Nested within these routes are child routes with their own paths and elements.

The top-level route contains the RootPage element, an ErrorNotFound element for handling errors, and an array of child routes. These child routes include the home page, public routes for login, registration, and user validation, a “forgot” route for password reset, a private route for logout, and several other routes with different paths.

When a user navigates to a specific path, the corresponding element for that route is rendered. The RouterProvider component wraps the entire routing structure, allowing the defined routes to be used throughout the application.



```jsx
const App = () => {
    const router = createBrowserRouter([
        {
            path: "/",
            element: <RootPage />,
            errorElement: <ErrorNotFound />,
            children: [
                {
                    path: "",
                    element: (
                        <>
                            <HomePage />
                            <ShortcutsWithArticles />
                        </>
                    ),
                },
                {
                    element: <PublicRoute />,
                    children: [
                        {
                            path: "login",
                            element: <Login />,
                        },
                        {
                            path: "register",
                            element: <Register />,
                        },
                        {
                            path: "validate-user",
                            element: <ValidateUser />,
                        },
                    ],
                },
                {
                    element: <ForgotRoute />,
                    children: [
                        {
                            path: "password-reset",
                            element: <ForgotPassword />,
                        },
                    ],
                },
                {
                    element: <PrivateRoute />,
                    children: [
                        {
                            path: "logout",
                            element: <Logout />,
                        },
                    ],
                },
                {
                    path: "ditt-sykehus/*",
                    element: <DittSykehusWithArticles />,
                },
                {
                    path: "foreldrerollen/*",
                    element: <ForeldreRollenWithArticles />,
                },
                {
                    path: "kontaktpersoner",
                    element: <KontaktPersonerWithArticles />,
                },
                {
                    path: "premature-barn/*",
                    element: <PrematureBarnWithArticles />,
                },
                {
                    path: "artikkel/:slug",
                    element: <ArticlePage />,
                },
            ],
        },
    ])

    return <RouterProvider router={router} />
}
```

Figure 9 - React router in “App.jsx”

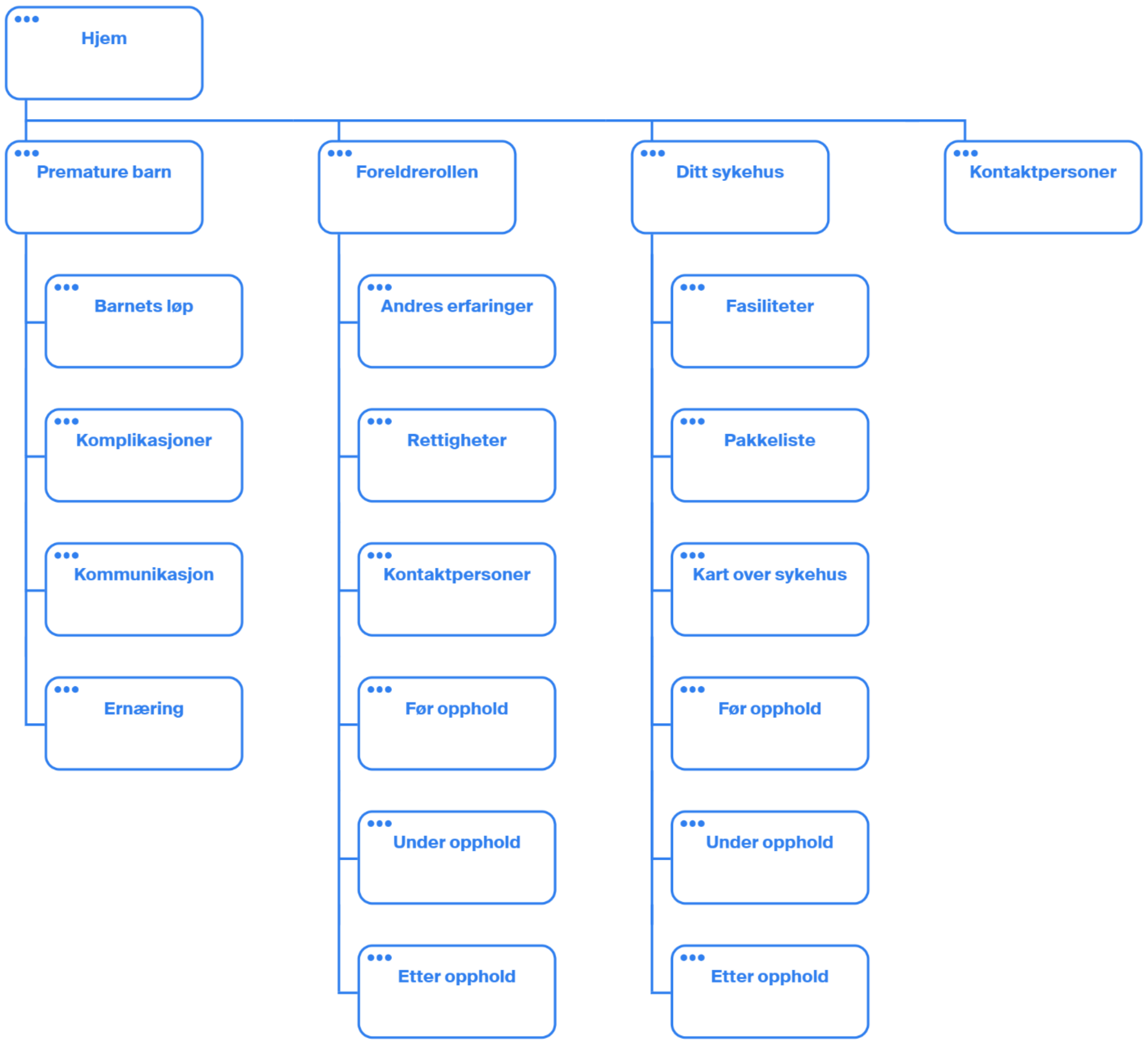
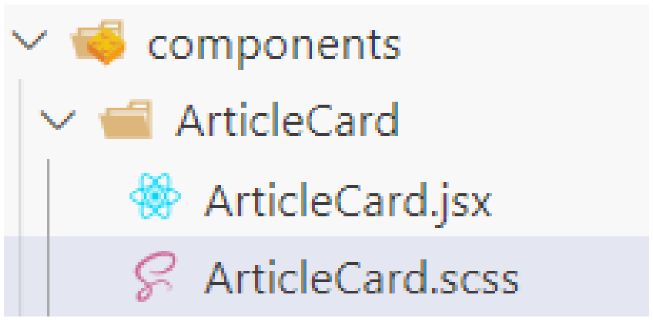


Figure 10 - Sitemap of the website

Setting up SCSS

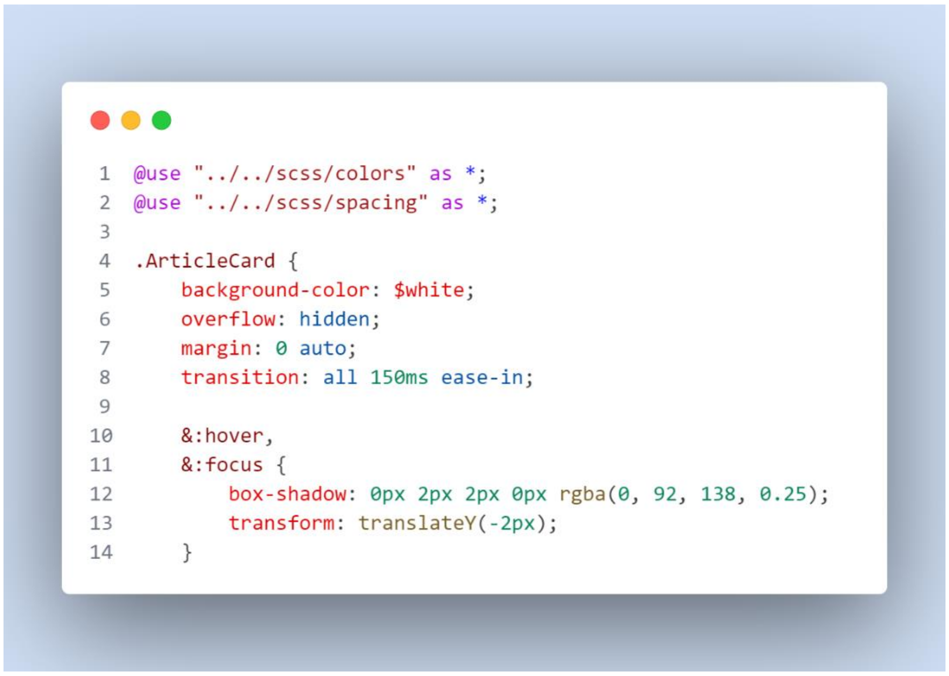
Each component resides in its own folder, accompanied by a respective SCSS file. To prevent global styling conflicts, styles are nested under the component's className. A global SCSS file manages overarching styles throughout the app.

Separate files store color and spacing variables, enabling easy access by importing them when necessary. The color variables encompass the entire color palette for the platform, while spacing variables define pixel-based distances.



```
components
  ArticleCard
    ArticleCard.jsx
    ArticleCard.scss
```

Figure 11 - The ArticleCard folder contains the "ArticleCard.jsx" and "ArticleCard.scss"

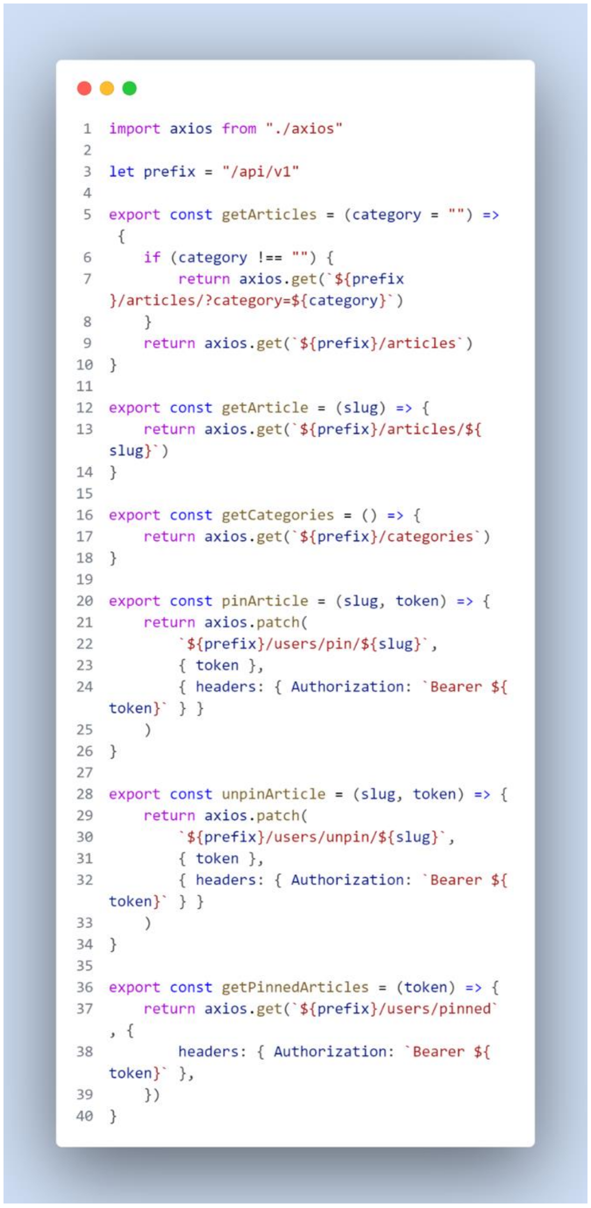


```scss
@use "../../scss/colors" as *;
@use "../../scss/spacing" as *;

.ArticleCard {
    background-color: $white;
    overflow: hidden;
    margin: 0 auto;
    transition: all 150ms ease-in;

    &:hover,
    &:focus {
        box-shadow: 0px 2px 2px 0px rgba(0, 92, 138, 0.25);
        transform: translateY(-2px);
    }
```

Figure 12 – Example of how a component is styled: Every element is styled and nested inside ".ArticleCard" className, and color and spacing variables are imported in the file

Connecting to the backend

Axios enables browser XMLHttpRequests. XHR objects facilitate server interaction, allowing data retrieval from a URL without requiring a complete page refresh. This capability enables web pages to update specific sections without interrupting the user's experience (mdn web docs, 2023). Axios was used to create API calls to the backend API to retrieve and store data in MongoDB.



```javascript
import axios from "./axios"

let prefix = "/api/v1"

export const getArticles = (category = "") => {
    if (category !== "") {
        return axios.get(`${prefix}/articles/?category=${category}`)
    }
    return axios.get(`${prefix}/articles`)
}

export const getArticle = (slug) => {
    return axios.get(`${prefix}/articles/${slug}`)
}

export const getCategories = () => {
    return axios.get(`${prefix}/categories`)
}

export const pinArticle = (slug, token) => {
    return axios.patch(
        `${prefix}/users/pin/${slug}`,
        { token },
        { headers: { Authorization: `Bearer ${token}` } }
    )
}

export const unpinArticle = (slug, token) => {
    return axios.patch(
        `${prefix}/users/unpin/${slug}`,
        { token },
        { headers: { Authorization: `Bearer ${token}` } }
    )
}

export const getPinnedArticles = (token) => {
    return axios.get(`${prefix}/users/pinned`, {
        headers: { Authorization: `Bearer ${token}` },
    })
}
```

Figure 13 – API calls to interact with the backend using Axios

Context & Authentication

The authentication context is established for the React application utilizing a class component called AuthProvider. This context offers various authentication-related functionalities, such as user login, logout, registration, validation, password reset, and the generation of headers containing tokens for authorized API requests.

AuthProvider is responsible for maintaining the authentication state, which includes user authentication status, fetch loading state, user token, user details, and any potential errors. The component state is initialized with a predefined initial state.

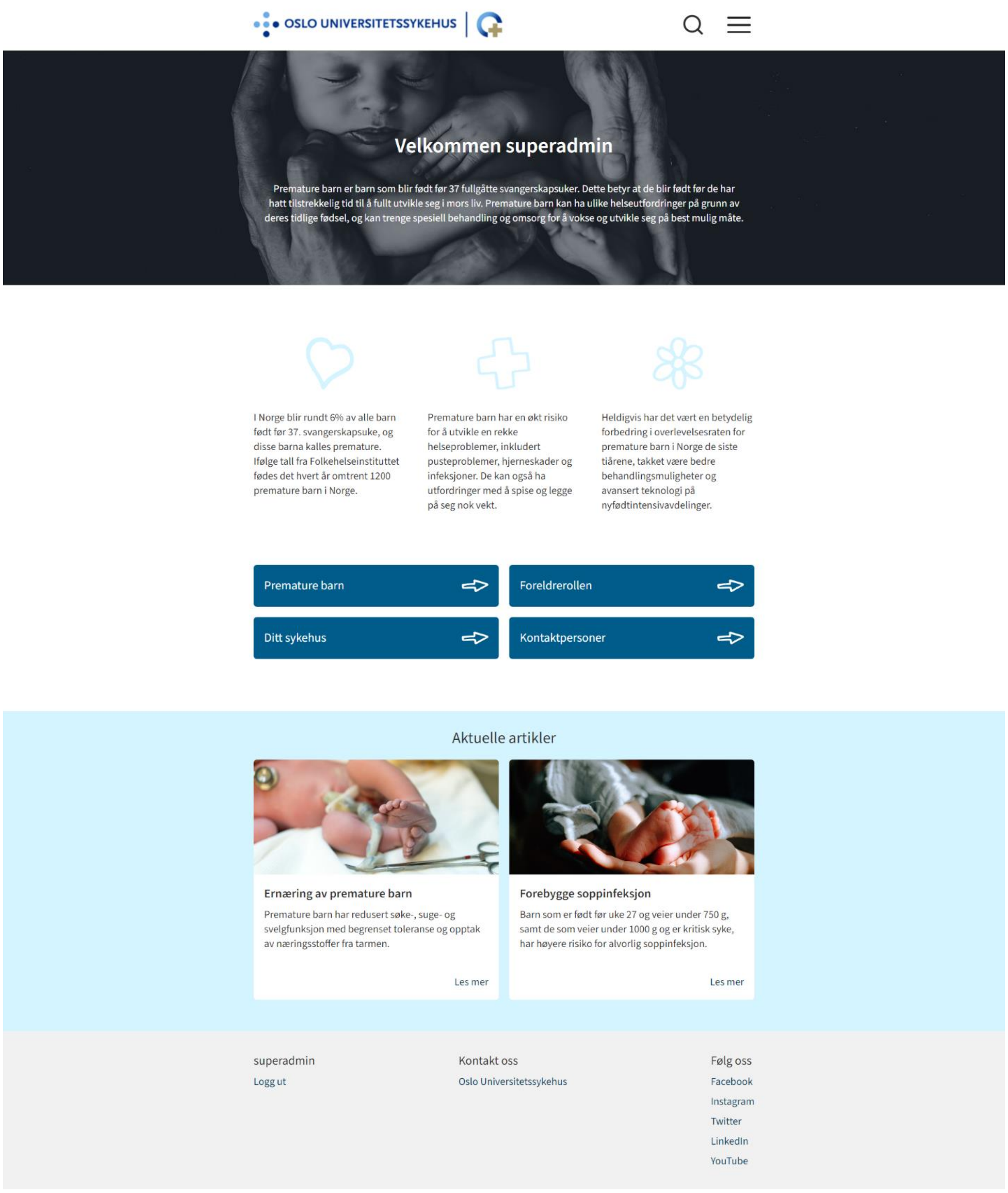
When the component is mounted, a token refresh method is called to refresh the token if necessary. The class component provides several methods for handling authentication, such as user login, logout, registration, validation, and password reset. These methods initiate the corresponding API functions and update the component state based on the received API response.

A method is implemented to set a timer to refresh the token one minute before its expiration, while another method is responsible for terminating the timer. The JWT token is decoded to ascertain the token's expiration time.

In conclusion, the authentication context is provided to the children components, delivering the authentication state and methods as the context's value. A consumer component is exported as well, enabling its use in other components to access the authentication context. In the following section, we will discuss the finished design.

4.2.3 The finished design

The finished application features a homepage with general information about premature babies. From the homepage, users can navigate through the website using React Router links.


OSLO UNIVERSITETSSYKEHUS

Velkommen superadmin

Premature barn er barn som blir født før 37 fullgåtte svangerskapsuker. Dette betyr at de blir født før de har hatt tilstrekkelig tid til å fullt utvikle seg i mors liv. Premature barn kan ha ulike helseutfordringer på grunn av deres tidlige fødsel, og kan trenge spesiell behandling og omsorg for å vokse og utvikle seg på best mulig måte.

I Norge blir rundt 6% av alle barn født før 37. svangerskapsuke, og disse barna kalles premature. Ifølge tall fra Folkehelseinstituttet fødes det hvert år omtrent 1200 premature barn i Norge.

Premature barn har en økt risiko for å utvikle en rekke helseproblemer, inkludert pusteproblemer, hjerneskader og infeksjoner. De kan også ha utfordringer med å spise og legge på seg nok vekt.

Heldigvis har det vært en betydelig forbedring i overlevelsesraten for premature barn i Norge de siste tiårene, takket være bedre behandlingsmuligheter og avansert teknologi på nyfødtintensivavdelinger.

Aktuelle artikler

Ernæring av premature barn

Premature barn har redusert søke-, suge- og svelgfunksjon med begrenset toleranse og opptak av næringsstoffer fra tarmen.

Les mer

Forebygge soppinfeksjon

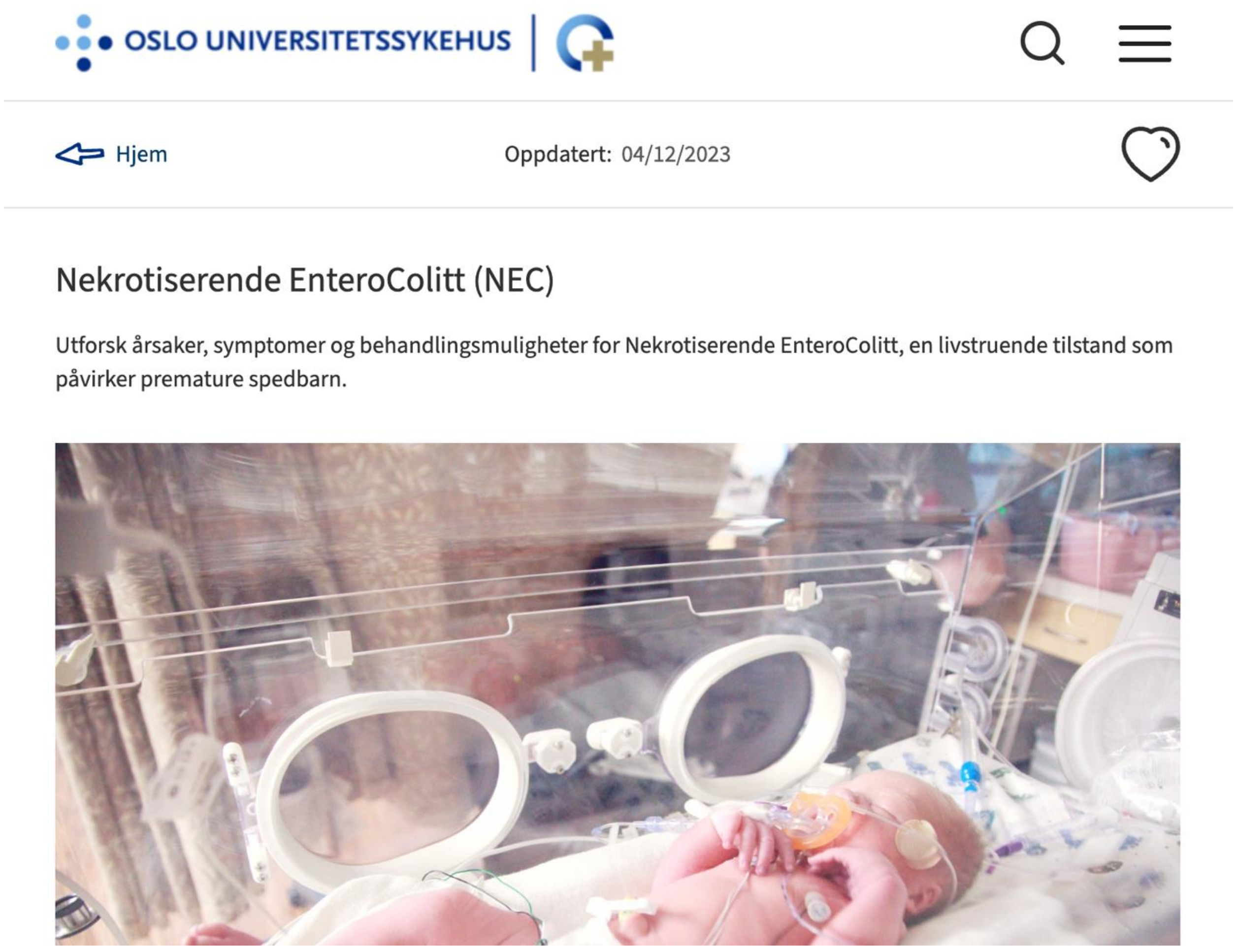
Barn som er født før uke 27 og veier under 750 g, samt de som veier under 1000 g og er kritisk syke, har høyere risiko for alvorlig soppinfeksjon.

Les mer

Figure 14 - The finished design of the website

Articles

Users can access CMS-authored articles across the site, with each featuring a thumbnail including an image, title, and description. Clicking the thumbnail directs to the article page, displaying the update date, title, description, text, media (images/videos), and sources, based on CMS input.



Oppdatert: 04/12/2023

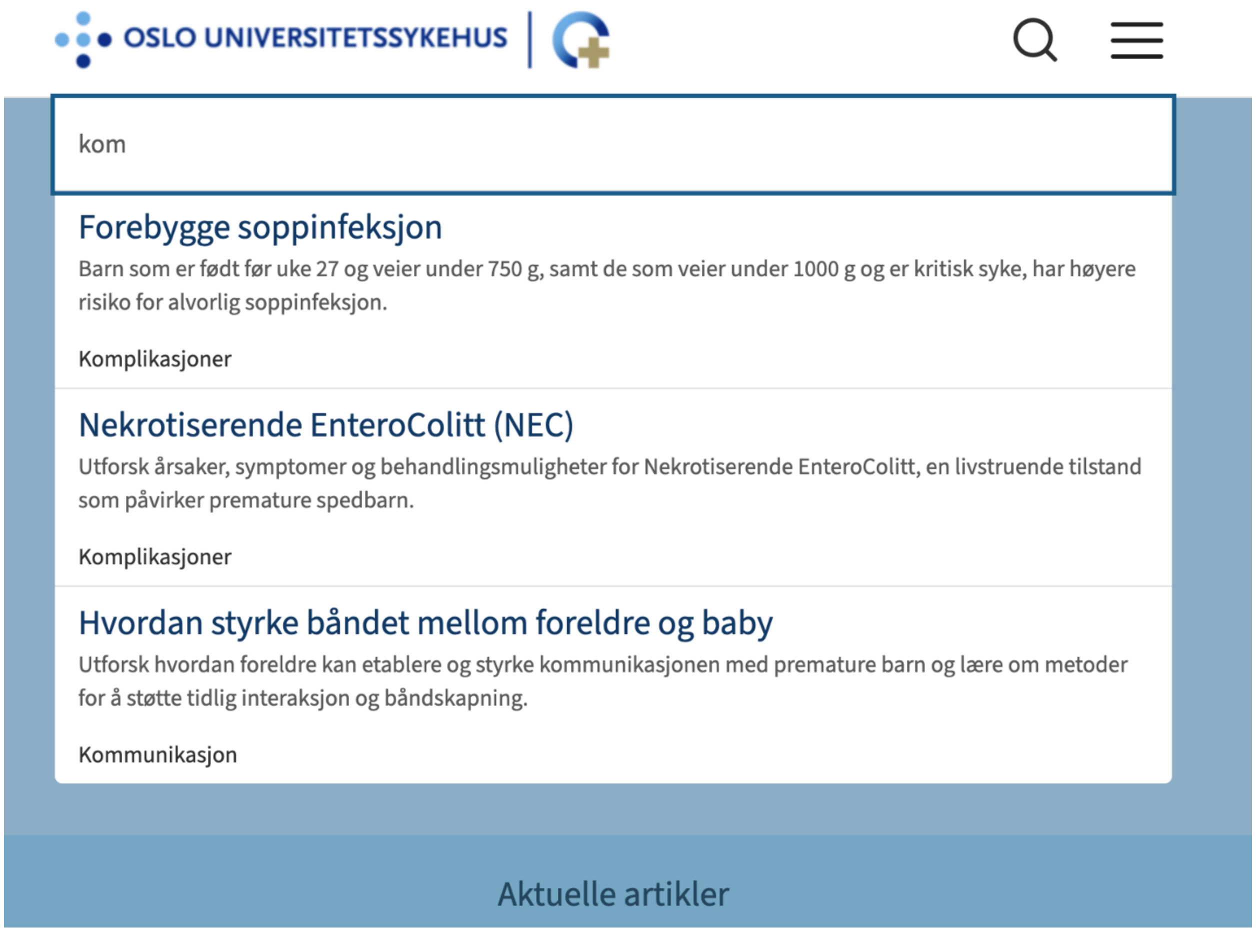
Nekrotiserende EnteroColitt (NEC)

Utforsk årsaker, symptomer og behandlingsmuligheter for Nekrotiserende EnteroColitt, en livstruende tilstand som påvirker premature spedbarn.

Figure 15 - The article page

Search

A user can search for articles via the search bar found in the navigation bar at the top of the page. Upon entering letters, a list dynamically updates results based on category, title, excerpt.



kom

Forebygge soppinfeksjon
Barn som er født før uke 27 og veier under 750 g, samt de som veier under 1000 g og er kritisk syke, har høyere risiko for alvorlig soppinfeksjon.
Komplikasjoner

Nekrotiserende EnteroColitt (NEC)
Utforsk årsaker, symptomer og behandlingsmuligheter for Nekrotiserende EnteroColitt, en livstruende tilstand som påvirker premature spedbarn.
Komplikasjoner

Hvordan styrke båndet mellom foreldre og baby
Utforsk hvordan foreldre kan etablere og styrke kommunikasjonen med premature barn og lære om metoder for å støtte tidlig interaksjon og båndskapning.
Kommunikasjon

Aktuelle artikler

Figure 16 - The search bar, users can search for excerpt, title and category

Navigation

Each page, except for the lowest level ones, include links to deeper nested pages. Breadcrumbs facilitate navigation back and visually represent hierarchy, while a hamburger menu enables site-wide access.

