



---

# KEA – BACHELOR PROJECT

---

Grundtvigs Dyreklinik – Booking System



## **Abstract**

This Report is our final bachelor project in Software Development. The report aims at describing the process and development of the system/application Grundtvigs Dyreklinik. Grundtvigs Dyreklinik is a small clinic that wants a new booking system, so that the clients can book a time for their pets, and order products online. The application is going to be developed in the ASP.Net MVC framework with C#, and to help us develop the application, we are going to use a software development process, UML charts, database diagrams and different testing techniques.

The final purpose of the report is to continuously document, analyze and evaluate the process so that we can show how the system is intended to run and show which considerations we have made, regarding the different methods we have used. All this will then accumulate in a conclusion on what we have learned and reflect on what we could have done differently.

JANUARY 5, 2018

GROUP: ELVIS SARONJIC, BENYAM NEAMEN & MERT INCE

Supervisor: Jarl Tuxen

BA Software Development - Software Team 9 Fall

## Table of Contents

1. Introduction & Problem .....	3
1.2 The Organizational Form.....	3
1.3 Business Platform.....	3
1.1 Background and motivation.....	4
1.4 Problem Definition .....	5
1.5 Problem Formulation .....	5
1.6 Project Scope .....	6
2. Development Methodology .....	7
2.1 Project Planning and Scheduling.....	8
3. FURPS .....	9
3.1 User Stories .....	11
4. Analysis & Choice of Technologies.....	14
4.1 Risk Analysis .....	14
4.2 Use Case Diagram .....	15
4.3 Task planning and controlling .....	16
4.4 Tools and Development Environment .....	17
4.4.1 Visual Studio.....	17
4.4.2 Visual Studio Team Services (VSTS) .....	18
4.4.3 Trello .....	18
4.4.4 Apache JMeter .....	18
4.4.5 HP Unified Functional Testing (UFT) .....	18
4.4.6 Azure .....	19
5. Design.....	20
5.1 Design Class Diagram .....	20
5.2 Software Architecture.....	21
5.2.1 ASP.NET MVC .....	22
5.3 Database .....	23
5.3.1 Conceptual Design .....	23
5.3.2 Logical Design.....	24
5.3.3. Physical Design.....	25
6. SCRUM .....	26
Sprint 1 .....	27
Sprint 2.....	28

Sprint 3 .....	29
Sprint 4 .....	30
7. Implementation .....	31
7.1 Database and Entity Framework.....	31
7.1.1 Relations between tables.....	32
7.1.2 SQL and ORM .....	33
7.1.3 Trigger .....	34
7.2 Security .....	35
7.2.1 Identity management .....	35
7.3 Integration .....	39
7.3.1 Facebook.....	40
7.3.2 Booking with Full Calendar .....	42
7.4 Version Control .....	44
8. Test.....	45
8.1 Test Cases.....	45
8.2 Traceability Matrix .....	46
8.3 Usability Testing.....	46
8.4 Unit Test.....	49
8.5 Automated Functional Testing.....	51
8.6 Deploying on test environment (Azure) .....	53
8.7 Performance test .....	54
8.7.1 Test results.....	55
8.8 Acceptance Testing .....	56
9. Handing over the System .....	57
10. Conclusion.....	58
11. Reflection and Perspectivation .....	59
11.1 Scrum vs Waterfall .....	59
11.2 Usability Testing.....	59
11.3 Version Control .....	60
11.4 Test Driven Development .....	60
11.5 Risk Analysis .....	60
11.6 Improvements.....	61
11.7 Final Remarks .....	61
12. Bibliography .....	62

## 1. Introduction & Problem *(MI & ES)*

As our bachelor project we have found an animal clinic that need a new system. We got this opportunity through one of the group's members connection, who owns Grundtvig's Dyreklinik.

Grundtvig's Dyreklinik is located in Copenhagen, Denmark and was established in 1998, however, first as a visiting veterinarian, but later got more rooms and more employees.

Grundtvig's Dyreklinik today consists of a senior veterinarian, who is also the owner of the clinic. The clinic offers a different variety of treatments for domestic animals, such as dental treatment or laser therapy.

The main purpose for Grundtvigs Dyreklinik is to make their customers happy and have a glad and well-motivated staff. One of their visions for the future is to expand the clinic in Copenhagen and hire more employees.

We have taken contact to this small clinic and talked about their visions, which made us more interested in the company and its needs. Therefor we have accepted to create a new booking system that can help the company gain more recognition in the vet branch.

The system/application is going to be developed in ASP.NET MVC framework with C#, and to help ourselves with developing the system, we will use a software development methodology, UML-diagrams and tools. We will throughout the project document, evaluate and analyse the development process.

### 1.2 The Organizational Form *(ES)*

Grundtvigs Dyreklinik was established in 1998. In the beginning it functioned as an out-of-town visiting veterinarian, which means that the veterinarian went to the homes of the clients to treat the dog. Later on, they got more employees and therefor had the need to expanded to their own premises/rooms.

