University of Bergen
Department of informatics

Western Norway University of Applied Sciences
Department of Computing, Mathematics and Physics

Decision support framework for choosing treatment

Author: André Dyrstad
Supervisor: Adrian Rutle
Co-supervisor: Tori Smedal

May, 2019

Abstract

With the release of "Fritt behandlingsvalg" in 2015, Norwegian patients got the right to select where they want to attend special treatment. As of now, there is no easy way to compare treatment centers, as the information about their treatments can only be found at their respective websites.

During this study, we are going to look at the current problems with "Fritt behandlingsvalg" and try to develop a recommender system that helps patients select a treatment center that suits their needs. First, we implement a prototype based on input from the Norwegian Multiple Sclerosis Competence Centre. Then, we conduct a set of surveys and experiments to test our prototype and improve it through several iterations.

Based on experience and feedback, we present a proposal to a general framework that can collect data from treatment centers and recommend centers based on patient preferences and needs.

Acknowledgements

First and foremost, I want to thank my supervisor Adrian Rutle for helping me over the course of the master thesis.

I would also like to thank my co-supervisor Tori Smedal as well as Anne Britt Rundhovde Skår and Lars Bø from Norwegian Multiple Sclerosis Competence Centre for helping me test and develop my application as well as providing useful information about Multiple Sclerosis and "Fritt behandlingsvalg".

André Dyrstad
03 June, 2019

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In 2015, Norway introduced an arrangement called "Fritt behandlingsvalg". This suggests that people are allowed to choose where they want to attend medical treatment. Then, in 2017, the government introduced "Fritt rehabiliteringsvalg" where patients could select treatment centers within the specialist health service. Previously, patients could only choose among the treatment centers that were located within their region. Now, patients can choose between all public institutions as well as private centers that have been approved by Helfo. The goal with this arrangement was to shorten the wait times by distributing patients more evenly across all centers, as well as giving the patient the possibility to adjust the outcome when selecting a treatment center.

1.1 Background

As mentioned above, Norwegian patients have the right to choose their place of treatment. All rehabilitation centers within the specialist health service are listed on the website of helsenorge.no [11]. The problem is, that the list only contains the wait times for the treatment and does not say anything about what the treatment center has to offer. This kind of information can be crucial when selecting a treatment center and the patients should be able to access this information.

As of now, there is no easy way to determine which treatment center to choose. The choice is often made by a doctor and is based on his or her knowledge about the different treatment centers. While this might work to some extent, it is not ideal in the long run. We can not expect every doctor to know everything about all available treatment centers in Norway, nor keep the knowledge up to date.

1.2 Research questions

The goal of this research is to improve the current system for selecting treatment centers. Our approach is to make a website that can recommend treatment centers based on patients needs. In order to achieve this goal, we attempt to answer the following research questions:

- Is it possible to make a digital solution for recommending treatment centers?
- Is it possible to make a general framework that can be applied to a variety of different treatments and diseases?

To answer the first research question, we have decided to focus on making a recommendation system that works for Multiple sclerosis (MS). Then, we will try to adapt our recommendation system to work for different diseases. After completing these two steps, we should be able to answer our research questions.

In addition to our research questions, we want our application to follow a set of criteria. These criteria are used as a guideline to see if we actually improved the current solution for selecting a treatment center:

- The application should be easy to use for everyone involved.
- The recommender system should be fair. Treatment centers should not be recommended at random or based on the alphabetical order.
- The given recommendation should be easy to understand and should provide useful information when selecting a treatment center.

1.3 Chapter outline

Chapter 2: Motivation - This chapter describes the motivation behind the research. We are also going to look at MS and why we should improve the system currently in use.

Chapter 3: Related work - Provides information about other applications and systems that tries to improve treatment center selection. These systems are then compared with our own application

Chapter 4: Method - In this chapter, we first give a description of our design methodology and our software development methods. Later, we look at our research methodology.

Chapter 5: Design - Here, we describe all the modules in the application and how they communicate with each other. The chapter does also provide information about our client-server relationship, as well as a short introduction to Machine Learning, Rule-based Systems, and Representational State Transfer.

Chapter 6: Implementation - This chapter describes how the application is implemented and which tools and frameworks we used.

Chapter 7: Results - Here, we present the results from our surveys.

Chapter 8: Discussion - In this chapter, we discuss previously used frameworks, related work, and come with a conclusion for our project.

Chapter 9: Future work - In the final chapter, we look at some features we did not have the time or resources to complete.

Chapter 2

Motivation

This chapter describes the motivation behind the research. We are also going to look at MS and why we should improve the system currently in use.

2.1 Multiple sclerosis

MS is a chronic disease that can impact all areas of the brain and spinal cord. These structures make up the Central nervous system (CNS). The CNS is the source of all our thinking, feelings and actions. It controls most physical functions and is the receiving end of all our perceptions of our surroundings. MS can, therefore, disrupt a large variety of physical functions, from normal functions, like walking, to solving complicated cognitive tasks. Other common symptoms include pain, numbness, visual disturbances, bladder control problems, and fatigue, just to mention a few[41].

The Norwegian Directorate for e-health [35] states that everyone has a different experience with MS. MS-symptoms are highly individually variable, and the disease course is very unpredictable. Symptoms may initially be vague and are not specific for MS. Some symptoms are long-lasting, some last only for a short amount of time. This variability sometimes makes MS difficult to diagnose early. When the diagnosis is set, it is often possible to get a treatment that can slow down or stop the disease progression. Early treatment can prevent

future MS-attacks. As a result, early diagnosis and treatment are important when it comes to MS.

Because of the diversity of symptoms, it is important to give the correct recommendations and treatment to each patient. Some patients might need rehabilitation for bladder problems, while others may need rehabilitation for work –related problems and cognitive dysfunction. So far, an unsolved question has been: How do we ensure that each patient is recommended to the optimal treatment center? This is where our application comes in. By matching each patient needs to the services offered by the treatment centers, we should be able to help patients select the right treatment center to attend treatment[41].

2.2 Digitalization

Before the computer was commonly used, all tasks were done by hand. While these methods worked back then, it was a slow process and there where a lot of people involved. Then the computer came. A lot of tasks were moved to a digital platform and everything became more efficient. Instead of doing all the work yourself, you could just ask the computer to do it for you. While many companies adapted to these kinds of methods, some were left behind and stuck with the old methods.

2.2.1 What is digitalization?

Digitalization is the process of moving information from a physical form to a digital form. Paper documents can be scanned or rewritten to a Portable Document Format (PDF) file. Frequency modulation broadcasting is slowly being replaced by Digital Audio Broadcasting. Information and messages can be sent over the internet instead of using old fashion mail services. You don't even need to visit the bank to manage your account anymore. All these things are examples of digitalization and are affecting your life more than you can imagine.

2.2.2 Why digitalization?

Let's look a bit deeper into the "paper to PDF" scenario. When dealing with a lot of papers, you need a lot of storage. If you want to store one million pages in physical form, you will need about $24m^2$ of space to store all your documents[7]. If you want to store these pages electronically, you could fit the same amount of pages on a 64GB memory stick. The cost makes a huge difference as well. You can get a memory stick for 100 Norwegian Kroner, while a storage facility is a bit more costly.

Another benefit of digitalization is the ability to search and update documents. It is a lot quicker to search among files on a computer than a storage facility. Updates can be done by simply removing the old text and replace it with new information.

2.2.3 Digitalization within healthcare

Digitalization can also improve healthcare and help in increasing physical activity. Wearable technology can help people monitor their own health and give warnings if something is out of the ordinary. Virtual reality can help old people do basic exercises and give surgeons the possibility to practice before an important surgery. Artificial intelligence can guess the disease based on symptoms. Last but not least, digitalization can help people access information that otherwise would be difficult to obtain. The possibilities are endless.

Another great example of how digitalization can improve healthcare can be found at helsenorge.no [11]. This website has a module called "My health" where people can access their personal records, book a meeting with their general practitioner and ask questions about their health. Previously, all of these interactions would require you to call your general practitioner or the local hospital. Now you can access it through your own computer. Not only does this shorten the time it takes to find information, but health professionals spend more time helping people with critical problems rather than answering phone calls.

2.2.4 Digitalization within treatment selection

Given what we know, is it possible to use digitalization to implement an application for selecting treatment centers? The biggest step into digitalization is the gathering of treatment center data. Without a digital solution, we would have to collect data with a form. This would result in a manual process for both the sender and the receiver.

The new and improved system solves this problem by moving the form to a website. This website is always available, easy to access and the information is automatically updated whenever a treatment center submits a change. When submitted, the answers are automatically moved to a database and instantly used by the other modules. Norwegian Multiple Sclerosis Competence Centre is still able to access the answers and manage them if needed. The benefit of automating this pipeline is that updates can happen more frequently. Frequent updates give patients better, and more up-to-date information about each center.

2.3 Information access

As of now, the information gathered by the treatment centers who offers MS is not easily accessible. It is difficult to find, and not always up to date. And even if the patient finds it, it is a lot of information to process. With around 50 centers to choose from and a lot of treatments, you have to spend some time to find the best treatment center for you. The website improves this struggle by displaying the information as a survey. The patients are able to answer a few questions about their needs and preferences, and in return, they get a few recommendations about where they should attend treatment. The recommendations should then be discussed with the medical doctor, nurse or other health care professionals aiming to find the best solution for the patient. Not only does this shorten the time spent looking for information, but it also represents the information in an impartial manner.

2.4 A general framework for recommendation systems

If we manage to do something about the current system for selecting treatment centers for MS patients, what is stopping us from applying the same logic to other treatments

and diagnoses? By giving the admins the possibility to add, remove and edit questions, the application could (in theory) work on anything. By answering some questions, the user could get a recommendation on e.g. which restaurant they should visit based on food preferences, where you should live based on economy and family or maybe which type of dog you should buy. All you need is an expert to define some rules which are used when giving recommendations.

Chapter 3

Method

To carry out our research, we have selected a few methods that can help us develop and review our application. These methods include iterative design, kanban, systematic literature reviews, and empirical research. Afterwards, we describe our communication with different experts in healthcare and IT.

3.1 Iterative design

Iterative design is a methodology used to design user interfaces. The method involves making effective and user-friendly interfaces through several iterations. First, we start off by making a prototype with a design we would like to use. This can either be a functional program, or wireframe that shows the basics of the interface. Since we are not testing the system as a whole, we can use fake, hard-coded data to test the interface. When we have a working prototype, we do some user testing to get feedback on our interface. According to J. Nielsen [31] it is common to use around ten people during testing. He also says that is it important to test the interface on people who are actually going to use it, to get the best feedback possible. This includes both novice and expert users. When the testing is complete, we analyze the results and do it all over again until we are happy with our result.

To measure usability, Nielsen mentions five different quality attributes:

- **Easy to learn** - A user can quickly learn the basics of the system.
- **Efficient to use** - Once learned, the user can work in an effective manner.
- **Easy to remember** - You remember how the system works after not using it for a few months.
- **Few errors** - The user can complete their tasks without too many errors. They can also recover easily if an error occurs
- **Pleasant to use** - The user enjoy using your system

While these attributes are important to make a good interface, we do not have to focus equally on all five attributes. Which attribute we should focus on, depends on the project. In our decision support framework, we focused on a system that is easy to learn, has few errors and is pleasant to use. Since this is a system you use once (or rarely), it is less important to make it efficient to use and easy to remember. The first impression is the most important factor in our case. If the website is not appealing or easy to use, people will leave the website and find another alternative. The same goes with few errors. If they find it difficult to complete the survey or have to start over again because of errors, they won't bother doing it at all.

When it comes to the iterative part, we decided to split the process into three parts, one for each module. The reason why is that each module has a different target audience. The patient module is used by patients, the center module by people who work at the treatment centers and the admin module by Norwegian Multiple Sclerosis Competence Centre. In addition, we used employees at Haukeland University Hospital as our experts to help us improve the design. These experts where our main feedback source during the first iterations. When we had a working pipeline, we moved on to the novice users and used their feedback to improve the interface. More information about our user testing can be found in chapter 6.

3.2 Kanban

Kanban is a lean approach to software development. It is mostly used as a tool to manage and improve workflow when doing agile programming. While there is a lot of different ways to apply kanban, there are a few key elements that you should follow.

- Visualize workflow.
- Limit work in progress
- Focus on the flow

It is common to use a kanban board to visualize the workflow. The board contains columns that each represents a stage in the workflow. Tasks are added to the first column and then moved right for every completed stage. The board used in this project can be seen in figure 3.1

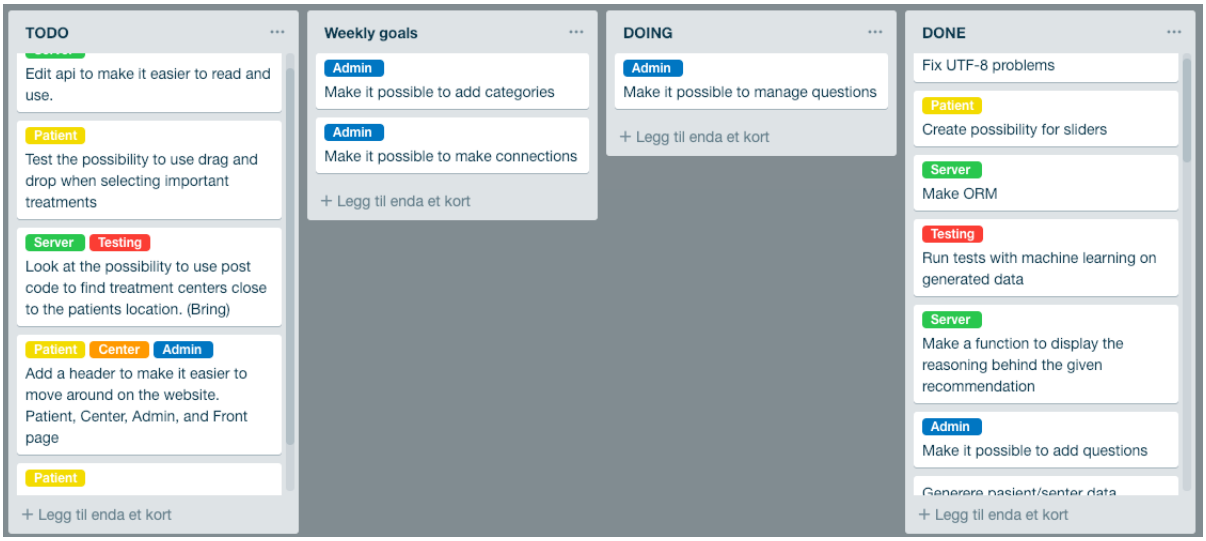


Figure 3.1: Kanban board

Kanban board used under development

Stop starting, start finishing is a common phrase used when working with kanban. By limiting the number of tasks in the *Doing* column, you can focus on your current task without thinking about anything else. A lot of ongoing tasks can also affect each other and you might end up having to redo a task because it does not fit in with the new additions. Finally, if you complete tasks before starting a new one, you can deploy the application after each completed task. This method made a huge difference when we had to deploy a working prototype before each meeting with the Norwegian Multiple Sclerosis Competence Centre.

Focus on the flow is more relevant when doing kanban in a group. What it means, is that you should have an even flow and prevent tasks from being blocked by other tasks. This is not a huge problem when doing kanban alone since you do not have to wait for anyone

else to complete their task before you can continue your own. A well-planned development process will prevent blocked tasks and help in generating a good workflow for your project.

3.3 Systematic literature review

To get a better overview of already existing research, we have decided to use Systematic literature review (SLR) to gather more information about our topic. In the paper: Procedures for Performing Systematic Reviews[25], Kitchenham describes SLR as:

> A systematic literature review is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest.

Knowing this, why should we do a SLR? Well, some common use cases are to identify gaps in the current research, compare your hypothesis with other papers, and summarise previous research [5]. In our case, we want to find systems similar to ours and compare them with our own application. While we might not find systems that are identical, we might get an indication of how we should approach when making our application.

As described by Kitchenham [25], SLR is split into three parts: identifying, evaluating and interpreting. In this section, we are going to focus on identifying related work and later in section 7.3, we are going to evaluate and interpret our findings.

3.3.1 Planning the review

The first phase of a SLR is the planning phase. In this phase, we start off by specifying the research questions that we want to answer during the review. In our case, we want to test our research questions stated in our introductory chapter. After making the questions, we want to make a review protocol that contains all the information needed in order to perform our review. This protocol should contain:

- Background

- Research question(s)
- Search strategy
- Criteria
- Quality assessment
- Data extraction strategy
- Synthesis of the extracted data

Review protocol

Research questions

Is it possible to make a digital solution for recommending treatment centers?
Is it possible to make a general framework that can be applied to a variety of different treatments and diseases?

Search sources

Scopus, Google Scholar

Search words

Recommender system OR Recommendation system
Rule-based
Doctor
Health
Treatment center
Treatment facility
Physician
Hospital

Search applied on

Title, abstract, keywords

Search period

2000-2019

Searched items
Conferences, papers, journals

Criteria - Include
All documents must be in English.