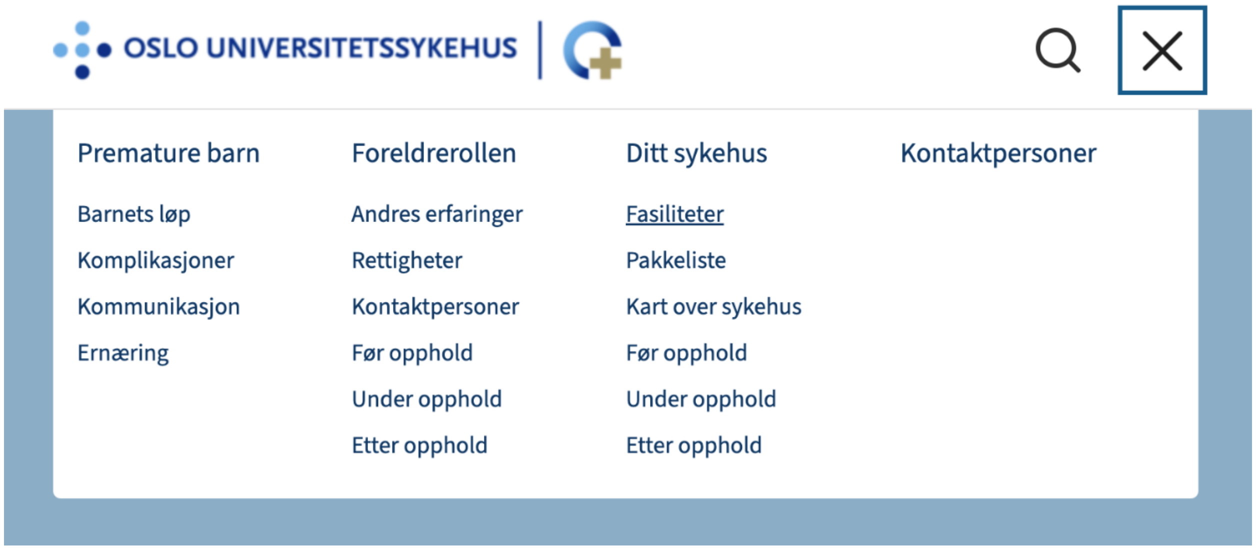


Figure 17 - The "Hamburger" menu provides easy navigation across the website

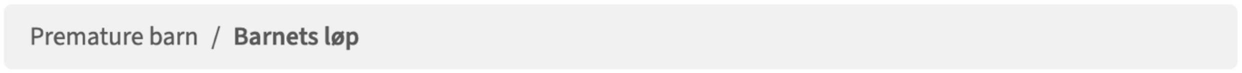


Figure 18 - Breadcrumbs are present on every page

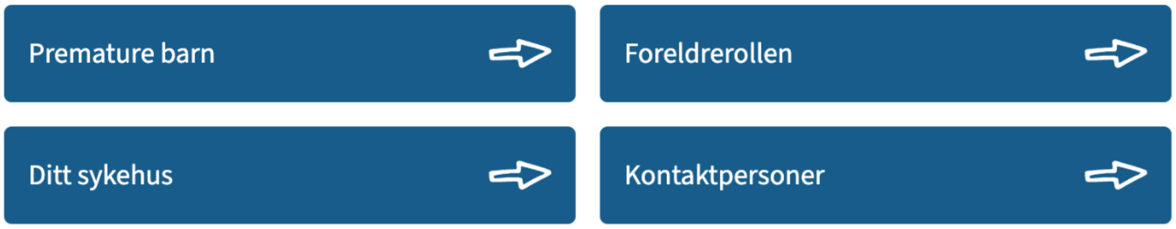
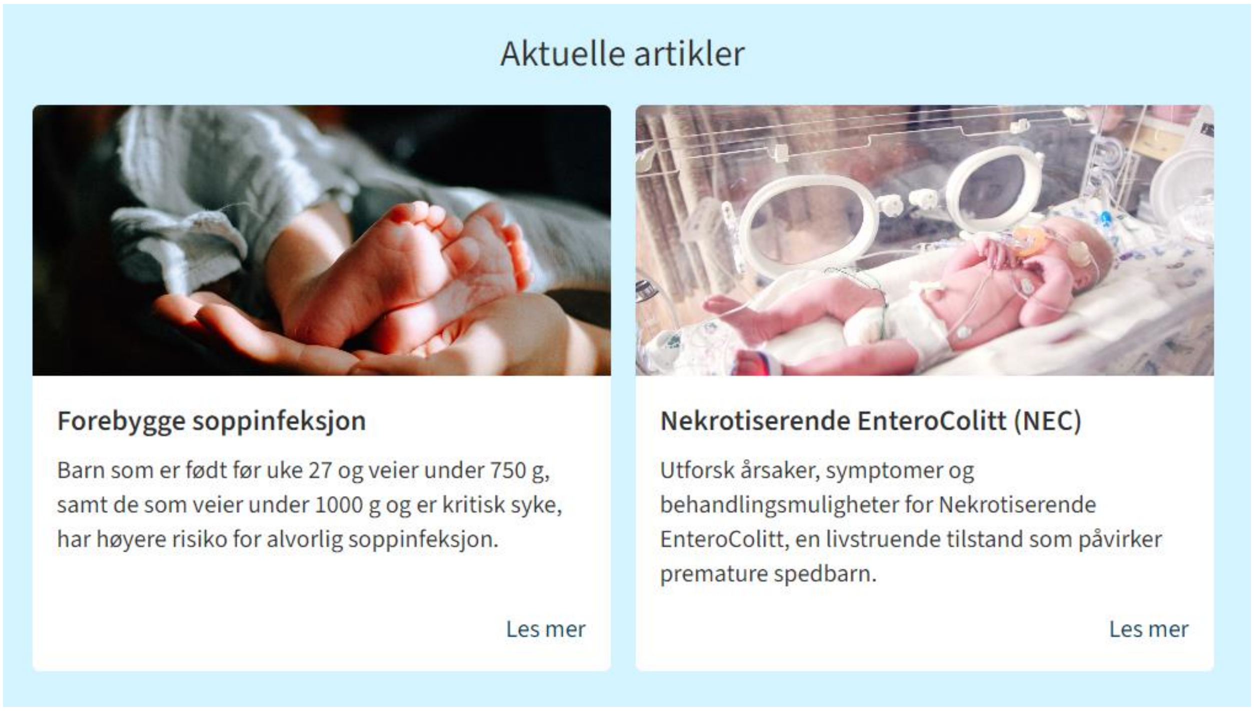


Figure 19 – Links to nested pages

Aktuelle artikler (relevant articles)

All but the lowest-level pages feature an "aktuelle artikler" section, displaying two random, location-relevant articles. For instance, if a user is on the "foreldrerollen" (parent role) page, only related articles will be shown.


Aktuelle artikler

Forebygge soppinfeksjon

Barn som er født før uke 27 og veier under 750 g, samt de som veier under 1000 g og er kritisk syke, har høyere risiko for alvorlig soppinfeksjon.

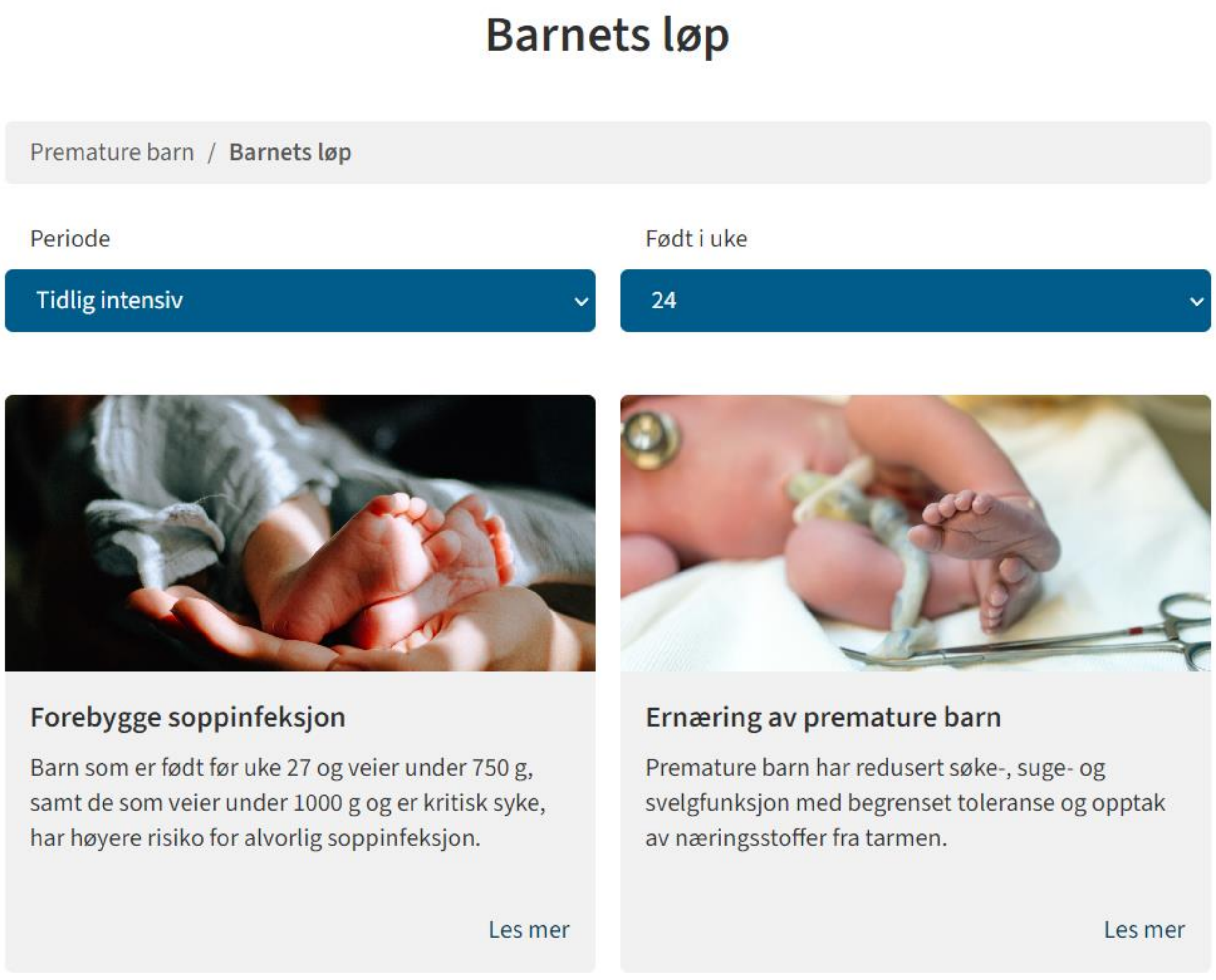
Nekrotiserende EnteroColitt (NEC)

Utforsk årsaker, symptomer og behandlingsmuligheter for Nekrotiserende EnteroColitt, en livstruende tilstand som påvirker premature spedbarn.

Figure 20 - A section with relevant articles

Barnets løp (Child’s journey)

“Barnets løp” is a page that offers a personalized experience for parents of prematurely born children. It features every article from the categories “komplikasjoner”(complications), “ernæring”(nutrition), and “kommunikasjon”(communication). Users can filter these articles based on the current period and the week their baby was born. For instance, a user can set the period to “Early Intensive” and indicate that the baby was born in the 24th week. Consequently, the user will only see articles relevant to these filters. When a user creates an account on the website, they are prompted to specify the week of their baby's birth. This information automatically sets the “Født I uke”(born in week) filter to the registered week by default.


Barnets løp

Premature barn / **Barnets løp**

Periode: Tidlig intensiv

Født i uke: 24

Forebygge soppinfeksjon

Barn som er født før uke 27 og veier under 750 g, samt de som veier under 1000 g og er kritisk syke, har høyere risiko for alvorlig soppinfeksjon.

Les mer

Ernæring av premature barn

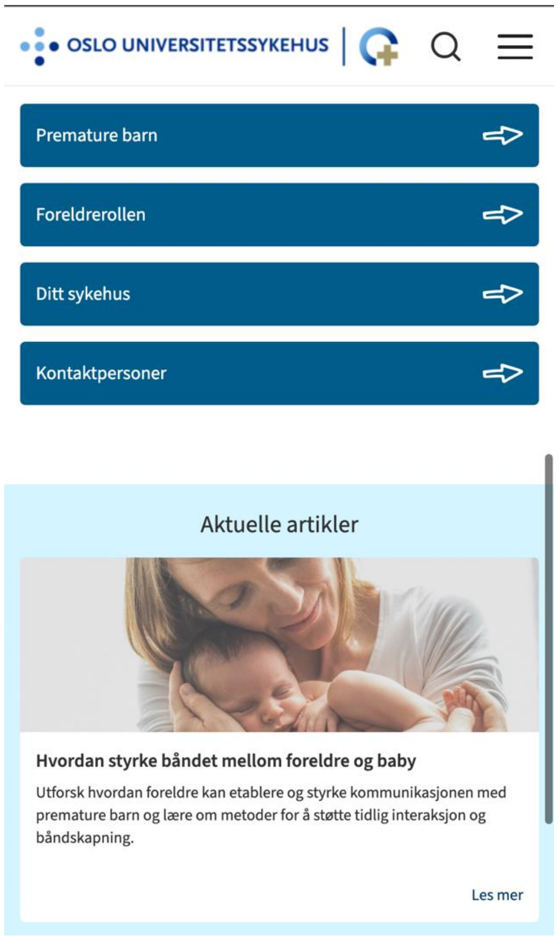
Premature barn har redusert søke-, suge- og svelgfunksjon med begrenset toleranse og opptak av næringsstoffer fra tarmen.

Les mer

Figure 21 - Barnets løp (Child's journey)

Responsiveness

The fully responsive website adapts to devices from 320px wide and up, offering smooth functionality on different screens. CSS media queries adjust elements like buttons and article thumbnails at 768px wide, while CSS grid organizes content into dynamic columns based on screen size.



Hvordan styrke båndet mellom foreldre og baby

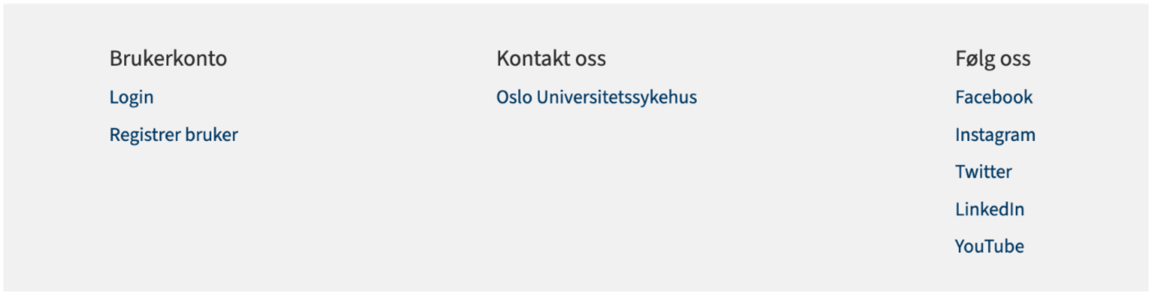
Utforsk hvordan foreldre kan etablere og styrke kommunikasjonen med premature barn og lære om metoder for å støtte tidlig interaksjon og båndskapning.

Les mer

Figure 22 - Homepage, Mobile view

Footer

A footer is present across every page. This section includes user management links (login, log out, register), contact info for Oslo Universitetssykehus, and its social media connections.

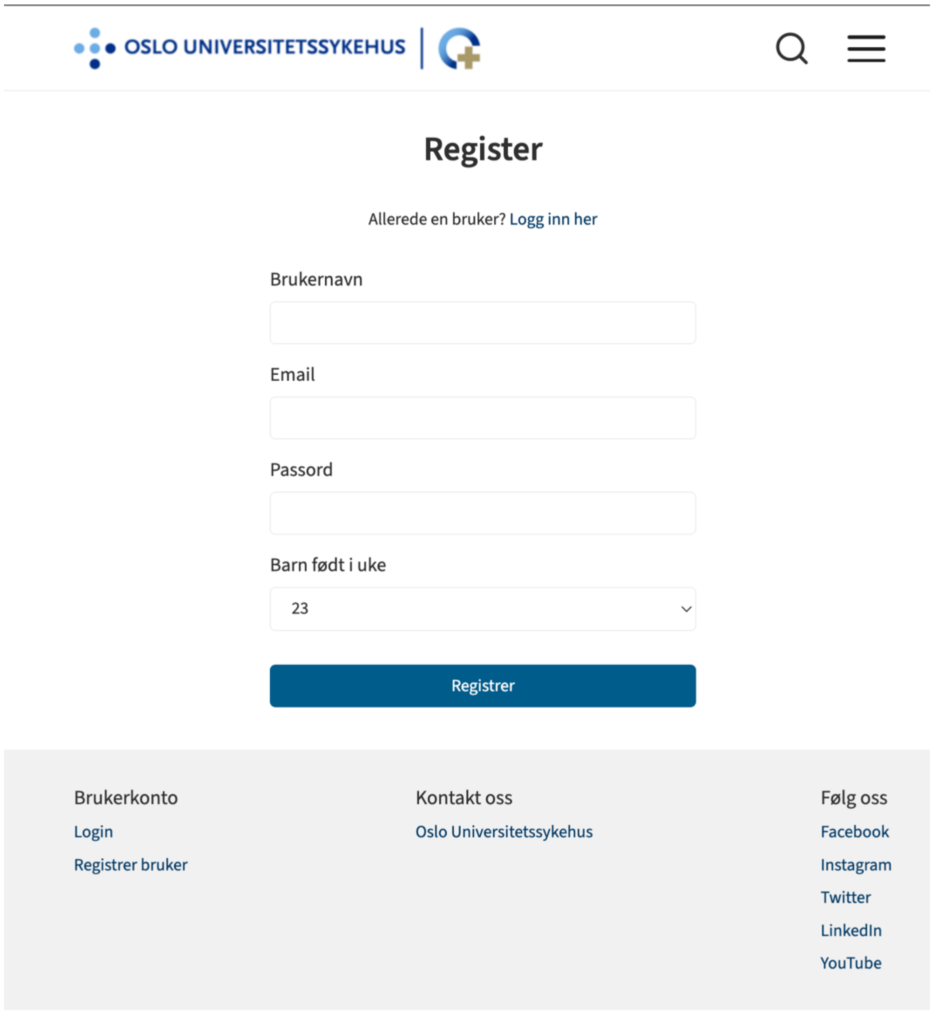


Brukerkonto	Kontakt oss	Følg oss
Login	Oslo Universitetssykehus	Facebook
Registrer bruker		Instagram
		Twitter
		LinkedIn
		YouTube

Figure 23 - Footer

User registration and login

Upon registering, users can input their premature baby's birth week. This is saved in AuthContext, enabling the "barnets løp" page to set the default filter. Users can also save articles as favorites, which are associated with their document in the database, though favorites display on the webpage is not yet available.



OSLO UNIVERSITETSSYKEHUS

Register

Allerede en bruker? Logg inn her

Brukernavn

Email

Passord

Barn født i uke

23

Registrer

Brukerkonto
Login
Registrer bruker

Kontakt oss
Oslo Universitetssykehus

Følg oss
Facebook
Instagram
Twitter
LinkedIn
YouTube

Figure 24 – User registration

4.2.4 Frontend summary

The frontend of the web application was built using React, SCSS, and Vite. Key dependencies include axios, jwt-decode, moment, react-router-dom, and sass. The frontend is organized with Vite's default folder structure, while React Router enables client-side routing for seamless navigation.

SCSS is used to manage styles, with separate files for color and spacing variables. Axios connects the frontend to the backend, handling data retrieval and storage in MongoDB. An authentication context, provided by AuthProvider, offers various functionalities like login, logout, registration, and validation.

The finished design includes a homepage, articles, search functionality, navigation, an "aktuelle artikler" section, responsiveness, a footer, and user registration/login capabilities, catering to end-users like parents of premature babies.

4.3 Backend

The backend of the application refers to the server-side of the application, which is responsible for managing the logic and data of the application. It is built using Node.js, which is a server-side JavaScript runtime environment (Node, 2023).

4.3.1 Backend technologies

Node.js is an asynchronous event-driven JavaScript runtime, *Node.js* is designed to build scalable network applications (Node, 2023). Simplified, Node.js is JavaScript that runs outside the browser (Subramanian, 2019, p. 7).

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications (Express, 2023). To summarize, the framework is a web server framework specifically for Node.js and it is not vastly different from other server-side frameworks (Subramanian, 2019, p. 9).

MongoDB is a non-relational document-oriented database that uses a JSON Object for CRUD (Create, Read, Update and Delete) operations (Subramanian, 2019, p. 10).

Database

The data structure of a web application is complex since it requires balancing the needs of the application, the performance characteristics of the database engine and the data retrieval patterns (MongoDB, Inc, 2023).

A key decision in designing data models for MongoDB applications is how the application represents relationships between data and the structure of documents. The data can be embedded. Embedded database structure store data relationships in a single document. The data can also be manually referenced in other documents, collections, and databases. The application then runs a second query to resolve the referenced fields (MongoDB, Inc, 2023).

4.3.2 Setting up the backend

Defining a database structure

As described in *Database*, the task of structuring a database is acknowledged to be challenging due to the need to simultaneously address the performance demands of the application and ensure a balance with its usage. Five of MongoDB's rules have been followed to structure the database (Karlsson, 2022).

The database structure is divided into five schemas. The schemas were created using Mongoose[4], a popular Object Data Modeling (ODM) library for Node.js that provides a straightforward way to interact with MongoDB databases. Each collection in the database defines the structure of each document stored within that collection. Below is a list of the schemas:

1. users
2. refreshtokens
3. media
4. categories
5. articles

Appendix 3 - Database schemas provide a detailed overview of the database's documents, fields, and data types. To summarize, the user-schema contains user specific information. The refreshToken schema contains information about the refresh token and is used for authentication; IP addresses are recorded against the token to help identify any anomalous or malicious activities. The article schema contains article-specific information. The category schema contains the various categories the article can have, and the media schema contains media-specific information. See *Figure 25 - DB-schema visualization* for a visualization of the database structure.

[4] Mongoose Documentation: https://mongoosejs.com/docs/

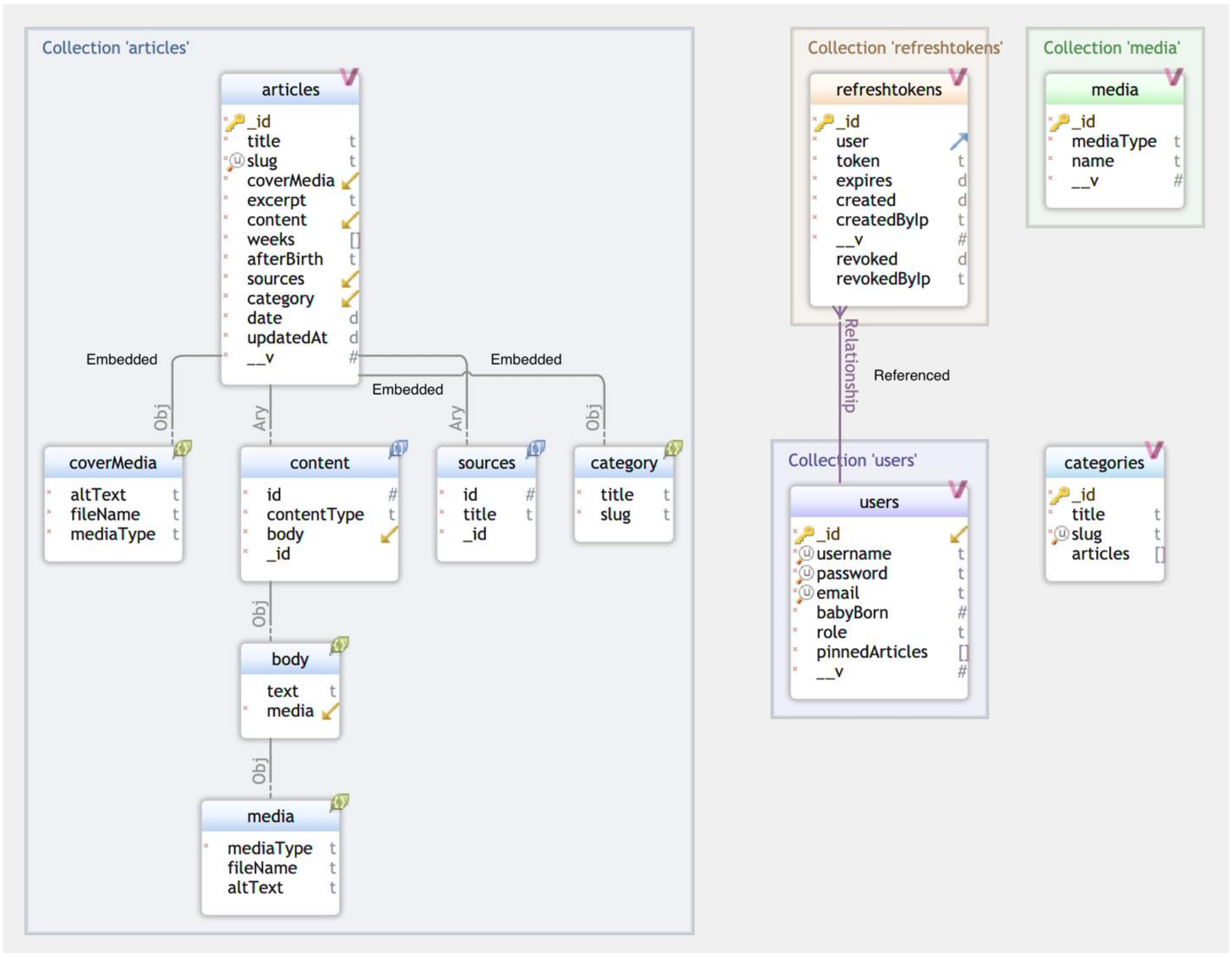


Figure 25 - DB-schema visualization

The following database structure choices were made following the five guidelines advocated by MongoDB (Karlsson, 2022):

Favor embedding

The content of each article is embedded inside the article schema. When accessing an article, the content is also needed – the same applies to sources where it is embedded also in the article schema.

Limit embedding

The need for accessing each category without accessing the list of articles under each category necessitated the extraction of the category from the article schema.

No JOIN or $lookup

With the current database structure, there is no need to utilize JOIN or $lookup.

Limit arrays

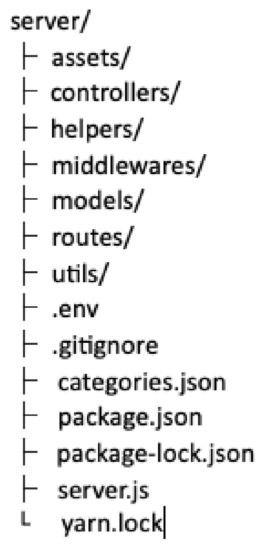
The only array that grows without bound is the articles inside the category, since there is no limit to the amounts of articles can have one category.

Defining a file structure

The file structure, see *Figure 26 - Backend file structure*, has been divided into 7 folders:

1. assets
2. controllers
3. helpers
4. middlewares
5. models
6. routes
7. utils

The “assets” folder contains the images from the image uploads from the CMS. The “controllers”- folder contains five controller files that control the logic for handling route requests and has matching files for each of the endpoints.



```
server/
 ├ assets/
 ├ controllers/
 ├ helpers/
 ├ middlewares/
 ├ models/
 ├ routes/
 ├ utils/
 ├ .env
 ├ .gitignore
 ├  categories.json
 ├  package.json
 ├  package-lock.json
 ├  server.js
 └   yarn.lock
```

Figure 26 - Backend file structure

The “*helpers*”- folder contains a single file, “*functions.js*”. This file exports various helper functions that are utilized throughout the project. The “*middlewares*”- folder contains five middleware files: “*authorize.middleware.js*”, “*authUser.js*”, “*hasCategoryQuery.middleware.js*”, “*hasMediaTypeQuery.middleware.js*” and “*role.middleware.js*”. The first file is used to authorize a single route endpoint. The second file is used to authorize a user when updating/changing their password. The third and fourth files are used to check if a route has a query in the URL, and if it does not it adds a default query. Lastly, the fifth file is used only to enable users with a specific role to access an endpoint.

The ”*models*”- folder contains all the schemas that define the structure and content of the database, further described in Defining a database structure. The “*routes*”- folder contains all the route specific information. The “*utils*”- folder contains three helper/utility function files: “*connectDB.js*,” “*roles.js*” and “*upload.js*”. The first utility function is used to connect to a MongoDB database. The second helper/utility function operates as an enumeration of each of the available roles used for authorization. The third utility function is used to upload the images/videos to the file system.

4.4 Content Management System (CMS)

A custom Content Management System (CMS) has been developed for this project to facilitate efficient management, organization, and publishing of content on the web application. The CMS enables non-technical users to create, edit, and maintain content without requiring extensive knowledge of web development (Barker, 2016). It is designed as a Single Page Application (SPA) using the Quasar framework, resulting in a Minimum Viable Product (MVP) that meets the project's requirements.

The custom CMS consists of six pages:

- **Login**: The login page enables users to authenticate themselves to access the CMS. It is the only public route in the application.
- **Oversikt**: This page provides an overview of the available pages, offering easy navigation for users.
- **Innholdsbehandler**: This page allows editors to create new articles for the main application through a four-step process: configuring settings, uploading the main picture, and adding title and description, adding sections (text, media, or text & media), and adding sources and publishing the article.
- **Eksisterende innhold**: This page displays previously created articles, with options to edit or delete them.
- **Mediebibliotek**: This page serves as a media library, displaying all previously uploaded images and videos, with an option to upload new media.
- **Administrer brukere**: Accessible only to users with the "*superadmin*" role, this page enables the creation, editing, and deletion of editor accounts.

A headless CMS is a modern approach to web development that separates the presentation layer (frontend) from the content management layer (backend), allowing developers to build custom frontend applications that access a database through APIs. This decoupling provides increased flexibility, scalability, and performance by enabling developers to choose their preferred frontend frameworks and tools (Oracle, 2023).

4.4.1 Technologies

This section will provide a brief explanation of the technologies used within development of the CMS, and the reason for using these technologies.

- **Vue 3:** Vue 3 is one of the most performant mainstream frontend frameworks, outperforming Angular and React in the js-framework-benchmark (js-framework-benchmark, 2023). By utilizing the Composition API, Vue 3 allows for TypeScript integration ensuring scaling capabilities (Vue.js, 2023).
- **Quasar Framework:** Quasar is an enterprise-ready cross-platform VueJs framework, containing a library of more than 70 Material Design web components which allows for quickly creating responsive web applications (Quasar, 2023).
- **Pinia Store:** Incorporating Pinia, the recommended state management library for Vue applications, provides essential features such as stronger conventions for team collaboration, Vue DevTools integration, Hot Module Replacement, and Server-Side Rendering support, while offering a simpler API and superior type inference support when used with TypeScript as compared to its predecessor, Vuex (Vue.js, 2023).
- **TypeScript:** TypeScript, a strongly typed programming language that builds upon JavaScript, provides enhanced tooling for better development at any scale by adding additional syntax for types, enabling early error detection, and ensuring compatibility with JavaScript environments such as browsers or Node.js (TypeScript, 2023).

The rationale for the chosen technologies can be summarized into the following key points:

- **Custom CMS:** While existing CMS solutions were considered, the decision to develop a custom CMS aimed to enhance the learning outcomes of this bachelor thesis and provide a tailored solution specific to the project's requirements.