Today Grundtvigs Dyreklinik has offices in Copenhagen and consists of a superior vet, who is also the owner of the clinic, two other veterinarians and a nurse. Due to the low number of employees, the clinic is under the basic organizational form, characterized by the fact that there is only one decision maker. Thus, the structure of the organization is organic. The organization is also referred to as a flat organization and the communication itself in the clinic is in line with the principle, which in this case does not create the classic disadvantages of long communication routes as the clinic has a small size.

### 1.3 Business Platform *(ES)*

Grundtvigs Dyreklinik earns money by healing sick animals and selling products like pet food and accessories. Their mission is to cure and provide a correct dietary guidance to these animals, thereby creating customer satisfaction. The concept at Grundtvigs Dyreklinik are "quality", that is something they are very focused on. By maintaining the high quality both in the procedure of treating animals and the customer service, the chances of customers returning gets higher which means the clinic's increase their turnover. So Grundtvigs Dyreklinik places great emphasis and time on having satisfied customers as satisfied customers will return. The goal for the clinic is to get less unsatisfied customers, in the last period the clinic has had some complaints about their system. Improving the system will lead to increased turnover, so they can afford to expand and get a new clinic somewhere else in Copenhagen.

## 1.1 Background and motivation *(MI & ES)*

The motivation and background for why Grundtvigs Dyreklinik wants to develop a new system and why we have decided to help them will be discussed in this segment.

The overall reason why we are going to be a part of this project, is both because it is a great opportunity to use as a bachelor project, where we can demonstrate what we have learned throughout our education at KEA. But also, because we will have the entire responsibility in developing this new web-application for an animal clinic, where we will be part of it from the beginning, to: - finding the requirements, designing the system and building the web-application is very exciting to be part of. And lastly, we know the wife's husband, which is the reason why we got this opportunity/chance and the reason why we want to help them increase the number of customers and give them a better reputation.

Grundtvigs Dyreklinik has in the past period experienced a potential loss of customers and customers who are dissatisfied with the system. Here they have expressed/complained that the system is too slow and that the flow is not intuitive. Some of the reason why these problems have occurred is because when the clinic bought the old web-application, they were never really part of the process of devolving the system. And the people behind the current web-application were not establish professional developers but more sort of shady contractors (*what I mean here is people that usually use a template and make home site for couple of thousand kroners*). That never really documented or tested the system for anything like; performance, the flow of the interface or even had it tester by users before deploying the web-application.

This has also resulted in Grundtvigs Dyreklinik missing some components/features in their current system, that could had help them make even more profit. Grundtvigs Dyreklinik sells products where they sell medicine for the pets at their clinic. But this functionality has not been implemented in their current application. We will therefore design the system in a way that will enable/make it possible for a product catalog to be implemented. This will solve the need for their customers who want to buy different veterinary products online

Another problem they have is the long process users have if they want to register/create a profile for their current system. Which occurs because the user must wait for an approval of their registration before they can use the system. And this approval can only be granted by a few employees in the clinic. So, the reason why this is a problem is because If the waiting time for the approvment is too long, the user can therefore get impatient and be a big factor for them to find another vet.

We have decided that will try to overcome Grundtvigs Dyrekliniks problems by making them a system with focus on the different issues they have.

## 1.4 Problem Definition *(All)*

At Grundtvigs Dyreklinik they are concerned with their old system, which has some issues regarding, booking time for your pet and when registering a new user. The clinic has made their concern clear, that they're losing/have unsatisfied patients due to:

- Slow process when booking a time for your pet
- Slow process (has to wait for acceptance) when creating a new user
- The flow isn't intuitive

We had a meeting with Grundtvigs Dyreklinik about the problems they had, and what we could do to solve them. We will solve these problems/issues by making a new system that makes it easier for patients to book a time for their pets. Where they'll have the option to pick a date and time from a live calendar, which the veterinarian is available. We will improve the overall flow of the system and also the process when creating a new user, so that you don't have to wait for an acceptance from the clinic.

## 1.5 Problem Formulation *(All)*

We are approaching the end of our education and in our last project we want to demonstrate our skills, by using the theory and practice that we have acquired during our education and apply this to develop a system:

How to develop a booking system for Grundtvigs Dyreklinik?

- How to manage the project?
- How to implement and manage the database?
- How to ensure the quality of the system?

## 1.6 Project Scope *(All)*

The purpose of this project is not to develop an end-product, but rather to build the core functionality, so that Grundtvigs Dyreklinik can in the future, further develop on it.

We have therefore talked with our client where we discussed what their minimum expectation from us is. And the conclusions we reached was that they expected us to have a functional prototype of a booking system for the employees and users *(which we will get into in this segment)*.

We also decided that our final deadline for making the prototype that Grundtvigs Dyreklinik expect us to deliver, is January 5<sup>th</sup>.

The plan is to implement more functionality if we're done with the core implementation before the deadline. But taking the time limitation into consideration we have together with Grundtvigs Dyreklinik made a prioritization of the different functionalities.