To be considered, the document must contain a description of a recommendation system that is related to recommending doctors, hospitals or treatment centers.

Criteria - Exclude
Papers about health tracking systems are not interesting when it comes to treatment center selection.

Since our recommender system is all about recommending where you should attend treatment, we do not include papers about recommending medication or type of treatment. This information is already known by the patient before using our application.

We want to prevent duplicates in our list of selected documents. This includes documents published by the same authors on the same topic.

3.3.2 Conducting the review

After the planning phase, we move on to conducting our review. In this phase, we use our protocol to find studies that might be relevant. Then we select relevant studies and finally extract and present the data.

As described in our review protocol, we used Google Scholar and Scopus to find documents. A list of our findings and search strategies are provided below. Since some of our strategies provided 60+ results, we have decided to only include documents that had a promising title or abstract. The documents not included contained information about recommendation systems that give you a diagnose based on symptoms and health tracking systems.

Table 3.1: Table of SLR findings

Title	Search	Selected	Reason
A Hybrid Recommender System for Patient-Doctor Matchmaking in Primary Care [19]	2	Yes	
Power to the patients: The HealthNet Social Network [12]	2	Yes	
A Patient-Centric Healthcare Model Based on Health Recommender Systems [4]	3	Yes	
A Novel Model for Hospital Recommender System Using Hybrid Filtering and Big Data Techniques [8]	1	Yes	
A Decision Support System for Prescription of Non-Medication-Based Rehabilitation [15]	4	No	Set diagnosis and recommends treatment
A Collaborative Filtering Recommender System in Primary Care: Towards a Trusting Patient-Doctor Relationship[18]	2	No	Duplicate
Building a Classification Model for Physician Recommender Service Based on Needs for Physician Information[27]	2	No	Does not recommend centers
Which Doctor to Trust: A Recommender System for Identifying the Right Doctors[17]	2	No	Only for finding Key opinion leaders
Recommending doctors and health facilities in the HealthNet Social Network[29]	2	No	Duplicate
How to find your appropriate doctor: An integrated recommendation framework in big data context[23]	2	No	
A Hospital Recommendation System Based on Patient Satisfaction Survey[24]	1	No	Mostly focused on analyzing feedback

During our search, we used a combination of our search words to reduce the number of irrelevant documents. All our documents were found with these search strings:

1. (Recommender system OR Recommendation system) AND Hospital
2. (Recommender system OR Recommendation system) AND Doctor
3. (Recommender system OR Recommendation system) AND Health AND Rule-based systems

4. (Recommender system OR Recommendation system) AND Treatment center

We did try some other combinations as well, but they either resulted in duplicates or nothing at all.

- (Recommender system OR Recommendation system) AND Treatment facility
- (Recommender system OR Recommendation system) AND Physician

All the relevant findings can be found in table 3.1

Further discussion about our finding can be found in section 7.3

3.4 Empirical research

As a part of our research methodology, we have decided to use Empirical research. In Empirical research, we want to gain knowledge with the help of observation and experience. This is a commonly used method in scientific research.

3.4.1 Quantitative and qualitative research

There are two well-known paradigms to carry out empirical research, *Quantitative research* and *Qualitative research*. Quantitative research is research concerning numbers and statistics. The data collected can later be displayed as a graph or a table to give a better overview. Common ways to collect such data is to run experiments as well as questionnaires with closed-ended questions. To produce quantitative data with questionnaires, you need to display questions as a scale (high, medium, low), categories (yes, no) or a numeric value (0-10). The numeric values and categories are given, can be used to find similarities or differences in the data. This is useful when you want to test already constructed hypotheses. [28]

Qualitative research is research without numbers. Rather than observing **if** something occurs, we are asking **why** or **how** something occurs. Common methods for collecting such data is reading old papers, records, images, etc., as well as using open-ended questionnaires. Qualitative data are useful if you want to know the reasoning behind a phenomenon. You can find out if there are any difficulties completing a task, with the help of quantitative research, but you need qualitative data to find out why it is difficult. [28]

3.4.2 Empirical research methods

In the book Empirical Research Methods in Software Engineering [6], Wohlin, Höst and Henningsson describe four different methods when doing empirical research: experiment, case study, survey and post-mortem analysis. To run our research, we have decided to use experiments and surveys in our research.

An experiment is a method where you test a small part of the project. While a case study positions the researcher as an observer, experiments are controlled by the researcher. The goal is to get a result on a specific problem by only controlling a few variables. Wohlin et al. mention two different types of variables: independent and dependent. The independent variables are what you want to test. Is solution A better than solution B? The dependent variables are variables that might be affected by solution A and B. The result is the dependent variables and how they are affected by the independent variables. This method can be applied quantitative research [6].

A survey is a method where you ask questions in the form of a questionnaire or an interview. This method is usually used after the project is completed. The goal of a survey is to gather a lot of data that can later be evaluated. To get the best possible data, the population should contain people who are involved with the application. In our case, our population consists of patients and treatment centers. It is also possible to use surveys early in the development process to get an overview of the population. This method can be applied to both qualitative and quantitative research.

3.4.3 Iterative design with empirical research

Iterative design and empirical research have a lot in common. You can run empirical research to gather more data, analyze the data and then use the result to improve your application with iterative design. You could, of course, apply empirical research without any iterations as we did with our Binary vs Number scale experiment (section 3.5.2). In our case, we use iterative design to implement, test and evaluate through many iterations, and empirical research to gain domain knowledge, evaluate prototypes and compare different methods. Iterative design is our design methodology, while empirical research is our research method.

3.5 Experiments and surveys

To test and evaluate our application, we made a few experiments and surveys. The experiments are used to compare two methods to find the best solution for our application, while the surveys are used to gather feedback from the target users.

3.5.1 Experiment: Jaccard index vs Numerical rating scale scale

One method of finding similarities between two sets is the mathematical formula called Jaccard Index. This formula compares the two sets by taking the number of values they have in common, and divide it by the total amount of unique elements in both sets. To prevent decimals, we multiplied the answer with 100. To test this method, we ran an experiment and compared Jaccard with a numeric scale. To get a more accurate result, we gave the same questions a score of 1 or higher in both tests.

3.5.2 Experiment: Binary vs Numerical rating scale

At the beginning of the project, the patient could only answer their questions as yes or no. While this binary method got the job done, we wanted to test another approach where the patients were given the possibility to give their answers on a scale from 0 to 10.

To run this test, we did an experiment where we first gave each question a score of 0 or 1, and then ran a few tests where we gave questions a score between 0 and 10. To make the test more accurate, if a question got a score of 0 in the binary test, the same question was given the score of 0 in the number scale test. The questions that got a score of 1 on the binary test, got a score of 1 or higher on the number scale. To remove some variables, we did not use the feedback scores when running this experiment. Since the test data consists of numeric data, we can classify this as quantitative research.

3.5.3 Survey: Patient questionnaire

To test if our recommender system worked as intended, we first gave MS patients access to our website and asked them to use the recommender system. Then, we gave the patients a questionnaire for them to answer. The questionnaire contained both open- and closed-ended questions to gather both qualitative and quantitative data. The quantitative data gave us an overview of the overall completion rate and satisfaction, while the qualitative questions gave us the reasoning of why an occurring problem existed. In addition, we had an observer from Haukeland University Hospital who helped the patients through the process, as well as taking notes. The goal with the research was to put our first research question to the test, see if patients found the application useful and find out if they would use it in a post-prototype scenario. The results of the survey were also used to improve the website.

To make the right product for the right customers, we did some anonymous user testing with real MS patients. To deal with privacy, we held the tests at Haukeland University Hospital on a local computer. Since we were not allowed to attend the user testing, we completed the questionnaire with the help of the Norwegian Multiple Sclerosis Competence Centre. Nine participants completed the test.

The test was split into two parts. First, the patients answered questions from the patient module and looked at the recommendations given. Then, they were given a short, anonymous survey with questions about their user experience and technological background. The survey can be found in appendix A.

3.5.4 Survey: Exploratory treatment center questionnaire

To gain more domain knowledge before going all-in on the development phase, we completed a short test where we gave 10 treatment centers the possibility to test our center module and then give feedback on it through a survey. This is known as an exploratory survey, where you gather data to improve further research. This research is qualitative research since our questionnaire only contained open-ended questions. Only 4 treatment centers submitted an answer due to connection problems.

3.5.5 Survey: Treatment center questionnaire

To test our first research question, 33 treatment centers in Norway were given the possibility to test our center component. After they completed the questions on the website, they were redirected to a Google Forms (See appendix D) questionnaire where they were told to answer some questions about their experience with the website. The questionnaire contained both open- and closed-ended questions. The goal with this survey, was much like the patient survey, to check if our application seemed to catch the treatment centers interest, as well as improving the website. Out of the 33 treatment centers, 15 answered.

3.5.6 Survey: Admin page questionnaire

To test our admin module, we first completed a survey where we interviewed fellow students. During the interview, they were given a small set of tasks shown in appendix C . The experience was later discussed and possible problems were uncovered. The goal of this test was to fix the big and obvious problems before running a bigger test on target users.

In the second test, we asked the Norwegian Multiple Sclerosis Competence Centre to play around with the admin module and report on any problems found during the testing. We consider this a more thorough test since we test it on users with a lesser technical background. Much like the other surveys, we wanted to find any problems with the website, as well as trying to get to a conclusion if our application is useful or not.

3.6 Communication with experts

In the course of this project, we have tried to communicate with people who work with MS and "Fritt behandlingsvalg". The Norwegian Directorate of Health has helped us understand the reasoning behind the idea of giving patients the possibility to choose a treatment center. They have also provided us with some statistics about wait times and how "Fritt behandlingsvalg" has affected the wait times.

Haukeland University Hospital and Norwegian Multiple Sclerosis Competence Centre have been our main companions in this project. They helped us:

- formulate questions
- follow the strict rules within healthcare
- use the correct medical terms
- communicate with treatment centers and other healthcare related people
- test the application on patients and treatment centers
- improve the application through many iterations

Issues were mostly discussed through email, but we also had a meeting every once in a while to discuss bigger changes and make a plan for the upcoming month. The application was deployed frequently to give Haukeland the possibility to test the application and provide feedback.

To get some feedback on the idea itself, we spoke with people from the Norwegian Directorate for e-health who is responsible for creating digital solutions within healthcare and Sunnaas which is the largest special hospital within rehabilitation in Norway. Both found the project interesting and gave some suggestions on how we could improve our application. Sunnaas gave us the idea to give patients additional information about questions that were difficult to understand. They also suggested the idea to rate questions based on importance, rather than giving patients the possibility to select as many answers as they want. The email with suggestions can be found in appendix G.

Chapter 4

Design

In this chapter, we will explain our architecture and recommendation system, as well as our road to a complete application.

4.1 Modules

To come up with our design, we made a domain model (figure 4.2) showing how our domain is connected together. As shown in the model, we have three clusters of boxes: things concerning patients on the left, treatment centers on the bottom, and admins on top and to the right. Everything is connected in the center, where we find our questions. Because of this distribution, we have decided to split our application into three different modules. By doing this, we can give each module its own client. This is beneficial, since each module/client has a different target user. An example is shown in figure 4.1

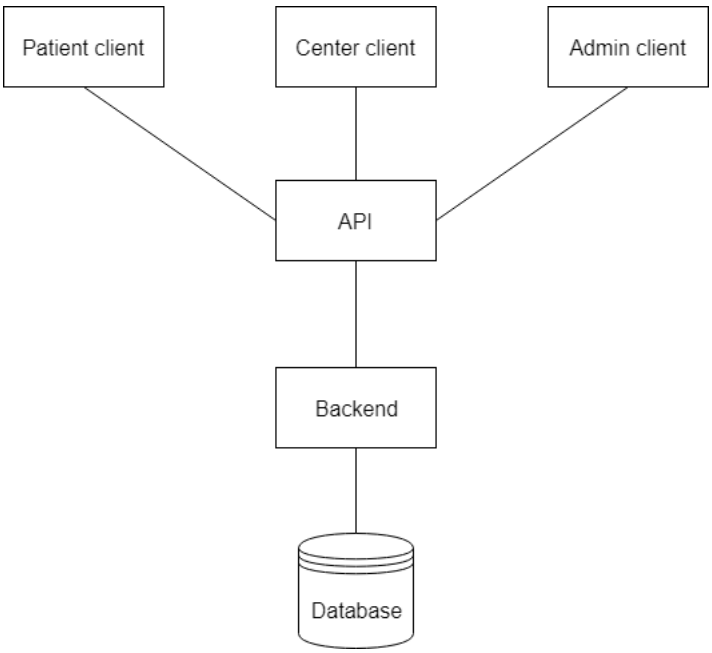


Figure 4.1: Server-client structure

A structure example using three different clients

4.1.1 Center module

The first module is a digital version of the survey initially developed, but not used, by the Norwegian Multiple Sclerosis Competence Centre to gather information about the different treatment centers. This survey involves questions about what kind of treatments the different centers have, information about the facility, and some basic contact information. The information is sent to the database, where it is used to whatever purpose needed.

4.1.2 Patient module

The second module is based on the patient. The patient can answer some questions about their needs and preferences about the facility. In return, they get a recommendation on which treatment centers seem to fulfill their needs. The recommendation is given by a Rule-based system (RBS) that calculates a score based on its rules. To make the RBS learn, we made a feedback system where patients can give feedback on their treatment.

4.1.3 Admin module

The last module consists of a set of pages where the administrators can customize the application. The idea behind this is to give admins the possibility to change the content of the application without having to write code or hire developers to make the change for them. The module gives them the possibility to:

- add new questions to the database
- select which questions that should be asked to the patients and treatment centers
- make connections between similar questions
- look at the feedback given by the patients
- look at the answers given by the treatment centers.

The admin module is key when building a generic recommendation system described in section 2.4.

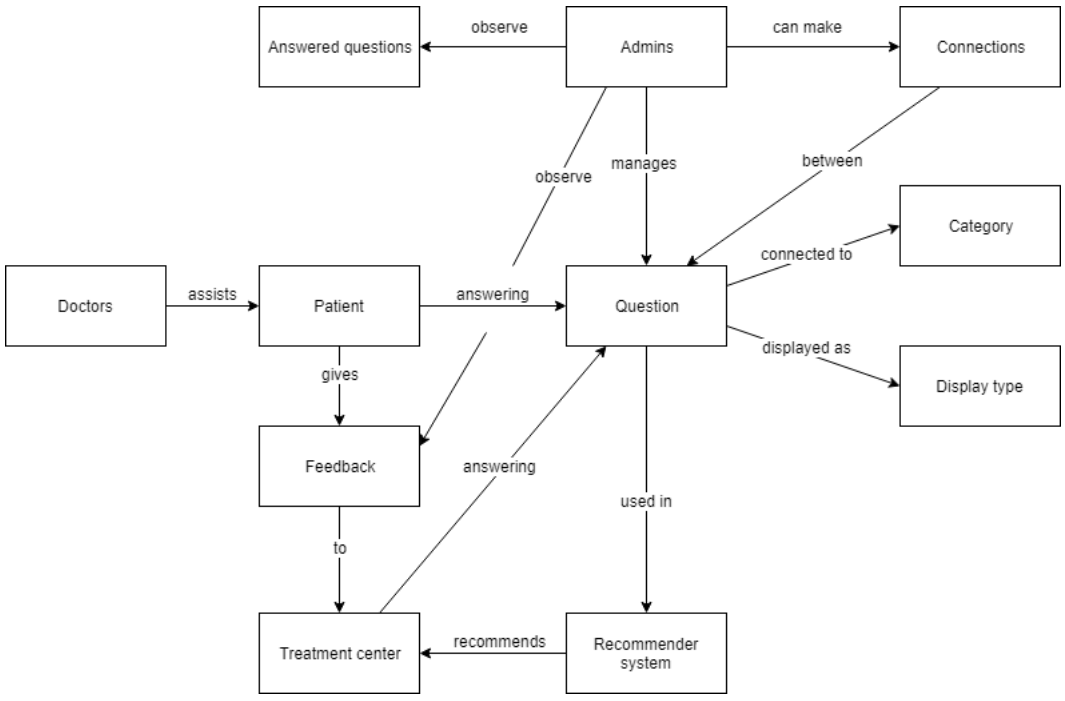


Figure 4.2: Domain model

An overview of our domain

4.2 Decision support framework architecture

To make the system easy to build and maintain, we have decided to use a Two-tier client-server architecture. In addition, we are going to talk about Software-as-a-Service (SaaS) and how we combined Two-tier architecture with SaaS to make our application.

4.2.1 Two-tier client-server architecture

The two-tier client-server architecture is an architecture where you split the application into two tiers, client and server. A tier is a process boundary where each tier can run on a different machine. A tier consists of one or more layers. The most common layers are presentation, data-handling, application processing, and database layer [38]. An image of our application structure can be found at 4.3

Client tier

The first tier is known as the Client tier. This is where you find all the clients that are used to communicate with the Server tier. This is the most common way for users to access web-based applications. It is usually made with HTML, JavaScript, and CSS or any framework supporting these languages. The Client tier communicates with the Server tier with the help of an Application Programming Interface (API).