- **Vue 3:** Vue 3 with the Composition API was selected over *React.js* to gain experience with an alternative syntax, as well as to leverage its modern and performant nature, making it more comparable to Angular and Svelte.
- **Quasar Framework:** The choice of a component library was motivated by the desire to simulate a professional working environment. Quasar was chosen over Vuetify[5], another popular Material Design based library for Vue (Vue Community, 2023), due to its comprehensive set of components, including those required for text editing, which Vuetify lacked.

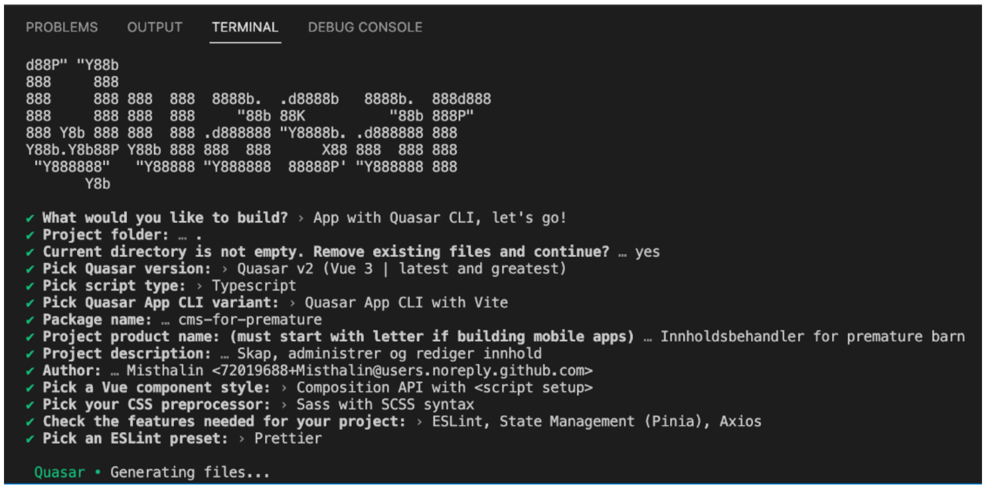
The technology choices were also driven by efficiency, scalability, and time constraints. Vue 3's performance and Quasar's component library enabled fast development. Pinia Store managed state effectively, and TypeScript ensured type safety, contributing to a maintainable and stable codebase. These technologies offered a solid foundation for the CMS, fulfilling project requirements and supporting future maintenance and growth.

[5] https://vuetifyjs.com/en/

4.4.2 Initial setup

The initial configuration process was carried out through the Quasar Command Line Interface (CLI) by executing the following commands:

```
$ yarn global add @quasar/cli
$ yarn create quasar
```

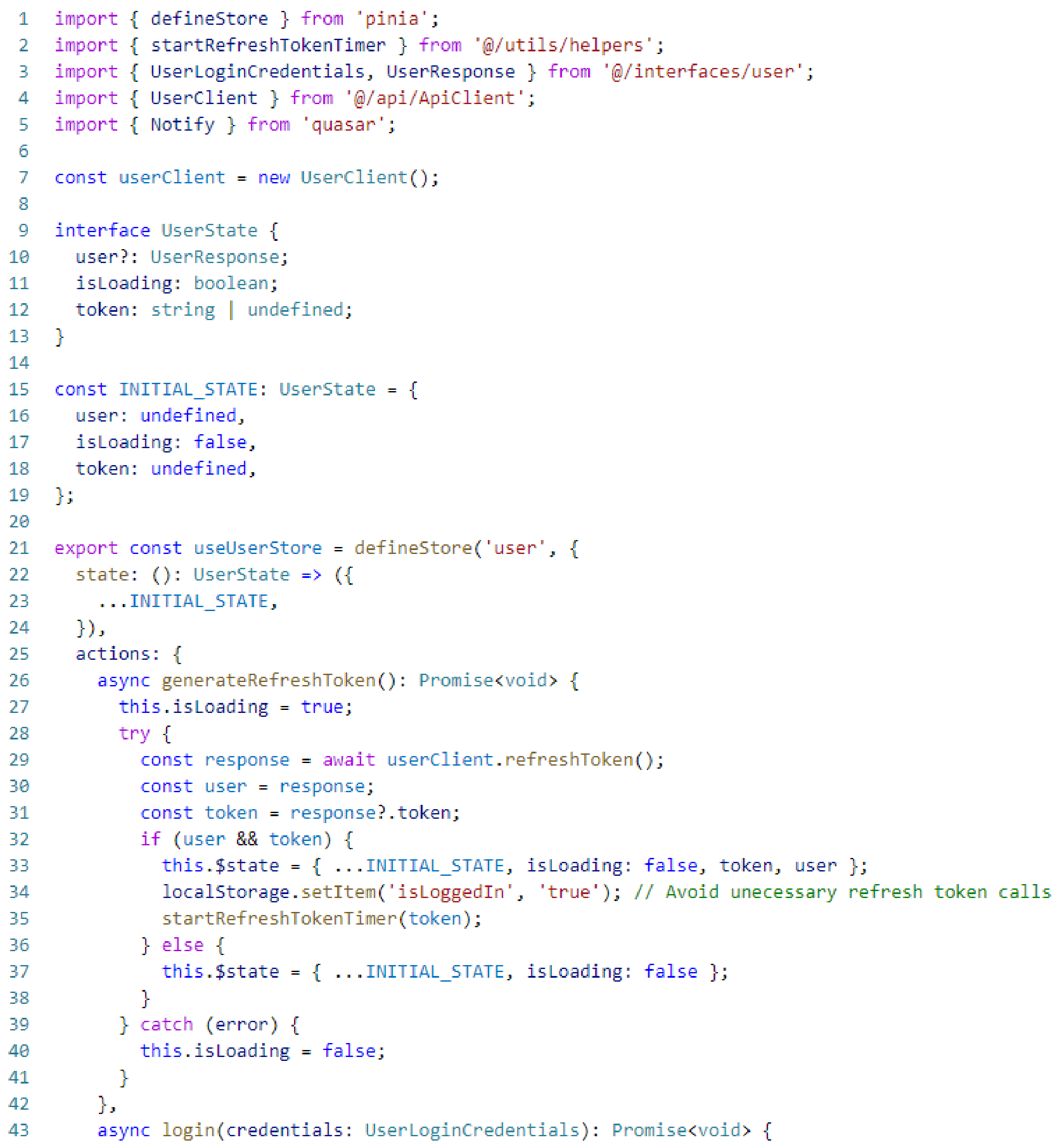


```
d88P" "Y88b
888     888
888     888 888  888  8888b.  .d8888b   8888b.  888d888
888     888 888  888     "88b 88K          "88b 888P"
888 Y8b 888 888  888 .d888888 "Y8888b. .d888888 888
Y88b.Y8b88P Y88b 888 888  888      X88 888  888 888
 "Y888888"   "Y88888 "Y888888  88888P' "Y888888 888
       Y8b

✔ What would you like to build? › App with Quasar CLI, let's go!
✔ Project folder: … .
✔ Current directory is not empty. Remove existing files and continue? … yes
✔ Pick Quasar version: › Quasar v2 (Vue 3 | latest and greatest)
✔ Pick script type: › Typescript
✔ Pick Quasar App CLI variant: › Quasar App CLI with Vite
✔ Package name: … cms-for-premature
✔ Project product name: (must start with letter if building mobile apps) … Innholdsbehandler for premature barn
✔ Project description: … Skap, administrer og rediger innhold
✔ Author: … Misthalin <72019688+Misthalin@users.noreply.github.com>
✔ Pick a Vue component style: › Composition API with <script setup>
✔ Pick your CSS preprocessor: › Sass with SCSS syntax
✔ Check the features needed for your project: › ESLint, State Management (Pinia), Axios
✔ Pick an ESLint preset: › Prettier

 Quasar • Generating files...
```

Figure 27 - CMS Setup, Quasar CLI configuration

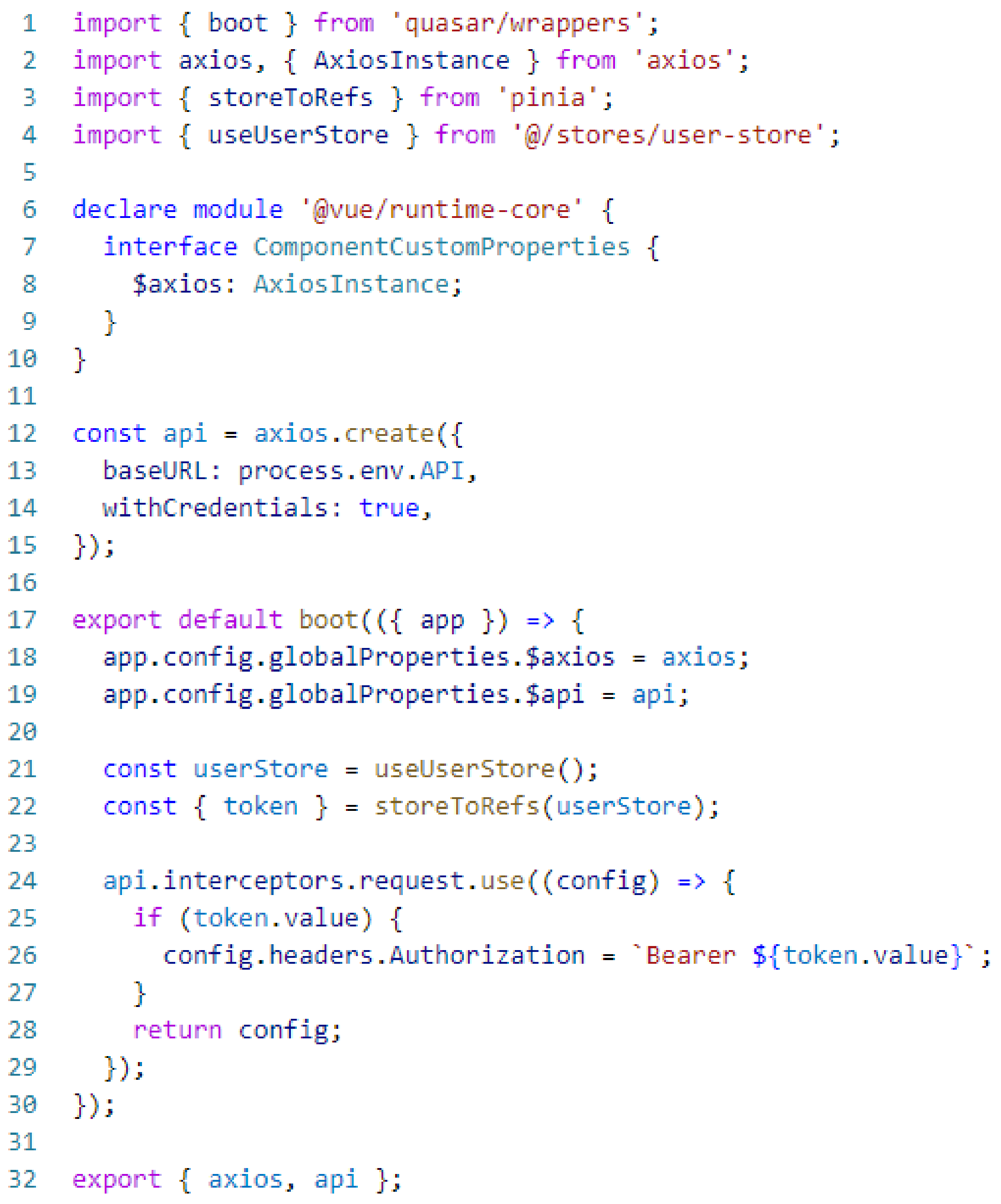
During the Quasar CLI configuration (as shown in Figure 27), the project was set up to utilize Vue 3 with the Composition API, TypeScript, Pinia for state management, Axios for API calls, and Sass with SCSS syntax. Upon completing the project installation, additional configurations for "*.editorconfig*", ESLint, and Prettier were implemented to ensure consistent coding style and automatic formatting upon saving changes.

With the project installation and configuration completed, the development phase began. The initial step involved creating all essential pages, along with establishing the corresponding routing setup. Following this, the integration of "*user-store.ts*", "*axios.ts*", and "*ApiClient.ts*" facilitated user login and communication with the backend. The "*user-store.ts*" file (Figure 28) manages user authentication and state, the "*axios.ts*" file (Figure 29) handles HTTP requests, and the "*ApiClient.ts*" file (Figure 30) serves as an interface to interact with the backend API.



```typescript
import { defineStore } from 'pinia';
import { startRefreshTokenTimer } from '@/utils/helpers';
import { UserLoginCredentials, UserResponse } from '@/interfaces/user';
import { UserClient } from '@/api/ApiClient';
import { Notify } from 'quasar';

const userClient = new UserClient();

interface UserState {
  user?: UserResponse;
  isLoading: boolean;
  token: string | undefined;
}

const INITIAL_STATE: UserState = {
  user: undefined,
  isLoading: false,
  token: undefined,
};

export const useUserStore = defineStore('user', {
  state: (): UserState => ({
    ...INITIAL_STATE,
  }),
  actions: {
    async generateRefreshToken(): Promise<void> {
      this.isLoading = true;
      try {
        const response = await userClient.refreshToken();
        const user = response;
        const token = response?.token;
        if (user && token) {
          this.$state = { ...INITIAL_STATE, isLoading: false, token, user };
          localStorage.setItem('isLoggedIn', 'true'); // Avoid unecessary refresh token calls
          startRefreshTokenTimer(token);
        } else {
          this.$state = { ...INITIAL_STATE, isLoading: false };
        }
      } catch (error) {
        this.isLoading = false;
      }
    },
    async login(credentials: UserLoginCredentials): Promise<void> {
```

Figure 28 - user-store.ts excerpt. Global state managing user authentication.



```
import { boot } from 'quasar/wrappers';
import axios, { AxiosInstance } from 'axios';
import { storeToRefs } from 'pinia';
import { useUserStore } from '@/stores/user-store';

declare module '@vue/runtime-core' {
  interface ComponentCustomProperties {
    $axios: AxiosInstance;
  }
}

const api = axios.create({
  baseURL: process.env.API,
  withCredentials: true,
});

export default boot(({ app }) => {
  app.config.globalProperties.$axios = axios;
  app.config.globalProperties.$api = api;

  const userStore = useUserStore();
  const { token } = storeToRefs(userStore);

  api.interceptors.request.use((config) => {
    if (token.value) {
      config.headers.Authorization = `Bearer ${token.value}`;
    }
    return config;
  });
});

export { axios, api };
```

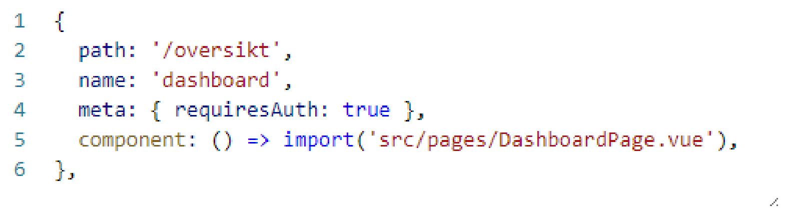
Figure 29 - axios.ts: Facilitates communication with the backend, ensuring the users authorization token gets sent as Authorization header for each request.

```
export class UserClient {
  async login(
    credentials: UserLoginCredentials
  ): Promise<UserResponse | undefined> {
    try {
      const response = await api.post('/login', credentials);
      if (response.data && response.data.token) {
        return response.data as UserResponse;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to log in: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }

  async revokeToken(token: string | undefined): Promise<void> {
    try {
      await api.post('/revoke-token', { token });
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to revoke token: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }

  async refreshToken(): Promise<UserResponse | undefined> {
    // Avoid unecessary refresh token calls
    if (localStorage.getItem('isLoggedIn') !== 'true') {
      return undefined;
    }

    try {
      const response = await api.post('/refresh-token');
      if (response.data && response.data.token) {
        return response.data as UserResponse;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to refresh token: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }
```

Figure 30 - ApiClient.ts, interface to communicate with the backend

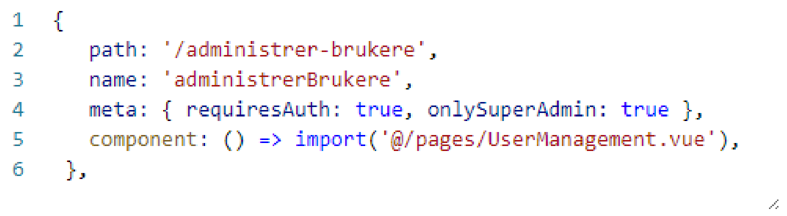
Upon integrating the login functionality, route protection was implemented to ensure proper access control. This was achieved by adding a "*meta: { requiresAuth: true }"* property to each route that necessitated authentication and an "*onlySuperAdmin: true*" property to routes restricted to the super admin role. For public routes, "*requiresAuth*" was set to false. A function was then created to check the "*user-store"* for authentication and role information, redirecting users accordingly based on their status.

The code snippets below demonstrate the three different types of routes: one that requires authentication (Figure 31), one that is restricted to the super admin role (Figure 32), and a public route (Figure 33).



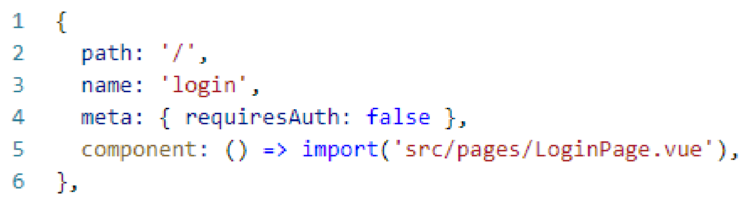
```
{
  path: '/oversikt',
  name: 'dashboard',
  meta: { requiresAuth: true },
  component: () => import('src/pages/DashboardPage.vue'),
},
```

Figure 31 - Route protection, requiresAuth



```
{
   path: '/administrer-brukere',
   name: 'administrerBrukere',
   meta: { requiresAuth: true, onlySuperAdmin: true },
   component: () => import('@/pages/UserManagement.vue'),
 },
```

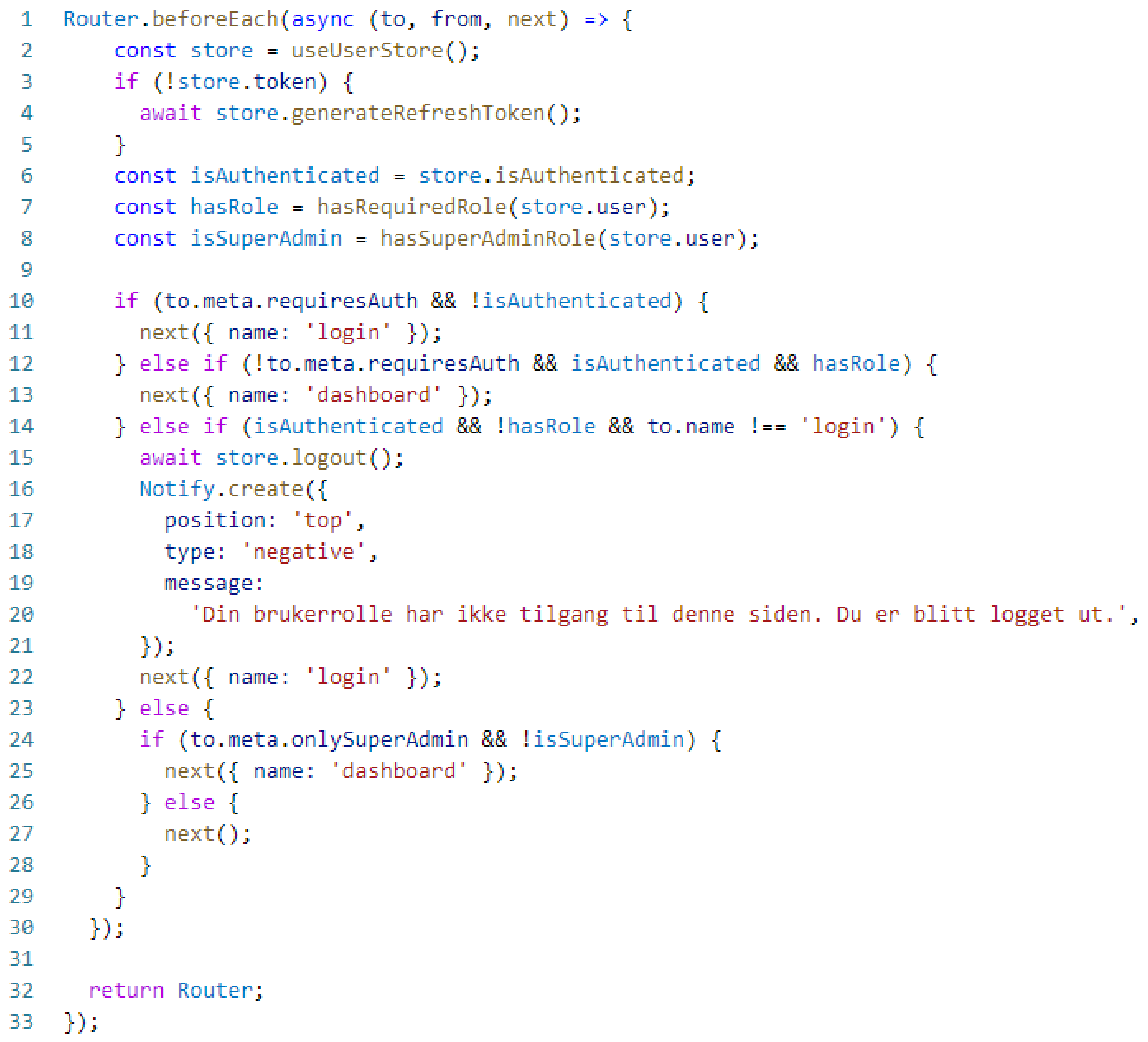
Figure 32 - Route protection, onlySuperAdmin



```
{
  path: '/',
  name: 'login',
  meta: { requiresAuth: false },
  component: () => import('src/pages/LoginPage.vue'),
},
```

Figure 33 - Route protection, public route

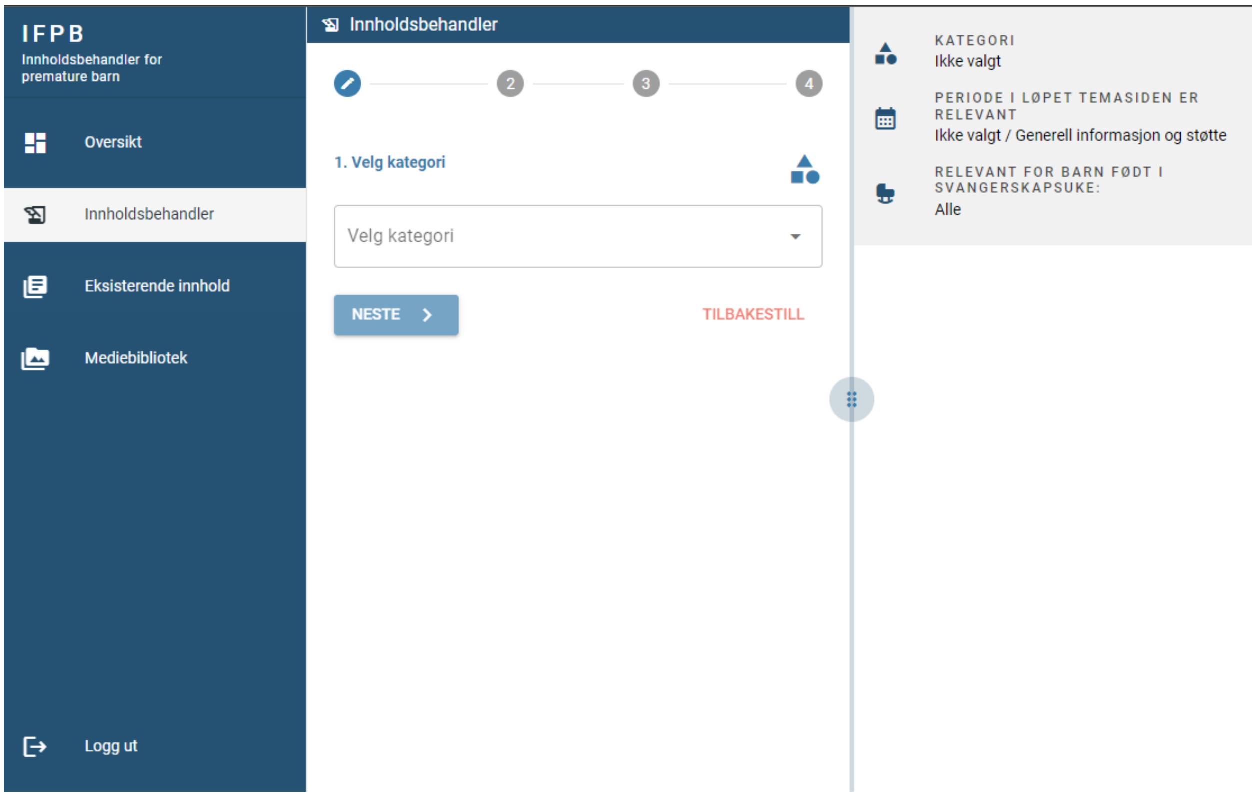
In addition to the route configurations, a “*beforeEach”* function was added to the router, as shown in Figure 34. This function checks the “*user-store”* for authentication and role information before allowing access to a route, ensuring proper access control is maintained.



```
Router.beforeEach(async (to, from, next) => {
    const store = useUserStore();
    if (!store.token) {
      await store.generateRefreshToken();
    }
    const isAuthenticated = store.isAuthenticated;
    const hasRole = hasRequiredRole(store.user);
    const isSuperAdmin = hasSuperAdminRole(store.user);

    if (to.meta.requiresAuth && !isAuthenticated) {
      next({ name: 'login' });
    } else if (!to.meta.requiresAuth && isAuthenticated && hasRole) {
      next({ name: 'dashboard' });
    } else if (isAuthenticated && !hasRole && to.name !== 'login') {
      await store.logout();
      Notify.create({
        position: 'top',
        type: 'negative',
        message:
          'Din brukerrolle har ikke tilgang til denne siden. Du er blitt logget ut.',
      });
      next({ name: 'login' });
    } else {
      if (to.meta.onlySuperAdmin && !isSuperAdmin) {
        next({ name: 'dashboard' });
      } else {
        next();
      }
    }
  });

  return Router;
});
```

Figure 34 - Route protection, before each route

4.4.3 ContentManager.vue / “Innholdsbehandler”



KATEGORI
Ikke valgt

PERIODE I LØPET TEMASIDEN ER RELEVANT
Ikke valgt / Generell informasjon og støtte

RELEVANT FOR BARN FØDT I SVANGERSKAPSUKE:
Alle

Figure 35 - ContentManager.vue initial view

Upon successful implementation of routing and login functionality, the development of the “*ContentManager.vue*” page, referred to as “Innholdsbehandler” in Norwegian, was initiated.

Keeping track of changes

The first objective was to make a store to keep track of the articles during their creation process. To achieve this, an article-store, “*article-store.ts*” was created and later iterated using Pinia store. The default state contains an empty article, which is updated as the creation progresses (see *Appendix 5 - article-store.ts).*

The “*article-store.ts*” contains the implementation of the Pinia store for managing articles in progress. It starts with importing the required modules and defining the default empty article object (EMPTY_ARTICLE). Following that, the

"*ExpandedSections*" type is defined, which is an object containing Boolean values for different sections.

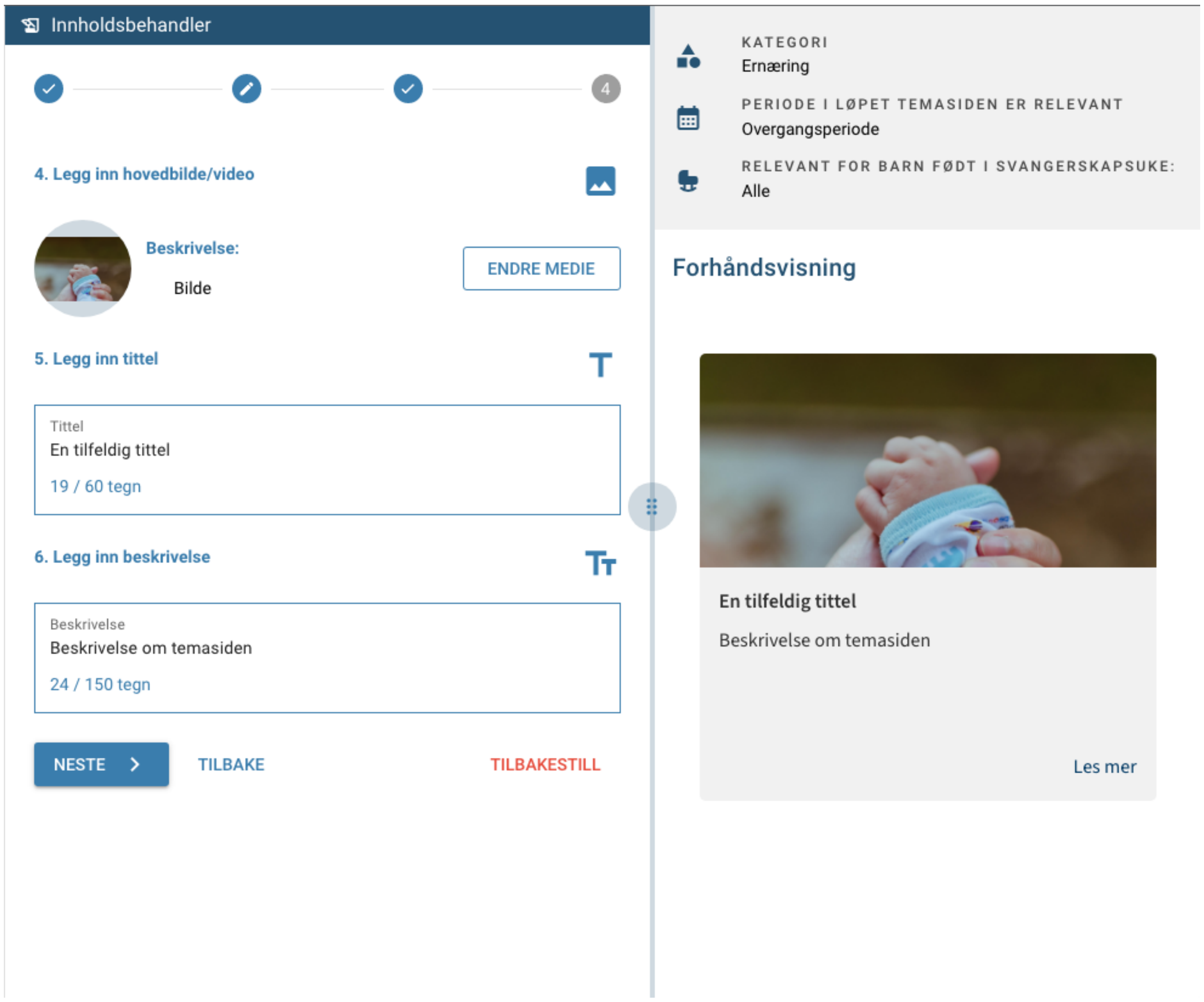
The "*useArticleStore*" function is then defined, setting up the store's state, actions, and getters. The state holds information about the current step, editing status, the current article, the original article, and expanded sections. Actions include functions for toggling and setting section expansion, as well as resetting the store to its initial state. The store also includes a getter to check if there are any unsaved changes in the current article.