The minimal implementation they expect, is that we deliver:

- Register and login to the system
- Create, delete, update treatments in the system
- Create, delete, update species, races and the user's pets in the system
- Book a treatment for the pet
- Register others in the employee role

We will move onto some of the other functionalities once the firstly prioritized implementations are done. Those afterwards will have a lower business value for the project and the system:

- Search for a specific user in the system
- Product catalog in the system for the customers to buy medical items for their pets. The employee will have to add products in the catalog and set prices
- The employee being able to delete or update a user in the system.
- Login using your social media account (Facebook or Google)

So, in our first draft we have found that these are all the functions that Grundtvigs Dyreklinik wants. And as it is now at this stage, we don't know yet if we will be able to implement it all. We will therefore together with Grundtvigs Dyreklinik make a more detailed prioritization of the functionalities, when making them into user stories *(Because in this stage, we will have estimated how long is going to take to finish implementing the functionalities)*.

## 2. Development Methodology *(MI)*

When developing a project, there is a lot of frameworks to use, that can help you manage and finish the project. We will in this chapter discuss which project management framework/methodology suits our project. We are going to analyze our options and chose the framework that would benefit us the most.

**Prince2** is a structured project management method where you divide the project into manageable and controllable stages. Prince2 would have been very complex for us to use in this project. The reason we didn't chose Prince2 was both because of our short deadline and that it would take more time for us to get familiar with this framework as we do not have much experience with it.

**Unified Process** is an iterative and incremental software development framework. With UP the project is divided into four phases; Inception, Elaboration, Construction and Transition. Where you furthermore divide the phases into iterations. We didn't feel comfortable with using UP, mostly because of the many artifacts you have to make in UP. At this point we are looking for a simpler framework.

**Waterfall** is one of the first software development models that has ever been used in a project. The Waterfall model is split into six phases; Requirement Gathering, System Design, Implementation, Integration and Testing, Deployment and Maintenance. This model is a linear approach when it comes to software development and it isn't easy to get back to any phase in the model once it is passed, which is one of the reasons we didn't chose to use this model. Another reason is because our customer is inexperienced in software development and therefore we expect that there will come changes throughout the project.

**Scrum** is an iterative software development model that is rather simple compared to e.g. Prince2. We have chosen to use this methodology for our project, as we all have good experience with Scrum and how to apply it. Scrum is flexible to use and we will get a good overview of the project if we use the different ceremonies, such as daily scrum meetings and sprint retrospective, that will help us improve the project progress. *(More detailed use and explanation of this method/framework can be found in the chapter: "Scrum")*



## 2.1 Project Planning and Scheduling (MI)

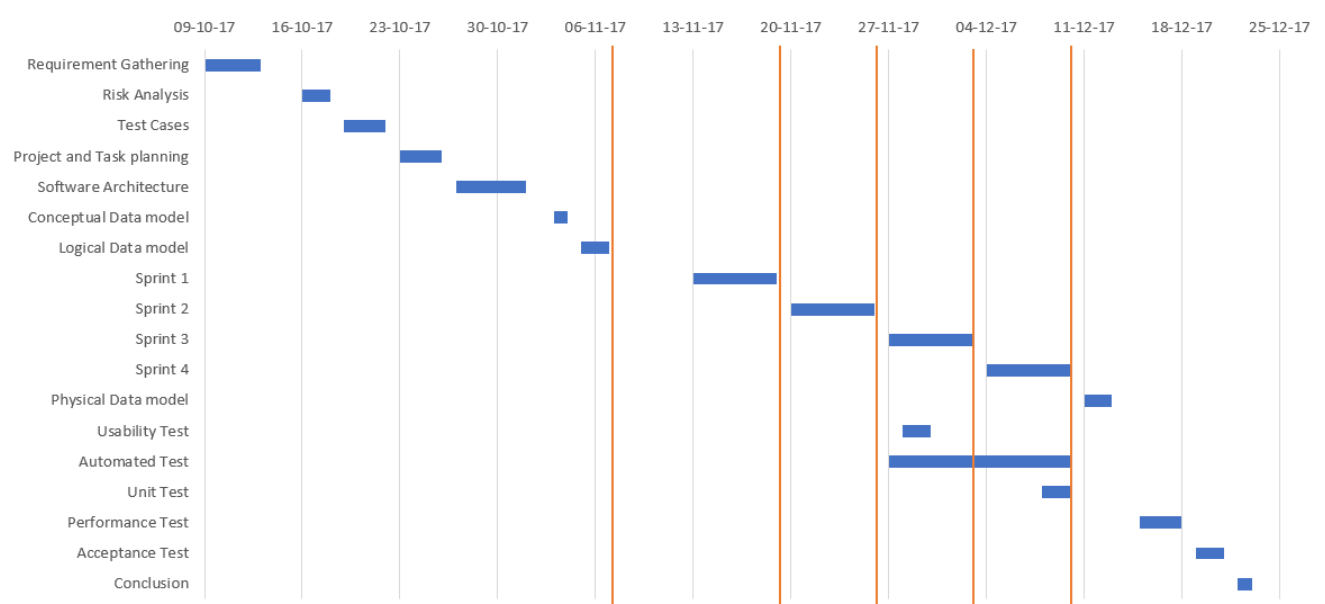
When starting on a project, there can often occur problems that can delay the delivery of the product. There may be a lot of reasons for this, but most often it is because you have not set a long enough deadline or problems that you didn't take into account occur, etc. By spending time to plan your project thoroughly, and by using a software project planning tool, you could avoid/minimize the chance that these things could happen. Software project planning is a method used to create a better overall overview in the early stages of project planning.