In our case, we use a React web-client as our frontend. This client takes part in the presentation layer and contains modules for gathering data from the treatment centers, recommend centers to patients, and managing the application as an admin. Each module works separately and could be split into three different clients if preferred.

The client described above is known as a thin client. A thin client has the presentation layer implemented on the client tier and the other three layers on the server tier. The benefit of using a thin client is that it can run on a normal web browser. There is also no need to reinstall the client whenever there is a new update. On the other hand, a thick client (which contains both presentation and application processing layer) can use the computational power of the client machine, whereas the thin client can only use the server for processing. Since our application has to work in a browser, we have decided to use a thin client.

Server tier

The last three layers are found in the server tier. The data-handling layer consists of an API that handles all communication to and from the client tier. Our API is made in Python with a library called flask. This API has all the methods necessary to move data from the presentation layer to the application processing layer, and back again.

The application processing layer is a separate file from the data-handling layer and contains all the application's logic. Most of the logic lies within the recommendation part of the application but does also involve methods for converting data to JavaScript Object Notation (JSON) and generating random strings.

Last but not least, the database layer holds methods for communicating with an SQLite database. These methods are made as queries with the help of SQLAlchemy and its Object-relational mapping (ORM). This layer also contains a few configuration files that hold information about how questions should be displayed.

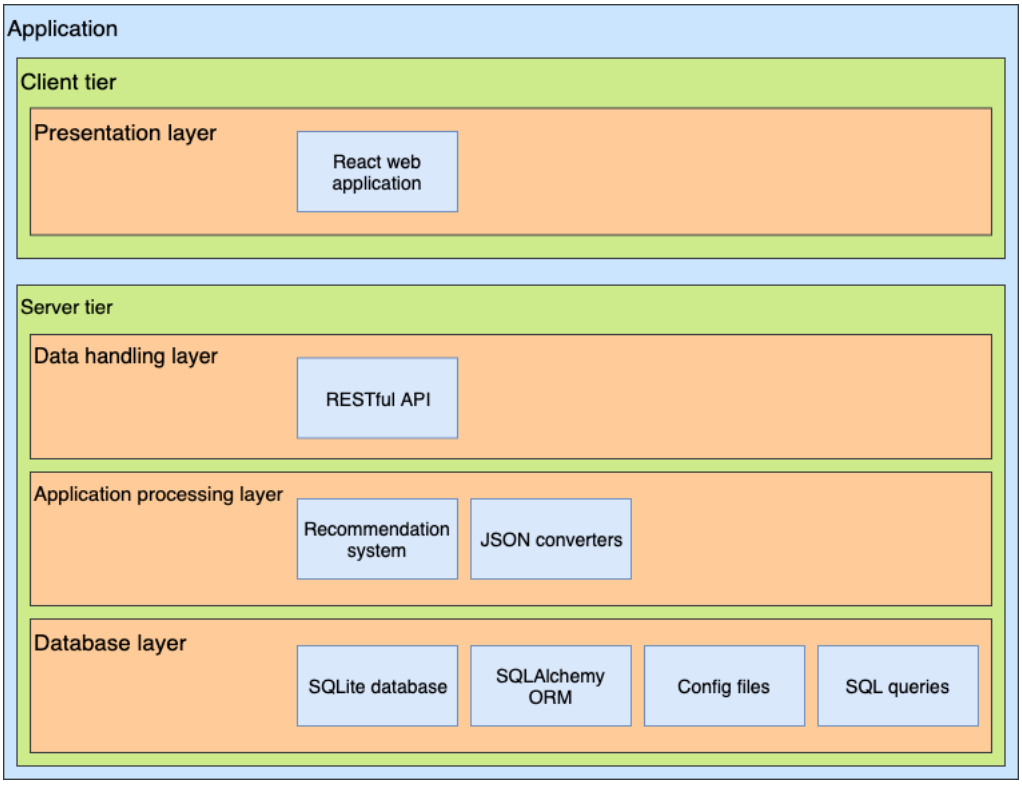


Figure 4.3: Application architecture

A model showing the current architecture

4.3 Cloud Computing

Cloud computing is a new method for delivering services and applications over the internet. These services are offered by many large companies like Google(Google Could Platform), Amazon(Amazon Web Services) and Microsoft(Azure). The motivation for cloud computing is to give companies and the general public the possibility to host an application in the cloud. There are four main service models used in cloud computing:

Infrastructure-as-a-Service (IaaS) - A service where you can rent and manage a virtual machine from the provider e.g. Google Compute Engine

Platform-as-a-Service (PaaS) - A service where users can develop, deploy and manage their application e.g. Heroku or Google App Engine. Everything else is managed by the provider.

Function-as-a-Service (FaaS) - A service where users can deploy single functions instead of a full application e.g. Google Cloud Functions.

Software-as-a-Service (SaaS) - A service where you give the user access to a complete application or a client. Everything else is managed by the provider.

Our application is deployed as a SaaS. The benefits of using SaaS is that we can give the users access to the service without them knowing anything about the server. It also gives us the possibility to use the thin client we mentioned earlier. A thin client opens the possibility to access the application from any device, anywhere in the world. [1]

4.4 Recommender system

A recommender system is an information filtering system that tries to remove unnecessary information by filtering data based on user preferences. They have many use cases. Netflix, Youtube, and Spotify use recommender systems to suggest movies/songs based on what you watch or listen to. Facebook and Twitter use it to show relevant ads and other types of content. Online stores use it to suggest items you might want to buy, based on previous purchases or browser history. Knowing this, can this method be applied to treatment selection?

4.4.1 Rule-based Systems

RBS is a method mentioned in the topic of artificial intelligence. The goal of RBS's, is to transform human knowledge into digital media. This is accomplished by making a set of if-then statements known as rules. These rules are based on human expertise and should simulate a real-life scenario by picking the best answer based on the input given. The more statements you give the system, the more accurate it becomes. [20]

An example of a RBS could be a system that decides if you should play football or not. One rule could check if it is a sunny day and return "true" if it is. Another may check if the football field is full and return "false" if it is. If you have enough of these statements, you should get a good indication whereas you are going to play football or not.

4.4.2 Machine learning

Machine learning (ML) is a more modern approach to artificial intelligence. Instead of giving your system a set of rules, a ML algorithm uses data from known scenarios to create a black box. This black box can be used in the same way as the RBS, to predict an outcome based on the input data. The big difference is that ML has the possibility to learn by itself, while in RBS, you have to add new rules manually. The problem is, that ML needs a lot of data to be accurate and it can be difficult to find enough training data. Since most ML algorithms works like a black box, it is also difficult to know which choices were made to produce the given result.

While there are a few different approaches when making a recommendation system with ML, collaborative filtering might be the most commonly used technique. It is used by companies to recommend movies based on user preferences. In short, this algorithm uses movie ratings given by the user and generates two matrices with numbers. These numbers are then used to fill the gaps in the user rating matrix. By doing this, the algorithm can guess what rating you would give unseen movies.

Another common method is called content-based filtering. This method tries to recommend items based on your previously collected data and is commonly used by online stores when recommending items that are similar to what the customer has bought before. In other

words, collaborative filtering suggests that similar people like the same movies/songs, etc., while content-based filtering suggests that people like items that are similar to what they already bought.

Finally, there is a method calledTerm frequency–inverse document frequency (tf–idf). A tf–idf score is calculated by the frequency of the word, down-weighted by the number of entities that contain this word. In other words, if a lot of treatment centers offer treatment A, it becomes less weighted than treatment B, which is only offered by a few centers. This would be a good way to find the differences between treatment centers.

4.4.3 Rule-based systems vs. Machine learning

Our first approach was to test the possibility of using ML. After some research, we found that neither collaborative nor content-based filtering would work. Collaborative based filtering would be difficult without any data, as it recommends treatment centers based on previous recommendations of similar patients. This is known as the cold start problem and is a common problem when it comes to recommender systems. Our other approach, content-based filtering, would recommend treatment centers that are similar to other centers that you found helpful. This is not very useful, as this recommender system has to work without any previously collected data from the patient.

tf–idf on the other hand, worked a bit better. It managed to find some differences, but because the treatment centers have a lot in common (and some almost identical), the algorithm had problems separating them from each other. Almost every tf–idf score had a difference less than 5 and there where no way to tell why a center got the score they got.

We then moved on to some testing with mathematical formulas and found the Jaccard index. This is a formula defined as the intersection divided by the union of the two sets: $\frac{|A \cap B|}{|A \cup B|}$ This formula would give the score of 1 if the sets are identical and the score of 0 if they have nothing in common. To make the results a bit more accurate, we removed all the data from the center set, that was not in the patient set. By removing these elements, we get an answer that only takes to consider what the patients ask for. We do not care if a center offers treatment A if the patient only asks for treatment B and C.

While Jaccard seemed pretty basic, it actually worked pretty well. The system recommended centers based on the patient's answers and we got an explanation on why the centers where recommended. Progress! But there was still a small problem. With this formula, patients could not grade the treatments based on their importance. The system was binary and a specific treatment would either be very important or not important at all. Back to the drawing board.

In the end, we ended up with a RBS, that gave patients the possibility to rate each question from 0 to 10, where 0 is not important and 10 is very important. **If** a center preforms treatment X and the patient gave X a score of 1 or better, **then** we have a match. For each treatment center, we use the formula to calculate our recommendation score:

$$\sum_{match}^{n} \frac{center\ score\ on\ current\ question}{100} * patient\ score\ on\ current\ question$$

The list of treatment centers is then sorted based on their recommendation score. The three best centers are then given to the user, as well as the reasoning behind the recommendation.

Some questions cannot be mapped as a one-to-one relation. As a result, we made it possible to connect questions with a one-to-many relation. For example, question X from the patient form might be connected to question Y and Z on the center side.

Why does this RBS work? Well, we solved the problem concerning binary rating by giving the patients the possibility to rate each question using a Numeric Rate Scale (NRS)[21], rating questions from 0 to 10. We removed the need for training data and we can give an explanation of why a specific treatment center was recommended. With this combination, we are able to offer a pretty good recommendation system that can recommend treatment centers and justify why each center fits the patient's needs or not.

To test if our numeric scale improved our RBS, we performed the experiments described in section 3.5.2. The results of this experiment endorse our solution. As shown in table 4.1, the numeric scale made a pretty big difference. The table on the left contains data gathered with binary patient input. As you can see, the scores are pretty similar. Six out of ten treatment centers share a score of three. This becomes a problem when we have to

recommend the top three treatment centers. As of now, the recommender would select the second and third place according to alphabetical order. We could randomize the treatment centers, but this would not be a fair solution.

On the right side of table 4.1, you find the results from the numeric scale tests. In this table, the scores are a bit more spread out. Now, we can recommend three treatment centers without having to pick them randomly as a tiebreaker. Since similar treatment centers can exist, we can not guarantee that a tie will not occur with a numeric scale, but it is at least a less common occurrence. Another benefit is that patients can change the recommendations by making small adjustments to their answers. These adjustments do not exist with binary input.

Given these arguments, it is, without a doubt, a better solution to use a numeric scale as our patient input. This is beneficial for both patients and treatment centers. The patients can affect the results by making minor changes and rate the importance of symptoms more accurately. In addition, there is a smaller chance that a treatment center loses the lottery by being unlucky with the tiebreaker.

The same problem occurred when we ran our Jaccard experiment described in section 3.5.1. Treatment centers with a lot in common tend to get the same score. The results can be found in table 4.2

Table 4.1: Result from Binary vs Scale experiment

Treatment center	score
Treatment center 8	4
Treatment center 1	3
Treatment center 2	3
Treatment center 3	3
Treatment center 4	3
Treatment center 5	3
Treatment center 7	3
Treatment center 6	2
Treatment center 9	2
Treatment center 10	2

Treatment center	score
Treatment center 8	28
Treatment center 3	27
Treatment center 7	27
Treatment center 5	25
Treatment center 2	22
Treatment center 4	22
Treatment center 1	18
Treatment center 9	14
Treatment center 10	13
Treatment center 6	12

The table on the left shows the score with a binary input data, while the left shows the scores with a scale from 0-10

Table 4.2: Result from Jaccard vs Scale experiment

Treatment center	Score
Treatment center 4	71
Treatment center 5	71
Treatment center 6	71
Treatment center 8	71
Treatment center 1	57
Treatment center 3	57
Treatment center 7	57
Treatment center 2	43

Treatment center	Score
Treatment center 5	34
Treatment center 4	29
Treatment center 6	29
Treatment center 8	26
Treatment center 1	25
Treatment center 3	24
Treatment center 7	24
Treatment center 2	22

The table on the left shows the score with Jaccard Index, while the left shows the scores with a scale from 0-10

4.5 Representational State Transfer

Representational State Transfer (REST) is a software architectural style that use the already existing Hypertext Transfer Protocol (HTTP). Everything is build around these four HTTP methods:

- GET - Retrieve data from API.
- POST - Used to send data to the API.
- PUT - Update or add an item to the given URI.
- DELETE - Remove an item from the given URI.

A web service that follows this style is called a RESTful web service. This kind of web service is often used to make connections between web clients and servers. It is also common to use RESTful as a way to distribute information by making an open API. Everyone can send requests to these API's and you can find information about everything between weather forecast's [32] and data from NASA [30].

To make APIs as user-friendly as possible, there are a lot of rules describing how you should design your API. This includes everything from URI format, response status codes and HTTP request methods. A full overview of these rules and other design methods can be found in O'Reilly's REST API - Design rulebook [26].

4.5.1 Benefits with RESTful

The biggest benefit of RESTful is the possibility to make systems with loose coupling. Loosely coupled systems have little to no knowledge about the other components and you can easily develop one component without having to think about the other ones. This also gives you the possibility to write the frontend in one language and the backend in another language. Another benefit is that there are a bunch of libraries and frameworks to make RESTful services. Almost every known language has some way to create or communicate with a RESTful API. Python has flask[37], Java has JAX-RS[33] and Node.js has express[3] and axios, just to mention a few.

4.5.2 REST vs SOAP

But, why did we select REST over Simple Object Access Protocol (SOAP)? While SOAP works great on huge industrial systems, REST is a much simpler approach. You only need HTTP for it to work, you can send files with different formats and you can easily update the API without having to change the client. SOAP is difficult to learn, can only send XML and has a WSDL file that needs to be updated whenever there is a new endpoint.

4.5.3 JavaScript Object Notation

To transfer data between our server and client, we use some specific file formats. We have decided to use JSON. JSON is a text-based file format that can be used to transfer object and other data structures over the internet. Since JSON and JavaScript Objects have a lot in common, it is common to use JSON in combination with JavaScript. In addition, the file structure is also very similar to Python's dictionaries which is convenient when we want to communicate with our server.

JSON has a basic syntax where the object starts and ends with a curly brace. Each object consists of key/value pairs that represent our data. This structure can be compared with the file structure on your computer. Each key is a folder and the value is the files within the folder. Each key can be easily accessed by traversing through the tree: `key1.key2`. An example of a JSON file can be found in listing 4.5.3

Listing 4.1: Json file example

```
{
  "key1": [
    {
      "key2": "value"
    },
    {
      "key3": "value"
    }
  ]
}
```

Our other alternative would be Extensible Markup Language (XML). While XML works great when dealing with metadata, JSON is more compact and can be transferred at a higher speed. JSON can also do the same amount of work with fewer words.[22]

4.6 Picking the right questions

At the beginning of the project, the plan was to mirror the questions from the word document found in appendix D . We found out rather quickly that this approach would not work. Some of the questions where open-ended and the treatment centers were required to give a few short text answers. While this works great when the answers are read by a human, it is difficult for a computer to understand the context of the answers. Because of this, we decided to remove or replace all open-ended questions with a binary yes-no question.

After our first user test, we found another problem concerning wait times and patient capacity of the treatment centers. The questions about wait times were mostly left empty and followed by a comment explaining why it was difficult to answer these questions accurately. The same goes for patient capacity. Because of this, we decided to remove these questions as well. In addition, we added a link to our patient module. This link redirects to helsenorge.no's list of wait times and can be used by the patients when selecting a treatment center.

We then moved on to the questions on the patient module. To start off, the Norwegian Multiple Sclerosis Competence Centre gave us a suggestion on which questions should be given to the patients. These questions where based on the questions asked in the center module. The first problem we found, was a large amount of redundancy. We basically asked the same questions three times by asking the patients to first pick the two most important symptoms, then the next three, and finally less important symptoms if needed. These questions are useful on a nondigital media but can be simplified on a computer.

A possible solution to this problem was to change the questions from binary yes-no questions to a numeric scale. After the numeric scale change, came the discussion about professions. While we asked treatment centers which professions they had, we did not use the data when recommending treatment centers to the patients. To solve this problem, we came up with the idea of connecting questions together. A center that had a psychologist and offered help with mental health, would be given a higher score than a center that offered mental health without a psychologist. While we did not use this technique with professions, in the end, we still implemented the feature so that it can be used if the problem should occur again. The questions about professions can still be found in the center module to collect data for later use.