Lastly, a "*deepCopy*" function is implemented to create deep copies of the objects, as the spread operator (…) does not suffice for deep cloning in this context. This function serves as a simplified version of the lodash[6] *"_.cloneDeep()"* method.

QSplitter with Configuration and Preview

In the "*ContentManager.vue*" page, the QSplitter[7] component is used to enable the user to interactively adjust the size of the configuration and preview sections. By default, the configuration section is displayed on the left side and the preview section on the right side, as shown in Figure 36.

[6] https://lodash.com/
[7] https://quasar.dev/vue-components/splitter#qsplitter-api



4. Legg inn hovedbilde/video

Beskrivelse: Bilde

5. Legg inn tittel

Tittel
En tilfeldig tittel
19 / 60 tegn

6. Legg inn beskrivelse

Beskrivelse
Beskrivelse om temasiden
24 / 150 tegn

KATEGORI
Ernæring

PERIODE I LØPET TEMASIDEN ER RELEVANT
Overgangsperiode

RELEVANT FOR BARN FØDT I SVANGERSKAPSUKE:
Alle

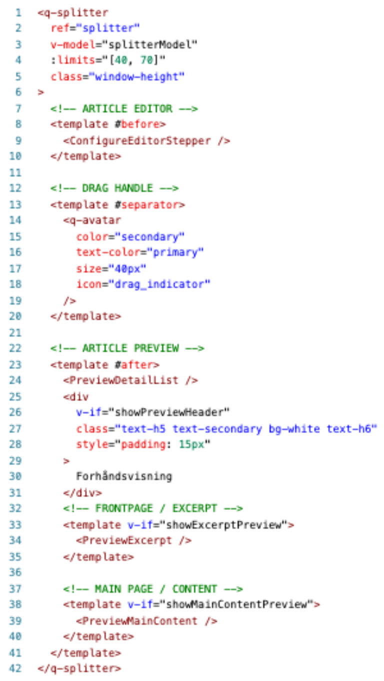
Forhåndsvisning

En tilfeldig tittel

Beskrivelse om temasiden

Figure 36 - QSplitter with Configuration and Preview

The QSplitter component is part of the Quasar Framework and allows for the creation of resizable and flexible layouts by dividing a container into separate, adjustable sections (Quasar, 2023). The primary purpose of implementing QSplitter is to provide an intuitive user interface, allowing users to easily adjust the configuration and preview sections based on their preferences and observe the article's responsiveness across various screen widths while creating or editing content in the CMS.



```
<q-splitter
  ref="splitter"
  v-model="splitterModel"
  :limits="[40, 70]"
  class="window-height"
>
  <!-- ARTICLE EDITOR -->
  <template #before>
    <ConfigureEditorStepper />
  </template>

  <!-- DRAG HANDLE -->
  <template #separator>
    <q-avatar
      color="secondary"
      text-color="primary"
      size="40px"
      icon="drag_indicator"
    />
  </template>

  <!-- ARTICLE PREVIEW -->
  <template #after>
    <PreviewDetailList />
    <div
      v-if="showPreviewHeader"
      class="text-h5 text-secondary bg-white text-h6"
      style="padding: 15px"
    >
      Forhåndsvisning
    </div>
    <!-- FRONTPAGE / EXCERPT -->
    <template v-if="showExcerptPreview">
      <PreviewExcerpt />
    </template>

    <!-- MAIN PAGE / CONTENT -->
    <template v-if="showMainContentPreview">
      <PreviewMainContent />
    </template>
  </template>
</q-splitter>
```

Figure 37 - QSplitter code snippet

The code above creates a QSplitter component, which is bound to a "*splitterModel*" reference that holds the current percentage value of the splitter. The limits prop sets the minimum and maximum limits for the splitter's adjustment, ensuring the sections do not become too small or too large.

Inside the QSplitter component, three slots are defined: before, separator, and after. The before slot contains the configuration section "*ConfigureEditorStepper*", the separator slot contains the drag handle for resizing the sections, and the after slot contains the preview section "*PreviewDetailList*", "*PreviewExcerpt*", and "*PreviewMainContent*". The drag handle is styled using the QAvatar[8] component with an appropriate icon to indicate its purpose.

In conclusion, the QSplitter component is aimed at enhancing the user experience in the "*ContentManager.vue*" page by providing a flexible and adjustable layout for the configuration and preview sections.

[8] https://quasar.dev/vue-components/avatar/

QStepper

The QStepper component, a part of the Quasar Framework, is utilized to guide users through the process of creating or editing articles by dividing the process into multiple, sequential steps (Quasar, 2023). It offers a clear and organized approach to content creation, with the goal of allowing users to easily navigate through the required steps while maintaining a clear understanding of the process.

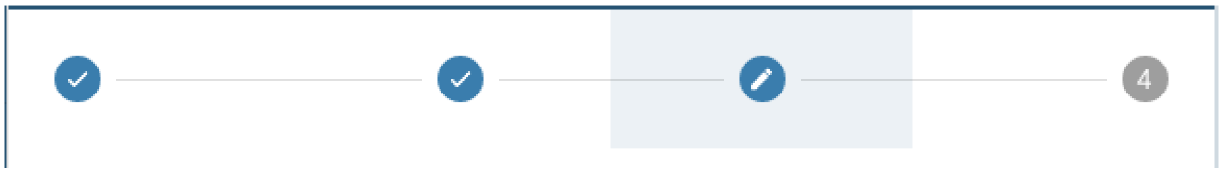


Figure 38 - QStepper example

As illustrated in Figure 38, the QStepper component is integrated into the configuration section of the “*ContentManager.vue*” page. Each step represents a distinct stage in the content creation process, and users are guided through the following four steps:

- **Settings:** In this step, users configure the basic settings of the article, such as its category, which period after birth the article is relevant, and relevancy based on the gestational week a child is born.
- **Main Content:** Users upload the main image for the article and provide a title and description.
- **Sections:** Users add various sections to the article, which may include text, media, or a combination of text and media.
- **Sources and Publishing:** Users input the sources for the article and publishing the article.

Creating and organizing article sections

The creation and organization of article sections are essential components of the content creation process in the “*ContentManager.vue*” page. Users can add, edit, and delete sections, as well as reorder them as needed, providing a flexible approach to content creation.

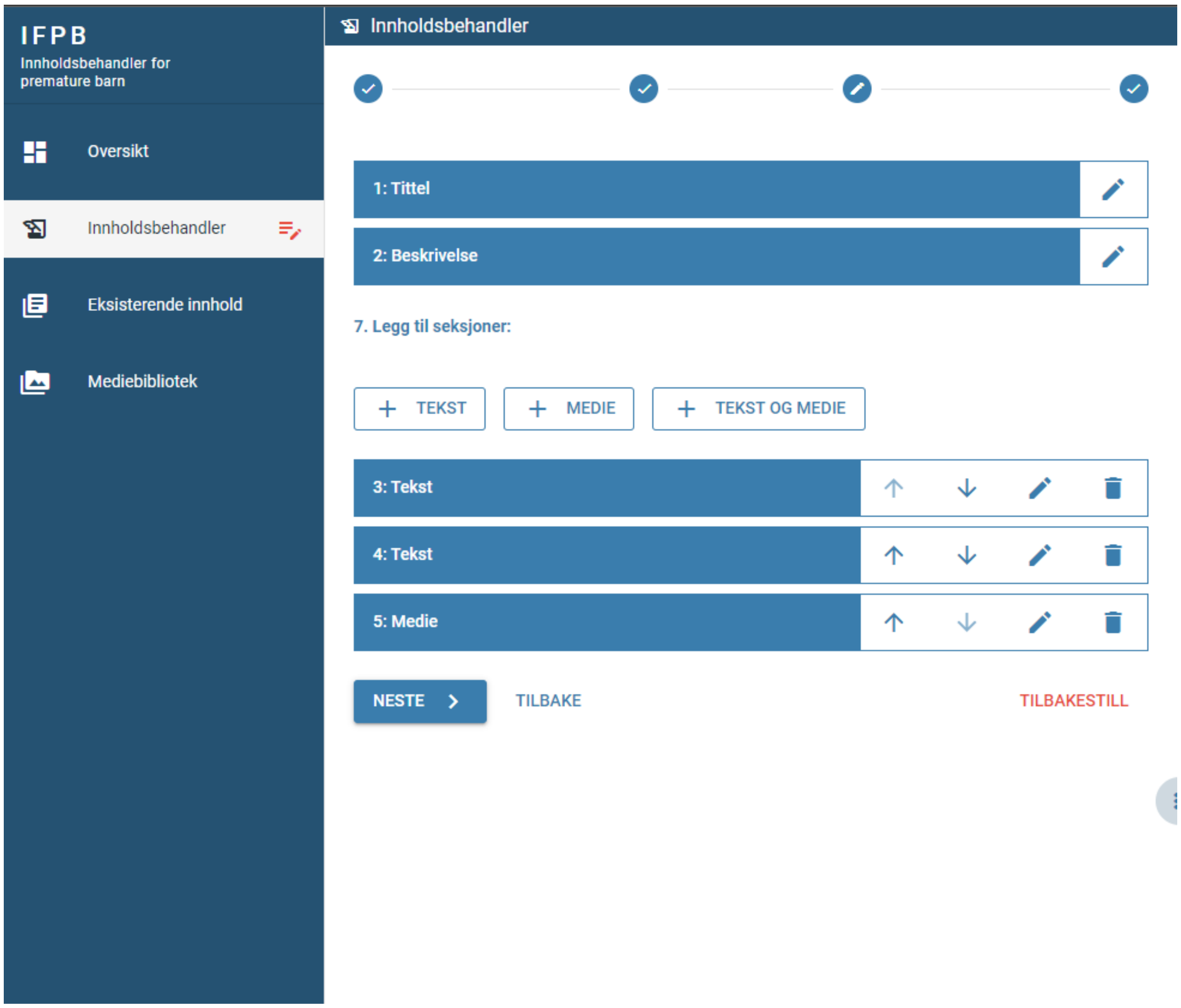


Figure 39 - Article section management

Add sections: Users can add new sections using three distinct buttons – "Text", "Media", and "Text & Media". *(See Appendix 6 - ConfigureMainContentSectionAddButtons.vue).*

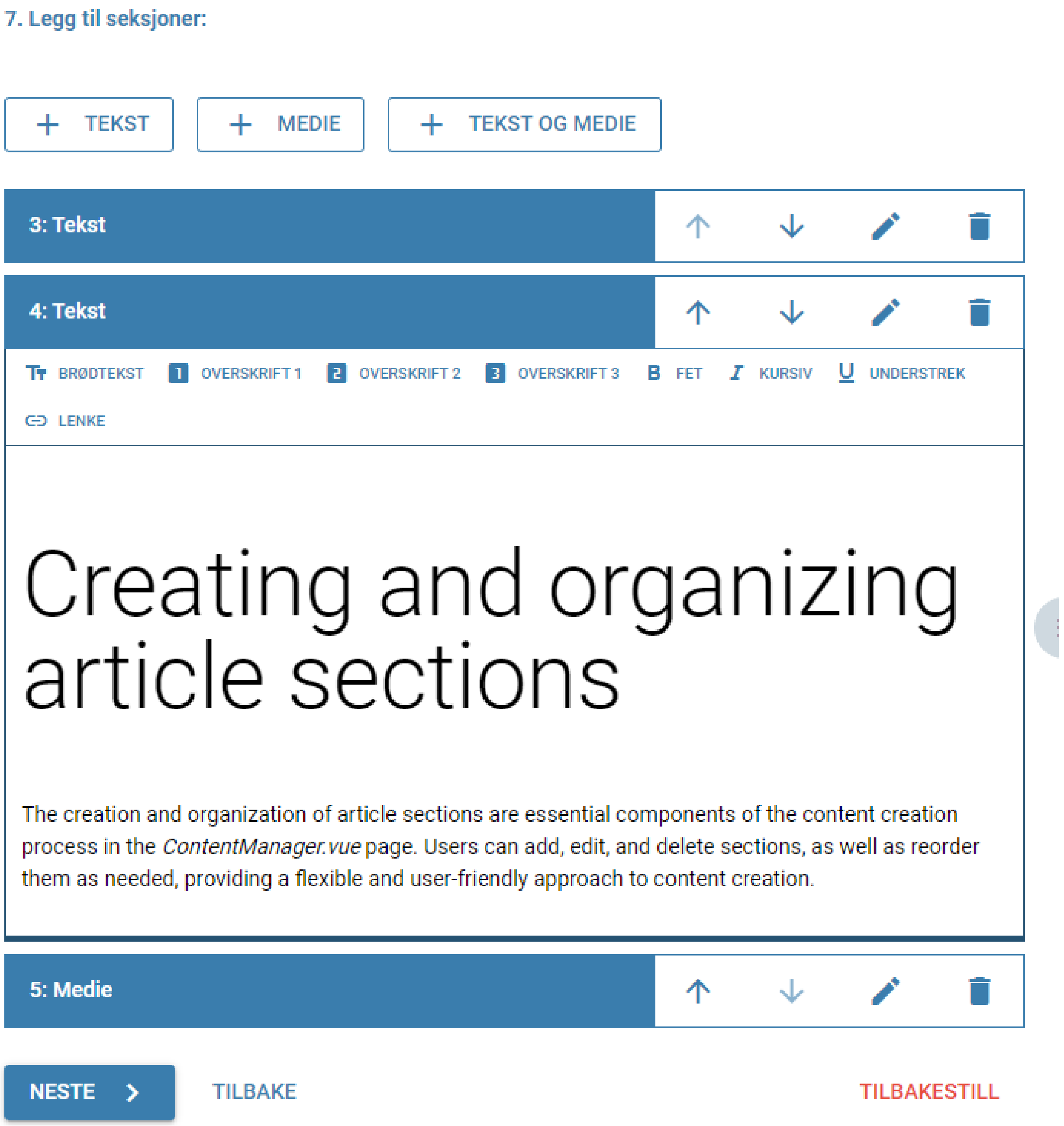
- **"Text":** Clicking this button introduces a new section with a QEditor[9], providing a text editing interface for content input.

[9] https://quasar.dev/vue-components/editor/

- **"Media":** Selecting this option triggers a modal, enabling users to choose their preferred media type (e.g., image or video) and input alternative text for accessibility purposes.

- **"Text & Media":** This choice unveils a modal that permits users to specify the media type, provide alternative text, and utilize a QEditor for text editing. Furthermore, users can personalize the layout by aligning the media to the right or left of the text.

Edit and delete sections: Users can edit or delete existing sections using the appropriate buttons displayed within each section. The edit button expands the section with its current content, facilitating any necessary modifications, or contracts the section if already open. The delete button opens a confirm dialog with the option to remove the section from the article.



Creating and organizing article sections

The creation and organization of article sections are essential components of the content creation process in the *ContentManager.vue* page. Users can add, edit, and delete sections, as well as reorder them as needed, providing a flexible and user-friendly approach to content creation.

Figure 40 - Expanded section (Text)

Reorder sections: Users can reorder sections by using the arrow buttons. The main components involved in this reordering process are:

- ***"article-store.ts":*** This is the store that holds the state and actions related to the articles. It maintains a record of expanded sections using an object called "*expandedSections*". Each key in this object represents a section id, and the corresponding Boolean value indicates whether the section is expanded or not. *(Code preview in Appendix 5).*

- ***"ConfigureMainContentSectionButtons.vue":*** This component defines the buttons responsible for reordering the sections. The "*moveUp*" and "*moveDown*" functions handle the movement of sections by swapping their positions in either the "*content*" or "*sources*" array, depending on the section type. The "*swapItems*" function is responsible for performing the actual swap. *(Code preview in Appendix 9).*

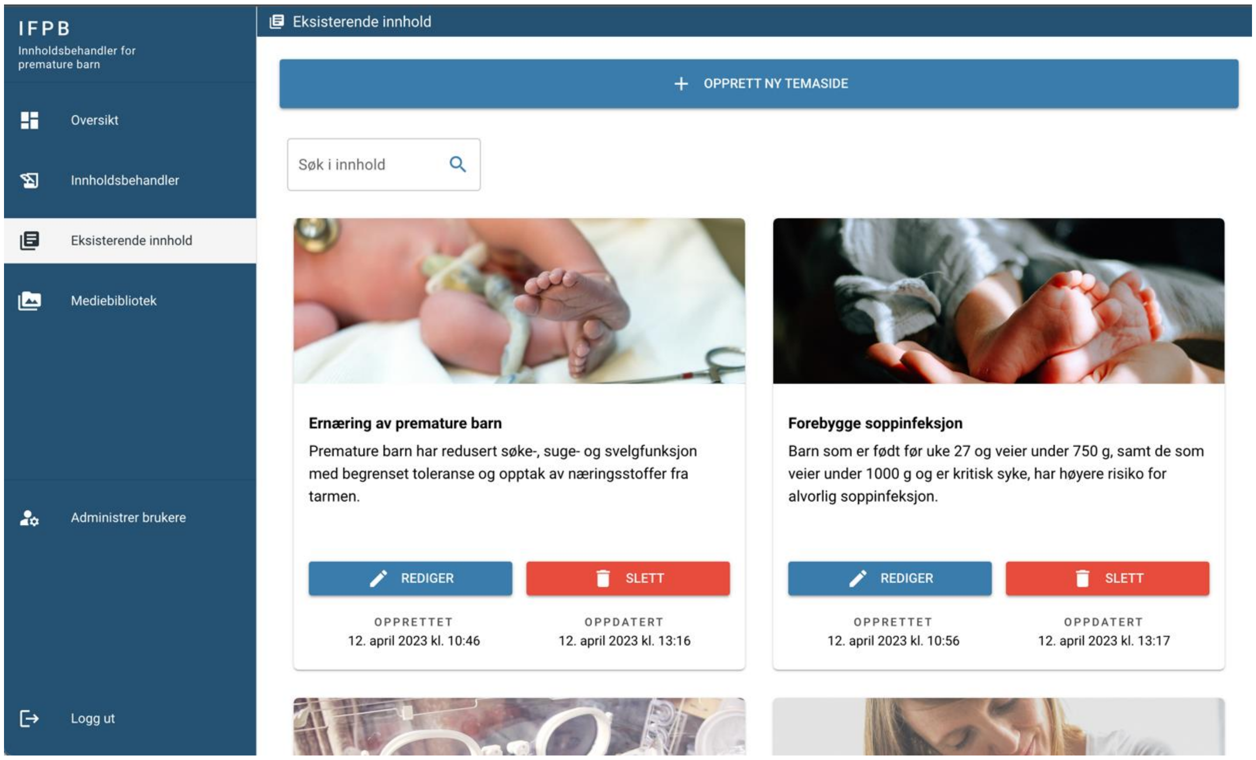
- ***"ConfigureMainContentSection.vue":*** This component renders the individual sections and handles their expansion state. It sets the expansion state of a section using the "*setSectionExpanded*" action from the store when the model value of the expansion item is updated. *(Code preview in Appendix 8).*

- ***"ConfigureMainContent.*vue":** This component renders all the sections and is responsible for organizing the layout of the sections. *(Code preview in Appendix 7).*

The combination of these functionalities allows users to create, edit, and organize article sections efficiently, contributing to an intuitive and streamlined content creation process within the custom CMS.

4.4.4 ContentLibrary.vue / "Eksisterende innhold"

The "*ContentLibrary.vue*" component, referred to as "Eksisterende innhold" in Norwegian, is responsible for displaying a list of previously created articles in the custom CMS. The component provides users with the ability to search for articles, edit, and delete them. Additionally, it contains a button to create a new article that redirects the user to the "*ContentManager.vue*" component for article creation.



Ernæring av premature barn
Premature barn har redusert søke-, suge- og svelgfunksjon med begrenset toleranse og opptak av næringsstoffer fra tarmen.

OPPRETTET 12. april 2023 kl. 10:46 · OPPDATERT 12. april 2023 kl. 13:16

Forebygge soppinfeksjon
Barn som er født før uke 27 og veier under 750 g, samt de som veier under 1000 g og er kritisk syke, har høyere risiko for alvorlig soppinfeksjon.

OPPRETTET 12. april 2023 kl. 10:56 · OPPDATERT 12. april 2023 kl. 13:17

Figure 41 - ContentLibrary.vue

The component uses the Quasar QTable[10] component to display articles in a grid format, with pagination hidden. The QTable component is designed to be responsive, with configuration options to display the table as cards, providing an optimal viewing experience across various devices.

A search input field is provided at the top-left corner of the table, allowing users to filter the displayed articles based on their search query. This functionality is achieved by binding the "filter" property of the QTable component to a Vue ref named "filter". The search input field uses a debounce value of 300 milliseconds to

[10] https://quasar.dev/vue-components/table#qtable-api

ensure a smooth user experience and reduce the strain on the browser during rapid input changes.

The component fetches the list of articles from the API using the “*ArticleClient”* class when the component is mounted. The list of articles is stored in a Vue ref named “*articles”.* A “*loading*” ref is used to indicate whether the component is fetching data from the API. When the “*loading*” ref is set to true, the QTable component will display a loading spinner.

The table columns are configured to display the following information for each article:

- Cover media (image or video)
- Title
- Excerpt
- Creation date
- Last update date

Each article row in the table contains two action buttons: one to edit the article and another to delete it. When a user clicks the edit button, they are redirected to the “*ContentManager.vue”* component, where the selected article’s data is preloaded for editing. If the user has unsaved changes in the “*ContentManager.vue”* component, a confirmation dialog will appear, warning the user that they will lose their unsaved changes if they proceed.

When a user clicks the delete button, a confirmation dialog is shown, asking the user to confirm the deletion. If the user confirms, the article is deleted, and the list of articles is updated.

In summary, the “*ContentLibrary.vue”* component serves as an interface for managing previously created articles, where users can search, edit, and delete articles.

4.4.5 MediaLibrary.vue / “Mediebibliotek”

The “*MediaLibrary.vue”* page, referred to as “Mediebibliotek” in Norwegian, is responsible for managing and displaying the uploaded media files in the CMS, such as images and videos. It provides an interface to view and manage the uploaded media files. The implementation of this page consists of two main components: "*MediaLibrary.vue*" and "*MediaLibraryContentUploader.vue*".

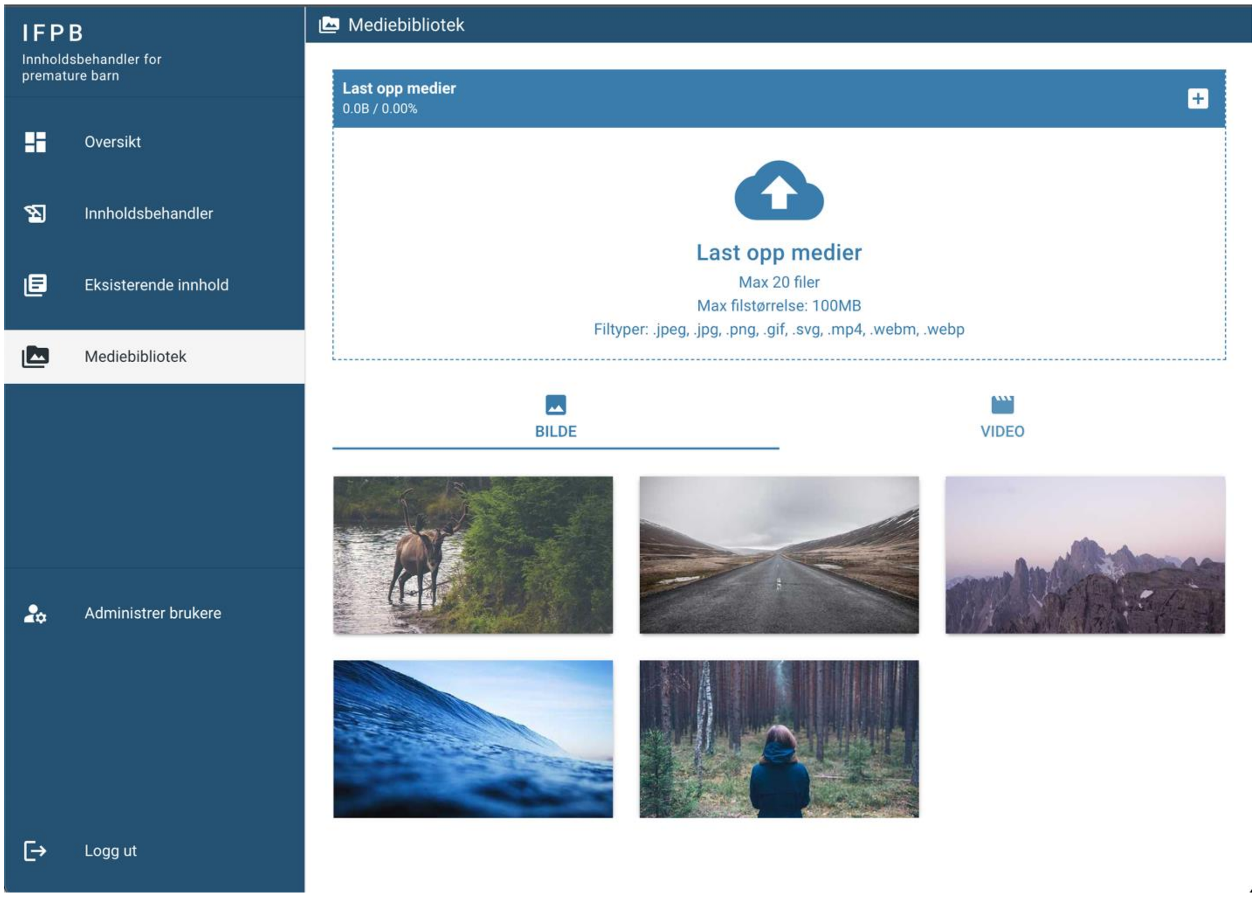


Figure 42 - MediaLibrary.vue

The "*MediaLibrary.vue*" component contains a tabbed interface to separate images and videos, making it easy for users to navigate between different media types. It utilizes the Quasar's QInfiniteScroll[11] component to display a grid of media items,

[11] https://quasar.dev/vue-components/infinite-scroll/

providing a smooth browsing experience for users. The media items are filtered based on the selected tab, either showing images or videos accordingly.