To do this, we have used something called Gantt Chart, which is a type of bar chart that illustrates the project's schedule, thus creating an overall overview. Gantt Chart illustrates the start and end dates, for one's tasks in the project. Overall it shows a summary of the project, it also shows which tasks are related to each other and which tasks are to be done first. If the project has been properly planned, this form will become a kind of roadmap that defines which tasks and milestones are to be monitored and controlled as the project progresses.

We have divided the planning into two different parts, first of all we have the project planning, where we will make a Gantt chart to plan the whole project. The Gantt chart will help us plan the different milestones that is going to take place in the project. Then we have the product task planning that will help us implement the tasks.

We have created a Gantt chart using the Smartsheet application, which is a project planning tool for creating such charts for overview. We have made some milestones for the whole project, starting with the requirement gathering which is the first thing to be done in a project, we expect that it'll take a couple of days, till we are sure that we got all the requirements. Then we will make a risk analysis based on our knowledge and the requirements we have gathered.

Some of the milestones can take longer than expected, because of the complexity of it, but we have calculated the amount of work it should take to finish the milestone. Our longest milestone in this Gantt chart is the beginning phase, that will take approximately a month. We will in the later chapters explain which activities and implementations we're doing in each sprint.



Here you can see our Gantt chart and the different milestones

In the product task planning we will plan the fulfilment of the different tasks. For that we will use the Kanban board from Trello to plan and keep track of the tasks.

After talking with Grundtvigs Dyreklinik they have agreed that it is maybe unrealistic for us to be able to develop the whole application, when considering the short deadline for the bachelor.

The task planning will be in the analysis chapter, where you will see the tasks and the status of the implementation.

### 3. FURPS *(MI & ES)*

In this project, we wanted to use FURPS to find the specifications/requirements for the application of Grundtvigs Dyreklinik. We chose to use FURPS because it is a helpful tool, which allow us to categorize the way we find requirements. By using FURPS as a checklist, we can reduce the risk of not considering some important fact of/in the system.

#### **Functionality:**

First of, the application Grundtvigs Dyreklinik is going to have three different types of actors; The Employee that is going to act like the administrator, and the Customer who is either going to be the active end-user or the passive end-user of the system. Each role is going to have access to different parts of the system and its features.

Some of the application's functionality will only be available if you register to the system and get an account. If you do not have an account, you will have the option to make one. To get an account, a user and employee will have to go through the registration process, where they will have the option to register to the system, either through the application's own system/functionality or through an API, that uses the Facebook or Google account.

To log into the system, they will have the option of either doing it with their username/email and password or through Facebook and Google login, where they will be redirected.

The features an **employee** is going to have as the **administrator** are:

1. Create, update and delete functionality to both treatments and products, that way the normal user can see which treatments their pets can get and which product they can buy from the clinic.
2. The employee will also be able to add other employees to admin-roles.
3. The employee should be able to delete users that is either violating the system laws/agreements or if the user has stop using the system. He or she should also be able to update the user account, in case the user needs help changing some information.
4. To make the work and flow for the employee easier and faster he or she should be able to search for a specific user.

The features a **customer** who has registered in the system and become an **active end-user** are:

1. Associate pets in his *(the users)* name by register/create them in the system. By doing that he can thereafter see all the pets, he has registered and if necessary delete or edit the pets that is associated with him the user.
2. Book an appointment to treat his *(the users)* pets for disease or other treatment. The user should also be able to see all the appointment/bookings he has made and if necessary cancel them.
3. A user should also be able to see the product the clinic is selling and be able to buy and order them.

**Usability:**

We are planning to make the design clean and simple and to achieve that, we are going to use Mandel's golden rules. Mandel's golden rules is about 1. Place the user in control, 2. Reduce user's memory load, 3. Make the interface consistent. By doing this, the user is going to feel that he is in control when he is working with the system. In addition, we intend to make understandable error messages, so that the user can get help, if he should fail action.

Our GUI will be a little different, based on who is using it (user/admin) because the admin will have a few more possibilities and views. The GUI of the system will be designed; having in mind that it is going to be a web-based application and the interface is going to be in Danish.

The idea with making a simple GUI is, that we don't have any designers in the team who can make a colorful and design stylish GUI. Therefore we are not going to focus much on the design aspects, but on making an easy and simple to use flow.

After the project the company will be able to hire a designer to make a better GUI, by using their knowledge in design to create the different wireframes and applying them.

**Reliability:**

The system should be stable, and should be able to handle smaller errors on runtime. If there happens to be errors the system should throw an exception/pop-up telling the user what the problem is. It's important that the system is reliable, and doesn't crash. To make sure that doesn't happen we will check our logs on the server to find out where the limit of the system is.

If the system crashes for any reasons, it should be restarted (automatically) as fast as possible, and the system should be up and running again.