4.7 A brief history of the project

Rome wasn't built in a day and the same goes for software. Making a good application takes time and there is a lot of decisions to be made. In this section, we are going to look at the project's history and how we ended up with the application we have today.

The first step was to make a pilot for gathering information about treatment centers. The only resource we had to start with, was the document proposed to collect data from centers. The document can be found in appendix E. We used this document to make our earliest prototype. This was a static website written in Angular 4, connected to a basic Node.js API and a NoSQL database. Everything was hosted on IBM Cloud as a SaaS. The website was then sent to 10 treatment centers for testing and data collecting.

With the data analyzed and the bugs fixed, we moved on to the patient module. We used the questions from the center module as a reference to make the questions for the patient

module. The first proposal can be found in appendix B. With the questions made, we tried to make the website more dynamic by reading the questions from a JSON file rather than adding them directly in the HTML. The JSON files were added to the backend and passed through the API to the frontend. Because of all the trouble discussed in chapter 7.1, we decided to change our stack from Angular 4 and Node.js to React and Python. The server we had at IBM was also shut down, hence we moved the frontend to Heroku, and the API and backend to a local server at the Western Norway University of Applied Sciences (HVL).

After many iterations of reworking questions and making the frontend more dynamic, we ended up with a pretty good pipeline. It was time to take a look at the recommender system. The process is written in detail in section 4.4.3, but in short, we ended up with a RBS. Since RBSs uses predefined rules, we made a feedback function to give the RBS the possibility to learn.

With all the new data generated from patients and centers, it became difficult to keep the data organized. We decided to change the database from a Not only SQL (NoSql) database to SQL database in order to make relations between the data. The technique of ORM was used to improve communication with the database, and we made a file full of useful SQL queries. All the questions where moved to the database and we made configuration files that told the client how to display each question. Meanwhile, the client got an overhaul where users were given the possibility to rate questions from 0 to 10 (NRS) with the help of a slider. The RBS was updated to work with the new sliders.

We now had a working website that could collect data, recommend treatment centers, and give feedback, but there where no way to manage the website. It was time to start working on the admin module. The backend got some functions for editing configuration files. Meanwhile, the client was given the possibility to make new questions, edit configuration files, make connections between questions and look at patient feedback.

Chapter 5

Implementation

In this chapter, we introduce all the frameworks and languages we used to make our application, as well as give a thorough description of our implementation.

5.1 Presentation layer

Our presentation layer holds our user interface. The interface is made with the help of a React and contains all the methods necessary to communicate with our server. This section describes the frameworks and libraries we used, followed by our implementation.

5.1.1 React

React[39] is a JavaScript library used to make frontend applications. With the help of Cascading Style Sheets (CSS) and JavaScript XML (JSX) we can use React to make advanced user interfaces. The library is component-based, which means that we can make many separate components and connect them together to make a complete application. Each component can maintain its own state and will rerender each time the state is updated. A state update can be as simple as moving a slider value from four to five or removing the loading icon when the page is done loading.

JSX is an extension to JavaScript, used as a replacement for HyperText Markup Language (HTML). While JSX has a lot in common with HTML, it has some extra features that makes it a lot easier to use. The biggest change, is the possibility to use JavaScript inside the HTML tags. As an example, `<p>{1+8}</p>` would render as `<p>9</p>`. It can also be used with methods (`<div>{this.unpackList()}</div>`) and variables (`<p>{this.state.introduction}</p>`).[13]

5.1.2 Libraries

To simplify our development process a bit, we used some libraries containing different components and methods. These libraries are easily accessible through Node's packet manager, npm.

React-router-dom

React-router-dom gives us the possibility to move between the different pages. Each page is given a route that defines where you need to go on the site to access this page. `<Route path="/patient" component={Patient}/>`. will redirect you to the patient page, if you end your URL with "/patient".

React-bootstrap

React-bootstrap is a library that contains a lot of components. Some of the components are just an improved version of an already existing component, while others are completely new. The benefit with bootstrap is that the components have a set of predefined styles you can use when developing. This saves you a lot of time when it comes to styling and making components.

React-final-form

React-final-form (RFF) is a library that improves the already existing HTML forms. While the input components are pretty similar, RFF has improved the output when you submit the form. HTML gives you a form object, while RFF gives you a JavaScript object. JavaScript objects can be directly translated into JSON, which gives us the opportunity to send the data through the API without converting it first.

React-rangeslider

React-rangeslider is a library that contains a slider component. Each slider has a state which tells us the value of the slider. Since a slider component is difficult to make from scratch, we decided to use this library to shorten the development process.

React-table

React-table contains a table object that works a lot better than the original HTML table. Instead of iterating through the data and place every element manually, react-table takes a JSON file as input, and does all the work for you. All you need to do is to define the column headers and tell which JSON key it should put beneath each header. React-table does also come with a pre-styled table with the possibility to sort the table based on columns.

Axios

Axios is a library made to simplify requests. When using axios, you can make a API call with only one line of code, `axios.post("example.com/api/url",jsonFile)`. Axios does also support promises.

Fuse

Fuse is a library that can be used to filter a JSON file. To make it work, you need to provide it with two JSON files. One that contains the data you want to filter and one that contains the configuration of the search. Fuse will then return a new JSON file that only contains the key-value pairs which passed the filter.

5.1.3 Center component

The Center component is where we gather our data from the treatment centers. This component fetches data from the API and displays them as a web form. An introduction is found on the top of the page, followed by a set of questions. To make it easier to use, we split each category into a separate page. A forward and backward button is given to move between the different categories. When the user reaches the final page, a submit button becomes visible. The forward or backward button is disabled when you are on the first or last page to prevent confusion when filling the form. When the user submits the form, they get a message that says if the task was successful or not. A screenshot of the described component can be found in figure 5.1

The questions are fetched from the API with the help of Axios and then saved as a state. Loading is then set to false, and the questions are displayed on the page with the help of the `showIntro, showFullPage` and `showPage` functions. The `showFullPage` function will first use `showIntro` to display the introduction part of the JSON file. Then, it iterates through each category and sends the data to the `showPage` function. This function will iterate through all the questions in the category and display them with the help of RFF. In the end, we add the buttons to navigate the page. A submit function given to submit the answers with the help of Axios. Any response given by the API is displayed with our Response component mentioned later.

To make it possible to switch between categories, we made a help function called `changeDisplayedPage`. This function will change the `displayValue` each time the user hits the forward or backward buttons.

Figure 5.1: Screenshot of the center component

An image showing the user interface of the center component

5.1.4 Patient component

The Patient component works much like the Center component, but there are a few differences. In addition to the RFF library, we use react-rangeslider to give the patients the possibility to rate their answers from 0 to 10. Since react-rangeslider is not a part of the RFF library, we had to make a minor modification to the submit function. This modification would merge the slider data into the RFF JSON file. To keep the slider states updated, we added a function that updates the states whenever a slider is used. We also made a method for initializing the sliders after the first API call.

The second change does also take place in our submit function. Whenever the patient submits their answers, instead of giving a success/fail response, the data is moved to the RBS. The recommendation is then displayed on the page with the help of our Place and Recommendation components. If the patients are not happy with their recommendation, they have the possibility to return to the Patient component and change their answers. When the recommendation is given, the patients also get a unique id, which is later used in the Feedback component. A screenshot of our patient component can be found in figure 5.2

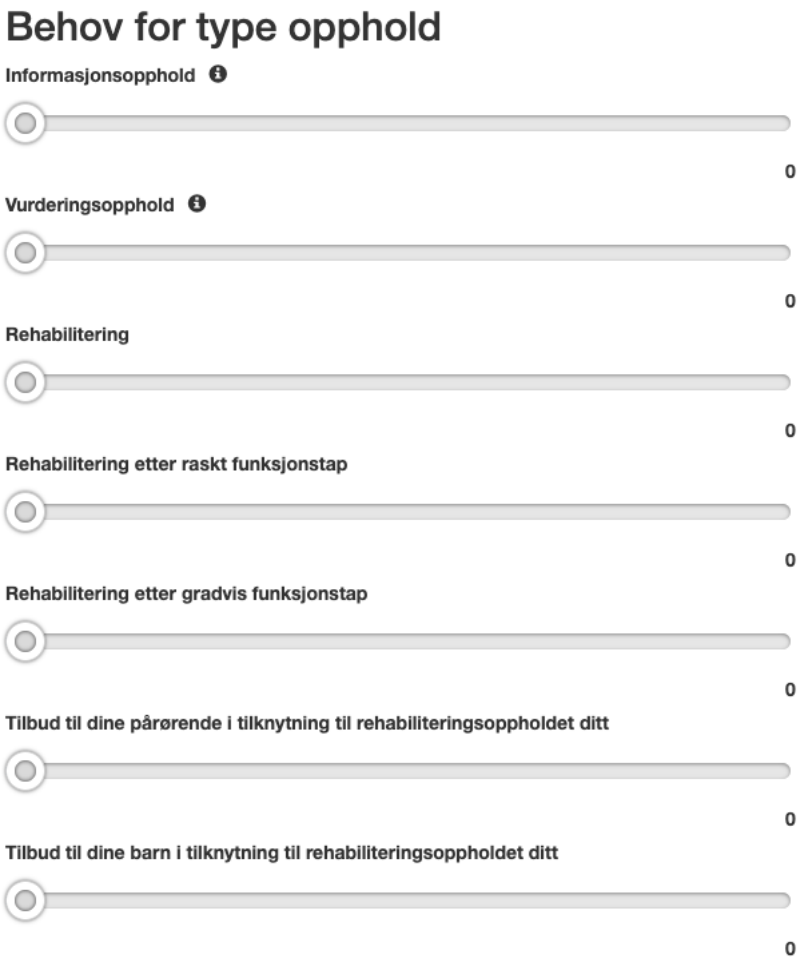


Figure 5.2: Screenshot of the patient component

An image showing the user interface of the patient component

5.1.5 Feedback

The Feedback component is used by the patients to give feedback to a treatment center. By using the unique id given in the Patient component, the patient will get access to all the questions that were given the score of 1 or higher. Much like the Patient component, each question has a slider where the patient can rate the question from 0 to 10. Since we do not know which treatment center the patient attends, we asked them about this information as well. This is done by giving the patients access to a dropdown menu, which contains every treatment center in the database. A submit button is given to submit the answers.

This component works much like the Patient and Center components. We get the question from the API, display them as sliders (without RFF) and submits the answers with Axios. A response is given to tell the user if the task was successful or not. The only addition is the `submitPatientId` function that is used to submit the unique id. Instead of adding a submit button, the function will wait until the input field has a string of length twenty. When this requirement is fulfilled, the application will automatically send a request to the API and ask for the questions corresponding to the unique id.

5.1.6 Admin sites

The admin sites are where the administrators can manage questions and look at the data in the database. To make it a bit more user-friendly, we have decided to split it into five different components:

Add question

Add question is a simple component for adding a new question to the database. The questions will not be added directly into the patient/center pages but will appear in the "Manage questions" component. The component consists of three `<input type="text"/>` elements: question, id and extra information.

Manage questions

This is where the admins can decide which questions should be displayed in the patient and center components. This is done by first selecting which component you want to manage, then selecting the category as shown in figure 5.3. The component will then display all questions in the database, as well as the questions already existing on the patient/center component. You can now choose to add a question to the patient/center component by selecting the question you want to add, as well as how you want to display it. You can also remove a question by clicking on it and then click the remove button. If you want, you can add or remove multiple questions by shift- or ctrl-clicking on multiple questions. The submitted changes will be sent to the backend where they will be added to the configuration

file. You can also manage categories on this page. You can add a category by simply entering a name in the text field and click add. If a category becomes empty, it will automatically be removed. An image of these features can be found in figure 5.4

First, the component makes an API call to the server, asking for all questions in the database and the patient configuration file. The categories and questions found in the configuration file, are added to two different lists. The categories are displayed in a dropdown menu, while the questions are shown is a RFF multi-select component. The list of all questions is added to another multi-select component. The rest of the page is static and is generated without any help from the backend.

In addition to the render methods, we have some functions for managing the selected question and categories, `addToList, removeFromList` and `addCategory`. These functions will update the lists mention earlier. If we add a new question or category, the function will update its corresponding list. If we remove a question, the `removeFromList` function will find the question in the list and remove it. If there are no questions left in the given category, the application removes the category as well. The updates will not be added to the configuration files until the user clicks the update button.

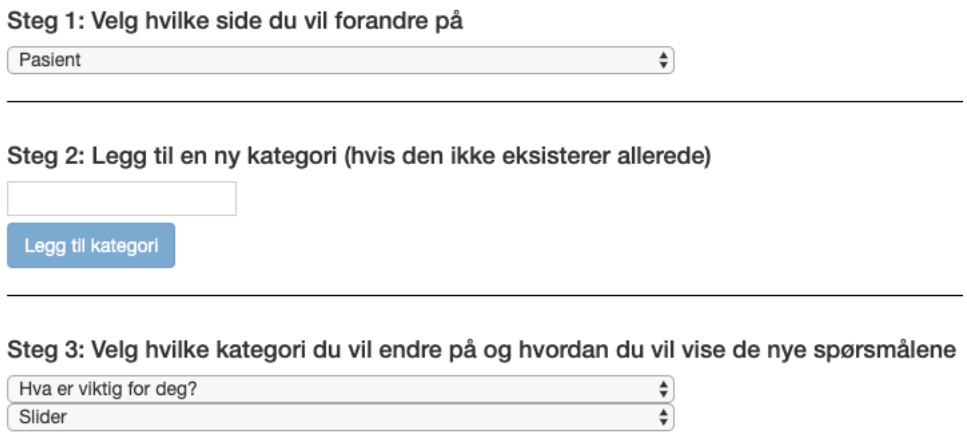


Figure 5.3: Screenshot of the manage questions component

An image showing the user interface of the manage question component

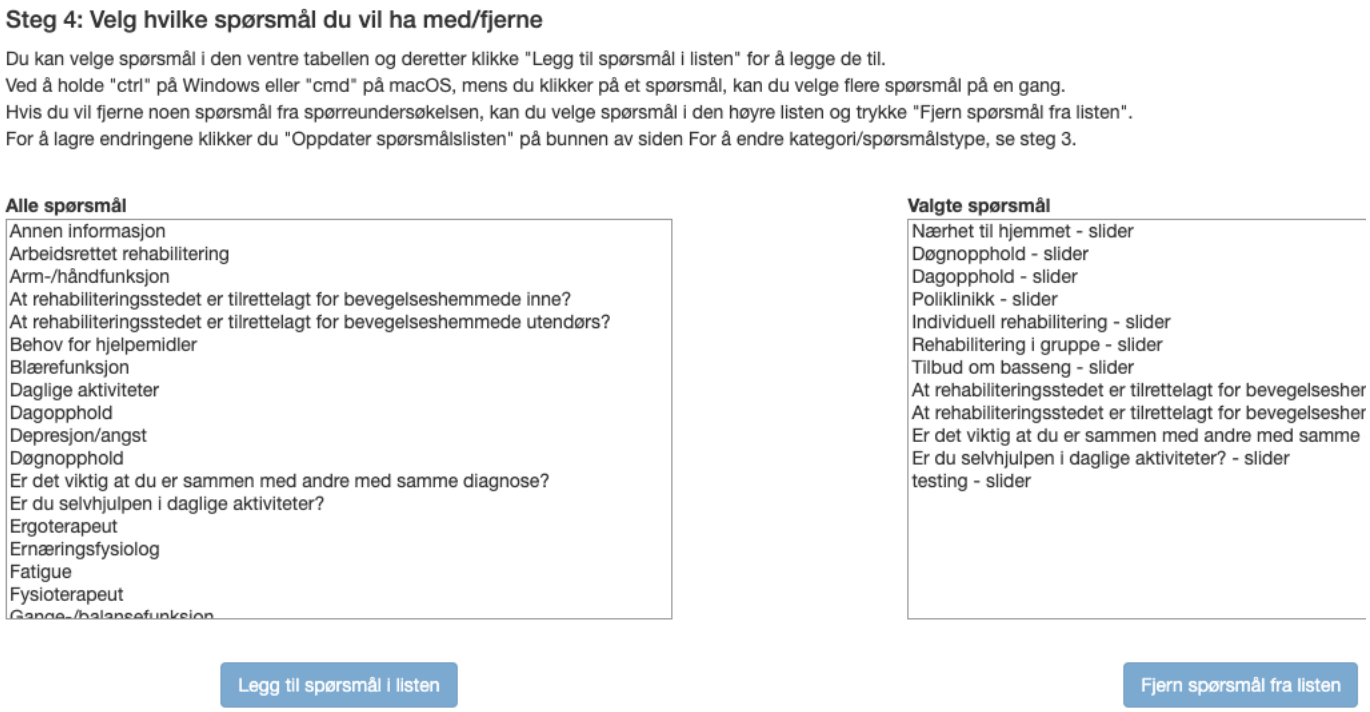


Figure 5.4: Screenshot of the manage questions component

An image showing the user interface of the manage question component

Add connection

In this component, we are able to add connections between two questions. The page contains two dropdown menus with all available questions as well as a table from react-table showing all current connections. To add a connection, simply select the questions we want to connect, and submit. The table will be updated immediately to show the user that the task is complete.