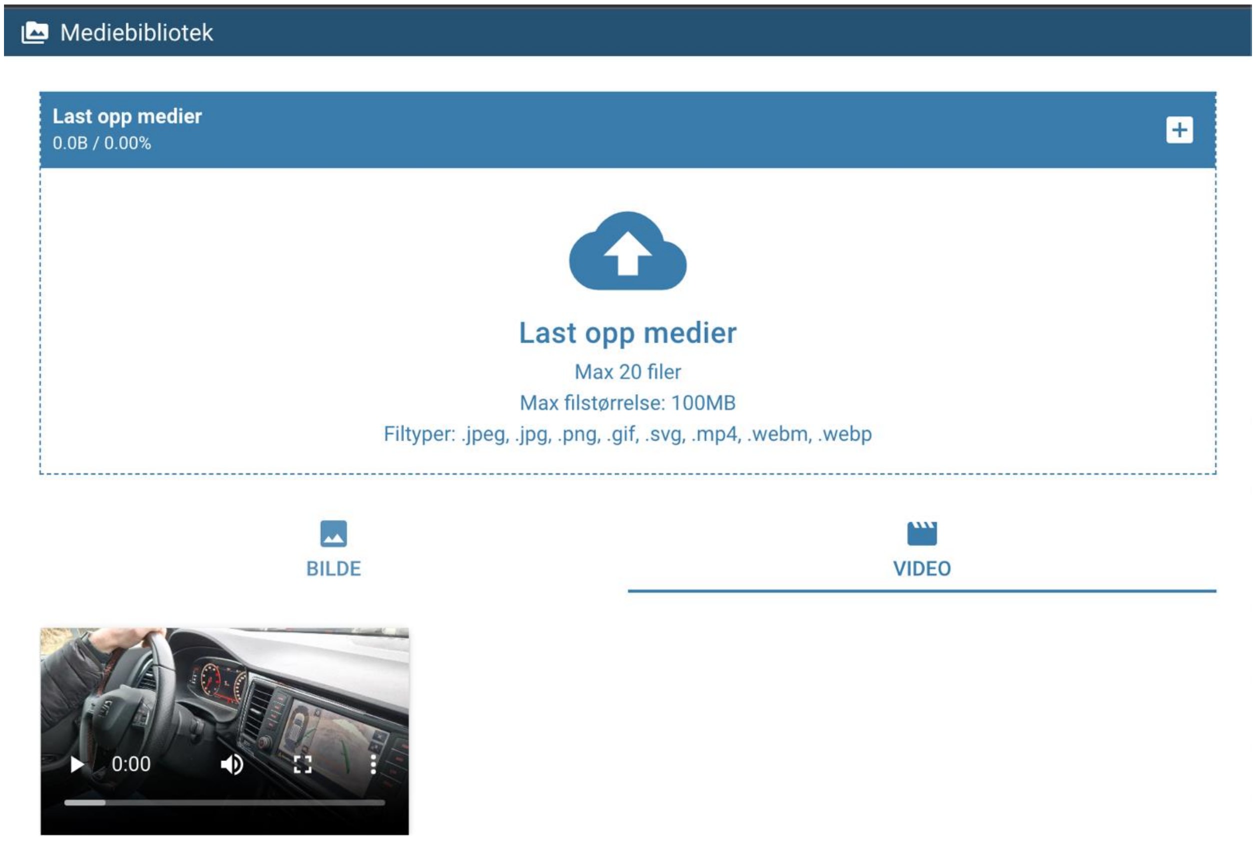
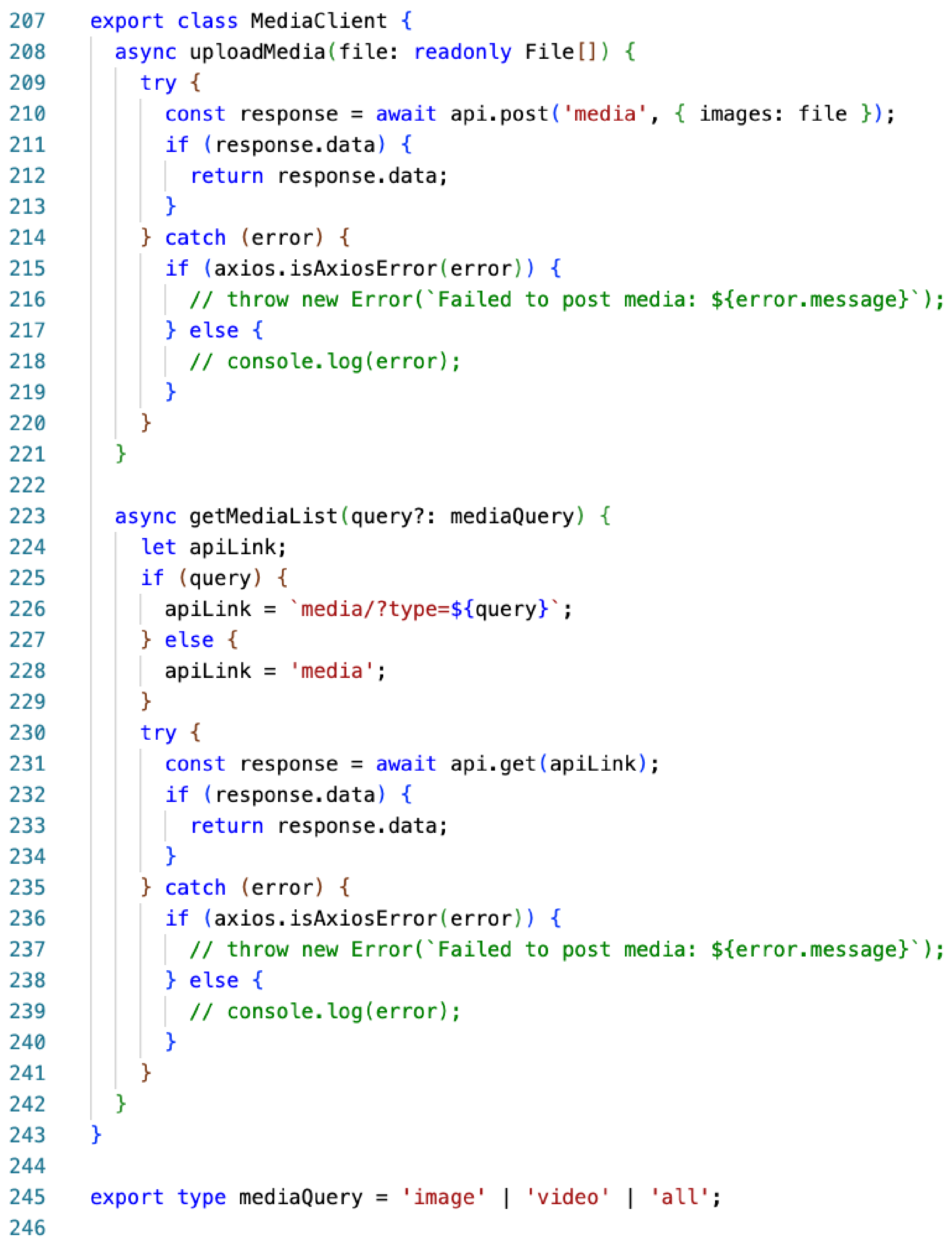
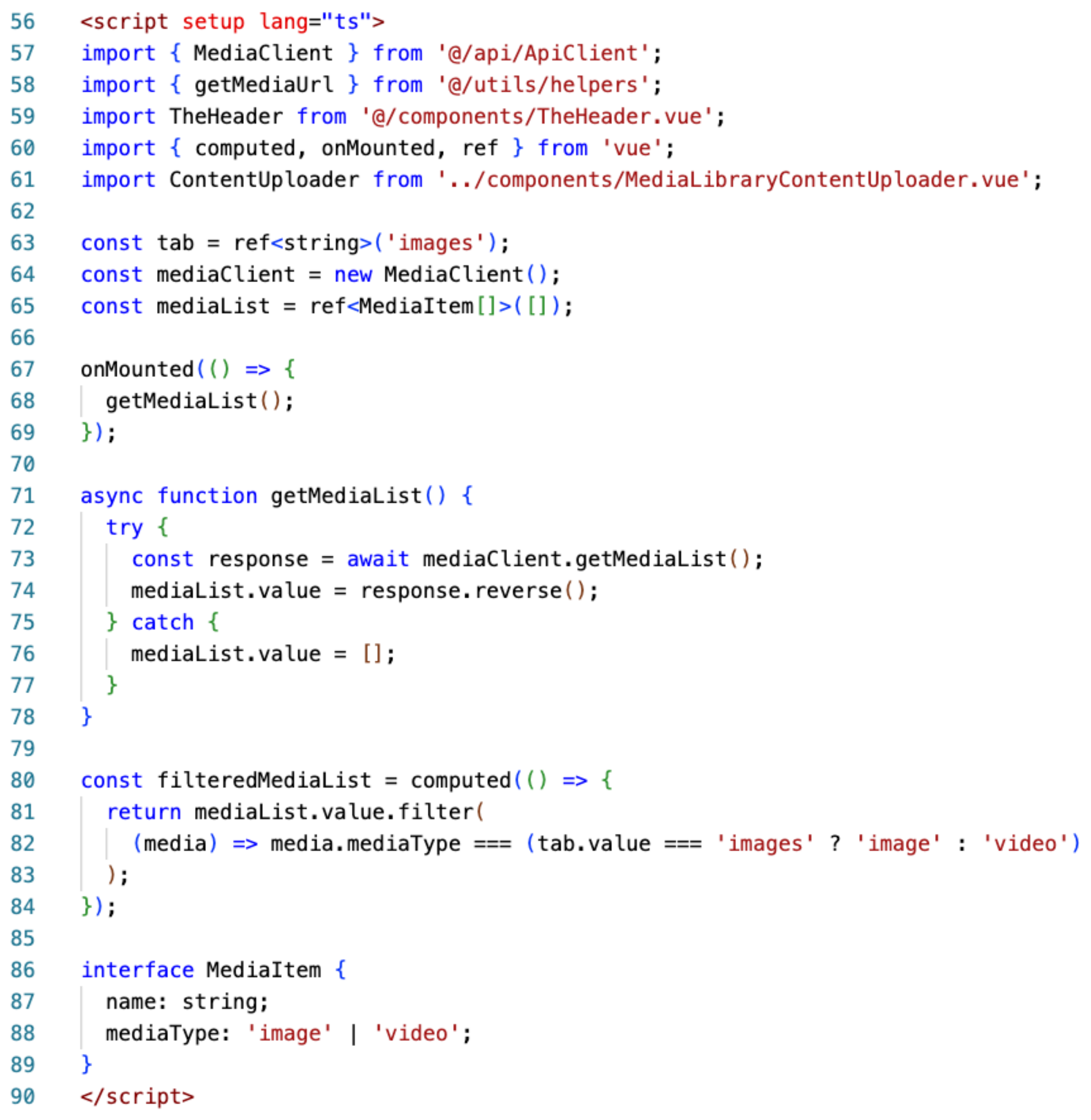


Figure 43 - MediaLibrary.vue, Video tab



```
export class MediaClient {
  async uploadMedia(file: readonly File[]) {
    try {
      const response = await api.post('media', { images: file });
      if (response.data) {
        return response.data;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to post media: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }

  async getMediaList(query?: mediaQuery) {
    let apiLink;
    if (query) {
      apiLink = `media/?type=${query}`;
    } else {
      apiLink = 'media';
    }
    try {
      const response = await api.get(apiLink);
      if (response.data) {
        return response.data;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to post media: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }
}

export type mediaQuery = 'image' | 'video' | 'all';
```

Figure 44 - ApiClient.ts - MediaClient class

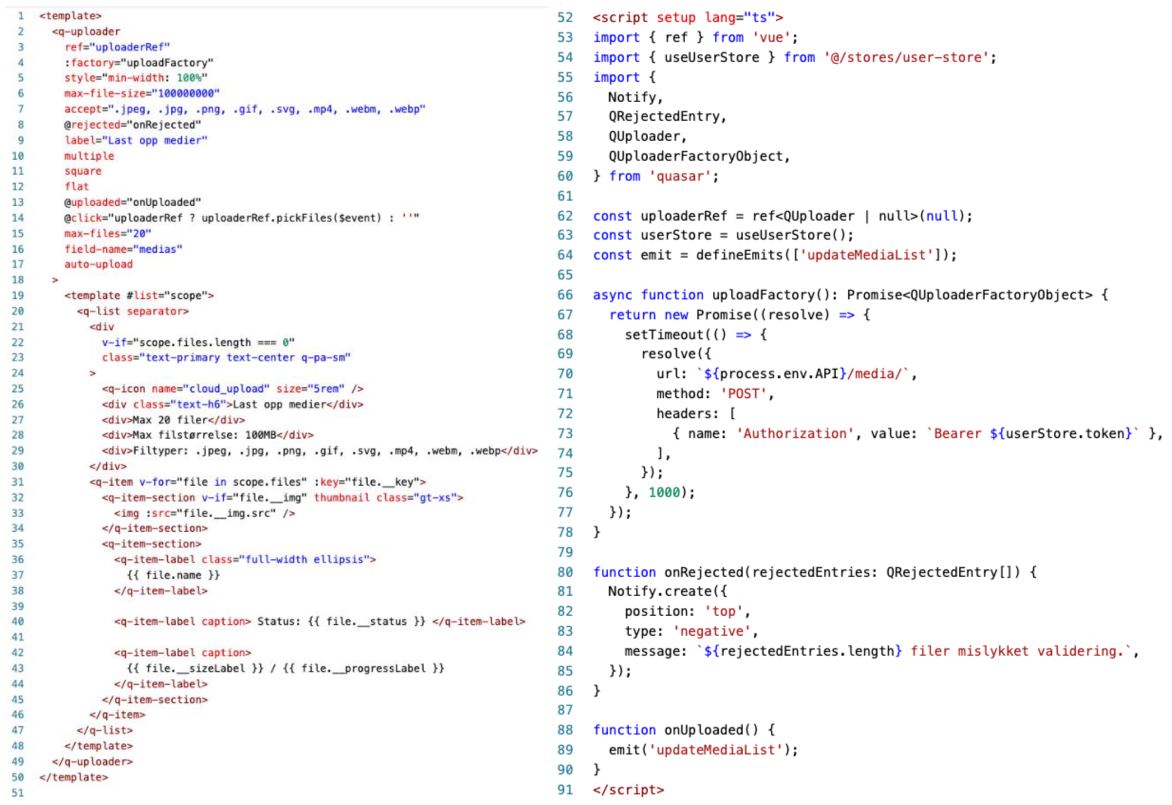


```
<script setup lang="ts">
import { MediaClient } from '@/api/ApiClient';
import { getMediaUrl } from '@/utils/helpers';
import TheHeader from '@/components/TheHeader.vue';
import { computed, onMounted, ref } from 'vue';
import ContentUploader from '../components/MediaLibraryContentUploader.vue';

const tab = ref<string>('images');
const mediaClient = new MediaClient();
const mediaList = ref<MediaItem[]>([]);

onMounted(() => {
  getMediaList();
});

async function getMediaList() {
  try {
    const response = await mediaClient.getMediaList();
    mediaList.value = response.reverse();
  } catch {
    mediaList.value = [];
  }
}

const filteredMediaList = computed(() => {
  return mediaList.value.filter(
    (media) => media.mediaType === (tab.value === 'images' ? 'image' : 'video')
  );
});

interface MediaItem {
  name: string;
  mediaType: 'image' | 'video';
}
</script>
```

Figure 45 - MediaLibrary.vue - script setup

On the mounted lifecycle hook, the “*getMediaList*” function is called to retrieve the list of media items from the backend using the “*MediaClient*” class from the “*ApiClient.ts*” file. The response is then assigned to the “*mediaList*” reactive variable, which is used to render the media items on the page.

The “*MediaLibraryContentUploader.vue*” component (Figure 46) allows users to upload new media files to the CMS. It uses the Quasar QUploader[12] component to handle file uploads, providing a comprehensive interface with support for drag-and-drop, multiple file uploads, and file type validation. The uploader is configured to send the media files to the backend API using the “*uploadFactory*” function, which sets the appropriate URL, method, and headers for the HTTP request.



```
<template>
  <q-uploader
    ref="uploaderRef"
    :factory="uploadFactory"
    style="min-width: 100%"
    max-file-size="100000000"
    accept=".jpeg, .jpg, .png, .gif, .svg, .mp4, .webm, .webp"
    @rejected="onRejected"
    label="Last opp medier"
    multiple
    square
    flat
    @uploaded="onUploaded"
    @click="uploaderRef ? uploaderRef.pickFiles($event) : ''"
    max-files="20"
    field-name="medias"
    auto-upload
  >
    <template #list="scope">
      <q-list separator>
        <div
          v-if="scope.files.length === 0"
          class="text-primary text-center q-pa-sm"
        >
          <q-icon name="cloud_upload" size="5rem" />
          <div class="text-h6">Last opp medier</div>
          <div>Max 20 filer</div>
          <div>Max filstørrelse: 100MB</div>
          <div>Filtyper: .jpeg, .jpg, .png, .gif, .svg, .mp4, .webm, .webp</div>
        </div>
        <q-item v-for="file in scope.files" :key="file.__key">
          <q-item-section v-if="file.__img" thumbnail class="gt-xs">
            <img :src="file.__img.src" />
          </q-item-section>
          <q-item-section>
            <q-item-label class="full-width ellipsis">
              {{ file.name }}
            </q-item-label>

            <q-item-label caption> Status: {{ file.__status }} </q-item-label>

            <q-item-label caption>
              {{ file.__sizeLabel }} / {{ file.__progressLabel }}
            </q-item-label>
          </q-item-section>
        </q-item>
      </q-list>
    </template>
  </q-uploader>
</template>

<script setup lang="ts">
import { ref } from 'vue';
import { useUserStore } from '@/stores/user-store';
import {
  Notify,
  QRejectedEntry,
  QUploader,
  QUploaderFactoryObject,
} from 'quasar';

const uploaderRef = ref<QUploader | null>(null);
const userStore = useUserStore();
const emit = defineEmits(['updateMediaList']);

async function uploadFactory(): Promise<QUploaderFactoryObject> {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve({
        url: `${process.env.API}/media/`,
        method: 'POST',
        headers: [
          { name: 'Authorization', value: `Bearer ${userStore.token}` },
        ],
      });
    }, 1000);
  });
}

function onRejected(rejectedEntries: QRejectedEntry[]) {
  Notify.create({
    position: 'top',
    type: 'negative',
    message: `${rejectedEntries.length} filer mislykket validering.`,
  });
}

function onUploaded() {
  emit('updateMediaList');
}
</script>
```

Figure 46 - MediaLibraryContentUploader.vue

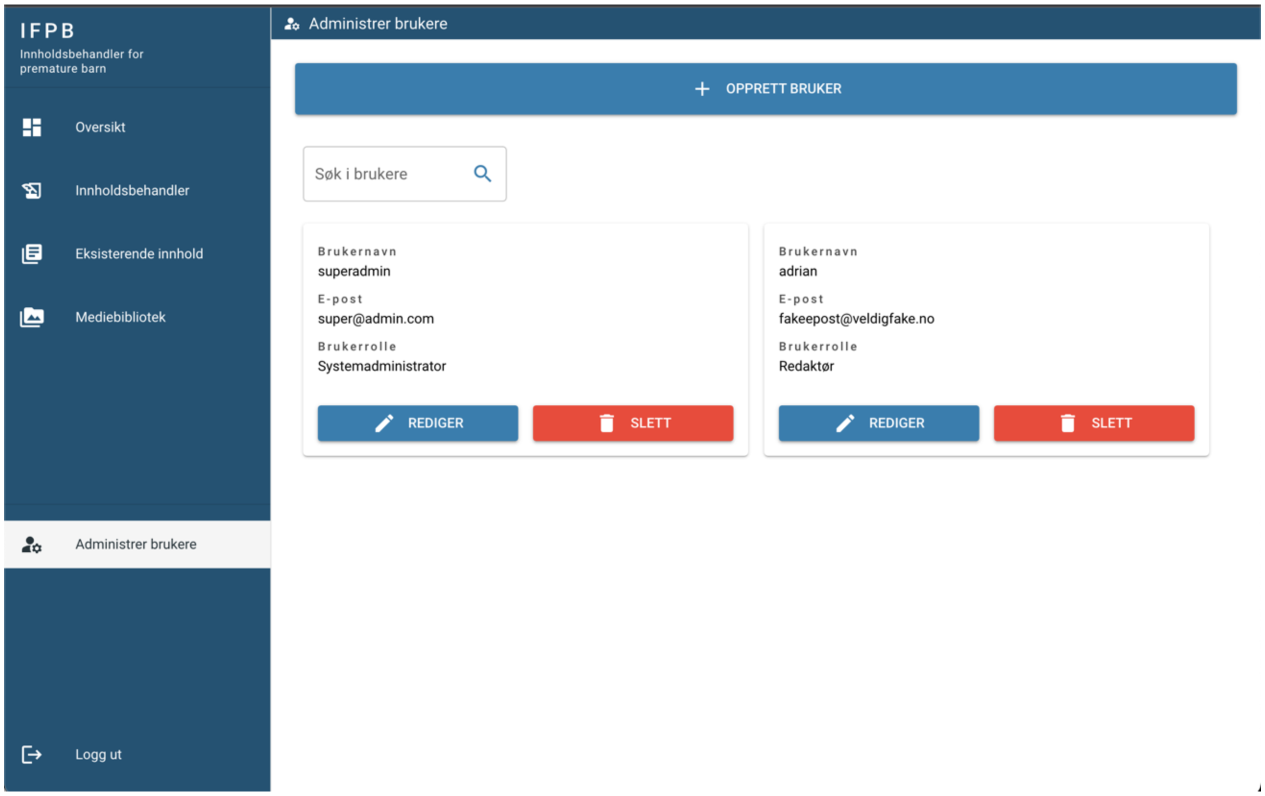
When a file upload is completed, the “*onUploaded*” function is called to emit an “*updateMediaList*” event, which is listened for in the parent “*MediaLibrary.vue*” component. This event triggers the “*getMediaList*” function to update the displayed media list with the newly uploaded media files.

[12] https://quasar.dev/vue-components/uploader

4.4.6 UserManagement.vue / “Administrer brukere”

The *“UserManagement.vue”* component, referred to as “Administrer brukere” in Norwegian, is a dedicated interface for managing user accounts. Accessible only to users with the “superadmin” role, this component provides functionality for creating, editing, and deleting editor accounts.

“*UserManagement.vue*” consists of a search input field, an "Opprett bruker" (Create User) button, and a table displaying a list of user accounts. Each account is represented by a card containing the username, email, and user role. Additionally, there are two action buttons within the card: "rediger" (edit) and "slett" (delete).



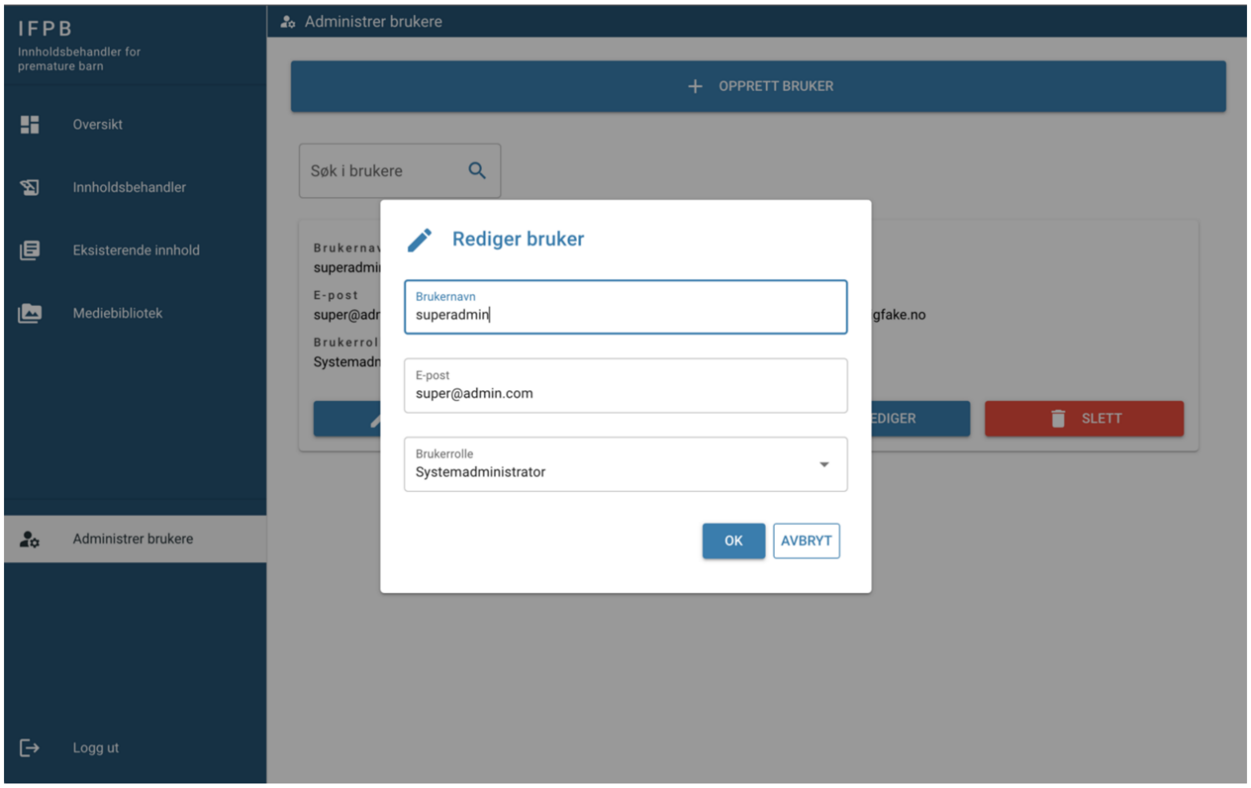
Brukernavn
superadmin
E-post
super@admin.com
Brukerrolle
Systemadministrator

Brukernavn
adrian
E-post
fakeepost@veldigfake.no
Brukerrolle
Redaktør

Figure 47 - UserManagement.vue

The component retrieves user information from the backend API using the “*UserClient*” class provided in the *ApiClient.ts* file (*Appendix 11 - ApiClient.ts)*. This class contains methods to handle actions like logging in, revoking and refreshing tokens, registering users, updating users, and deleting users.

When the "Opprett bruker" button is clicked, the *“openUserModal”* function is called, which triggers the display of the “*UserManagementEditUserModal.vue*” component. This modal allows users to input the new account's username, email, role, and password.



Brukernavn: superadmin

E-post: super@admin.com

Brukerrolle: Systemadministrator

Figure 48 - UserManagementEditUserModal.vue

The edit and delete buttons on each user account card also utilize the “*openUserModal*” and “*deleteUser”* functions, respectively. When the edit button is clicked, the “*UserManagementEditUserModal.vue*” component is displayed with pre-filled fields corresponding to the existing account information. On the other hand, clicking the delete button prompts a confirmation dialog, and upon confirmation, the “*deleteUser*” function is called to remove the account.

4.4.7 File structure and naming conventions

In this project, we have tried to follow the naming conventions outlined in the Vue.js Style Guide[13]. The file structure and naming conventions for this project are as follows:

Directory structure *(Appendix 4 - CMS file structure)*

- **src**: Contains the primary source code for the application.
  - **api**: API related files.
  - **boot**: Boot files which run before the root Vue instance is instantiated.
  - **components**: Vue components used in the application.
  - **css**: Global SCSS styling and variables.
  - **interfaces**: TypeScript interfaces.
  - **pages**: The primary view files, which are made up of components.
  - **router**: Router configuration files for the application.
  - **stores**: Pinia store configuration files for managing the application's state.
  - **utils**: Utility files and helper functions.
- **public**: Static files, such as the favicon

Naming conventions

- **Components:** Vue components are named using PascalCase (e.g., "*AppDialogConfirm.vue*"). Single-instance components should be prefixed with "The" (e.g., "*TheSidebar.vue*").
- **Filenames:** All file names should be in kebab-case (e.g., "*article-store.ts*").
- **JavaScript Variables:** Variables should be named in camelCase (e.g., "*handleClick*").
- **Event names:** Custom event names should use kebab-case (e.g., "*toggle-edit"*).

[13] https://v2.vuejs.org/v2/style-guide/

- **HTML Attributes:** Attributes should be written in kebab-case (e.g., “content-type”).
- **Pinia Store:** Modules, actions, mutations, and getters should be named using camelCase (e.g., “*useUserStore*”, “*generateRefreshToken*”, “*isAuthenticated*”).

4.5 Deploying the application

The application was deployed through Railway, a comprehensive platform that facilitates the development, deployment, and scaling of applications. Railway’s built-in support for Node.js and MongoDB made it a suitable choice for the project (Railway Corporation, 2023).

Due to the mono-repo structure of the project, which houses all applications in a single repository, a deployment solution was required that could link the repository and deploy it on each folder. Railway enables this process.

Railway services are deployment destinations for the application. These services connect with GitHub and deploy automatically on each commit. The Railway dashboard displays the services created, as depicted in *Figure 49 - Railway service overview*.

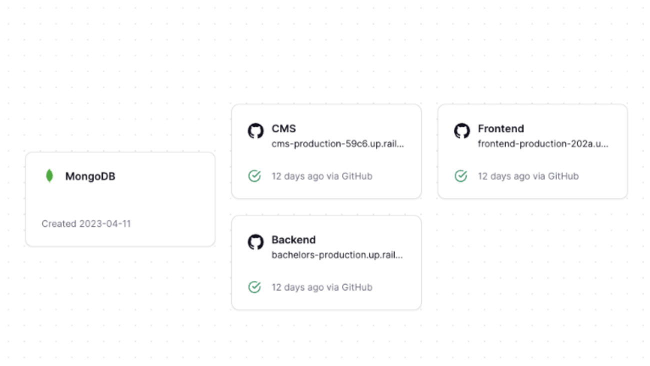


Figure 49 - Railway service overview

In the following section we will evaluate the final product and the results of the final user test.

4.6 Application evaluation

In this section, we will evaluate the outcomes of our user testing after discussing the design techniques, development procedures, and results of our project. Our evaluation will be divided into two main sections: the Content Management System (CMS) and the frontend of the application. We will examine the strengths and weaknesses that were uncovered by the test results in each section, referring to the data documented in the Excel spreadsheet *(see Appendix 14 - User testing final product)* as appropriate.

4.6.1 Conducting the test

Following the development process of the project the group conducted a final user test to gather feedback and an insight into what could be a further improvement of the product. The test participants consisted of two healthcare workers who are also the product owners, experienced men aged 50-67, and two student men aged 23-25. The mix of ages and backgrounds provided a valuable perspective for evaluating the product's usability and potential improvements.

Four tasks were created for both the Content Management System and the frontend, concluding in 8 tasks in total:

Content Management System

1. Log in with username "superadmin" and password
2. Create a new article
3. Edit the same article you created
4. Add a new user

Frontend

1. Log in with username "superadmin" and password
2. Navigate to "barnets løp" and filter to period "tidlig intensive" and born in week 30
3. Search for an article and add to your favorites
4. Navigate to «Hvordan styrke båndet mellom foreldre og baby» without using the search field and, hamburger menu or "aktuelle artikler".

Throughout the testing, task data like time used, errors, and notes were documented in an Excel spreadsheet *(see Appendix 14 - User testing final product).*

4.6.2 Results

Content Management System:

The test results for the CMS *(see Appendix 14 - User testing final product)* showed that users were able to complete the tasks with minimal errors and within a reasonable amount of time. The step-by-step process and the ability to preview the article while it's being made were well-received, and users appreciated the intuitive interface. However, some suggestions for improvement include reversing the article array for easier access to newer content, changing the color of the “Create User” button for better differentiation, ability to filter media library content by category, and using more descriptive button labels like “Save” or “Create” instead of “OK”.

Frontend:

For the frontend, users encountered some difficulties in navigating to specific content without using the search bar or hamburger menu. There were also some inconsistencies in the placement and visibility of certain elements, such as the favorites icon, which could be improved for a better user experience. However, the overall design and aesthetics were praised, and users found the website visually appealing and easy to use.

4.6.3 Summary

In conclusion, the user testing provided valuable feedback for improving the usability and functionality of the application. Both the CMS and the frontend demonstrated strengths in their design and implementation. By addressing the identified weaknesses and incorporating user suggestions, the application can be further refined and enhanced to provide an even better user experience. This process of evaluation and iteration is crucial for the successful development of any application, and our bachelor thesis has shown the importance of incorporating user feedback in the design process.

5 Reflection

In this chapter, we reflect upon how our solution aligns with the problem statement, the learning outcomes, and challenges encountered during the development of the web application. We discuss the knowledge and skills acquired throughout the project and analyze the impact of our assumptions on the final product.