**Performance:**

The system should have a normal response time, where the user is not kept waiting too long. We have in mind that there will be several users (*up to 150 users*) using the system at the same time, and it shouldn't affect the performance of the system.

Based on some researches the response time for the user on the system should be around 0.1sec<sup>1</sup>, and the biggest delay shouldn't be more than 10sec. The recovery time from the backup system should be approximately 30min.

**Supportability:**

The system should be supported by the three most used web-browsers, which is Google Chrome, Firefox and Edge. We will only test our system on the newer version of these browsers. The version of Chrome that the system will be supported on is from version 59.0.3071 to the newest version 61.0.3163.

The system will also be supported by the newest version of Firefox 47.0.2 and the latest version of Edge (version 40.15063.0.0).

---

<sup>1</sup> <https://www.nngroup.com/articles/website-response-times/>

### 3.1 User Stories (ES)

Finding out what to code and how you are going to allocate the product function/requirement of the system is a very crucial part in development. Therefore, we need a tool, a process; that is fast, easy to manage and most important having experience in using it. For us it was User stories.

User Stories are short simple descriptions of a feature told from the perspective of the person who wants the new capacity/requirements. User Stories are often written on cards or sticky notes, and arranged up on walls or tables to facilitate planning and discussion. When doing user stories, it is important to switch focus from writing about the functions, to discussing them. In fact, these discussions/reflections are much more important than what is written in the text.<sup>2</sup> It is also important to prioritize and estimate the User Stories, and the more experience you have, the more accurate are the estimations.

We have prioritized our User Stories: **1** priority, **2** priority, **3** priority, because that way we are sure to have implemented the most important/critical parts of the system first. We have also put our user stories in Time Boxes 1, 2, 4, 8, 12 hours, etc.

#### Employee:

**US 01** - As an employee, I want to be able to create/delete/update treatments to the system, so that the user/customer can chose which treatment their pets need (*Total 8hr*). Priority 1

- Task 1 - Make CRUD (*Create, Read, Update and Delete*) functionality to treatments (8hr).

**US 12** - As an employee, I want to be able to create/delete/update species to the system, so that the user/customer can chose which kind of species their pet is (*Total 8hr*). Priority 1

- Task 1 - Make CRUD (*Create, Read, Update and Delete*) functionality to species (8hr).

**US 13** - As an employee, I want to be able to create/delete/update animal race to the system, so that the user/customer can chose which kind of race their pet is (*Total 8hr*). Priority 1

- Task 1 - Make CRUD (*Create, Read, Update and Delete*) functionality to race (8hr).

**US 02** - As an employee, I want to be able to search for a specific user/customer, in order to find them and make the process (*of different features*) faster and easier for the user and employee. (*Total 8hr*). Priority 3

- Task 1 - Make a search bar where it finds user/customer via name (8hr).

---

<sup>2</sup> <https://www.mountaingoatsoftware.com/agile/user-stories>

**US 03** - As an employee, I want to be able to create/delete/update products to the system, so that the user/customer can buy products to their pets (*Total 8hr*). Priority 3

- Task 1 - Make CRUD (*Create, Read, Update and Delete*) functionality to product (*8hr*).

**US 04** - As an employee, I want to have the right to be able to delete or update a user that is in the system, that way I can either force a user to not use the system anymore or help change some information on his account. (*Total 12hr*). Priority 3

- Task 1 - Make delete functionality to remove a user (*6hr*).
- Task 2 - Make changes to the users account (*6hr*).

**US 05** - As an employee, I want to be able to register others in employee roles, so that they can administrate the system. (*Total 16hr*). Priority 2

- Task 1 - Expand existing user account (*4hr*).
- Task 2 - Upload and modify the employee's profile picture (*4hr*).
- Task 3 - Give the employee administrative right (*8hr*).

#### **User/Customer:**

**US 06** - As a user/customer, I want to be able to create/delete/update my pets into the system and in my account, so that they can get treatments (*Total 16hr*). Priority 1

- Task 1 - Make CRUD (*Create, Read, Update and Delete*) functionality to pets (*6hr*).
- Task 2 - Assign pets to the users account (*6hr*).
- Task 3 - Make a page, where the user/customer can see all the pets (*4hr*).

**US 07** - As a user/customer, I want to be able to register myself, in order to use the system. (*Total 8hr*). Priority 1

- Task 1 - Expand existing user account (*4hr*).
- Task 2 - Upload and modify the user's profile picture (*4hr*).

**US 08** – As a user/customer, I want to be able to book a treatment for my pets in the system, in order to help/cure the pets. *(Total 30 hr)*. Priority 1

- Task 1 - Make CRUD (*Create, Read, Update and Delete*) functionality to booking (6hr).
- Task 2 - Assign the booking to the users account (4hr).
- Task 3 - Assign the booking to the pet (4hr).
- Task 4 - Create a schedule (16hr)

**US 09** - As a user/customer, I want to be able to buy/order products from the system, that my pets need  
*(Total 16hr)*. Priority 3

- Task 1 - Make a page, where the user/customer can see all the products (8hr).
- Task 2 - Buy/Order products from the system (8hr).