Review feedback

To prevent our recommender system from becoming a black box, we added a Review feedback component, where the admins can access all feedback given by patients. The data is displayed in a react-table, showing a treatment center's score on a specific question. This component also gives us the possibility to change these scores. By using the table, admins can decide if a score should be increased or decreased based on patient feedback.

Center information

This component contains a table that shows all the data given by the treatment centers. This is where we get the benefit of react-table's sorting function. If we want to find all the information about a specific treatment center, we can sort the treatment center column. If we want to see how the different centers scored on a specific question, we can first sort according to question and then according to score. This will give us a table showing which centers that have the highest score on a specific question.

With the help of Fuse, we made a search function, where the users can enter a keyword, a question or a treatment center. In return, the search function will try to find all the rows this fulfills the query. This feature was one of the suggestions given during survey 3.5.6

5.1.7 Other components

In addition to the large components that make up a web page on their own, we have a few smaller components and features. These components are used in some or all of the larger components.

Header

The header works as a menu and is used to navigate the site. It is made with the help of bootstrap's Navbar component. Each button in the header is connected to a route and will redirect to the specified page when clicked. The header does also have a dropdown menu for all the different admin pages.

Information boxes

The information boxes are also made with the help of bootstrap. It is displayed in the client as a Glyphicon (i). An OverlayTrigger is used to show the message when the user hovers over the boxes, and the Popover component contains the information displayed on hover. The information icon component is used in the patient- and center forms to give the user more information about the question asked. When used, the component requires a header and a message, `<InformationBox header={"header"} text={"Some text"}/>`.

Place and recommendation

The Place component is a template on how a treatment center should be displayed. It displays information like name, score, link to the treatment center and a list containing all treatments/symptoms that the patient and center have in common. A Place is displayed in the Recommendation component. This component will place X amount of Places on the screen. The X is decided by the number of treatment centers given by the backend.

Response

When the user clicks a button, they expect some kind of feedback from the application. This is where the Response component comes in. Whenever a user clicks a button and no other response is given, we display a Response component to show the user that the task was successful or failed. To display this message, we use the Alert bootstrap component. The Response component is red if something went wrong, or green if the task was a success.

Loading and printing

The last two components are not really components but are still a small part of the program. The loading bar is just a `<div/>` with some CSS that makes the loading bar spin in circles. It is displayed when a component is loading. The final feature is a button on the header that prints the page currently showing.

5.2 Data handling layer

The data handling layer contains our API that we use to send data between our presentation layer and application processing layer. Our API is written in Flask[37], a framework for making web applications in Python. While Flask has a lot of features, we are only going to focus on two extensions: flask_restful and flask_cors.

5.2.1 Flask_restful

Flask_restful is an extension that can be used to make REST API's in Python. It is a lightweight library that is easy to learn and can make API's with only a few lines of code. A debugging mode is available when developing the API. This mode compiles and restarts your API every time you save your code and provides improved error messages if something goes wrong. Flask_restful supports all the basic HTTP methods: GET, POST, PUT and DELETE.

Listing 5.1: POST and GET methods in Flask_RESTful

```
class Centers(Resource):
    def get(self):
        return sql.get_questions_by_id("center")

    def post(self):
        json_data = request.get_json(force=True)
        sql.add_new_center(json_data)
        return {"status": "success"}
```

5.2.2 Flask_cors

Flask_cors help us deal with Cross-Origin Resource Sharing (CORS). CORS is a mechanism that makes it possible to share resources across two domains. Because of security reasons, our client which is at Heroku is not allowed to request resources from our API because it is on a different domain (HVL). To solve this problem, we can either make a whitelist and add Heroku as a valid domain, or we could tell the API to accept requests from any source. Flask_cors gives us the possibility to accept requests from any source with only three lines of code.

Listing 5.2: Flask_cors setup

```
from flask_cors import CORS
cors = CORS(app)
app.config['CORS_HEADERS'] = 'Content-Type'
```

5.2.3 Pipeline

When receiving a request from the client, the API uses methods found in our database layer to query our database. The result is then moved to the application processing layer where

it is converted into JSON. Finally, the JSON file is moved to the API, which sends the data back to the client. If the given request is a POST request, the API will move the data to a method that inserts the given data into the database. POST request that does not have a return statement, returns a status message instead. A system sequence diagram showing our pipeline can be found in figure 5.5

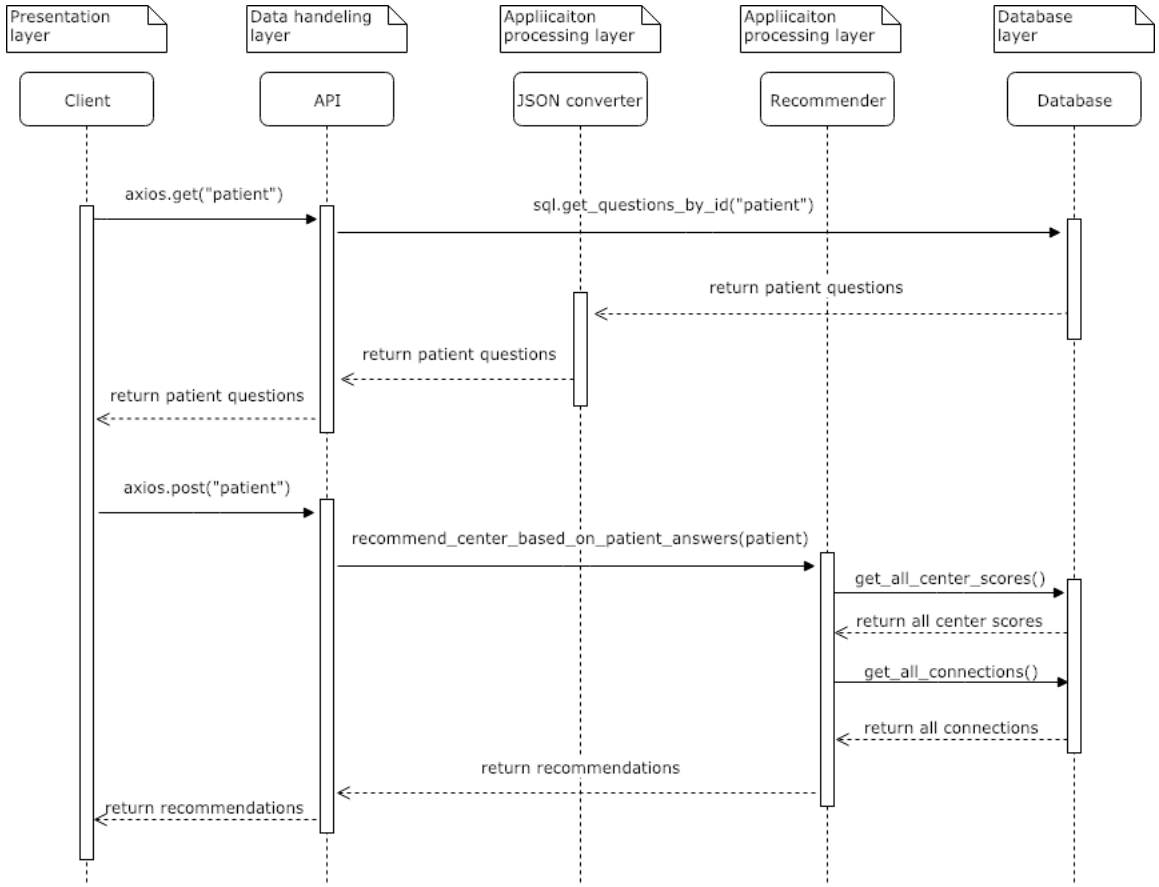


Figure 5.5: System sequence diagram of the patient pipeline

A system sequence diagram showing the data flow during a treatment center recommendation

5.3 Application processing layer

The application processing layer contains methods for giving recommendations, converting data to JSON and generating random strings. Everything in this layer is written in Python.

5.3.1 Python

Python[14] is an object-oriented language that is easy to learn and read. It has a large standard library and a huge amount of easily accessible, nonstandard libraries that can solve a lot of issues with a few lines of code. With libraries like Scikit-learn and TensorFlow, it is known to be the best language when it comes to machine learning.

We have chosen to use Python in this application because of its simplicity and easy setup. The application is small and there is no need to split the code into many files and classes like you would with Java. While Python works on small projects, Java might be better when making a large project with many users and requests. Since we do not need this kind of large scale project, we prefer a fast and effective language like Python.

As mentioned above, Python works great when doing machine learning. Since our initial plan was to use machine learning to recommend treatment centers, Python was the preferred language. While we did not use machine learning in our final product, it is possible to use a machine learning approach when we have collected more data. Changing from Python would remove this possibility and it would be time-consuming to rewrite the entire backend in another language. [16]

5.3.2 Feedback system

To give our system the possibility to learn, we added a feedback functionality where the patients could give feedback on the treatment centers they have visited. When a treatment center submits their information for the first time, every question is given the score of 50. The score will then increase or decrease based on the patient feedback, where the minimum score is zero and the maximum score one. This score is not connected to the treatment center

as a whole but is supposed to rate a specific treatment that the center has to offer. If the patients are happy with treatment **A**, they will give **A** a good score, which would increase the center's score and reputation on treatment **A**.

There are two ways to update the feedback score. The first one happens automatically when a patient gives feedback on a treatment center. The score is calculated with the formula

$$current\_score + \frac{score\_given\_by\_patient - 5}{10}$$

. In other words, a perfect score of 10 from a patient, would increase the feedback score with 0.5 and the lowest score of 0, would decrease the feedback score by 0.5. The other methods are used by the admins, where they can manually update the score to whatever they want. This is mostly used if the admins find the score unfair.

5.3.3 Recommender system

To give a good recommendation, we use the collected data from the treatment centers and the patients to find similarities between the two. We start off by removing all the questions where the patient gave the score of zero. By removing the questions early, we can shorten the time it takes to run the algorithm by a large amount. If the patient gave five questions a score of one or higher, we would run the inner loop about 67500 times with 50 treatment centers. If we use all 38 questions, we would run the inner loop about 513000 times.

After removing the questions, we start iterating through the *center_scores* variable. This variable is a list of *Score* objects containing:

- A Question object Q
- A Center object C
- A Score given to a Center C on a specific Question Q.

To make the code a bit easier to read, we made a function that separates the *Score* objects into dictionaries according to which *Center* they belong to (5.3.3). In other words, each treatment center has a key in the dictionary, where the value is a list of *Scores* connected to the given *Center*. Now, instead of looping through the *Scores* and checking which *Center*

the *Score* belongs to, we can simply loop over each *Center*, and then loop over each *Score*. The same result could be achieved by doing one query to the database for each treatment center. However, this approach would move more workload to the database.

Listing 5.3: Dictionary of Scores

```
{
  centerA:[scoreObject1, scoreObject2, ...],
  centerB:[scoreObject1, scoreObject2, ...],
  ...,
}
```

We now have a specific *Score* object. The next step is to check if the *Question* Q is answered by the Patient. This is done by looping through all the patient answers and comparing them to Q. If the patient has given Q a score of 1 or more, we add the patient score $*$ center score to the result score ($result\_score+ = 4 * 0.5$). Finally, if the patient has answered *Question* Q, we check if Q has a connection to any other questions. If a connection exists, we use the same formula as above. Whenever we find a match or connection, we add *Question* Q to a list. This list is used to show the reasoning behind the recommendation given.

After each iteration of the outer loop, we save the result score and reasoning list and wipe all variables. The resulting score for each center is then sorted and the three treatment centers with the highest score are selected. All known information about these centers is then added to a JSON file, together with the reasoning and returned to the client.

Listing 5.4: Pseudocode of a converter

```
remove_all_patient_scores_bellow_threshold()
split_all_center_scores_to_their_correct_list()

for all centers:
  reset all variables

  for all scores given to a center

    for all patient scores
      if both patient and center has answered the same question
        add score given by the patient to the result score
        add the question to the relevant questions list

      for all connections in the database
        if the current patient score has a connection
          add score given by the patient to the result score
          add the question to the relevant questions list

return convert_result_to_json()
```

5.3.4 Utilities

The remaining methods can be found in our utilities-file. This file contains methods for converting database files to JSON, generating random strings and communicating with the configuration files.

Converters

The converters are used to convert database objects to JSON. This is done by iterating through the list of database objects and convert them into a JSON object. We use these converters when we want to transfer the given data to the client with the API.

Listing 5.5: Pseudocode of a converter

```
1 for object in listOfObjects
2   json.add({"element1": object.element1, "element2": object.element2, ..})
3 return json
```

Random string generator

When patients have completed the survey, they get a random id which they can use to give feedback on their treatment. Since this id hides personal information about the patient's symptoms, we need to make this id as safe as possible. To generate this random id, we use a method that picks a random character from a list of size 64. As of now, the generated string is 20 characters long. With some quick math, this gives us

$$64^{20} = 1329227995784915872903807060280344576 \approx 1.33 * 10^{36}$$

different combinations.

According to PWDtools [36] and other similar calculators, it would take a stupid amount of years to brute force an id. Even if you manage to find the id, there is no indication on whom this id belongs to. To be on the safe side, we run a check to see if the id is already assigned to a patient before assigning it to a new patient.

Configuration files

The last method is used to update and read the configuration files used to display questions on the client. When an admin updates the questions through the client, this method will transform the data into a specific syntax and overwrite the old configuration file with the new data.

5.4 Database layer

Now that we have a client and some application logic, we need a way to store the collected data. While there are a few different ways to store data, we have decided to use a relational database. Our Relational Database Management System (RDBMS) is called SQLite and is a more lightweight RDBMS embedded in the program itself. It is commonly used as storage in clients and mobile apps, but it also works in a two-tier architecture where the database is part of the server-tier. SQLite has the same functionality as any other SQL database, with the exception of domain integrity. In other words, SQLite cannot guarantee that the correct element type is stored in the database.

Our database consists of eight tables:

- Entity - Is either a center or a patient
- Address - Contains information about an entity's address
- Center - Contains information about a specific treatment center
- Patient - Contains information about a specific patient
- Question - A question that can be answered by patients and/or treatment centers
- Score - A score that rates an entity on a specific question
- Feedback - A feedback given by a patient after completed treatment
- Connection - A connection between two questions that is used when recommending a treatment center.

The motivation behind this structure is to follow the rules of database normalization. Database normalization is a way to structure your database to improve data integrity and

reduce redundancy. There are a total of 11 normal forms that describe how you can improve your database structure. While you can achieve all 11 normal forms, it is common to stop at Boyce–Codd normal form (BCNF) or 3NF. Most BCNF tables are free of insertions, deletions and update abnormalities, which is what we need for our project. A more detailed view of the structure can be seen in figure 5.6. [34]

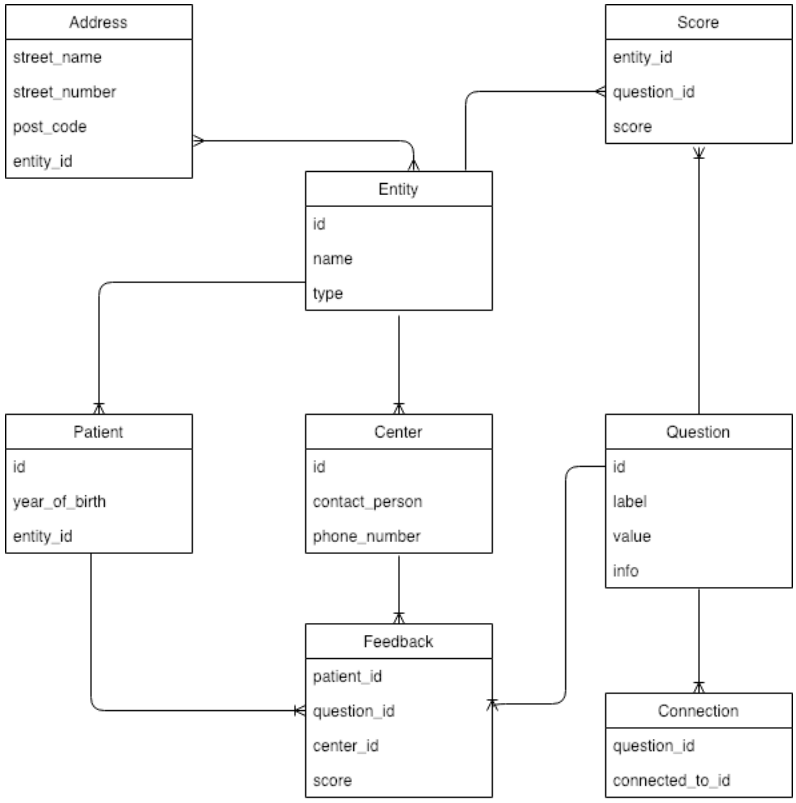


Figure 5.6: Entity-relationship model

Entity-relationship model of the database

5.4.1 SQLAlchemy and Object-relational mapping

To simplify the communication between the application processing layer and database layer, we use a technique called ORM. What ORM does, is making objects from data, much like object-oriented programming. By giving every table in the database its own object, we can easily read and write to the database by making new objects. For example, for every new submission from a patient, we make an entity, address and patient object as well as a score object for each question answered by the patient. The ORM is written in a library called

SQLAlchemy [40]. The library contains methods for working with SQL and makes it easy to add data to the database with the help of ORM.