5.1 Aligning the Final Deliverable with the Problem Statement

The problem statement, *"How can we develop an online platform to provide up to date, customized, and accurate information about premature infancy to support parents of premature infants?"*, was answered through the research questions:

1. *How can web development techniques be applied to create a user-friendly interface that addresses the specific needs of parents during these critical stages of their parenting journey?*
2. *How can we use web technologies to develop a solution that both answers the client's request and fits the user's needs?*
3. *How can we allow the healthcare providers to efficiently update and disseminate information relevant to parents during the hospital stay and homecare period?*

To address the first research question, we focused our efforts into developing a usable website. The extent to which we achieved delivering a user-friendly website was assessed in the final evaluation. By conducting user research, we were able to identify the most relevant features and information that parents needed, which were then incorporated into the application's design. This approach ensured that the platform was both easy to navigate and provided customized information tailored to individual needs. Feedback was positive and only minor suggestions for improvement have been raised.

For the second research question, we utilized various web technologies such as MERN/MEVN stack, React, Vue, Quasar, SCSS/SASS, and backend technologies to develop a solution that not only addressed the client's request but also met the user's needs. By combining these technologies, we were able to create a seamless user experience that efficiently delivered up-to-date information to parents of premature infants.

To tackle the third research question, we developed a custom content management system (CMS) that allowed healthcare providers to efficiently update and disseminate information relevant to parents during the hospital stay and homecare period. This CMS featured an intuitive interface and user roles for easy content management, ensuring that information was always current and accessible for parents.

In conclusion, the final deliverable successfully aligns with the problem statement by providing an online platform that offers customized, accurate, and up-to-date information about premature infancy to support parents of premature infants. Through careful consideration of user needs, application of appropriate web development techniques, and development of a custom CMS for healthcare providers, we have created a solution that addresses the challenges identified in the problem statement and meets the needs of both parents and healthcare providers.

5.2 Learning outcomes

The completion of this web development bachelor's project has led to several key learning outcomes, which are discussed below:

1. **Comprehensive understanding of the web development process:**
   Through the various stages of research, design, development, and deployment, the group gained a thorough understanding of the web development process. This includes the importance of user research, competitor analysis, and the use of design methodologies in creating an effective web application.
2. **Proficiency in web development technologies:**
   The project provided an opportunity to gain hands-on experience with web development technologies such as MERN/MEVN stack, React, Vue, Quasar,

SCSS/SASS, and backend technologies. This experience has not only improved the technical skills of the group, but also contributed to a deeper understanding of how these technologies work together to create a seamless user experience.

3. **User research and user-centered design:**
   The group’s engagement in user research, including product owner’s needs, interviews, affinity mapping, personas, and priority matrix, has emphasized the importance of incorporating user needs and preferences in the design and development process. This approach fosters a user-centered design, ensuring that the final product is tailored to the target audience.
4. **Competitor analysis:**
   By conducting a competitor analysis, the group learned how to identify strengths and weaknesses in existing websites and use this information to create a more competitive and effective product. This analytical skill is crucial for future web development projects and for staying ahead in the industry.
5. **Custom content management systems:**
   The development of a custom CMS using the Quasar framework and Vue 3 allowed the group to learn how to design and implement a custom solution tailored to the project’s requirements. The experience of creating a custom CMS has provided valuable insights into the intricacies of content management, user roles, and the underlying technologies, such as Vue 3, Quasar, Pinia Store, and TypeScript.
6. **Deployment and project management:**
   The group learned the importance of effective project management in ensuring that the web application was deployed successfully. This experience has demonstrated the need for proper planning, communication, and time management in the development process.
7. **Evaluation and reflection:**
   By evaluating the final product and reflecting on the learning outcomes and challenges, the group has gained a deeper understanding of the web development process and the impact of assumptions on the final product.

This insight will be valuable for future projects, as it highlights areas for improvement and growth.

5.3 Challenges during development

Throughout the web development project, several challenges were encountered, which provided valuable learning experiences and opportunities for growth. This section discusses some of the key challenges faced during the development process and the solutions adopted to overcome them.

1. **Understanding and integrating new technologies:**
   One of the challenges was familiarizing oneself with the Quasar framework, Vue 3, Pinia Store, and TypeScript, as these technologies were new for the group. To overcome this, the group spent time researching and learning these technologies through online resources, documentation, and hands-on practice.
2. **Balancing functionality and usability:**
   Creating a custom CMS that provides necessary features while maintaining a user-friendly interface was a challenge. To address this, the group conducted user tests and sought feedback from the product owner and other stakeholders to ensure that the CMS was intuitive and easy to use.
3. **Managing project scope and time constraints:**
   Ensuring that the project remained within its scope and was completed on time proved to be a challenge, especially considering the complexity of creating a custom CMS. To tackle this issue, the group employed effective project management techniques, such as setting clear objectives through a Gantt chart *(Appendix 10),* prioritizing tasks, and maintaining regular communication with the supervisor and other stakeholders.
4. **Dealing with unanticipated technical issues:**
   During development, the group encountered unexpected technical issues, such as bugs or compatibility problems between different technologies. To resolve these issues, the group relied on problem-solving skills, online resources, and sought guidance from the supervisor and peers when necessary.

5. **Maintaining code quality and organization:**
   As the project progressed, maintaining clean and organized code became increasingly challenging. The group addressed this issue by adopting best practices for code organization, implementing proper naming conventions, and using version control systems such as Git for tracking changes and collaborating effectively.
6. **Adapting to changing requirements:**
   Throughout the project, the group faced changes in requirements or priorities due to user feedback or new insights gained during the development process. Adapting to these changes required flexibility, effective communication with stakeholders, and the ability to reevaluate and adjust the development plan.

5.4 Recommendations to the product owner

In conclusion, the project group wishes to make some concrete recommendations to the product owner. Through this study, the group has documented our path to a solution. Although our final product is fully usable as is, we have still found areas which can be considered for further improvements. The list follows:

1. If the product is to be further developed, it should be tested in a real environment with real users. Meaning parents of premature infants at a hospital and after hospital stay.
2. If the solution is to be developed further, it is also recommended to ensure the product is performant and scalable. This can be addressed by conducting performance tests and optimizing the code wherever possible.
3. Implement feedback from final evaluation. This includes changing the background image on the homepage, improving naming conventions, minor changes on the sitemap, and adding links to various resources.
4. Implement a data trafficker to check if the product is being used by parents at all.
5. If the project is to be continued, it is recommended to implement a full set of articles through the Headless CMS.

6 Conclusion

In this bachelor thesis, we have explored the problem of providing up-to-date, customized, and accurate information about premature infancy to support parents of premature infants. We have addressed this problem by developing an online platform through a user-centered design approach, applying relevant web development techniques, and creating a custom Content Management System (CMS) for healthcare providers.

The application evaluation, as discussed in *Chapter 4.6,* revealed that our solution has several strengths in terms of design and functionality. Both the CMS and the frontend demonstrated positive user experiences, while the feedback obtained from user testing indicated areas for further improvement. By addressing these issues, the application can be refined and enhanced to better serve the needs of parents and healthcare providers.

In our reflection (*Chapter 5*), we discussed how the final deliverable aligns with the problem statement and the various learning outcomes achieved throughout the project. We also examined the challenges encountered during the development process, which provided valuable insights and opportunities for growth.

In conclusion, this bachelor thesis has successfully demonstrated the importance of a user-centered approach in web development and the value of incorporating user feedback in the design process. Our solution, which combines a user-friendly interface with a custom CMS, effectively addresses the needs of both parents of premature infants and healthcare providers. By continuing to refine and enhance the application based on user feedback and emerging technologies, this platform has the potential to become an invaluable resource for parents navigating the challenges of premature infancy.

7 Reference list

Arancio, S. (2021, August 5). *Medium*. Retrieved from ReactJS: A brief history. A peak into the evolution of one of the world’s most popular programming libraries. | by Stephen Arancio | Medium: https://medium.com/@sjarancio/reactjs-a-brief-history-3c1e969a477f

Barker, D. (2016). Web Content Management: Systems, Features, and Best Practices. In D. Barker, *Web Content Management: Systems, Features, and Best Practices* (pp. 1-13). Sebastopol: O'Reilly Media, Inc.

Baxter, K., Courage, C., & Caine, K. (2015). *Understanding your users: A practical guide to user research methods* (2 ed.). Morgan Kaufmann.

Dam, R. F., & Siang, T. Y. (2022). *Interaction Design Foundation*. Retrieved Februar 13, 2023, from https://www.interaction-design.org/literature/article/affinity-diagrams-learn-how-to-cluster-and-bundle-ideas-and-facts

Design Council. (2019). *Framework for Innovation: Design Council's evolved Double Diamond - Design Council*. Retrieved April 14, 2023, from https://www.designcouncil.org.uk/our-work/skills-learning/tools-frameworks/framework-for-innovation-design-councils-evolved-double-diamond/

Express. (2023). *Express*. Retrieved April 11, 2023, from https://expressjs.com/

Fileformat. (2023, April 17). *Fileformat.* Retrieved from SCSS File Format - Sass Cascading Style Sheet: https://docs.fileformat.com/web/scss/

Google. (2023, April 21). *Design Sprints*. Retrieved from Phase 3: Sketch, Crazy 8's: https://designsprintkit.withgoogle.com/methodology/phase3-sketch/crazy-8s js-framework-benchmark. (2023). *Js Framework Benchmark*. Retrieved April 12, 2023, from https://rawgit.com/krausest/js-framework-benchmark/master/webdriver-ts-results/table.html

Karlsen, J. T. (2021). *Prosjektledelse - fra initiering til gevinstrealisering* (5. utgave ed.). Oslo: Universitetsforlaget.

Karlsson, J. (2022). *MongoDB Schema Design Best Practices*. Retrieved April 12, 2023, from https://www.mongodb.com/developer/products/mongodb/mongodb-schema-design-best-practices/

mdn web docs. (2023, February 18). *mdn web docs*. Retrieved from XMLHttpRequest - Web APIs | MDN: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

mdn web docs. (2023, April 14). *SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. Retrieved from mdn web docs: https://developer.mozilla.org/en-US/docs/Glossary/SPA

Miller, J., & Osmani, A. (2022). *Rendering on the Web*. Retrieved April 19, 2023, from https://web.dev/rendering-on-the-web

MongoDB, Inc. (2023). *Data Modeling Introduction*. Retrieved April 12, 2023, from https://www.mongodb.com/docs/v5.3/core/data-modeling-introduction/#data-modeling-introduction

MongoDB, Inc. (2023, April 21). *MERN Stack Explained*. Retrieved from MongoDB: https://www.mongodb.com/mern-stack

Node. (2023). *About Node.js*. Retrieved April 11, 2023, from https://nodejs.org/en/about

Norsk Helseinformatikk. (2021, April 15). *Norsk Helseinformatikk*. Retrieved from For tidlig fødsel (prematuritet) - NHI.no:

https://nhi.no/sykdommer/barn/nyfodtmedisin/for-tidlig-fodsel-prematuritet/?page=1

Oracle. (2023). *What is a Content Management System*. Retrieved April 12, 2023, from https://www.oracle.com/content-management/what-is-cms/

Oslo Universitetssykehus. (2020, September 20). *Oslo Universitetssykehus*. Retrieved from For tidlig fødte barn (prematur) på Ullevål sykehus - Oslo universitetssykehus: https://oslo-universitetssykehus.no/behandlinger/for-tidlig-fodte-barn-prematur?sted=nyfodtintensiv-pa-ulleval-sykehus

Quasar. (2023). *Introduction to Quasar*. Retrieved April 12, 2023, from https://quasar.dev/introduction-to-quasar

Quasar. (2023). *QSplitter*. Retrieved April 18, 2023, from https://quasar.dev/vue-components/splitter#qsplitter-api

Quasar. (2023). *Stepper | Quasar*. Retrieved April 18, 2023, from https://quasar.dev/vue-components/stepper#qstepper-api

Railway Corporation. (2023, April 26). *Railway Documentation*. Retrieved from Getting Started: https://docs.railway.app/getting-started

React Router. (2023, April 14). *Feature Overview v6.10.0 | React Router*. Retrieved from React Router: https://reactrouter.com/en/main/start/overview#client-side-routing

Richards, A. (2020, October 2). *Medium*. Retrieved from Learn the SCSS (Sass) Basics in 5 Minutes | by Andrew Richards | The Startup | Medium: https://medium.com/swlh/learn-the-scss-sass-basics-in-5-minutes-73002653b443

Subramanian, V. (2019). *Pro MERN Stack* (2nd edition ed.). Bangalore, Karnataka, India: Apress Media LLC.

Team Asana. (2022, October 24). *asana*. Retrieved from Priority Matrix: Identify What Matters and Get More Done [2023] [2022] • Asana: https://asana.com/resources/priority-matrix

TypeScript. (2023, April 13). *TypeScript: JavaScript With Syntax For Types.* Retrieved from TypeScript: https://www.typescriptlang.org/

Vite. (2023, April 18). *Vite*. Retrieved from Why Vite: https://vitejs.dev/guide/why.html

Vue Community. (2023, April 26). *UI Libraries | Vue Community*. Retrieved from Vue Community: https://vue-community.org/guide/ecosystem/ui-libraries.html

Vue.js. (2023). *Frequently Asked Questions | Vue.js*. Retrieved April 12, 2023, from https://vuejs.org/about/faq.html

Vue.js. (2023, April 13). *State Management | Vue.js*. Retrieved from Vue.js: https://vuejs.org/guide/scaling-up/state-management.html#ssr-considerations

8 Figure list

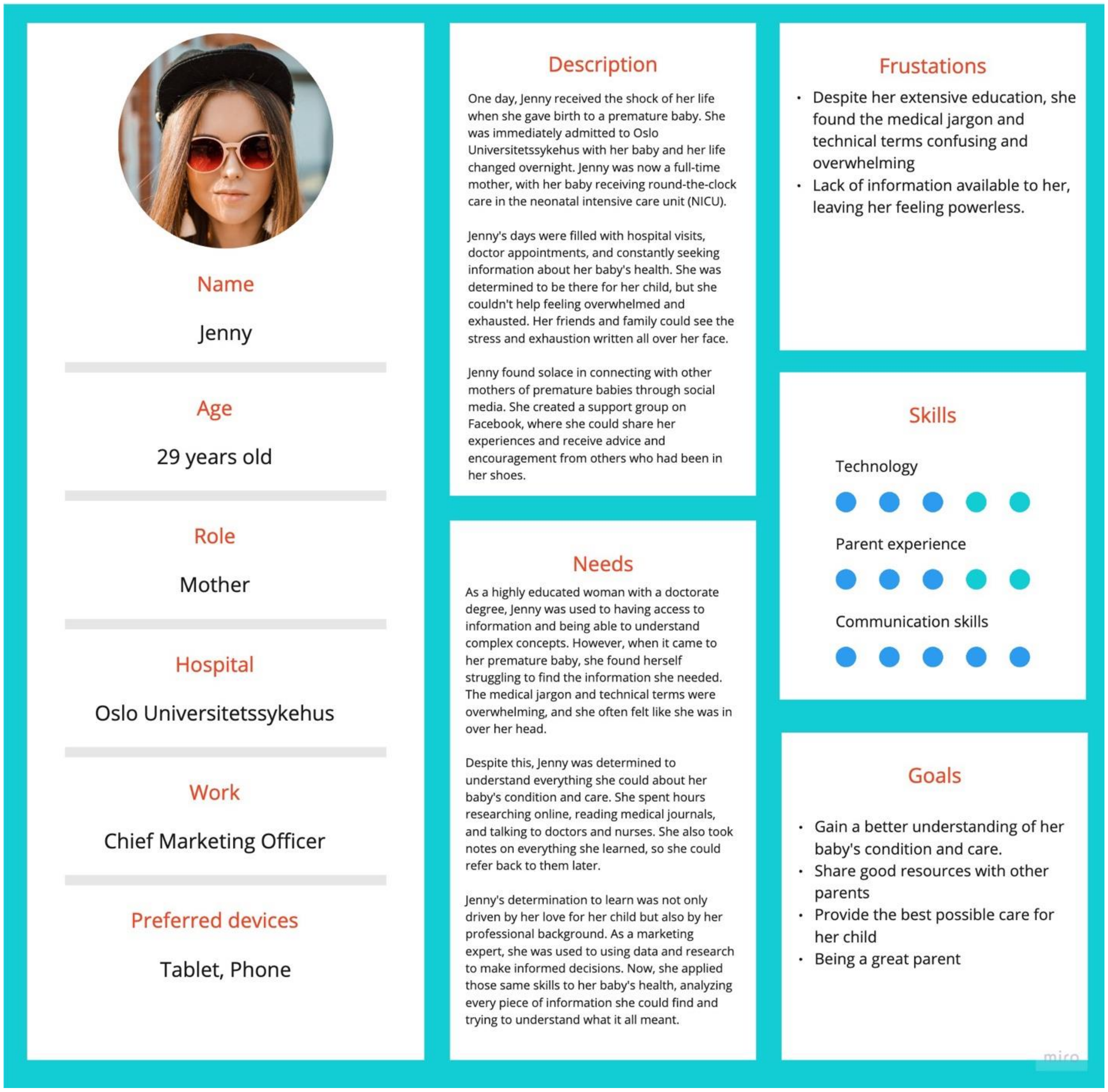
FIGURE LIST

9 Table list

10 Appendix

Appendix 1 – Personas

Primary



Name

Jenny

Age

29 years old

Role

Mother

Hospital

Oslo Universitetssykehus

Work

Chief Marketing Officer

Preferred devices

Tablet, Phone

Description

One day, Jenny received the shock of her life when she gave birth to a premature baby. She was immediately admitted to Oslo Universitetssykehus with her baby and her life changed overnight. Jenny was now a full-time mother, with her baby receiving round-the-clock care in the neonatal intensive care unit (NICU).

Jenny's days were filled with hospital visits, doctor appointments, and constantly seeking information about her baby's health. She was determined to be there for her child, but she couldn't help feeling overwhelmed and exhausted. Her friends and family could see the stress and exhaustion written all over her face.

Jenny found solace in connecting with other mothers of premature babies through social media. She created a support group on Facebook, where she could share her experiences and receive advice and encouragement from others who had been in her shoes.

Needs

As a highly educated woman with a doctorate degree, Jenny was used to having access to information and being able to understand complex concepts. However, when it came to her premature baby, she found herself struggling to find the information she needed. The medical jargon and technical terms were overwhelming, and she often felt like she was in over her head.

Despite this, Jenny was determined to understand everything she could about her baby's condition and care. She spent hours researching online, reading medical journals, and talking to doctors and nurses. She also took notes on everything she learned, so she could refer back to them later.

Jenny's determination to learn was not only driven by her love for her child but also by her professional background. As a marketing expert, she was used to using data and research to make informed decisions. Now, she applied those same skills to her baby's health, analyzing every piece of information she could find and trying to understand what it all meant.

Frustrations

- Despite her extensive education, she found the medical jargon and technical terms confusing and overwhelming
- Lack of information available to her, leaving her feeling powerless.

Skills

Technology

● ● ● ○ ○

Parent experience

● ● ● ○ ○

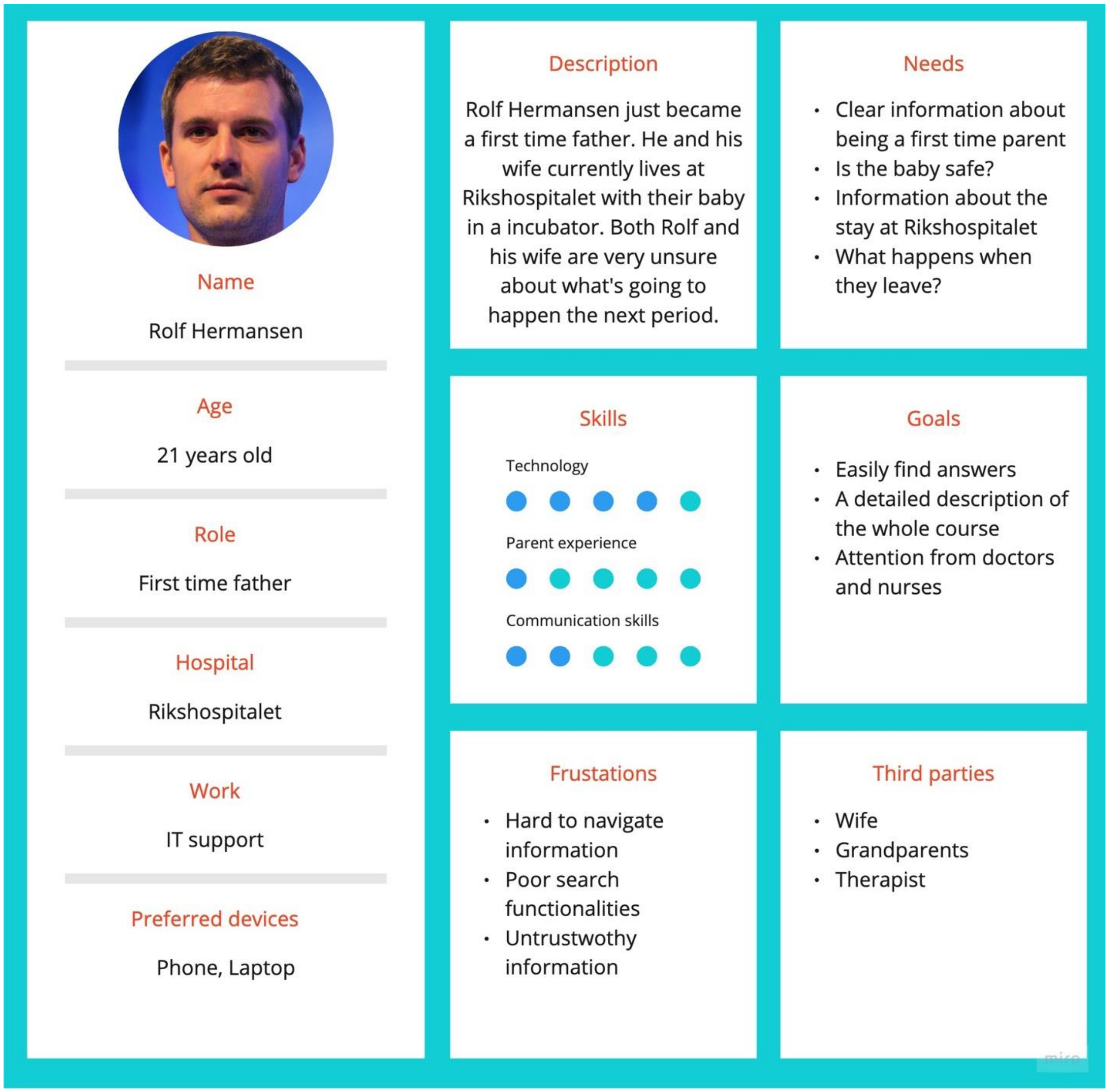
Communication skills

● ● ● ● ●

Goals

- Gain a better understanding of her baby's condition and care.
- Share good resources with other parents
- Provide the best possible care for her child
- Being a great parent

Secondary



Name

Rolf Hermansen

Age

21 years old

Role

First time father

Hospital

Rikshospitalet

Work

IT support

Preferred devices

Phone, Laptop

Description

Rolf Hermansen just became a first time father. He and his wife currently lives at Rikshospitalet with their baby in a incubator. Both Rolf and his wife are very unsure about what's going to happen the next period.

Needs

- Clear information about being a first time parent
- Is the baby safe?
- Information about the stay at Rikshospitalet
- What happens when they leave?

Skills

Technology

● ● ● ● ○

Parent experience

● ○ ○ ○ ○

Communication skills

● ● ○ ○ ○

Goals

- Easily find answers
- A detailed description of the whole course
- Attention from doctors and nurses

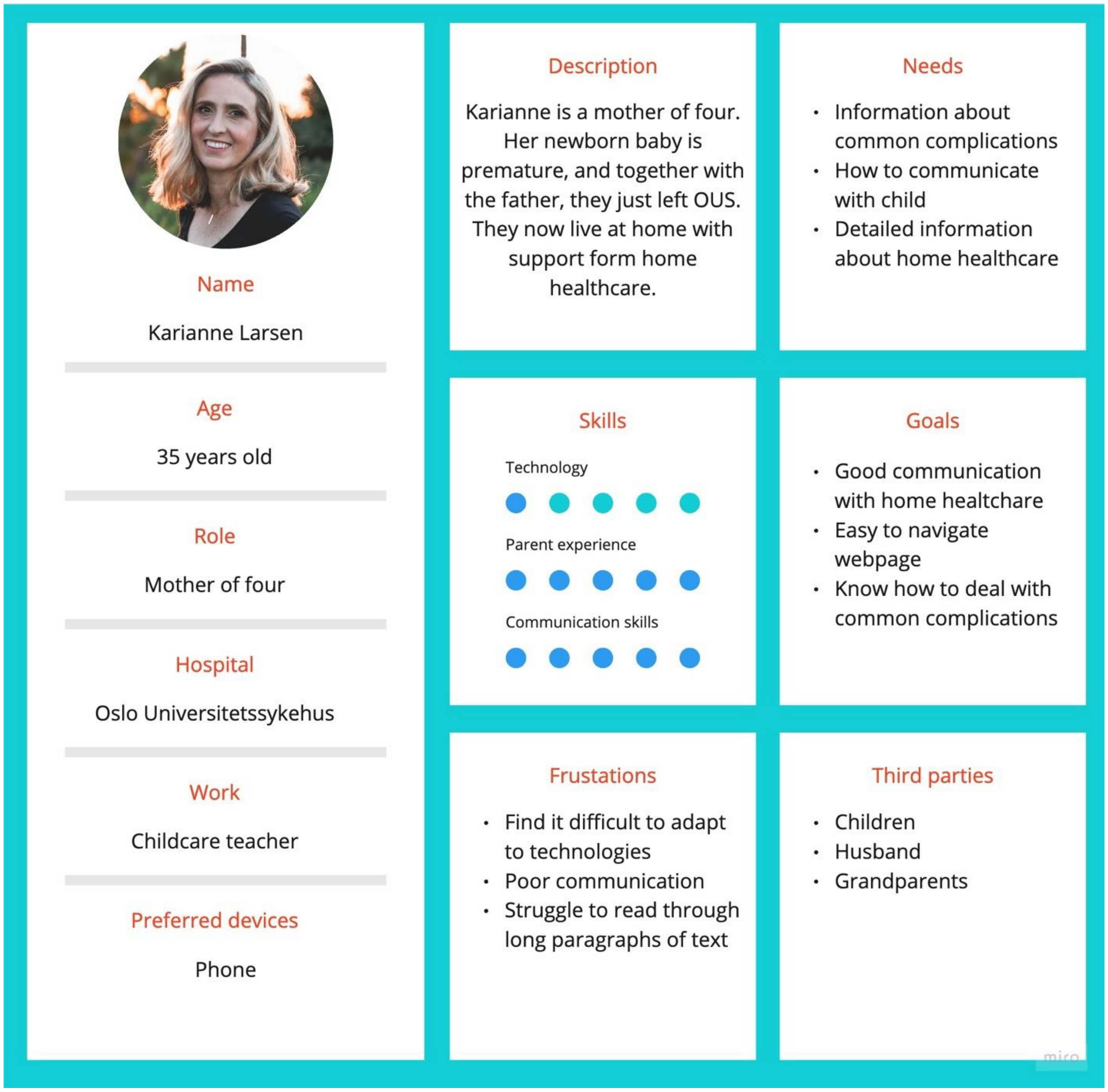
Frustations

- Hard to navigate information
- Poor search functionalities
- Untrustwothy information

Third parties

- Wife
- Grandparents
- Therapist

Secondary



Name

Karianne Larsen

Age

35 years old

Role

Mother of four

Hospital

Oslo Universitetssykehus

Work

Childcare teacher

Preferred devices

Phone

Description

Karianne is a mother of four. Her newborn baby is premature, and together with the father, they just left OUS. They now live at home with support form home healthcare.