**All:**

**US 10** - As a user/customer and employee, I want to be able to Login/logoff, in order to use the system *(Total 4hr)*. Priority 1

- Task 1 - Create login functionality (2hr).
- Task 2 - Create logoff functionality (2hr).

**US 11** - As a user/customer and employee, I want to be able to login/register by using my Facebook or Google account, in order to use the system *(Total 8hr)*. Priority 2

- Task 1 - Login functionality by using google (4hr).
- Task 2 - Login functionality by using Facebook (4hr).

**US 14** - As a user/customer and employee, when registered through Facebook I want to retrieve my information from Facebook *(Total 14hr)*. Priority 3

- Task 1 – Retrieve information from Facebook (14hr).

## 4. Analysis & Choice of Technologies

In this chapter we will use various diagrams and tools to make a deeper analyzation of the requirements. We will use the techniques and diagrams we have learned during our education in KEA.

We are also going to discuss the different tools and technologies that we are going to use in the project, and talk about why we have chosen them.

### 4.1 Risk Analysis *(MI & BN)*

The risk analysis is based on a mixture of our experience and the meetings we had with our customer Grundtvigs Dyreklinik. The meetings we had with Grundtvigs Dyreklinik gave us an idea about what kind of problems could occur and how we would manage it.

We have chosen to use the proactive approach when dealing with the risk that could occur in our project. We started by identify all potential risks in the project by making a Risk Table, this way we found out which risk is the most critical and what kind of risk management we could apply.

We have chosen to use the numbers between one to five to indicate the impact and probability of a risk in our project to calculate the risk exposure/risk factor. The risk exposure helped us to prioritize our risk so that those with highest value got first priority. We found the risk exposure (RF) by multiplying the probability of the risk that could occur with the impact it has on our project. (RF = Probability \* Impact).

Risk Identification		Risk Evaluation			Risk Management
Risk Name	Description	Probability	Impact	Risk Factor	Mitigation
R1	Unable to complete all requirements	5	5	25	Weekly status meeting and start with the requirements that are significant. Having a sprint review after each sprint
R2	Wrong estimation of user stories	5	4	20	Using project management estimation like comparing prior user stories
R3	The project being too big and resulting in not being able to delivered within deadline	3	5	15	Assesment using the burndownchart and using tools like trello.
R4	The Application performance	4	4	16	Have a daily scrum meeting, if necessary call for a meeting with Grundtvigs Dyreklinik
R5	Slow response time on application	4	3	12	Stress and load balance test on the system
R6	Database size/performance problems	3	4	12	Azure service and target regions
R7	Unauthorized alteration of data	2	5	10	Mapping of the database should go through the three normalization stages and do calculation of the size beforehand
R8	Disease or personal problems resulting in not being able to meet up	5	2	10	Secure the database so that only authenticated have access, make role based authentication for the application
R9	Security against sql injections	1	5	5	This thing no one can foresee but we should all have a good understand what a group member is doing so that other can take over while there is a problem
R10	The groups ability to work together	1	5	5	Make stored procedure
R11	Losing track of the development process	1	4	4	the probability is low but since the impact is high we have chosen to vote on everything and we should always priotate the project
R12	Missing experience/knowledge with development tool	3	1	3	Weekly review, scrum meetings etc.
R13	Requirement change	1	2	2	All for one, one for all. The others should help on spare time to explain
					This shouldn't happen and any alteration should be done on the consideration of the project time

*Red line tells where our cut off line is where everything above this line must be manage (Zoom in)*

As it can be seen on the risk table our most concerned risk for this project is about not being able to meet deadlines or unable to implement all requirements. This is firstly based on our time limitation, that we can do nothing with, and having in mind that the project can be bigger than expected. The second most concern risk was that of the performance and maintaining of the application, we thought about those risks because it can sometimes be hard to keep the performance at the same level due to limited resources. To prevent those risk, we made some tests and thought about how to structure our application in general.

In general, the risk analysis table have been helpful to get an overview of the risks so that we could find solutions to prevent the damages they could have caused for the whole project.

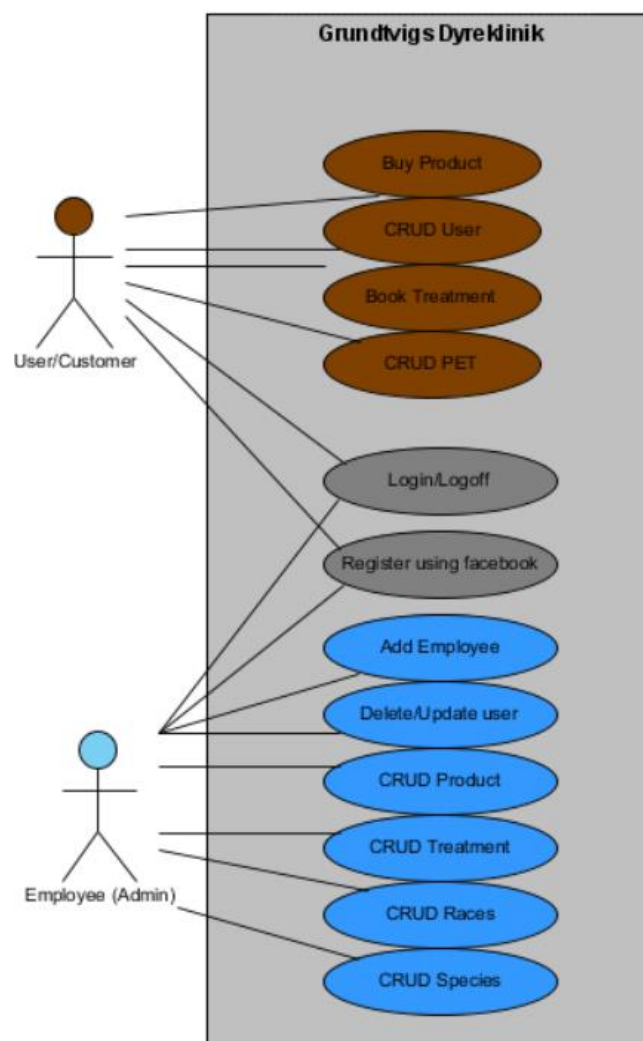
## 4.2 Use Case Diagram (MI & ES)