Listing 5.6: Implementation of a question object

```
class Question(Base):
    __tablename__ = 'question'
    id = Column(Integer, primary_key=True)
    label = Column(String, nullable=False)
    value = Column(String, nullable=False,  unique=True)
    info = Column(String)
```

As mentioned above, SQLAlchemy is a great tool when making queries to the database. Instead of using the traditional SQL syntax, SQLAlchemy gives you the possibility to write queries with python syntax. While this does not make a huge difference, it is a bit more convenient to write python code rather than a large query.

Listing 5.7: Query to get all the treatment center and their scores with SQLAlchemy

```
session.query(Entity, Question, Score)
    .join(Score)
    .join(Question)
    .filter(Entity.type == "center")
    .all()
```

Listing 5.8: Query to get all the treatment center and their scores with SQL

```
SELECT * FROM Entity
INNER JOIN Score ON Entity.id == Score.entity_id
INNER JOIN Question ON question.id == Score.question_id
WHERE Entity.type == "center"
```

5.4.2 Configuration files

The project contains a few configuration files that decide which questions should be displayed in the patient and center module. These files are simple JSON files that contain information about categories, question id's and how we should show each question on the client. The *displayAs* key, supports all the different input types that can be found in the web client:

- text - Text input
- slider - Sliders used in the patient module
- radio - Radio buttons
- checkbox - Check boxes

- textarea - Text areas

Listing 5.9: Configuration file example

```
{
  "What is important for you?":[
    {
      "id": 1,
      "displayAs": "slider"
    }
  ]
}
```

Chapter 6

Results

In this chapter, we present the results of our surveys.

6.1 Results from patient testing

The feedback from the user testing was overwhelmingly positive. Everyone managed to complete the survey and almost everyone got some useful information from the recommendation given. There were a few people who had some trouble when answering the questions, but these were mostly related to how the questions where asked, not the system itself. Our effort to reduce the number of questions asked has paid off as well. No one had any problems with the length of the survey.

While the information icon is explained in the introduction, there were a few patients who said they did not find it during the survey. This suggests that we have to do something about it. Either make it bigger or move it somewhere else to make it more visible to the user. Some also found it difficult to use the sliders with a touchpad. While it is possible to click on the slider, we forgot to mention this in the introduction. An update to the intro would solve this problem.

When we asked the patients if they found the tool useful, eight out of nine said yes. We got some positive comments as well, where the participants said that it was a good

measure. This indicates that the tool is working and can give patients useful information about treatment centers in the future. A table of the quantitative data can be found below.

Table 6.1: Results from patient testing based on response from 9 patients

Question	Yes/Good	No/Bad
Were the motivation explained in an understandable manner?	7	2
Were the survey self explanatory?	8	1
What do you think about the length of the survey?	9	0
Was the questions self explanatory?	7	2
Did you use the information icon?	7	2
Did you manage to complete the survey?	9	0
Was the results presented in an understandable manner?	8	1
Is there a need for this kind of application?	8	1

6.2 Results from treatment center testing

The feedback from the first test described in section 3.5.4 was a bit mixed. The biggest concern was the difficulty of estimating wait times. To keep this information up to date, they would have to update it weekly. Because of this, we decided to remove all questions about wait times. We also got some feedback on missing professions and treatments. These were later added to the website.

Another problem we noticed during testing, was that we didn't have access to the person who filled the form. Any questions we had about their submission was left unanswered because we did not know whom to contact. As a result, we added a text field where people could enter their personal email addresses.

Our second test described in section 3.5.5 was a lot more successful. In the quantitative, closed-ended part of our questionnaire, we only got positive answers. There where a couple of cases where no answer was given, but the reason behind it, is unknown. So, if we only look at the quantitative data from the people who answered, we have 100% approval rate.

During the test, we discovered a problem we had encountered before when running the website within Haukeland University Hospital. People could not access the website on the secure hospital networks. We got a few emails about this problem, but we did not find a

solution to it. Luckily, the treatment centers were able to access the website through phones and tables connected to a guest network.

We got a few comments concerning the lack of short text answers when some alternatives are missing or an answer would only apply in a specific scenario. Such fields were available in a previous iteration, but we removed them because they were not used when recommending treatment centers. We could add text fields for writing missing symptoms and professions, and then use the information to manually add them to the website later.

But, when it comes to specific scenarios, it is difficult to find every corner case where something only applies given something else. As an example, someone mentioned that they did not offer cognitive therapy by itself, but could offer it to patients with other needs as well. If we found all these scenarios, the list of questions would be ridiculously long and there would be a chance that the treatment centers would not take the time to answer the questions.

Another comment we got when it comes to the survey length, was redundant questions. We found out that questions concerning accessibility within the treatment centers are redundant since every treatment center must offer full accessibility to everyone, including wheelchair users etc. Information like this can help us reduce our list of questions.

Table 6.2: Results from second center testing based on response from 15 treatment centers

Question	Yes	No
Was the motivation explained in an understandable manner?	13	0
Was the survey self explanatory?	14	0
Was the questions self explanatory?	13	0

6.3 Results from admin testing

During the first set of admin tests, we got a few suggestions on how we could improve our application. The biggest concern was the lack of feedback from the application itself. Some buttons did not give any response when clicked and our users were not sure if they actually clicked the button or not. Another suggestion was to sort the list of questions shown in the

Manage Questions component. This would reduce the time it took to find the question you were looking for.

The final suggestion was to make a search function to our treatment center table in the admin module. This would reduce the time needed to find the information you need. It would also give the users the possibility to do simple queries for statistical use.

During the test with Norwegian Multiple Sclerosis Competence centers, we discovered more problems concerning feedback from the application. This problem was solved immediately. We also got a request to do something about the creation of new categories. This was resolved by moving everything that concerns categories to step 2.

6.4 Meeting with the Department of Rheumatology

To answer our second research question, we arranged a meeting with a doctor from the Department of Rheumatology. In this meeting, we discussed the similarities between MS and rheumatology when it comes to "Fritt behandlingsvalg", symptoms, treatment centers and wait times. The goal was to determine if our application would work with rheumatology as well as MS.

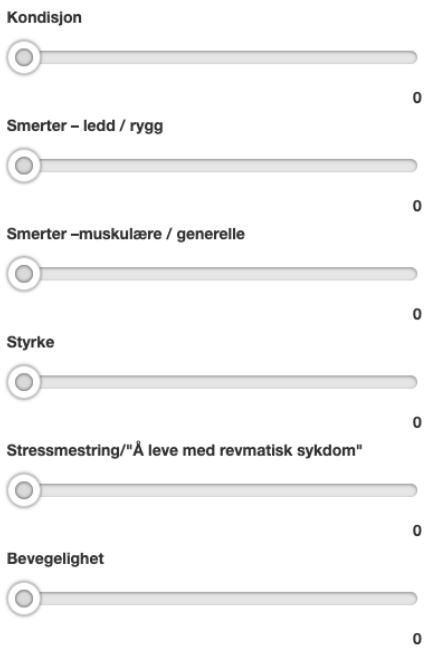
During our meeting, we talked about rheumatology and "Fritt behandlingsvalg". Both MS and rheumatology patients can attend treatment wherever they want and they can find available treatment centers and their wait times at helsenorge.no. They also share the same problems when it comes to a lack of information. The list of wait times is rather sparse, and no one really knows which treatments the different centers offer.

While a lot of patients from Hordaland attended treatment at Haukeland University Hospital, the doctor from the Department of Rheumatology did not deny the possibility that some treatment centers might offer better treatment on some symptoms. Our application solves this problem with the help of our feedback feature by recommending treatment centers with a good reputation for a specific problem (e.g. fatigue, pain, etc).

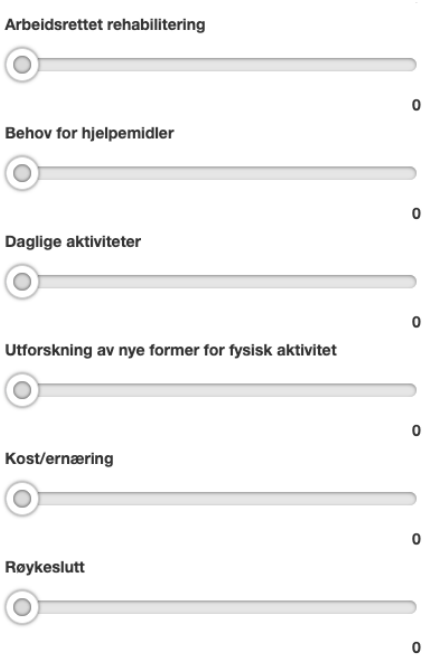
To summarize the information gathered during the meeting, the doctor from the Department of Rheumatology found our application useful and thought it could solve some of their problems. The main problem concerning information access can be solved with our application and may split patients between different centers rather than sending everyone to the large hospitals in Norway.

6.4.1 Testing our application with Rheumatology

To test if our application would work with Rheumatology, we ask the doctor from the Department of Rheumatology if she could provide us with questions related to Rheumatology. With the questions given, we modified our website to work with the new set of questions. All modifications were made with the help of our admin module. No change of code nor configuration files. The result from our test can be seen in figure 6.1.



(a) Rheumatology questions 1



(b) Rheumatology questions 2

Figure 6.1: A figure showing the results of our rheumatology test

Chapter 7

Discussion

In this chapter, we first present previously attempted frameworks and languages. Then we discuss our research questions and come to a conclusion.

7.1 Tested frameworks and languages

During the course of this project, we have done some testing with different frameworks and languages. Some did not make the cut and were replaced by other languages and frameworks later in the project. This section describes the reasoning behind our choices by comparing the currently used frameworks and languages with the old ones.

7.1.1 Angular 4

Our first attempt at making a front-end client was written in Angular 4. While Angular did what it was supposed to, it was incredibly difficult to learn and everything seemed a bit clunky. In order to make an API call with Angular, you had to import a bunch of libraries, make an HTTP-service, edit some configuration files and then write the API call. In React, you can simply install one library and everything works with a simple one-liner. Since Angular uses Typescript, it is a bit more difficult to find libraries that work with an

Angular application. In React, you can find libraries for just about anything. Finally, writing code and debugging was a slow process with Angular and it was difficult to find answers on the internet. React had a lot of easily accessible information and was well known by fellow students.

On the technical side of things, React use a virtual DOM, which only renders the needed components each time something changes, while Angular uses a real DOM where everything is re-rendered on change. This means, that each time we move a slider in the patient module, we only render that specific element in React, but in Angular, we have to re-render everything. Because of this, React is a bit faster when rendering[2].

7.1.2 Node.js API

Our first goal was to get a working pipeline we could use to collect information about treatment centers. Because of our knowledge with Node.js, we could quickly set up a working API from scratch. When we later decided to explore machine learning, we switched both the API and back-end to Python to make to make it easier to exchange data between the data handling layer and the application processing layer. Most machine learning libraries are found in Python, which is why decided to write the backend in Python and not Node.js.

7.1.3 NoSQL database

We started off with a MongoDB database and used it to save the data we collected from the treatment centers. We did not have a specific database structure in mind at this point, which is why we went with a NoSql database. In the later stages of the development, we made a new data structure where the different tables had a lot of connections to each other. Since SQLite is a relational database management system, we decided to switch to SQLite to make it easier to connect our data together.

7.2 Answering research questions

To put our research question to the test, it would be beneficial if we had some old data to compare our recommendation system with. There are no data that tells us if the patients are satisfied with the treatment center or not. This becomes a problem when we want to compare the currently used system with our application. To compensate, we added a question to our survey asking the patients if there is a need for a decision support application like ours. This question would give us an indication of whether there is a need for our application or not.

Is it possible to make a digital solution for recommending treatment centers?

Since most of the feedback is positive and 8 out of 9 patients found the need for this application, we could conclude that it is possible to make an application for selecting treatment centers. On the other hand, there are still a few problems concerning wait times and data gathering. Without a proper way to gather wait times, we can not recommend centers based on availability. If needed, patients can find this information by themselves by looking at the overview at helsenorge.no, but as of now, we can not add these wait times to our application. Because of this, we might recommend a treatment center with a large queue when there are centers with available space.

The other problem occurs in the center module. The problem is that there are so many different ways to name and describe a symptom or treatment. To satisfy all treatment centers, we would need to have a text field after each category to gather information about all the different corner cases that exist. This becomes a problem when we want to keep the survey short and simple. If it is too long, people will not bother updating their information frequently. If it is too short, we miss valuable information. To find the perfect amount of questions is key, but very challenging.

A large amount of similarities between centers does also become a huge problem. After looking through the data from our second treatment center survey, we found that many centers have a lot in common and some centers were completely identical. As of now, the only way we can differentiate two identical treatment centers, is through our feedback system. A center with positive feedback would get a better score than the rest. Other tiebreakers include wait times or to recommend centers based on location preferences. Both of these solutions are discussed in section 8.4

As of now, there is no way to tell the frontend that a field is required. Another missing feature is the possibility to show/hide some fields if the user answers yes or no on a specific question. An example is our questions concerning age restrictions. Some treatment centers might only offer treatment to people who are younger or older than a specific age. As of now, there is no way of hiding the input fields if a treatment center answered no when asked if they have an upper or lower age restriction. In this case, it would be nice if we could show the "Enter upper/lower age restriction" field, only if the center had answered yes on the previous question. While these features are not mandatory, it would be a good way to improve our application.

On a positive note, everything else seems to work as intended. The positive feedback from the patients, treatment centers, Norwegian Directorate for e-health and the Norwegian Directorate of Health shows us, that with a bit of tuning and access to more information about wait times, we could make an application that can recommend treatment centers pretty accurately.

Is it possible to make a general framework that can be applied to a variety of different treatments and diseases?

To say that our application could be applied to different treatments and diseases, we need an application that is easy to use and that can be managed by people without a technical background. While we have been successful in most of these requirements, there are still a few problems that occur.

For everything to work properly, each disease needs to have its own instance of our application. In other words, we would have to deploy a new version of our application for every disease. This is easily achievable by deploying the application in the cloud and make a new instance for each disease. The problem is, that this process can be a bit difficult for people without any technical background. Because of this, it would be beneficial to have some sort of IT department that could help to initialize new instances.

After the application is deployed, it can be customized with the help of our admin module. Ideally, an expert on the given disease should take the place as admin. They know which questions should be asked to give a good recommendation. From this module, admins

can manage both the patient and the center module by making questions, categories, and connections. Everything else is sorted out by the application itself by either matching two identical questions or by matching connections defined by the admins.

The results from our rheumatology test (section 6.4.1) show that our application may work well with other diseases. Within 15 minutes, we managed to modify the website to work with rheumatology without manually interacting with the backend. While this indicates that it is possible to modify our questions to work with different diseases, the rheumatology example was not tested as thoroughly as MS since we did not go through the process of asking treatment centers and patients for feedback, etc. In addition, it would be beneficial to test the application on more than two diseases, but that is something for the future.

7.3 Related Work

Recommender systems are quite common when it comes to personal health and healthcare. There is a lot of examples online where people use recommender systems to suggest a disease based on symptoms. There are also papers that describe applications for monitoring your own health, and that would recommend you to go to the doctor if there is something out of the ordinary. Finally, we found a few papers describing a recommender system that could recommend doctors and treatment centers based on symptoms.

7.3.1 HealthNet

HealthNet (HN) is a social network where patients can store personal health data, and share conditions and experiences with each other [12]. The data stored, is used to make a recommender system much like ours. By connecting people with the same conditions, the HN recommender system can recommend doctors and hospitals based on other people's experiences. HN is only available in Arizona, California, Oregon, and Washington.

In the article, HN talk about their experience when testing four different algorithms. The algorithms where:

- Collaborative filtering

- Cosine Similarity
- HN's homemade algorithm
- A hybrid between HN's algorithm and cosine similarity

Just like us, HN did not have success with Collaborative filtering despite having access to more data. Their homemade algorithm, on the other hand, worked pretty well. The algorithm combines Jaccard Index, Inverse Document Frequency and Cosine Similarity. We tested these algorithms separately with a decent result but did not combine them, in the same way, HN did. The problem with their algorithm, much like our previous attempts, is that the algorithm is binary. The symptoms are either really important or not important at all. While the HN algorithm might work with our recommendation system, it has to be changed to work with symptoms rated from 0-10.

Another big difference is that they try to find patients similarly to you, and then recommend doctors and hospitals based on their experience. Our approach is to find similarities between the patient's symptoms and what the treatment centers have to offer. To compare two patients, we would need a huge amount of data. We do not have access to this kind of data, which makes it difficult to use their methods. It is possible to change the algorithm when we have collected enough data, but this could take a while. Instead, we have given the patients the possibility to give feedback after ended treatment. This gives us the possibility to train the model based on user experience.