Needs

- Information about common complications
- How to communicate with child
- Detailed information about home healthcare

Skills

Technology

●○○○○

Parent experience

●●●●●

Communication skills

●●●●●

Goals

- Good communication with home healthcare
- Easy to navigate webpage
- Know how to deal with common complications

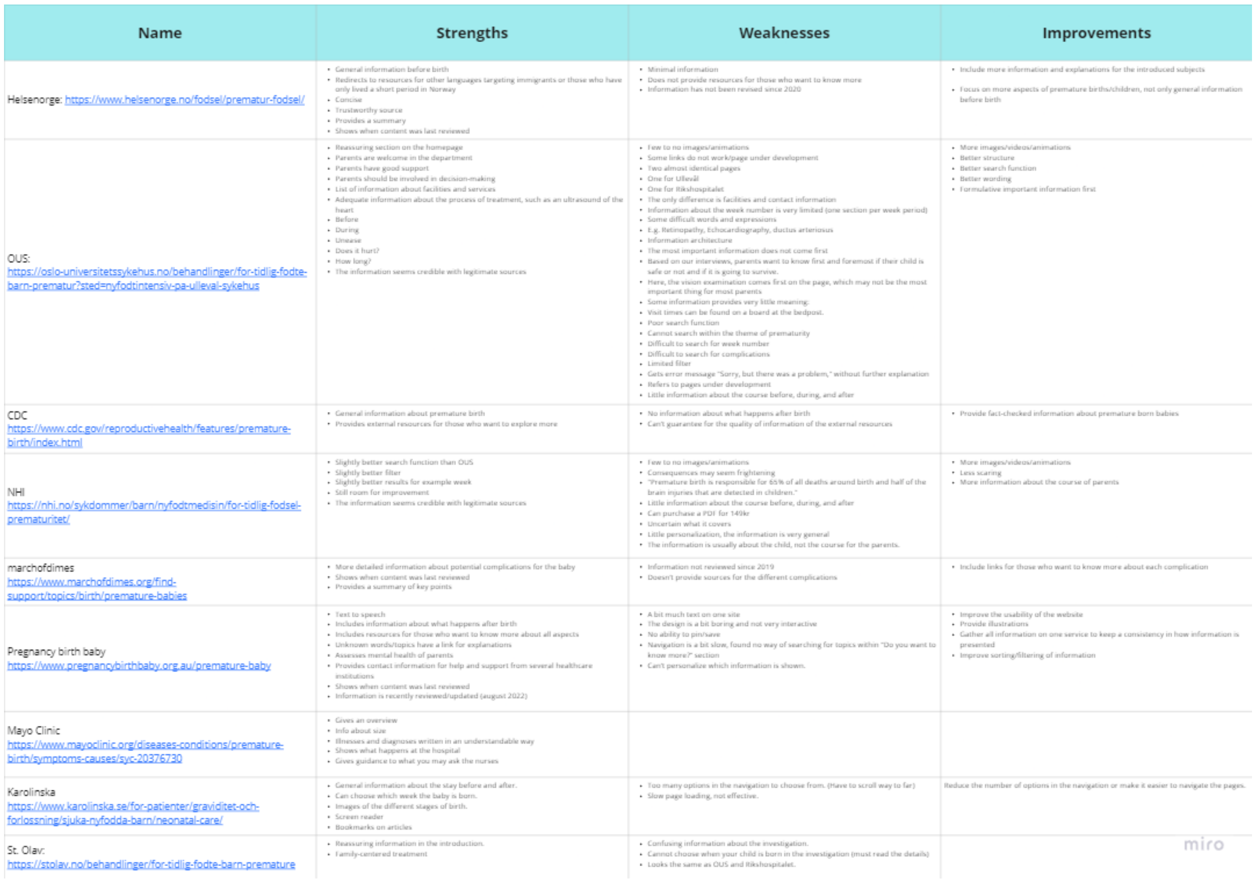
Frustrations

- Find it difficult to adapt to technologies
- Poor communication
- Struggle to read through long paragraphs of text

Third parties

- Children
- Husband
- Grandparents

Appendix 2 - Competitor analysis



Name	Strengths	Weaknesses	Improvements
Helsenorge: https://www.helsenorge.no/fodsel/prematur-fodsel/	• General information before birth • Redirects to resources for other languages targeting immigrants or those who have only lived a short period in Norway • Concise • Trustworthy source • Provides a summary • Shows when content was last reviewed	• Minimal information • Does not provide resources for those who want to know more • Information has not been revised since 2020	• Include more information and explanations for the introduced subjects • Focus on more aspects of premature births/children, not only general information before birth
OUS: https://oslo-universitetssykehus.no/behandlinger/for-tidlig-fodte-barn-prematur?sted=nyfodtintensiv-pa-ulleval-sykehus	• Reassuring section on the homepage • Parents are welcome in the department • Parents have good support • Parents should be involved in decision-making • List of information about facilities and services • Adequate information about the process of treatment, such as an ultrasound of the heart • Before • During • Unease • Does it hurt? • How long? • The information seems credible with legitimate sources	• Few to no images/animations • Some links do not work/page under development • Two almost identical pages • One for Ullevål • One for Rikshospitalet • The only difference is facilities and contact information • Information about the week number is very limited (one section per week period) • Some difficult words and expressions • E.g. Retinopathy, Echocardiography, ductus arteriosus • Information architecture • The most important information does not come first • Based on our interviews, parents want to know first and foremost if their child is safe or not and if it is going to survive. • Here, the vision examination comes first on the page, which may not be the most important thing for most parents • Some information provides very little meaning: • Visit times can be found on a board at the bedpost. • Poor search function • Cannot search within the theme of prematurity • Difficult to search for week number • Difficult to search for complications • Limited filter • Gets error message "Sorry, but there was a problem," without further explanation • Refers to pages under development • Little information about the course before, during, and after	• More images/videos/animations • Better structure • Better search function • Better wording • Formulative important information first
CDC https://www.cdc.gov/reproductivehealth/features/premature-birth/index.html	• General information about premature birth • Provides external resources for those who want to explore more	• No information about what happens after birth • Can't guarantee for the quality of information of the external resources	• Provide fact-checked information about premature born babies
NHI https://nhi.no/sykdommer/barn/nyfodtmedisin/for-tidlig-fodsel-prematuritet/	• Slightly better search function than OUS • Slightly better filter • Slightly better results for example week • Still room for improvement • The information seems credible with legitimate sources	• Few to no images/animations • Consequences may seem frightening • "Premature birth is responsible for 65% of all deaths around birth and half of the brain injuries that are detected in children." • Little information about the course before, during, and after • Can purchase a PDF for 149kr • Uncertain what it covers • Little personalization, the information is very general • The information is usually about the child, not the course for the parents.	• More images/videos/animations • Less scaring • More information about the course of parents
marchofdimes https://www.marchofdimes.org/find-support/topics/birth/premature-babies	• More detailed information about potential complications for the baby • Shows when content was last reviewed • Provides a summary of key points	• Information not reviewed since 2019 • Doesn't provide sources for the different complications	• Include links for those who want to know more about each complication
Pregnancy birth baby https://www.pregnancybirthbaby.org.au/premature-baby	• Text to speech • Includes information about what happens after birth • Includes resources for those who want to know more about all aspects • Unknown words/topics have a link for explanations • Assesses mental health of parents • Provides contact information for help and support from several healthcare institutions • Shows when content was last reviewed • Information is recently reviewed/updated (august 2022)	• A bit much text on one site • The design is a bit boring and not very interactive • No ability to pin/save • Navigation is a bit slow, found no way of searching for topics within "Do you want to know more?" section • Can't personalize which information is shown.	• Improve the usability of the website • Provide illustrations • Gather all information on one service to keep a consistency in how information is presented • Improve sorting/filtering of information
Mayo Clinic https://www.mayoclinic.org/diseases-conditions/premature-birth/symptoms-causes/syc-20376730	• Gives an overview • Info about size • Illnesses and diagnoses written in an understandable way • Shows what happens at the hospital • Gives guidance to what you may ask the nurses		
Karolinska https://www.karolinska.se/for-patienter/graviditet-och-forlossning/sjuka-nyfodda-barn/neonatal-care/	• General information about the stay before and after. • Can choose which week the baby is born. • Images of the different stages of birth. • Screen reader • Bookmarks on articles	• Too many options in the navigation to choose from. (Have to scroll way to far) • Slow page loading, not effective.	Reduce the number of options in the navigation or make it easier to navigate the pages.
St. Olav: https://stolav.no/behandlinger/for-tidlig-fodte-barn-premature	• Reassuring information in the introduction. • Family-centered treatment	• Confusing information about the investigation. • Cannot choose when your child is born in the investigation (must read the details) • Looks the same as OUS and Rikshospitalet.	

miro

Appendix 3 - Database schemas

Schema	Fields	Requirements
User	username	String, required, trim, unique
	password	String, required, trim
	email	String, required, trim, unique, lowercase
	babyBorn	Number, required: function () { return this.role === User },
	role	String, required, enum: ["superadmin", "admin", "user"], defaulf: "user"
	pinnedArticles	Array of strings
Refreshtoken	user	ObjectID, ref: "User"
	token	String
	expires	Date
	created	String
	createdByIp	String
	revoked	Date
	revokedByIp	String
	replacedByToken	String
Article	title	String, required
	slug	String, required, unique
	coverMedia	altText: String, required; fileName: String, required; mediaType: String, enum: ["image", "video"], default: "image",
	excerpt	String, required, trim
	content	Array of objects
	content.id	Number, required
	content.contentType	String, required, enum: ["media", "text", "media_and_text"], default: "text"
	content.body	Object
	body.text	String, trim
	body.media	Object
	media.mediaPositionLeft	Boolean
	media.fileName	String, trim
	media.altText	String, trim
	media.mediaType	String, required, enum: ["image", "video"], default: "image",
	date	Date, required, default: Date.now

	updatedAt	Date, required, default: Date.now
	weeks	Array of numbers
	afterBirth	String, required, enum:["early-intensive","stabilization","transition","homecare"," all",], default: "all"
	sources	Array of objects
	sources.id	Number, required
	sources.title	String, required
	category	title: String required; slug: String, required
Media	mediaType	String, required, enum: ["video", "image"], default: "image",
	name	String, required
Category	title	String, required
	slug	String, required, unique
	articles	Array of ObjectId, ref: "Article"

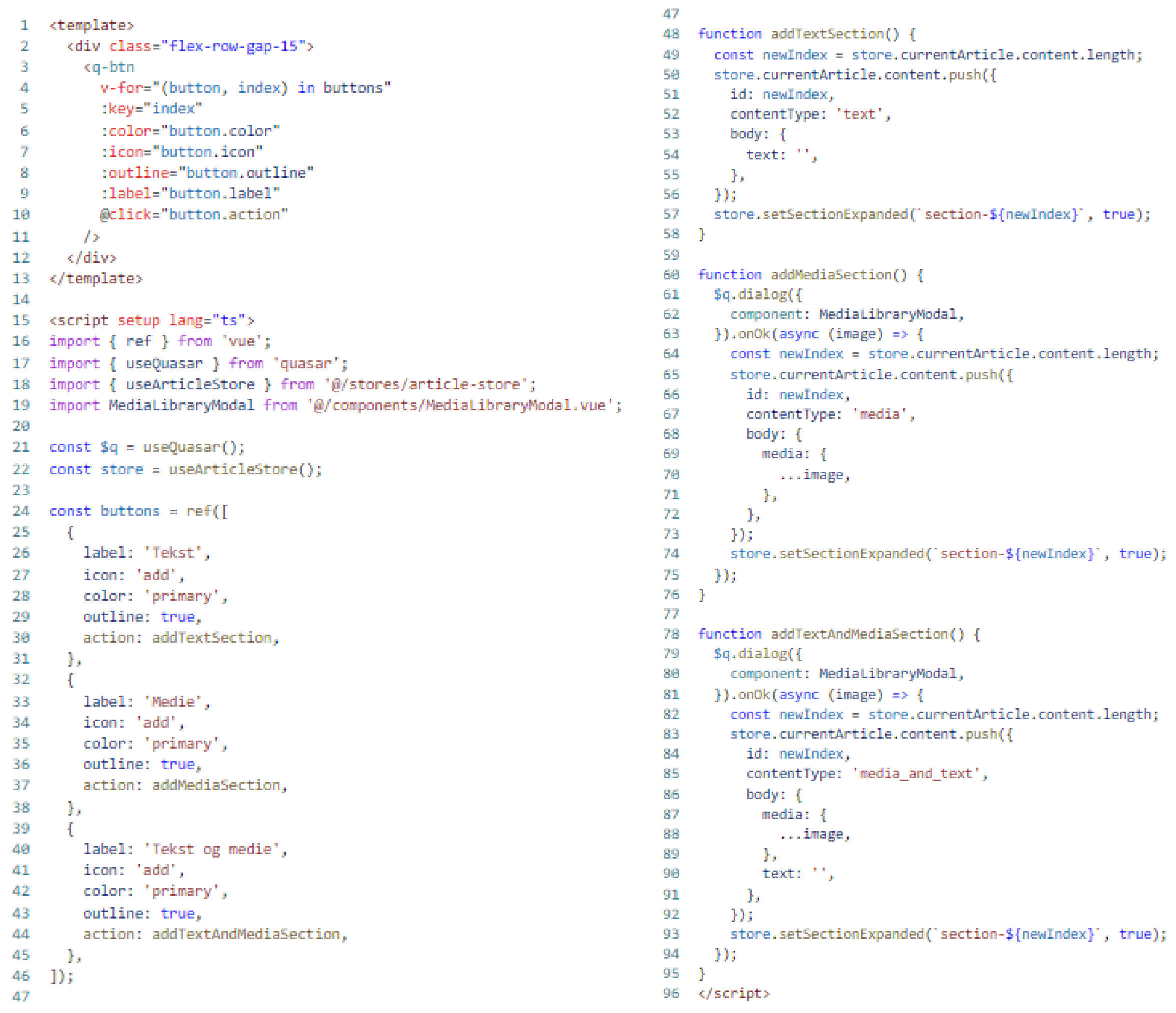
Appendix 4 - CMS file structure

```
CMS/
├─ .quasar/
├─ .vscode/
│  ├─ extensions.json
│  └─ settings.json
├─ public/
├─ src/
│  ├─ api/
│  │  └─ ApiClient.ts
│  ├─ boot/
│  │  ├─ axios.ts
│  │  └─ notify-defaults.ts
│  ├─ components/
│  │  └─ (various .vue files)
│  ├─ css/
│  │  ├─ app.scss
│  │  └─ quasar.variables.scss
│  ├─ interfaces/
│  │  ├─ article.ts
│  │  └─ user.ts
│  ├─ pages/
│  │  ├─ ContentManagerComponents/
│  │  │  └─ (various .vue files)
│  │  └─ (various .vue files)
│  ├─ router/
│  │  ├─ index.ts
│  │  └─ routes.ts
│  ├─ stores/
│  │  ├─ article-store.ts
│  │  ├─ index.ts
│  │  ├─ store-flag.d.ts
│  │  └─ user-store.ts
│  ├─ utils/
│  │  └─ helpers.ts
│  └─ App.vue
├─ .editorconfig
├─ .eslintignore
├─ .eslintrc.js
├─ .gitignore
├─ .npmrc
├─ .prettierrc
├─ index.html
├─ package.json
├─ postcss.config.js
├─ quasar.config.js
├─ README.md
├─ tsconfig.json
└─ yarn.lock
```

Appendix 5 - article-store.ts

```
import { ArticleItem } from '@/interfaces/article';
import { defineStore } from 'pinia';

const EMPTY_ARTICLE: ArticleItem = {
  title: '',
  coverMedia: {
    altText: '',
    fileName: '',
    mediaType: 'image',
  },
  excerpt: '',
  content: [],
  sources: [],
  weeks: null,
  afterBirth: null,
  category: null,
  slug: '',
};

type ExpandedSections = {
  [key: string]: boolean;
};

export const useArticleStore = defineStore('article', {
  state: () => ({
    step: 1,
    isEditing: false,
    currentArticle: deepCopy(EMPTY_ARTICLE),
    originalArticle: deepCopy(EMPTY_ARTICLE),
    expandedSections: {} as ExpandedSections,
  }),
  actions: {
    toggleSectionExpanded(sectionId: string) {
      this.expandedSections = {
        ...this.expandedSections,
        [sectionId]: !this.expandedSections[sectionId],
      };
    },
    setSectionExpanded(sectionId: string, expanded: boolean) {
      this.expandedSections = {
        ...this.expandedSections,
        [sectionId]: expanded,
      };
    },
    reset() {
      this.step = 1;
      this.isEditing = false;
      this.currentArticle = deepCopy(EMPTY_ARTICLE);
      this.expandedSections = {};
    },
  },
  getters: {
    hasUnsavedChanges(): boolean {
      return (
        JSON.stringify(this.currentArticle) !==
        JSON.stringify(this.originalArticle)
      );
    },
  },
});

// function since spread operator(...) doesn't go deep enough
// very simplified version of lodash _.cloneDeep()
function deepCopy<T>(obj: T): T {
  return JSON.parse(JSON.stringify(obj));
}
```

Appendix 6 - ConfigureMainContentSectionAddButtons.vue



```
<template>
  <div class="flex-row-gap-15">
    <q-btn
      v-for="(button, index) in buttons"
      :key="index"
      :color="button.color"
      :icon="button.icon"
      :outline="button.outline"
      :label="button.label"
      @click="button.action"
    />
  </div>
</template>

<script setup lang="ts">
import { ref } from 'vue';
import { useQuasar } from 'quasar';
import { useArticleStore } from '@/stores/article-store';
import MediaLibraryModal from '@/components/MediaLibraryModal.vue';

const $q = useQuasar();
const store = useArticleStore();

const buttons = ref([
  {
    label: 'Tekst',
    icon: 'add',
    color: 'primary',
    outline: true,
    action: addTextSection,
  },
  {
    label: 'Medie',
    icon: 'add',
    color: 'primary',
    outline: true,
    action: addMediaSection,
  },
  {
    label: 'Tekst og medie',
    icon: 'add',
    color: 'primary',
    outline: true,
    action: addTextAndMediaSection,
  },
]);


function addTextSection() {
  const newIndex = store.currentArticle.content.length;
  store.currentArticle.content.push({
    id: newIndex,
    contentType: 'text',
    body: {
      text: '',
    },
  });
  store.setSectionExpanded(`section-${newIndex}`, true);
}

function addMediaSection() {
  $q.dialog({
    component: MediaLibraryModal,
  }).onOk(async (image) => {
    const newIndex = store.currentArticle.content.length;
    store.currentArticle.content.push({
      id: newIndex,
      contentType: 'media',
      body: {
        media: {
          ...image,
        },
      },
    });
    store.setSectionExpanded(`section-${newIndex}`, true);
  });
}

function addTextAndMediaSection() {
  $q.dialog({
    component: MediaLibraryModal,
  }).onOk(async (image) => {
    const newIndex = store.currentArticle.content.length;
    store.currentArticle.content.push({
      id: newIndex,
      contentType: 'media_and_text',
      body: {
        media: {
          ...image,
        },
        text: '',
      },
    });
    store.setSectionExpanded(`section-${newIndex}`, true);
  });
}
</script>
```

Appendix 7 - ConfigureMainContent.vue

```
<template>
  <div class="flex column gap-24">
    <div class="flex column gap-8">
      <ConfigureMainContentSection :index="-2" content-type="title" />
      <ConfigureMainContentSection :index="-1" content-type="description" />
    </div>
    <p class="text-primary text-bold">7. Legg til seksjoner:</p>
    <ConfigureMainContentSectionAddButtons />
    <div class="flex column gap-8">
      <template
        v-for="(section, index) in store.currentArticle.content"
        :key="section.id"
      >
        <ConfigureMainContentSection
          :index="index"
          :content-type="section.contentType"
        />
      </template>
    </div>
  </div>
</template>

<script setup lang="ts">
import { useArticleStore } from '@/stores/article-store';
import ConfigureMainContentSection from '@/pages/ContentManagerComponents/ConfigureMainContentSection.vue';
import ConfigureMainContentSectionAddButtons from '@/pages/ContentManagerComponents/ConfigureMainContentSectionAddButtons.vue';

const store = useArticleStore();
</script>

<style scoped></style>

```

Appendix 8 - ConfigureMainContentSection.vue

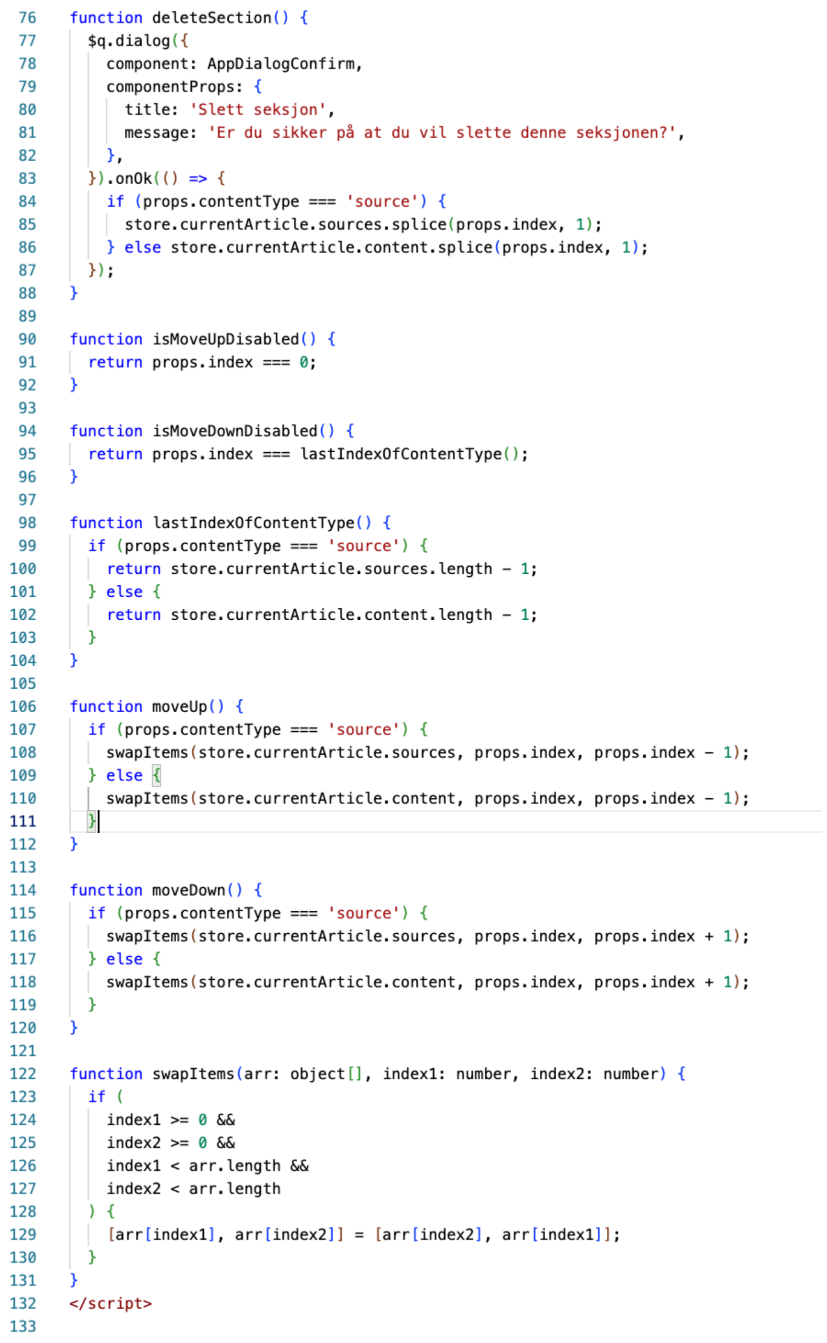
```
<template>
  <div class="expansion-item-container">
    <q-expansion-item
      :model-value="expanded"
      @update:model-value="
        expanded = $event;
        store.setSectionExpanded(
          contentType === 'source' ? `source-${index}` : `section-${index}`,
          $event
        );
      "
      :label="`${labelIndexComputed}: ${labelComputed}`"
      expand-icon-class="text-white"
      header-class="header"
      style="width: 100%"
    >
      <template #default>
        <!-- CONTENT WHEN OPEN -->
        <q-card class="card-border">
          <!-- TITLE -->
          <template v-if="contentType === 'title'"><TitleEdit /></template>

          <!-- DESCRIPTION -->
          <template v-if="contentType === 'description'">
            <DescriptionEdit />
          </template>

          <!-- TEXT -->
          <template v-if="contentType === 'text'">
            <TextEdit
              :model-value="
                store.currentArticle.content[index]?.body.text ?? ''
              "
              :index="index"
              @update:model-value="updateTextSectionBody(index, $event)"
            />
          </template>

          <!-- MEDIA -->
          <template v-if="contentType === 'media'">
            <MediaEdit
              :index="index"
              :model-value="
                store.currentArticle.content[index]?.body.media ?? emptyMedia
              "
              @update:model-value="updateMediaSectionBody(index, $event)"
            />
          </template>

          <!-- MEDIA AND TEXT -->
          <template v-if="contentType === 'media_and_text'">
            <TextEdit
              :model-value="
                store.currentArticle.content[index]?.body.text ?? ''
              "
              :index="index"
              @update:model-value="updateTextSectionBody(index, $event)"
            />
            <MediaEdit
              :index="index"
              :model-value="
                store.currentArticle.content[index]?.body.media ?? emptyMedia
              "
              @update:model-value="updateMediaSectionBody(index, $event)"
            />
            <div class="border-top padding-15">
              <span class="text-bold text-primary" style="padding-right: 15px"
                >Velg medieposisjon:</span
              >
              <q-btn-toggle
                :model-value="
                  store.currentArticle.content[index]?.body.media
                    ?.mediaPositionLeft ?? true
                "
                @update:model-value="updateMediaPositionLeft(index, $event)"
                toggle-color="primary"
                color="white"
                text-color="primary"
                :options="[
                  { label: 'Venstre', value: true },
                  { label: 'Høyre', value: false },
                ]"
              />
            </div>
          </template>

          <!-- SOURCE -->
          <template v-if="contentType === 'source'">
            <SourceEdit
              :model-value="store.currentArticle.sources[index]?.title ?? ''"
              :index="index"
              @update:model-value="updateSourceTitle(index, $event)"
            />
          </template>
        </q-card>
      </template>
    </q-expansion-item>

    <!-- BUTTONS -->
    <SectionButtons
      :index="index"
      :content-type="contentType"
      class="section-buttons-absolute"
      @toggle-edit="toggleSectionExpanded"
    />
  </div>
</template>
```

```

<script setup lang="ts">
import { computed, ref } from 'vue';
import { MediaItem } from '@/interfaces/article';
import { useArticleStore } from '@/stores/article-store';
import TextEdit from '@/pages/ContentManagerComponents/EditText.vue';
import MediaEdit from '@/pages/ContentManagerComponents/EditMedia.vue';
import TitleEdit from '@/pages/ContentManagerComponents/EditTitle.vue';
import SourceEdit from '@/pages/ContentManagerComponents/EditSource.vue';
import DescriptionEdit from '@/pages/ContentManagerComponents/EditDescription.vue';
import SectionButtons from '@/pages/ContentManagerComponents/ConfigureMainContentSectionButtons.vue';

const store = useArticleStore();

const props = defineProps({
  index: {
    type: Number,
    required: true,
  },
  contentType: {
    type: String,
    required: true,
  },
});

const expanded = ref(
  store.expandedSections[
    props.contentType === 'source'
      ? `source-${props.index}`
      : `section-${props.index}`
  ] || false
);

function toggleSectionExpanded() {
  expanded.value = !expanded.value;
  store.setSectionExpanded(
    props.contentType === 'source'
      ? `source-${props.index}`
      : `section-${props.index}`,
    expanded.value
  );
}
const labelComputed = computed(() => {
  return props.contentType === 'source'
    ? 'Kilde'
    : props.contentType === 'text'
    ? 'Tekst'
    : props.contentType === 'media'
    ? 'Medie'
    : props.contentType === 'media_and_text'
    ? 'Tekst og medie'
    : props.contentType === 'title'
    ? 'Tittel'
    : 'Beskrivelse';
});

const labelIndexComputed = computed(() => {
  return props.contentType !== 'source' ? props.index + 3 : props.index + 1;
});

const emptyMedia = { fileName: '', altText: '' };

function updateSourceTitle(index: number, newValue: string) {
  store.currentArticle.sources[index].title = newValue;
}

function updateTextSectionBody(index: number, newValue: string) {
  store.currentArticle.content[index].body.text = newValue;
}

function updateMediaSectionBody(index: number, newValue: MediaItem) {
  store.currentArticle.content[index].body.media = newValue;
}
```

```

function updateMediaPositionLeft(index: number, newValue: boolean) {
  if (
    !store.currentArticle.content[index] ||
    !store.currentArticle.content[index].body ||
    !store.currentArticle.content[index].body.media
  )
    return;
  // @ts-expect-error It will not be undefined. I promise!
  store.currentArticle.content[index].body.media.mediaPositionLeft = newValue;
}
</script>

<style lang="scss">
.header {
  color: white;
  font-weight: 500;
  background-color: $primary;
}

.border-top {
  border-top: 1px solid $secondary;
}

.card-border {
  border-left: 1px solid $secondary;
  border-right: 1px solid $secondary;
  border-bottom: 3px solid $secondary;
}

.section-buttons-absolute {
  position: absolute;
  right: 0;
  z-index: 99999;
}

.expansion-item-container {
  display: flex;
  flex-direction: row;
  align-items: flex-start;
  position: relative;
}
</style>