A use case diagram is a way to show the users interaction with the system. It identifies the different types of users and their interactions with the functions (user stories). We have not used use cases but instead made user stories, mainly because we choose to use scrum to manage our project.

So why make a use case diagram when you don't have any use cases. It was important for us to give an overview on which user will interact with which functions. This means that a simple visualization is very useful for e.g. stakeholders to see the actors and the function/requirements.

The use case diagram contains three different factors, the first thing to be sure of when making a use case diagram is, who the actors are. Then you list the functionalities/requirements represented as use cases (*in our case user stories*). And the last visualization is the relationship between the actors/users and the functionalities.

We have used different colors to make it easier for people to see the relations between the user stories and the actors. We have used brown for the user/customer user stories, blue for the employee user stories and grey for the user stories that are related to both actors.





### 4.3 Task planning and controlling (MI)

To be able to keep track of our tasks and know the overall status, we have decided to use the Kanban board from Trello. Trello is an online tool where you can easily create virtual boards, that you can use for different purposes.

We chose to use Trello because of our previous experiences, from the first semesters, where it turned out to be a great tool to handle your tasks.

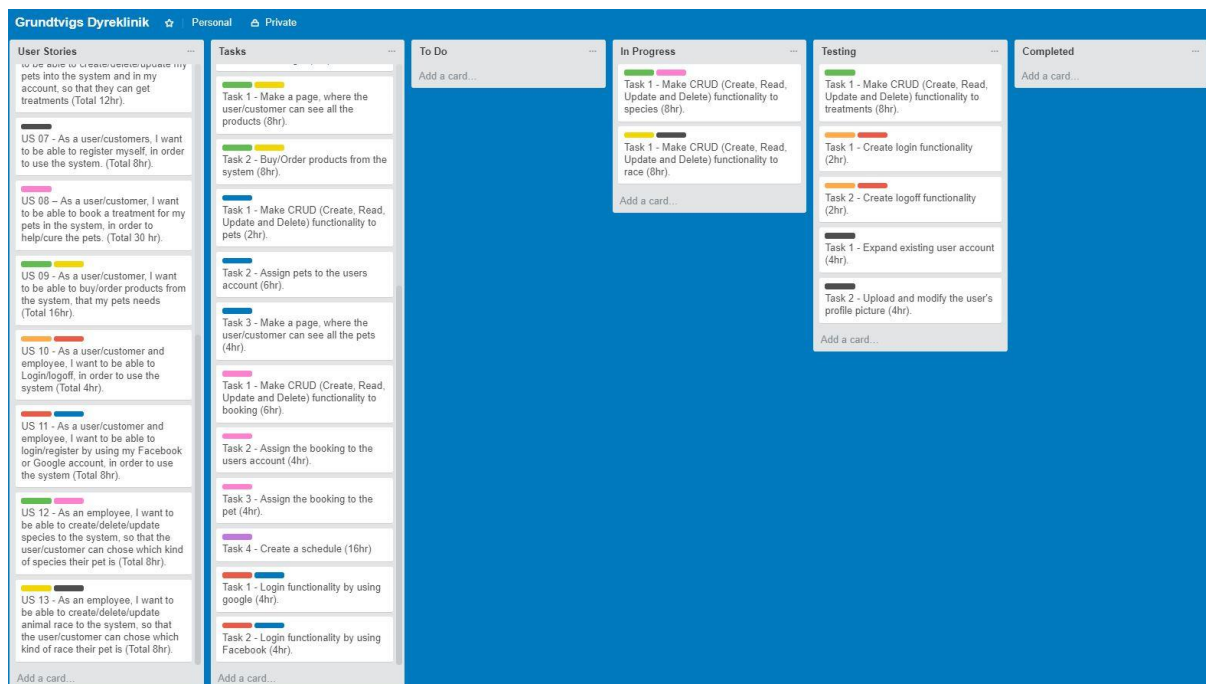
The Kanban board gives us the opportunity to divide the board, so that we'll all know what the implementation status is for the single task. We will divide it in groups of; User Stories, Tasks, To Do, In Progress, Testing and Completed.