7.3.2 A Patient-Centric Healthcare Model

The model described in this paper is a bit similar to HN [4]. The biggest difference, it that they removed the social network part and replaced it with personal health records, electronic health records, and electronic medical records. This model does also compares patients but combines this information with the doctor's expertise. The model is only a suggested model by the authors and is not tested before the paper was written.

The method described in this paper would be an ideal solution to the problem. If we could access health records, we could use these records along with the information given by the patient to recommend a treatment center. The problem is, as mentioned in the paper, privacy, and security. A possible solution to this problem is discussed in section 8.1. As of now, there is no way we could access the health records or guarantee safe communication when accessing the records.

7.3.3 A Novel Model for Hospital Recommender System Using Hybrid Filtering and Big Data Techniques

This paper [8] describes a recommender system much like ours. Their goal is to recommend a hospital based on user preferences. To solve this problem, they have used a combination of collaborative filtering and content-based filtering, called hybrid filtering. The idea behind it, it that they make one vector based on user preferences and add them to a matrix. This matrix is then used to recommend hospitals based on cosine similarity. A feedback system is also available after the recommendation is given. They also suggest a method to give recommendations based on the hospital's position according to the patient.

While the other papers describe methods for recommending doctors based on other patients experience, this system recommends hospitals based on the similarity between the hospital and the patient. This is the same approach we took when making our application. They also describe the problems with a cold start and sparse data and claim to solve it with hybrid filtering.

7.3.4 A Hybrid Recommender System for Patient-Doctor Matchmaking in Primary Care

This paper [19] describes a method for recommending family doctors based on previous interactions between patients and doctors. Much like the other papers, they suggest a hybrid method that is based on previously collected data. This paper differs from the others by applying different techniques based on the amount of data they have about each patient. If the data is sparse, they compare the recommendations based on demographic similarity. If the patient has visited hospitals but never had a family doctor, the system recommends based on hospital visits. Finally, if the patient has a lot of data, they give recommendations based on hybrid filtering.

7.3.5 Summary

All the methods described in this section take to account that you already have or can collect a large amount of data. Because of privacy reasons, we can not apply these methods, as we

do not have the possibility to store or access personal data. If we could move our application to helsenorge.no [11], we could apply the method of hybrid filtering when recommending treatment centers.

7.4 Conclusion

Then comes the conclusion. We have a working pipeline, we are able to recommend treatment centers and most importantly, the users seem happy with our product. Feedback indicates that patients would use this application to find a place to attend treatment, and many treatment centers find this application useful. While there are a few problems with our application, it still works as intended. Because of these arguments, we would like to declare this project a success!

Chapter 8

Future work

While most of our planned features were implemented, there are still a few additions we would like to add to our application. There where also a few tests we would like to carry out but did not have the time nor resources to complete. These tests and features are discussed in this chapter.

8.1 HelseNorge

To make our application more secure, we would like to deploy it to helsenorge.no[11]. This website is the main source of information when it comes to Norwegian healthcare. There is also a section where Norwegian citizens can access and manage personal data. If we could deploy our application behind the same, secure wall, we could:

- add the patient data to the journal,
- save the result(s) given by the recommender system to be accessed later,
- remove the unique id given to the patients and let them give feedback through their helsenorge identity instead,
- guarantee encrypted communication with the website,
- and guarantee the safety of their data. [10]

Since some treatment centers have an upper and/or lower age limit, we could use the patient data to access their date of birth. With this information, we could remove all the possibilities that do not match your current age. As of now, we have chosen to avoid asking for such information for safety and privacy reasons.

8.2 Further testing and more iterations

As mentioned in section 2.4, we made a framework that should (in theory) work on any kind of treatment. While we did not have the time or resources to test this hypothesis, it would be interesting to test the system on other diseases and see if it worked as intended. This would include more user testing and new iterations to find out if the application works properly in every scenario.

8.3 Language support

During user testing, we discovered a language barrier that we had not expected. Some of the patients had difficulties understanding some Norwegian words and phrases used in the application. To solve this problem, we should give the application the possibility to use other languages as well as Norwegian. We could solve this problem by using our current connection functionality, but it is not an ideal solution. A better idea would be to give each question the possibility to contain a lot of variations as shown in figure 8.1. To make this work, we would also need a configuration file for each language as it contains categories written in Norwegian.

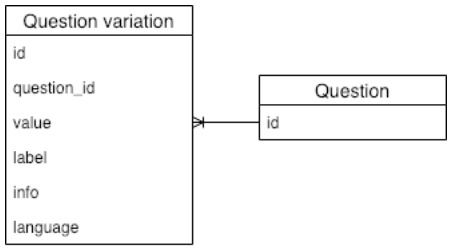


Figure 8.1: Questions that support different languages

An edited model showing how to solve the problem with different languages

8.4 Postcodes, distances and wait times

To make treatment center selection a bit more convenient, we had an idea to implement a way to calculate distances between the patient and the treatment center. If the patient said it was important to stay close to home, we could find centers within a specific radius and give these centers a higher score. The plan was to use postcodes to mark their positions and then use an API to find the distance between the two. We found a free API that managed to find the distances between some locations, but a lot of suggestions were completely wrong[9]. While there are some APIs that can do the work, they cost a lot of money. If someone wants to implement this feature, they can subscribe to one of these APIs, and give the patients recommendations based on location.

Another improvement we talked about, was to add the wait times as a variable. By looking at the wait times, we could recommend centers that have a short wait time over centers with a long wait time. To make this happen, we need a way to find the current wait times of all the centers in Norway. As mentioned before, helsenorge.no [11] has a list of wait times, but they are incomplete and rarely updated. If they managed to fix and maintain this list, we could use it to recommend relevant centers with the shortest wait time.

List of Acronyms and Abbreviations

API	Application Programming Interface.
BCNF	Boyce–Codd normal form.
CNS	Central nervous system.
CORS	Cross-Origin Resource Sharing.
CSS	Cascading Style Sheets.
FaaS	Function-as-a-Service.
HN	HealthNet.
HTML	HyperText Markup Language.
HTTP	Hypertext Transfer Protocol.
HVL	Western Norway University of Applied Sciences.
IaaS	Infrastructure-as-a-Service.
JSON	JavaScript Object Notation.
JSX	JavaScript XML.
ML	Machine learning.
MS	Multiple sclerosis.
NoSql	Not only SQL.
NRS	Numeric Rate Scale.
ORM	Object-relational mapping.
PaaS	Platform-as-a-Service.
PDF	Portable Document Format.
RBS	Rule-based system.
RDBMS	Relational Database Management System.
REST	Representational State Transfer.
RFF	React-final-form.
SaaS	Software-as-a-Service.
SLR	Systematic literature review.
SOAP	Simple Object Access Protocol.
SQL	Structured Query Language.
tf–idf	Term frequency–inverse document frequency.
XML	Extensible Markup Language.

Bibliography

[1] V. Madisetti A. Bahga. "Cloud Computing". In: A. Bahga and V. Madisetti, 2014, pp. 22–24.

[2] Altexsoft. *React vs. Angular Compared: Which One Suits Your Project Better?* 2018. URL: https://www.altexsoft.com/blog/engineering/react-vs-angular-compared-which-one-suits-your-project-better/ (visited on 02/14/2019).

[3] *API*. URL: https://expressjs.com/en/4x/api.html (visited on 05/22/2019).

[4] R. Bateja, S.K. Dubey, and A. Bhatt. "A Patient-Centric Healthcare Model Based on Health Recommender Systems". In: *Recent Findings in Intelligent Computing Techniques*. Springer, Singapore, 2018, pp. 269–276.

[5] D. Budgen and P. Brereton. "Preforming Systematic Literature Reviews in Software Engineering". In: *ICSE '06 Proceedings of the 28th international conference on Software engineering*. 2006, pp. 1051–1052.

[6] K. Henningsson C. Wohlin M. Höst. "Empirical Research Methods in Software Engineering". In: *Empirical Methods and Studies in Software Engineering. Lecture Notes in Computer Science, vol 2765*. Springer, Berlin, Heidelberg, 2003, pp. 7–23.

[7] ILM Corp. *Digital Storage Calculator*. 2018. URL: https://www.ilmcorp.com/tools-and-resources/digital-storage-calculator/ (visited on 01/25/2019).

[8] R. Devika and V. Subramaniyaswamy. "A novel model for hospital recommender system using hybrid filtering and big data techniques". In: *2018 2nd International Conference on I-SMAC*. IEEE, 2018, pp. 267–271.

[9] *DISTANCE API*. URL: https://no.avstand.org/api.xhtml (visited on 01/22/2019).

[10] Helsedirektoratet for E-Helse. *Bruksvilkår for tjenester på helsenorge.no – behandling av personopplysninger*. 2018. URL: `https://helsenorge.no/bruksvilkar-for-min-helse` (visited on 02/05/2019).

[11] Helsedirektoratet for E-Helse. *Ventetider for Rehabilitering: Nevrologiske og nevromuskulære sykdommer, MS*. 2018. URL: `https://helsenorge.no/velg-behandlingssted/ventetider-for-behandling?bid=347` (visited on 01/31/2019).

[12] G. Semeraro F. Narducci P. Lops. "Power to the patients: The HealthNet social network". In: *Information Systems*. Elsevier Ltd., 2017, pp. 111–122.

[13] A. Fedosejev. "React.js Essentials". In: Packt Publishing, 2015.

[14] Python Software Foundation. *Python*. 2019. URL: `https://www.python.org/` (visited on 05/03/2019).

[15] I. V. Gorbunov et al. "A Decision Support System for Prescription of Non-Medication-Based Rehabilitation". In: *Biomedical Engineering*. 2017, pp. 393–397.

[16] P. Gries, J. Campbell, and J. Montojo. "Practical Programming, Third Edition – An Introduction to Computer Science Using Python 3.6". In: The Pragmatic Bookshelf, 2017.

[17] L. Guo et al. "Which Doctor to Trust: A Recommender System for Identifying the Right Doctors". In: *Journal of Medical Internet Research, Vol 18, No 7*. 2016, pp. 28–38.

[18] Q. Han et al. "A Collaborative Filtering Recommender System in Primary Care: Towards a Trusting Patient-Doctor Relationship". In: *2018 IEEE International Conference on Healthcare Informatics*. 2018, pp. 377–379.

[19] Q. Han et al. "A Hybrid Recommender System for Patient-Doctor Matchmaking in Primary Care". In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2018, pp. 481–490.

[20] F. Hayes-Roth. "Rule-based systems". In: *Communications of the ACM. Volume 28, Issue 9*. 1985, pp. 921–932.

[21] M. J. Hjermstad et al. "Studies Comparing Numerical Rating Scales, Verbal Rating Scales, and Visual Analogue Scales for Assessment of Pain Intensity in Adults: A Systematic Literature Review". In: *Journal of Pain and Symptom Management. Volume 41, Issue 6*. 2011, pp. 1073–1093.

[22] C.J. Ihrig. “Pro Node.js for Developers”. In: Apress, Berkeley, CA, 2013, pp. 263–270.

[23] H. Jiang and W.Xu. “How to find your appropriate doctor: An integrated recommendation framework in big data context”. In: *2014 IEEE Symposium on Computational Intelligence in Healthcare and e-health (CICARE)*. IEEE, 2014.

[24] M. R. Khoie et al. “A Hospital Recommendation System Based on Patient Satisfaction Survey”. In: (2017).

[25] B. Kitchenham. “Procedures for Performing Systematic Reviews”. In: (2004).

[26] M. Massé. “REST API – Design rulebook”. In: O’Reilly, 2012.

[27] M.Chiu and W. Cheng. “Building a Classification Model for Physician Recommender Service Based on Needs for Physician Information”. In: *HCI in Business, Government, and Organizations: Information Systems*. 2018, pp. 28–38.

[28] S. A. McLeod. *What’s the difference between qualitative and quantitative research?* 2017. URL: `https://www.simplypsychology.org/qualitative-quantitative.html` (visited on 03/18/2019).

[29] F. Narducci et al. “Recommending doctors and health facilities in the HealthNet Social Network”. In: (2017).

[30] NASA. *NASA APIs*. 2018. URL: `https://api.nasa.gov/index.html` (visited on 01/29/2019).

[31] J. Nielsen. “Iterative user-interface design”. In: *Computer (Volume: 26 , Issue: 11 , Nov. 1993)*. 11. IEEE, 1993, pp. 32–41.

[32] NRK. *Weather API*. 2018. URL: `https://api.met.no/` (visited on 01/29/2019).

[33] Oracle. *Building RESTful Web Services with JAX-RS*. URL: `https://docs.oracle.com/javaee/6/tutorial/doc/giepu.html` (visited on 05/22/2019).

[34] M. Owens. “The Definitive Guide to SQLite”. In: Apress, Berkeley, CA, 2006.

[35] Helsebiblioteket/BMJ Best practice. *Multippel sklerose (MS)*. 2017. URL: `https://helsenorge.no/sykdom/hjerne-og-nerver/multippel-sklerose-(ms)` (visited on 04/22/2019).

[36] PWDTools. *Brute-force password recovery time calculator*. URL: `https://pwd.tools/` (visited on 02/26/2019).

[37] A. Ronacher. *Flask - Web development one drop at a time*. 2019. URL: `http://flask.pocoo.org/` (visited on 05/03/2019).

[38] I. Sommerville. “Software Engineering”. In: Pearson Education, 2016, pp. 490–516.

[39] Facebook open source. *React - A JavaScript library for building user interfaces*. 2019. URL: `https://reactjs.org/` (visited on 05/02/2019).

[40] SQLAlchemy. *SQLAlchemy - The Python SQL Toolkit and Object Relational Mapper*. 2019. URL: `https://www.sqlalchemy.org/` (visited on 05/03/2019).

[41] A. J. Thompson et al. “Multiple sclerosis”. In: *Lancet 2018; Volume 391*. 2018, pp. 1622–1636.

Appendices

A User testing patient survey

Valgomat for personer med MS

http://valgomat.herokuapp.com/

(Åpne i Chrome)

Det er viktig at personer med MS skal få et så godt tilpasset rehabiliteringstilbud som mulig. Som del av et mastergradsprosjekt utvikler Nasjonal kompetansetjeneste for MS i samarbeid med Høgskolen på Vestlandet et verktøy som skal gjøre det lettere å velge rett rehabiliteringssted; en såkalt «rehabiliteringsvalgomat». Ut fra svar på et elektronisk skjema der vi spør om behovet for rehabilitering, skal det fremkomme hvilket behandlingssted som synes best egnet ut fra den enkeltes behov. Resultatet er ikke en fasit på hva en bør velge, men kan gi en indikasjon på hvilke behandlingssteder som tilbyr behandlingen den enkelte trenger. Vi har nå utviklet en nettside med en foreløpig testversjon. Denne inneholder kun fiktive behandlingssteder, og den som svarer, trenger ikke å gi opplysninger i tråd med virkeligheten. I denne testversjonen ønsker vi kun å vite om spørsmålene er forståelige og om verktøyet virker greit å bruke.

Dersom du ønsker å delta i utprøving av denne testversjonen, vil vi be deg om to ting:

1. Først vil du bli bedt om å fylle ut skjemaet på en nettside. Dette skjemaet inneholder spørsmål om hvilke type opphold du trenger, hvilke problemstillinger du trenger behandling for, og hva som er viktig for deg under behandlingen. Til slutt vil du få en anbefaling på hvilke(t) behandlingssted som passer dine behov. Som tidligere nevnt, trenger du ikke å svare i tråd med det som passer for deg; du kan gjerne dikte svaret.

Dine svar til bli lagret, men vi har ikke mulighet til å koble svarene opp mot deg. Undersøkelsen er helt anonym.

2. Den andre delen er en kort spørreundersøkelse hvor vi stiller noen spørsmål om nettsiden du besøkte i del 1. Resultatet av undersøkelsen vil bli brukt som forskningsmateriale til masteroppgaven og bli brukt til å forbedre nettsiden.

Del 2

For å forbedre nettsiden, ønsker vi å stille deg noen spørsmål om din brukeropplevelse.

Hvis det er noen spørsmål du ikke ønsker å svare på, kan du la feltet stå tomt.

Undersøkelsen er anonym og vi ber deg derfor om å IKKE skrive noen andre personopplysninger på dette arket.

Sett ring rundt det svaret som passer best med din opplevelse.

Angående beskrivelsen øverst på siden:

Ble hensikten med undersøkelsen beskrevet på en god måte?

Ja

Nei

Var det tydelig hvordan du skulle fylle ut spørsmålene på nettsiden?

Ja

Nei

Var det noe du savnet?

Tekst

Selve undersøkelsen:

Hva synes du om lengden på undersøkelsen i del 1?

Kort

Lang

Passelig

Var spørsmålene lett å forstå/selvforklarende?

Ja

Nei

Var det noen spesifikke spørsmål/kategorier som var vanskelig å forstå?

Tekst

Brukte du informasjonsikonet?