```

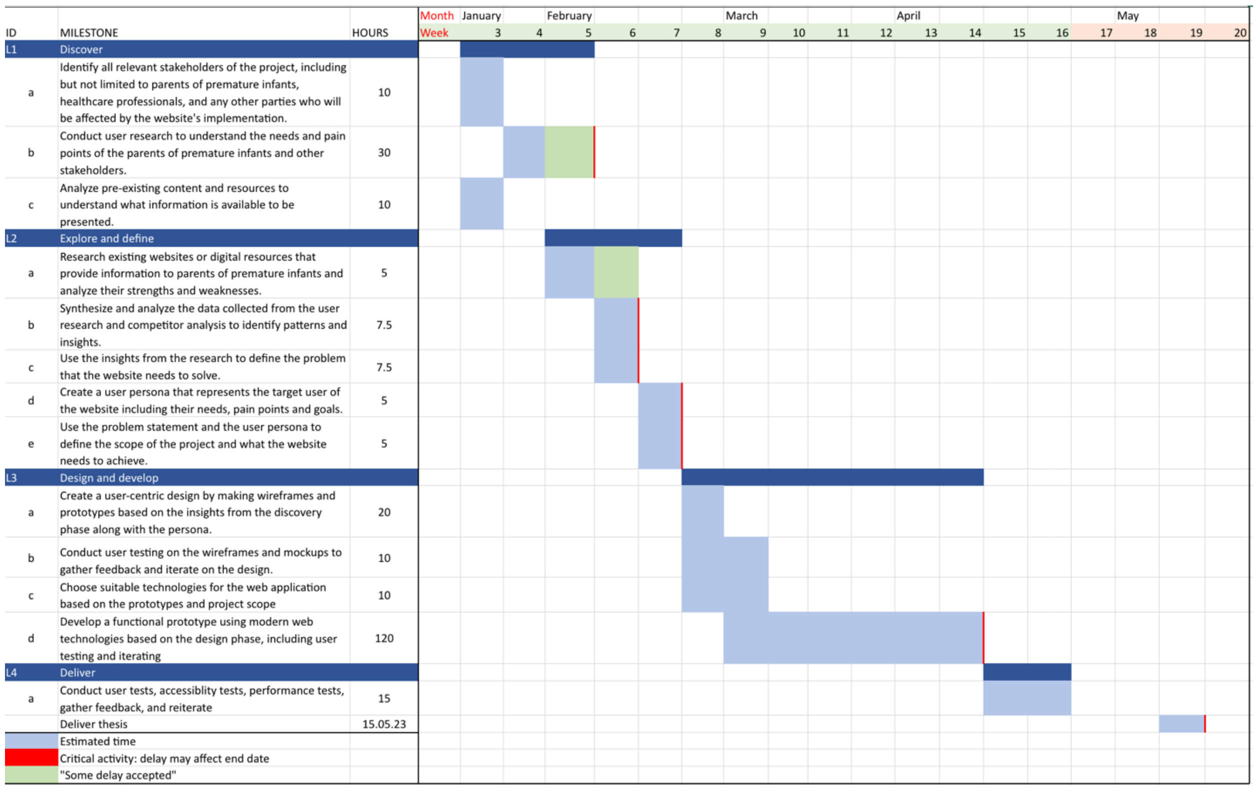
Appendix 9 - ConfigureMainContentSectionButtons.vue

```
<template>
  <div :class="buttonClass" v-if="buttonComputed">
    <q-btn
      v-for="(action, idx) in actions"
      :key="idx"
      flat
      :icon="action.icon"
      class="height-46"
      color="primary"
      @click="action.onClick"
      :disabled="action.isDisabled?.()"
    />
  </div>
  <div v-else class="edit-button">
    <q-btn
      flat
      icon="edit"
      class="height-46"
      color="primary"
      @click="emit('toggle-edit')"
    />
  </div>
</template>

<script setup lang="ts">
import { computed } from 'vue';
import { useQuasar } from 'quasar';
import { useArticleStore } from '@/stores/article-store';
import AppDialogConfirm from '@/components/AppDialogConfirm.vue';

const $q = useQuasar();
const store = useArticleStore();

const props = defineProps({
  index: {
    type: Number,
    required: true,
  },
  contentType: {
    type: String,
    required: true,
  },
});

const emit = defineEmits(['toggle-edit']);

const buttonComputed = computed(() =>
  ['text', 'media', 'media_and_text', 'source'].includes(props.contentType)
);

const actions = [
  {
    icon: 'arrow_upward',
    onClick: moveUp,
    isDisabled: isMoveUpDisabled,
  },
  {
    icon: 'arrow_downward',
    onClick: moveDown,
    isDisabled: isMoveDownDisabled,
  },
  {
    icon: 'edit',
    onClick: () => emit('toggle-edit'),
  },
  {
    icon: 'delete',
    onClick: deleteSection,
  },
];
```



```
function deleteSection() {
  $q.dialog({
    component: AppDialogConfirm,
    componentProps: {
      title: 'Slett seksjon',
      message: 'Er du sikker på at du vil slette denne seksjonen?',
    },
  }).onOk(() => {
    if (props.contentType === 'source') {
      store.currentArticle.sources.splice(props.index, 1);
    } else store.currentArticle.content.splice(props.index, 1);
  });
}

function isMoveUpDisabled() {
  return props.index === 0;
}

function isMoveDownDisabled() {
  return props.index === lastIndexOfContentType();
}

function lastIndexOfContentType() {
  if (props.contentType === 'source') {
    return store.currentArticle.sources.length - 1;
  } else {
    return store.currentArticle.content.length - 1;
  }
}

function moveUp() {
  if (props.contentType === 'source') {
    swapItems(store.currentArticle.sources, props.index, props.index - 1);
  } else {
    swapItems(store.currentArticle.content, props.index, props.index - 1);
  }
}

function moveDown() {
  if (props.contentType === 'source') {
    swapItems(store.currentArticle.sources, props.index, props.index + 1);
  } else {
    swapItems(store.currentArticle.content, props.index, props.index + 1);
  }
}

function swapItems(arr: object[], index1: number, index2: number) {
  if (
    index1 >= 0 &&
    index2 >= 0 &&
    index1 < arr.length &&
    index2 < arr.length
  ) {
    [arr[index1], arr[index2]] = [arr[index2], arr[index1]];
  }
}
</script>
```

Appendix 10 - Gantt chart



			Month	January		February				March				April					May		
ID	MILESTONE	HOURS	Week	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L1	Discover			■	■	■															
a	Identify all relevant stakeholders of the project, including but not limited to parents of premature infants, healthcare professionals, and any other parties who will be affected by the website's implementation.	10		🟦																	
b	Conduct user research to understand the needs and pain points of the parents of premature infants and other stakeholders.	30			🟦	🟩🟥															
c	Analyze pre-existing content and resources to understand what information is available to be presented.	10		🟦																	
L2	Explore and define					■	■	■													
a	Research existing websites or digital resources that provide information to parents of premature infants and analyze their strengths and weaknesses.	5				🟦	🟩														
b	Synthesize and analyze the data collected from the user research and competitor analysis to identify patterns and insights.	7.5					🟦🟥														
c	Use the insights from the research to define the problem that the website needs to solve.	7.5					🟦🟥														
d	Create a user persona that represents the target user of the website including their needs, pain points and goals.	5						🟦🟥													
e	Use the problem statement and the user persona to define the scope of the project and what the website needs to achieve.	5						🟦🟥													
L3	Design and develop								■	■	■	■	■	■	■						
a	Create a user-centric design by making wireframes and prototypes based on the insights from the discovery phase along with the persona.	20							🟦												
b	Conduct user testing on the wireframes and mockups to gather feedback and iterate on the design.	10							🟦	🟦											
c	Choose suitable technologies for the web application based on the prototypes and project scope	10							🟦	🟦											
d	Develop a functional prototype using modern web technologies based on the design phase, including user testing and iterating	120								🟦	🟦	🟦	🟦	🟦	🟦🟥						
L4	Deliver															■	■				
a	Conduct user tests, accessiblity tests, performance tests, gather feedback, and reiterate	15														🟦	🟦				
	Deliver thesis	15.05.23																		🟦🟥	
🟦	Estimated time																				
🟥	Critical activity: delay may affect end date																				
🟩	"Some delay accepted"																				

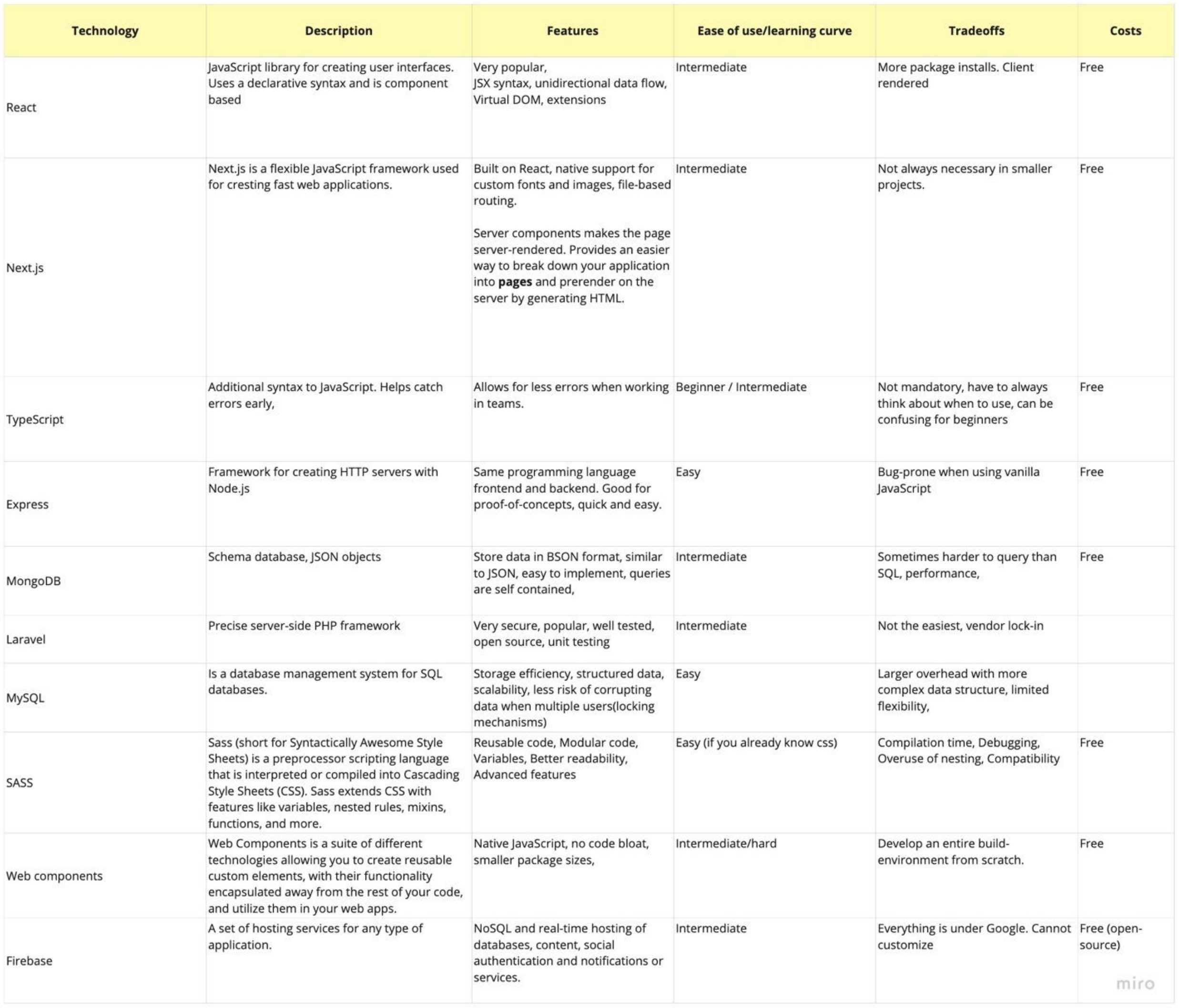
Appendix 11 - ApiClient.ts

```typescript
import axios from 'axios';
import {
  UpdateUserItem,
  UserCredentials,
  UserLoginCredentials,
  UserRegisterCredentials,
  UserResponse,
} from '@/interfaces/user';
import { api } from '@/boot/axios';
import { ArticleItem, CategoryResponseItem } from '@/interfaces/article';

export class UserClient {
  async login(
    credentials: UserLoginCredentials
  ): Promise<UserResponse | undefined> {
    try {
      const response = await api.post('/login', credentials);
      if (response.data && response.data.token) {
        return response.data as UserResponse;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to log in: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }

  async revokeToken(token: string | undefined): Promise<void> {
    try {
      await api.post('/revoke-token', { token });
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to revoke token: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }

  async refreshToken(): Promise<UserResponse | undefined> {
    // Avoid unecessary refresh token calls
    if (localStorage.getItem('isLoggedIn') !== 'true') {
      return undefined;
    }

    try {
      const response = await api.post('/refresh-token');
      if (response.data && response.data.token) {
        return response.data as UserResponse;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to refresh token: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }

  async register(
    credentials: UserRegisterCredentials
  ): Promise<UserResponse | undefined> {
    try {
      const response = await api.post('/users', credentials);
      if (response.data) {
        return response.data as UserResponse;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to register user: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }

  async getAllUsers(): Promise<UserCredentials[] | undefined> {
    try {
      const response = await api.get('/users');
      if (response.data) {
        return response.data as UserCredentials[];
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to get users: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }

  async updateUser(
    username: string,
    updatedUserCredentials: UpdateUserItem
  ): Promise<UserResponse | undefined> {
    try {
      const response = await api.put(
        `/users/${username}`,
        updatedUserCredentials
      );
      if (response.data) {
        return response.data as UserResponse;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to update user: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }

  async deleteUser(username: string): Promise<UserResponse | undefined> {
    try {
      const response = await api.delete<UserResponse>(`/users/${username}`);
      if (response.data) {
        return response.data;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to delete user: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }
}

export class ArticleClient {
  async getCategories(): Promise<CategoryResponseItem[] | undefined> {
    try {
      const response = await api.get('/categories');
      if (response.data) {
        return response.data;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to get categories: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }

  async getArticles(): Promise<ArticleItem[] | undefined> {
    try {
      const response = await api.get('/articles');
      if (response.data) {
        return response.data;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to get articles: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }

  async postArticle(article: ArticleItem) {
    try {
      const response = await api.post('/articles', article);
      if (response.data) {
        return response.data;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to post article: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }
  async putArticle(slug: string, article: ArticleItem) {
    try {
      const response = await api.put(`/articles/${slug}`, article);
      if (response.data) {
        return response.data;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to update article: ${error.message}`);
      } else {
        //console.log(error);
      }
    }
  }

  async deleteArticle(slug: string) {
    try {
      const response = await api.delete(`/articles/${slug}`);
      if (response.data) {
        return response.data;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to delete article: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }
}

export class MediaClient {
  async uploadMedia(file: readonly File[]) {
    try {
      const response = await api.post('media', { images: file });
      if (response.data) {
        return response.data;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to post media: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }

  async getMediaList(query?: mediaQuery) {
    let apiLink;
    if (query) {
      apiLink = `media/?type=${query}`;
    } else {
      apiLink = 'media';
    }
    try {
      const response = await api.get(apiLink);
      if (response.data) {
        return response.data;
      }
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // throw new Error(`Failed to post media: ${error.message}`);
      } else {
        // console.log(error);
      }
    }
  }
}

export type mediaQuery = 'image' | 'video' | 'all';

```

Appendix 12 - Client folder structure

```
client/
├─ public/
│  └─ vite.svg
├─ src/
│  ├─ api/
│  │  ├─ articleCalls.js
│  │  ├─ authCalls.js
│  │  └─ axios.js
│  ├─ assets/
│  │  ├─ backgrounds/
│  │  └─ icons/
│  ├─ components/
│  │  └─ (various component folders containing component file and .scss)
│  ├─ pages/
│  │  └─ (various component folders containing component file and .scss used as elements in React Router)
│  ├─ routes/
│  │  ├─ ForgotRoute.jsx
│  │  ├─ PrivateRoute.jsx
│  │  └─ PublicRoute.jsx
│  ├─ scss/
│  │  ├─ colors.scss
│  │  ├─ index.scss
│  │  └─ spacing.scss
│  ├─ utils/
│  │  ├─ AuthContext.jsx
│  │  └─ helpers.js
│  ├─ App.jsx
│  └─ main.jsx
├─ .env
├─ .gitignore
├─ index.html
├─ package.json
├─ vite.config.js
└─ yarn.lock
```

Appendix 13 - Technology evaluation



Technology	Description	Features	Ease of use/learning curve	Tradeoffs	Costs
React	JavaScript library for creating user interfaces. Uses a declarative syntax and is component based	Very popular, JSX syntax, unidirectional data flow, Virtual DOM, extensions	Intermediate	More package installs. Client rendered	Free
Next.js	Next.js is a flexible JavaScript framework used for cresting fast web applications.	Built on React, native support for custom fonts and images, file-based routing. Server components makes the page server-rendered. Provides an easier way to break down your application into **pages** and prerender on the server by generating HTML.	Intermediate	Not always necessary in smaller projects.	Free
TypeScript	Additional syntax to JavaScript. Helps catch errors early,	Allows for less errors when working in teams.	Beginner / Intermediate	Not mandatory, have to always think about when to use, can be confusing for beginners	Free
Express	Framework for creating HTTP servers with Node.js	Same programming language frontend and backend. Good for proof-of-concepts, quick and easy.	Easy	Bug-prone when using vanilla JavaScript	Free
MongoDB	Schema database, JSON objects	Store data in BSON format, similar to JSON, easy to implement, queries are self contained,	Intermediate	Sometimes harder to query than SQL, performance,	Free
Laravel	Precise server-side PHP framework	Very secure, popular, well tested, open source, unit testing	Intermediate	Not the easiest, vendor lock-in	
MySQL	Is a database management system for SQL databases.	Storage efficiency, structured data, scalability, less risk of corrupting data when multiple users(locking mechanisms)	Easy	Larger overhead with more complex data structure, limited flexibility,	
SASS	Sass (short for Syntactically Awesome Style Sheets) is a preprocessor scripting language that is interpreted or compiled into Cascading Style Sheets (CSS). Sass extends CSS with features like variables, nested rules, mixins, functions, and more.	Reusable code, Modular code, Variables, Better readability, Advanced features	Easy (if you already know css)	Compilation time, Debugging, Overuse of nesting, Compatibility	Free
Web components	Web Components is a suite of different technologies allowing you to create reusable custom elements, with their functionality encapsulated away from the rest of your code, and utilize them in your web apps.	Native JavaScript, no code bloat, smaller package sizes,	Intermediate/hard	Develop an entire build-environment from scratch.	Free
Firebase	A set of hosting services for any type of application.	NoSQL and real-time hosting of databases, content, social authentication and notifications or services.	Intermediate	Everything is under Google. Cannot customize	Free (open-source)

miro

Appendix 14 - User testing final product

Frontend

Oppgave	Tid brukt	Feil	Notater
Logg inn med brukernavn superadmin og passord passord	40s	2	går via hamburgermenyen først, ingen login der, finner login i footer senere, feiler pga. case sensitive brukernavn
Naviger til barnets løp og filtrer etter periode "tidlig intensiv" og født i uke 30	26s	0	går via hamburgermenyen
Søk etter en artikkel og legg den til i favoritter	26s	0	oversä hjertet først, scrollet litt langt, fant den på scroll opp, føler kanskje hjertet ikke hører til artikkelen, siden den er i en egen blokk
Naviger til "Hvordan styrke båndet mellom foreldre og baby", uten bruk av søkefelt, hamburgermeny eller "Aktuelle artikler"	28s	4	Går først til foreldrerollen. Kanskje en mer spesifikk sånn "Oss og barnet, meg og barnet"

CMS

Oppgave	Tid brukt	Feil	Notater
Logg inn med brukernavn superadmin og passord passord	5s	0	
Opprett en falsk temaside	4m30s	0	Liker segmenterte steg, tydelig hvor i prosessen man er, "tilbakestill" venstre, neste høyre, nice med preview og seksjoner, kanskje tydeliggjøre at den ene er forhåndsvisning av kortet og forhåndsvisning av artikkelen
Rediger den samme temasiden	19s	0	Reverser array, nyeste først, endre til oppdater under rediger
Legg til en ny bruker	53s	0	Liker at den er adskilt i sidemenyen, kanskje endre fargen på opprett bruker knappen for å skille mer fra resten (accent orange), lagre/opprett i stedenfor OK(?)

Frontend

Oppgave	Tid brukt	Feil	Notater
Logg inn med brukernavn superadmin og passord passord	55s	2	Finner ikke logg inn knappen, tror den er i hamburgermenyen
Naviger til barnets løp og filtrer etter periode "tidlig intensiv" og født i uke 30	1m 50s	1	Litt trøbbel med teknologi
Søk etter en artikkel og legg den til i favoritter	1m 30s	1	Søke etter, temasider var trøblete
Naviger til "Hvordan styrke båndet mellom foreldre og baby", uten bruk av søkefelt, hamburgermeny eller "Aktuelle artikler"	1m 32s	1	Først foreldrerollen, barnets løp, deretter kommunikasjon

CMS

Oppgave	Tid brukt	Feil	Notater
Logg inn med brukernavn superadmin og passord passord	8s	0	
Opprett en falsk temaside	8m 20s	3	Forsidebilde, sliter litt med stegene, kanskje noen beskrivelser av hvordan man lager en artikkel
Rediger den samme temasiden	8m 30s	0	
Legg til en ny bruker	10s	0	

Frontend

Oppgave	Tid brukt	Feil	Notater
Logg inn med brukernavn superadmin og passord passord	16s	1	går via hamburgermenyen
Naviger til barnets løp og filtrer etter periode "tidlig intensiv" og født i uke 30	19s	0	går via hamburgermenyen, finner fort, pil er for nært design
Søk etter en artikkel og legg den til i favoritter	6s	0	fant fort
Naviger til "Hvordan styrke båndet mellom foreldre og baby", uten bruk av søkefelt, hamburgermeny eller "Aktuelle artikler"	40s	12	går til foreldrerollen først, leter litt inne der, andres erfaringer, går hjem, prøver foreldrerollen igjen, etter opphold, andres erfaringer, kontaktpersoner, før opphold, premature barn, barnets løp. Litt vag forskjell mellom ordlyden på kategoriene. Det er mye som kan være under samme. Les mer forsvinner nedenfor kortet. Liker varslinger/toasters

CMS

Oppgave	Tid brukt	Feil	Notater
Logg inn med brukernavn superadmin og passord passord	3s	0	
Opprett en falsk temaside	4m 39s	0	Fin og naturlig gjennomgang av opprettelsesstegene. Føler han alltid har kontroll, siden det ligger forhåndsvisning.
Rediger den samme temasiden	40s	0	Så først på toppen av listen over eksisterende innhold, redigerer fort,
Legg til en ny bruker	32s	0	

Appendix 15 - Notes from interviews

[illegible]

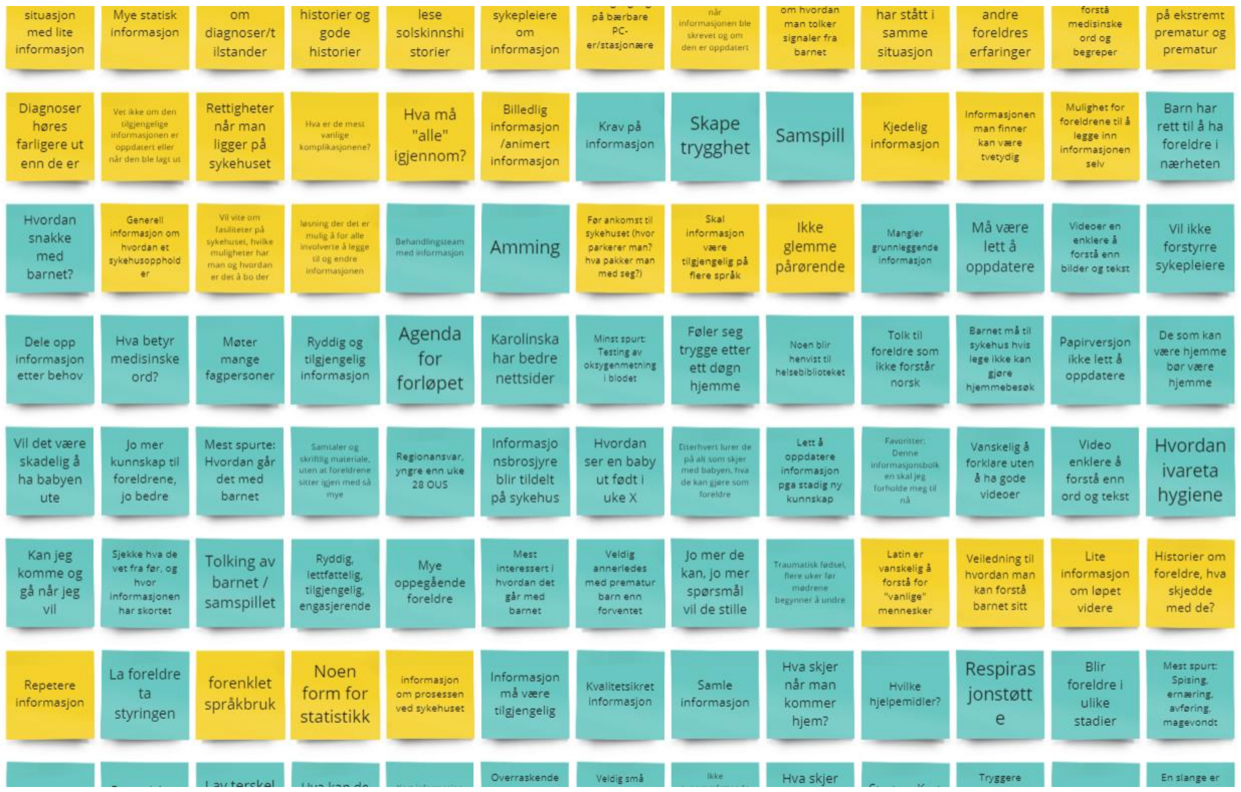
[illegible]

[illegible]

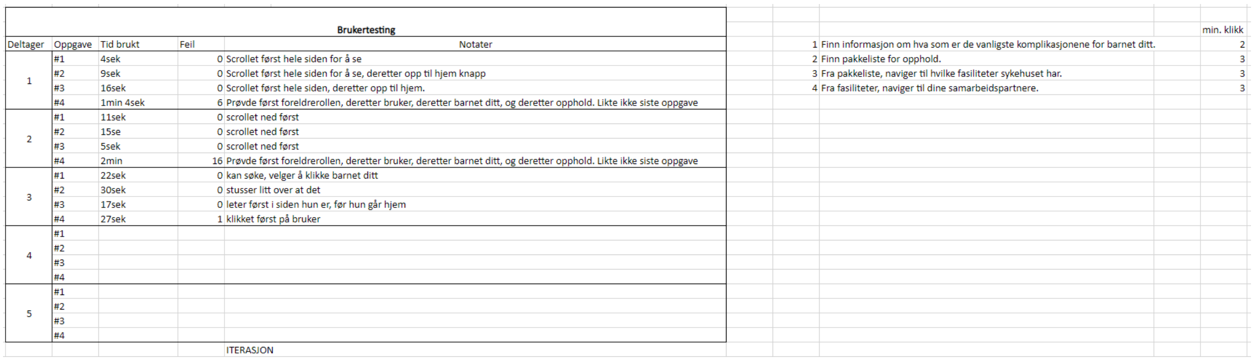
[illegible]

[illegible]

Appendix 16 - Affinity Diagram


situasjon med lite informasjon
Mye statisk informasjon
om diagnoser/tilstander
historier og gode historier
lese solskinnshistorier
sykepleiere om informasjon
på bærbare PC-er/stasjonære
når informasjonen ble skrevet og om den er oppdatert
om hvordan man tolker signaler fra barnet
har stått i samme situasjon
andre foreldres erfaringer
forstå medisinske ord og begreper
på ekstremt prematur og prematur
Diagnoser høres farligere ut enn de er
Vet ikke om den tilgjengelige informasjonen er oppdatert eller når den ble lagt ut
Rettigheter når man ligger på sykehuset
Hva er de mest vanlige komplikasjonene?
Hva må "alle" igjennom?
Billedlig informasjon /animert informasjon
Krav på informasjon
Skape trygghet
Samspill
Kjedelig informasjon
Informasjonen man finner kan være tvetydig
Mulighet for foreldrene til å legge inn informasjonen selv
Barn har rett til å ha foreldre i nærheten
Hvordan snakke med barnet?
Generell informasjon om hvordan et sykehusopphold er
Vil vite om fasiliteter på sykehuset, hvilke muligheter har man og hvordan er det å bo der
løsning der det er mulig å for alle involverte å legge til og endre informasjonen
Behandlingsteam med informasjon
Amming
Før ankomst til sykehuset (hvor parkerer man? hva pakker man med seg?)
Skal informasjon være tilgjengelig på flere språk
Ikke glemme pårørende
Mangler grunnleggende informasjon
Må være lett å oppdatere
Videoer en enklere å forstå enn bilder og tekst
Vil ikke forstyrre sykepleiere
Dele opp informasjon etter behov
Hva betyr medisinske ord?
Møter mange fagpersoner
Ryddig og tilgjengelig informasjon
Agenda for forløpet
Karolinska har bedre nettsider
Minst spurt: Testing av oksygenmetning i blodet
Føler seg trygge etter ett døgn hjemme
Noen blir henvist til helsebiblioteket
Tolk til foreldre som ikke forstår norsk
Barnet må til sykehus hvis lege ikke kan gjøre hjemmebesøk
Papirversjon ikke lett å oppdatere
De som kan være hjemme bør være hjemme
Vil det være skadelig å ha babyen ute
Jo mer kunnskap til foreldrene, jo bedre
Mest spurte: Hvordan går det med barnet
Samtaler og skriftlig materiale, uten at foreldrene sitter igjen med så mye
Regionansvar, yngre enn uke 28 OUS
Informasjonsbrosjyre blir tildelt på sykehus
Hvordan ser en baby ut født i uke X
Etterhvert lurer de på alt som skjer med babyen, hva de kan gjøre som foreldre
Lett å oppdatere informasjon pga stadig ny kunnskap
Favoritter: Denne informasjonsboken en skal jeg forholde meg til nå
Vanskelig å forklare uten å ha gode videoer
Video enklere å forstå enn ord og tekst
Hvordan ivareta hygiene
Kan jeg komme og gå når jeg vil
Sjekke hva de vet fra før, og hvor informasjonen har skortet
Tolking av barnet / samspillet
Ryddig, lettfattelig, tilgjengelig, engasjerende
Mye oppegående foreldre
Mest interessert i hvordan det går med barnet
Veldig annerledes med prematur barn enn forventet
Jo mer de kan, jo mer spørsmål vil de stille
Traumatisk fødsel, flere uker før mødrene begynner å undre
Latin er vanskelig å forstå for "vanlige" mennesker
Veiledning til hvordan man kan forstå barnet sitt
Lite informasjon om løpet videre
Historier om foreldre, hva skjedde med de?
Repetere informasjon
La foreldre ta styringen
forenklet språkbruk
Noen form for statistikk
informasjon om prosessen ved sykehuset
Informasjon må være tilgjengelig
Kvalitetsikret informasjon
Samle informasjon
Hva skjer når man kommer hjem?
Hvilke hjelpemidler?
Respirasjonstøtte
Blir foreldre i ulike stadier
Mest spurt: Spising, ernæring, avføring, magevondt
Lav terskel
Overraskende
Veldig små
Ikke
Hva skjer
Tryggere
En slange er

Appendix 17 - Low Fidelity User Testing



Brukertesting

Deltager	Oppgave	Tid brukt	Feil	Notater
1	#1	4sek	0	Scrollet først hele siden for å se
	#2	9sek	0	Scrollet først hele siden for å se, deretter opp til hjem knapp
	#3	16sek	0	Scrollet først hele siden, deretter opp til hjem.
	#4	1min 4sek	6	Prøvde først foreldrerollen, deretter bruker, deretter barnet ditt, og deretter opphold. Likte ikke siste oppgave
2	#1	11sek	0	scrollet ned først
	#2	15se	0	scrollet ned først
	#3	5sek	0	scrollet ned først
	#4	2min	16	Prøvde først foreldrerollen, deretter bruker, deretter barnet ditt, og deretter opphold. Likte ikke siste oppgave
3	#1	22sek	0	kan søke, velger å klikke barnet ditt
	#2	30sek	0	stusser litt over at det
	#3	17sek	0	leter først i siden hun er, før hun går hjem
	#4	27sek	1	klikket først på bruker
4	#1			
	#2			
	#3			
	#4			
5	#1			
	#2			
	#3			
	#4			
				ITERASJON

		min. klikk
1	Finn informasjon om hva som er de vanligste komplikasjonene for barnet ditt.	2
2	Finn pakkeliste for opphold.	3
3	Fra pakkeliste, naviger til hvilke fasiliteter sykehuset har.	3
4	Fra fasiliteter, naviger til dine samarbeidspartnere.	3

NTNU

Norwegian University of
Science and Technology