Ja

Ja, men spørsmålet var fortsatt uklart

Nei, hadde ikke brukt for ikonet

Nei, la ikke merke til ikonet

Resultat

Klarte du å gjennomføre undersøkelsen i del 1?

Ja

Nei

Hvis nei, hva var problemet?

Tekst

Var resultatet presentert på en forståelig måte?

Ja

Nei

Synes du det er behov for et slikt hjelpemiddel?

Ja

Nei

Annet

Har du noen andre kommentarer til nettsiden i sin helhet?

Tekst

Velg din aldersgruppe aldersgruppe:

Under 45

Over 45

Dine IT-kunnskaper

Hvor ofte bruker du smarttelefon/nettbrett?

Aldri

En gang i uken

Flere ganger i uken

Daglig

Hvor ofte bruker du PC/Mac?

Aldri

En gang i uken

Flere ganger i uken

Daglig

B First draft of patient questions

Rehab-Valgomat; Spm til Pasient/kliniker-skjema

Dersom du har behov for rehabilitering i *spesialisthelsetjenesten*, kan denne «valgomaten» hjelpe deg til å finne det tilbudet som passer best med dine behov.

Din alder:

Hva er viktig for deg:
Nærhet til hjemmet
Døgnopphold
Dagopphold
Poliklinikk
Individuell rehabilitering
Rehabilitering i gruppe
Tilbud om basseng
At rehabiliteringsstedet er tilrettelagt for bevegelseshemmede ute?
At rehabiliteringsstedet er tilrettelagt for bevegelseshemmede inne?
Er det viktig at du er sammen med andre med samme diagnose?
Er du selvhjulpen i daglige aktiviteter?

Behov for type opphold:
Informasjonsopphold
Vurderingsopphold
Rehabilitering
Rehabilitering etter raskt funksjonstap
Rehabilitering etter gradvis funksjonstap
Tilbud til voksne pårørende i tilknytning til rehabiliteringsoppholdet
Tilbud til barn pårørende i tilknytning til rehabiliteringsoppholdet

Hva er de to viktigste problemstillingen du ønsker rehabilitering for?
Arm-/håndfunksjon
Gange-/balansefunksjon
Spastisitet
Smerte
Fatigue
Blærefunksjon
Tarmfunksjon
Depresjon/angst
Kognitiv funksjon
Søvn
Tale/språk/svelg
Lungefunksjon
Seksualfunksjon

Arbeidsrettet rehabilitering
Behov for hjelpemidler
Daglige aktiviteter
Utforskning av nye former for fysisk aktivitet
Kost/ernæring
Stressmestring/"Å leve med MS"
Røykeslutt
Annet

Nevn 3 andre problemstillinger du har behov rehabilitering for?
Arm-/håndfunksjon
Gange-/balansefunksjon
Spastisitet
Smerte
Fatigue
Blærefunksjon
Tarmfunksjon
Depresjon/angst
Kognitiv funksjon
Søvn
Tale/språk/svelg
Lungefunksjon
Seksualfunksjon
Arbeidsrettet rehabilitering
Behov for hjelpemidler
Daglige aktiviteter
Utforskning av nye former for fysisk aktivitet
Kost/ernæring
Stressmestring/"Å leve med MS"
Røykeslutt
Annet

Har du flere rehabiliteringsbehov?
Arm-/håndfunksjon
Gange-/balansefunksjon
Spastisitet
Smerte
Fatigue
Blærefunksjon
Tarmfunksjon
Depresjon/angst
Kognitiv funksjon
Søvn
Tale/språk/svelg
Lungefunksjon
Seksualfunksjon
Arbeidsrettet rehabilitering
Behov for hjelpemidler
Daglige aktiviteter

Utforskning av nye former for fysisk aktivitet
Kost/ernæring
Stressmestring/"Å leve med MS"
Røykeslutt
Annet

Etter at du har klikket på send, vil du få ut et forslag til valg basert på dine svar. Dette kan tas med til din behandler, slik at dere sammen kan diskutere beste sted for deg. For informasjon om ventetid, anbefaler vi at du tar kontakt med det aktuelle rehabiliteringsstedet fordi ventetid varierer og avhenger blant annet av type opphold bosted i landet.

C Questions given to students during admin testing

Test av adminmodul

http://valgomat.herokuapp.com/

Alle verktøyene du trenger, kan du finne i menyen øverst på nettsiden.
Du kan finne flere sider ved å trykke på “Administrer nettsiden”

Bruk oversikten over behandlingsstedene til å svare på følgende spørsmål:

Ligger Åstveit Helsesenter i listen over behandlingssteder?
Har CatoSenteret Helsefagarbeider?
Hvor mange behandlingssteder tilbyr behandling for lungefunksjon?

Legg til og administrer spørsmål

Legg til et nytt spørsmål:
Spørsmålet skal være “Testspørsmål {klokkelsett}”. Eks: “Testspørsmål 1245” hvis klokken er 1245. ID-en skal være klokkeslettet (1245) og tilleggsinformasjon kan være hva du vil. Husk navnet på spørsmålet du la til! Du skal bruke det senere.

Administrer spørsmål:
Du skal nå legge inn spørsmålet ditt (fra forrige punkt) inn på pasientsiden. Spørsmålet skal legges inn i kategorien “Hva er viktig for deg?” som allerede eksisterer på nettsiden. Du trenger ikke lage ny kategori. Spørsmålet skal være et Slider spørsmål. Oppdater spørsmålslisten og bruk menyen på toppen for å se om spørsmålet ditt er kommet inn på pasientsiden.

Deretter skal du fjerne spørsmålet fra pasientsiden. Se om spørsmålet er fjernet.

Administrer koblinger:
Til slutt, vil jeg at du skal lage en kobling mellom ditt spørsmål og spørsmålet “Er du selvhjulpen til daglige aktiviteter?”.

D Questions given to treatment centers during center testing

Spørreundersøkelse til behandlingssteder

Angående beskrivelsen øverst på nettsiden

Ble hensikten med kartleggingen beskrevet på en god måte?

- ◯ Ja
- ◯ Nei

Var det tydelig hvordan du skulle fylle ut spørsmålene på nettsiden

- ◯ Ja
- ◯ Nei

Var det noe du savnet i introduksjonen?

Svaret ditt

NESTE

Send aldri passord via Google Skjemaer.

Spørreundersøkelse til behandlingssteder

Selve kartleggingen

Var spørsmålene lett å forstå/selvforklarende?

- ◯ Ja
- ◯ Nei

Var det noen spesifikke spørsmål/kategorier som var vanskelig å forstå?

Svaret ditt

Har du noen andre kommentarer til nettsiden i sin helhet?

Svaret ditt

TILBAKE SEND

Send aldri passord via Google Skjemaer.

E First treatment center form

KARTLEGGING AV REHABILITERINGSTILBUD I SPESIALISTHELSETJENESTEN FOR PERSONER MED MULTIPPEL SKLEROSE (MS)

Navn på institusjon/rehabiliteringsavdeling: **Nettadresse:** **Telefonnummer:**

Generelt om tilbud og innhold

Helseregion For private institusjoner: Hvilke(n) helseregion(er) er det gjort avtale med? Hvilke(n) annen/andre region(er) får dere pasienter fra?	Lag bokser her
Individuell rehabilitering?	Ja/nei
Rehabilitering i grupper?	Ja/nei
Vanlig varighet for opphold	
Tilbud om basseng?	Ja/nei
Tilrettelagt for rullestolbrukere?	Ja/nei
Beskrivelse av inntakskriterier: Selvhjulpen i daglige aktiviteter Alder Ja/nei (noen gir kun rehab under 35 år) Spesifiser for henholdsvis heldøgnsplasser og dagplasser	Ja/nei

Tilbud til personer med MS:

Inntak gruppevis for personer med MS	Ja/nei
Type opphold Informasjon Vurdering Rehabilitering etter raskt funksjonstap Rehabilitering etter gradvis funksjonstap Egne tilbud til voksne pårørende Egne tilbud til barn som pårørende	
Angi antall heldøgnsplasser og antall dagplasser, og angi hver for seg hvor mange av disse som er tiltenkt MS	Bokser for: - Antall heldøgn / antall av disse MS Antall dag/antall av disse
Vanlig ventetid for MS-rehabilitering	Uker
Faggruppene/profesjonene som er tilknyttet MS-rehabiliteringen (kryss av) ☐ Lege ☐ Sykepleier ☐ Hjelpepleier/ helsefagarbeider ☐ Klinisk psykolog	

☐ Nevropsykolog ☐ Sosionom ☐ Ergoterapeut ☐ Fysioterapeut ☐ Ernæringsfysiolog ☐ Uroterapeut ☐ Logoped ☐ Idrettspedagog/idrettsfysiolog el. lignende	

Problemstillinger dere har rehabiliteringstilbud for

Kryss av i boksene på listen under for de punktene der dere har et tilbud

Ved generelle behov for:	**Ja/nei**
• **Skole/utdanning/arbeidsrettet rehabilitering**	
• **Daglige funksjoner** o Vurdering av hjelpemidler o Vurdering av daglige aktiviteter o Samhandling med primærhelsetjenesten	
• **Livsstil** o Utforsking av muligheter for nye former for fysisk aktivitet o Kost/ernæringsveiledning o Stressmestring / «Å leve med MS» o Røykeslutt	

Tilbud ved problemstillinger knyttet til:	**Ja/nei**
Arm-/håndfunksjon	
Gangfunksjon • Utredning av indikasjon for /tilpasning av fotløftsystem (elektrisk nervestimulering) • Utprøving/tilpasning av ortoser • Utprøving av medikamentell behandling	
Spastisitet	
Smerte	
Fatigue	
Blærefunksjonen	
Tarmfunksjon	
Depresjon/angst	
Kognitiv funksjon	
Søvn	
Tale/språk/svelg	
Lungefunksjon	
Seksualfunksjon	

Generelle kommentarer/merknader/tilleggs-opplysninger:

Dato for utfylling:

F Statement from Norwegian Multiple Sclerosis Competence centre

Nasjonal Multippel Sklerose KOMPETANSETJENESTE

HELSE BERGEN

Haukeland universitetssjukehus, 11.04.2019

Uttalelse vedrørende arbeidet med en «Rehabiliteringsvalgomat»

Ved Nasjonal kompetansetjeneste for multippel sklerose (MS) har vi lenge vært opptatt av at pasienter med MS som trenger rehabilitering i spesialisthelsetjenesten, skal få rett rehabilitering på rett sted. Med et mangfold av problemstillinger som personer med MS kan møte og et mangfold av ulike rehabiliteringsinstitusjoner, er det behov for et digitalt verktøy som kan bidra til best mulig valg av rehabiliteringssted for den enkelte pasient.

Vi har lang og god erfaring med samarbeidet med Høgskolen på Vestlandet, Institutt for data- og realfag knyttet til IKT-løsninger i oppfølging av personer med MS. Vi var derfor svært glade for at masterstudent André Dyrstad med hovedveileder førsteamanuensis Adrian Rutle ønsket å utvikle et verktøy med en funksjonalitet som beskrevet over.

Vi har gjennom dette samarbeidet erfart at André har et sterkt engasjement og pågangsmot. Han er lydhør og åpen for innspill og har vist god evne til å tilpasse og justere løsninger i utvikling av en prototype, i tråd med fortløpende endringsforslag. Også i prosessen med utprøving blant pasienter og rehabiliteringssteder har han vært konstruktiv, nysgjerrig og forståelsesfull.

Rehabiliteringsvalgomaten er svar på et behov som er meldt til Helsedirektoratet, og vi har vært i kontakt med Direktoratet for e-helse for å informere om dette viktige arbeidet.

Vi har satt stor pris på samarbeidet med André og Adrian, og har opplevd André som en selvstendig og arbeidsom student. Vi har erfart betydningen av kombinasjonen IKT-kompetanse og helsefaglig kompetanse i utvikling av digitale løsninger i helsetjenesten. Dette mastergradsprosjektet er et svært godt eksempel på det.

Med Vennlig hilsen

Lars Bø
Professor dr. med/ leder for Nasjonal kompetansetjeneste for multippel sklerose
Nevrologisk avdeling
Haukeland universitetssjukehus

Nasjonalt multippel sklerose register
Nevrologisk avdeling – Haukeland Universitetssykehus – 5021 Bergen – Telefon: 55 97 55 03 – Telefaks: 55 97 59 01 – E-post: ms-senter@helse-bergen.no

G Sunnaas suggestions

Hei!
Veldig bra at dere jobber videre med dette.
Jeg var jo inne på forsøk med www.decidetreatment.org plattformen samvalgsløsningen fra Helse sørøst. Den er midlertidig på vent mens de store linjene ordnes opp i, så alle alternativ er interessante.

Synes dere har et veldig bra utgangspunkt.
Det jeg har av innspill har dere helt sikkert i tankene allerede:

1. Behov for å gjøre noe med språkbruk som kan være fremmed for mange pasienter – kanskje særlig nydiagnostiserte. Går an å teste på pasienter og justere ordlyden. Legge til en knapp med forklaring på hva det er, for hver kategori (f eks kognitiv rehabilitering).

2. Interessant hvor mye som er mulig å få til av dynamikk i valgomat– at utseendet forandrer seg basert på hva man svarer – at ting ommøbleres eller blir mer tydelig.

3. Litt opptatt av indre vekting mellom de ulike områdene, siden det er så mange. Hvis man f eks huker av Inkontinens sammen med 10 andre problemer drukner det lett hvis alt er vektet likt.Er det mulig å huke av max 2-3 valg som viktigst, så de får større betydning for hva som anbefales?

Følger gjerne med videre på dette.

H JSON example

Listing H.1: Translated JSON file sent during a GET request(with template)

```
{
  "questions": {
    "Category": [
      {
        "id": "Unique id",
        "label": "Question shown",
        "value": "ID to easily access special questions",
        "info": "Short description of the question",
        "displayAs": "How to display this question"
      }
    ],
    "What is important for you?": [
      {
        "id": 1,
        "label": "Close to home",
        "value": "closeToHome",
        "info": "You want treatment close to where you live",
        "displayAs": "slider"
      },
      {
        "id": 2,
        "label": "24 hour stay",
        "value": "24HourStay",
        "info": null,
        "displayAs": "slider"
      }
    ]
  }}
```

I Readme

Valgomat frontend

Link: https://github.com/AndreDyrstad/react-valgomat

This application is used to give patients the possibility to answer a few questions, and in return, get a recommendation on which treatment center they should pick.

Get started

To run the application, you need to have npm and node installed. You can install them by:

- installing them through their website
- running `sudo apt-get install nodejs` on linux
- running `brew install node` on mac.

To check if they are properly installed, type `node -v` and `npm -v`

Running the program

Now that you have npm up and running, you have to install all packages needed to run the application. This can be done by typing `npm install` in the terminal when you are in the root folder of the project. After all the packages are downloaded, simply type `npm start` to run the application. The application can be accessed in localhost:3000

If you want to change the API URI, simply change the return statement of the function found in global.js.

File overview

The components folder contains all JavaScript files used to make the website. The files are then split into small and large components, where the large components are full pages, and the small components are building blocks to make the large components.

The second main folder is the CSS folder. This folder contains (almost) all the CSS files used in the project. Finally, in our root folder, we have App.js and App.css which is the root of our application. Everything has to branch from these files.

Valgomat backend

Link: https://github.com/AndreDyrstad/api-valgomat

This application is used to give patients the possibility to answer a few questions, and in return, get a recommendation on which treatment center they should pick.

Get started

To run the application, you need to install Python 3. You can install it by:

- downloading it from their website
- typing brew install python3 on mac.
- typing `sudo apt install python3.X` on linux where X is version number.

To check if it works, type `python --version`. If you want to run the application in a virtual environment, type `pip install virtualenv` in the terminal to install a virtual environment library.

Running the program

First, you need to move to the root folder of the project. Then follow the path according to your operating system:

Windows

To run the application without a virtual environment, simply `type pip install -r requirements.py` to install the packages, then `python api.py` to run the application. If you want to run the application in a virtual environment, make a new environment by typing `virtualenv -p python venv`. To run the environment, type `source vent/Scripts/activate`. Finally, type `pip install -r requirements.txt` to install packages and `python api.py`to run the application. The API runs at localhost:8020 by default. To exit the environment, type `deactivate`.

Mac and linux

To run the application without a virtual environment, simply type `pip3 install -r requirements.py` to install the packages, then `python3 api.py` to run the application. If you want to run the application in a virtual environment, make a new environment by typing `virtualenv -p python3 venv`. To run the environment, type source `venv/bin/activate`. Finally, type `pip install -r requirements.txt` to install packages and `python api.py` to run the application.To exit the environment, type `deactivate`.

File overview

- config_files folder contains all our configuration files. These files have a strict setup and contain all the information needed to display questions to the frontend.
- storage folder contains a backup list of all questions.
- api.py file contains our API and everything needed to communicate with our frontend.
- database.py file has an overview of all our tables.
- sql_queries file contains all the SQL queries we use when we communicate with the database.
- rbs.py file contains our rule-based system and is used to recommend treatment centers.
- utilities.py contains a few converters and generator functions.