We have chosen to make an extra list for testing, which will be all the tasks, that is being tested before being moved to the completed list/column.

One of the very good things for us with using Trello was that you are able to put labels with colour, so that you can categories the different tasks. Our idea with doing that was to be able to see very fast which task belonged to which user story, without having to read it all.

After creating the user stories and tasks from the requirements we received from Grundtvigs Dyreklinik, we wrote it all on the Kanban board on Trello.

As we continued the development of the different tasks, we moved it to the lists, depending on their different statuses.



Zoom in to see Trello Kanban board (To see our progress in each sprint click on the link: <https://goo.gl/2HAoP9>)

## 4.4 Tools and Development Environment *(BN)*

Before any project start the most important thing to have in mind, is which tools to use. A wrong tool can make the development of project take longer and in some cases even not being able to complete the project. For that reason, we were very keen about the tools we wanted to choose to develop this project.

The tools we were going to use had to cover the; development, management and maintenance.

We chose our tools based on these three basic questions; Time, Update and Stability. How could the tools facilitate our work so we could save time? Are the tools up to date in the current market? And how stable is the tools so that we won't have implication while we are developing the project? In the segments below, we will describe which tools we have found, that could be essential for completing our project.

### 4.4.1 Visual Studio

We arrived to a conclusion that for the development environment we would chose a Microsoft Integrated Development Environment (IDE) called Visual Studio. Besides the obvious reason that over three million developers use this tool all over the world to develop computer programs, websites, web services, mobile apps and also the fact that it's the most used development tool in Denmark. For us to follow the big crowd wasn't enough to choose a tool, we looked on those three essential questions. Here is some of the obstructions that we think visual studio can help us overcome.

Visual Studio is a robust code editor that supports syntax highlighting and code completion. Using IntelliSense will help us make coding easier, it helps while coding with suggestions and auto filling the code if a developer can't remember the right syntax. The other time reducing is that visual studio has a background compiler this feature is a great feature because it will give feedback about syntax and compilation errors while writing the code.

Visual Studios Design have many features from web designer to database mapping, those designer's features are the ones that got most of our attention. It was the fact that it has made some of the developing work much shorter and less redundant coding. It implements a Asp.Net framework and the basic startup project with the necessary coding already implemented so that a developer could build up and change to fit the desired project. Once upon a time the developers would use many hours to implement simple things like connecting to a database and do the CRUD operation with a lot of redundant coding but with this tool the developer can use the time to do the more important things like configuring the business logic.

#### 4.4.2 Visual Studio Team Services (VSTS)

This is a Microsoft server that provide code, project and test management. In all the time we have been on KEA education we have been using different tools to do those management, especially when it comes to version control we have used GIT which in some time gave us a headache because it wasn't integrated with our development tools. We thought that this time we would try new tools that could combine all the management in one. Since this is a new tool and we needed to start the project management beforehand we decided to use Trello instead for project management and use VSTS for code and test management.

VSTS is integrated with Visual Studio which would make our version control tool in one place with our development tool.

We believe that it is a must to have version control for this kind of project because we are different individual persons with different tasks to implement. Version control will help us to track and control the output of the project and if something goes wrong then we can always roll back to last working version. We will use the VSTS to host our source code to an online repository and each team member can download the code to his local repository so that we can all work simultaneously on the project.

#### 4.4.3 Trello

We have chosen Trello for our task management, Trello is a collaboration tool that will organizes our tasks into boards. This way we can see what's being worked on, who's working on what, and where we are in our project process. We will use the board as a task planning and controlling tool in SCRUM but this will also serve us as an overview of the whole project.

#### 4.4.4 Apache JMeter

We will in this project use Apache JMeter, an open source performance testing tool, to make our performance tests, such as load, spike and stress test. The reason we chose Apache JMeter is that we from a previous semester had experience with it, and found out that the tool was actually pretty easy to use with a lot of documentation on their webpage.

#### 4.4.5 HP Unified Functional Testing (UFT)

This tool is an automated functional testing software, that will make it easier for us to test the application/system as fast as possible instead of testing it manually. We can with this tool control the execution of the different tests and compare the outcome to our expected outcomes. This automated test is done on the functions of the system.

#### 4.4.6 Azure

Azure is a Microsoft cloud solution for building, testing, deploying, and managing applications and services with a global network of Microsoft-managed data centers, we have chosen to use Azure as a testing environment for hosting the application on the web. By having Azure as a hosting environment, we would like to get some real information on how the application will behave once it's hosted on a server for production. We would like to test the response time and how the application behave on different web browsers, the other thing we want to get, is the log files for the application. Azure have an inbuilt server for log that include:

- Detailed Error Logging - Information of HTTP status failure (status code 400 or greater)  
We will use the information to know why the server return error so that we can handle them.
- Failed Request Tracing - Isolate what is causing a specific HTTP error to be returned with this information we would like to know what or how we can increase site performance.
- Web Server Logging - This information will be useful to know how many requests are from a specific IP address.

With all this information we think we will get an overview of how the application behaves so that we can chose the right host environment when it comes to size, performance and that we can test different clients web browser.

## 5. Design

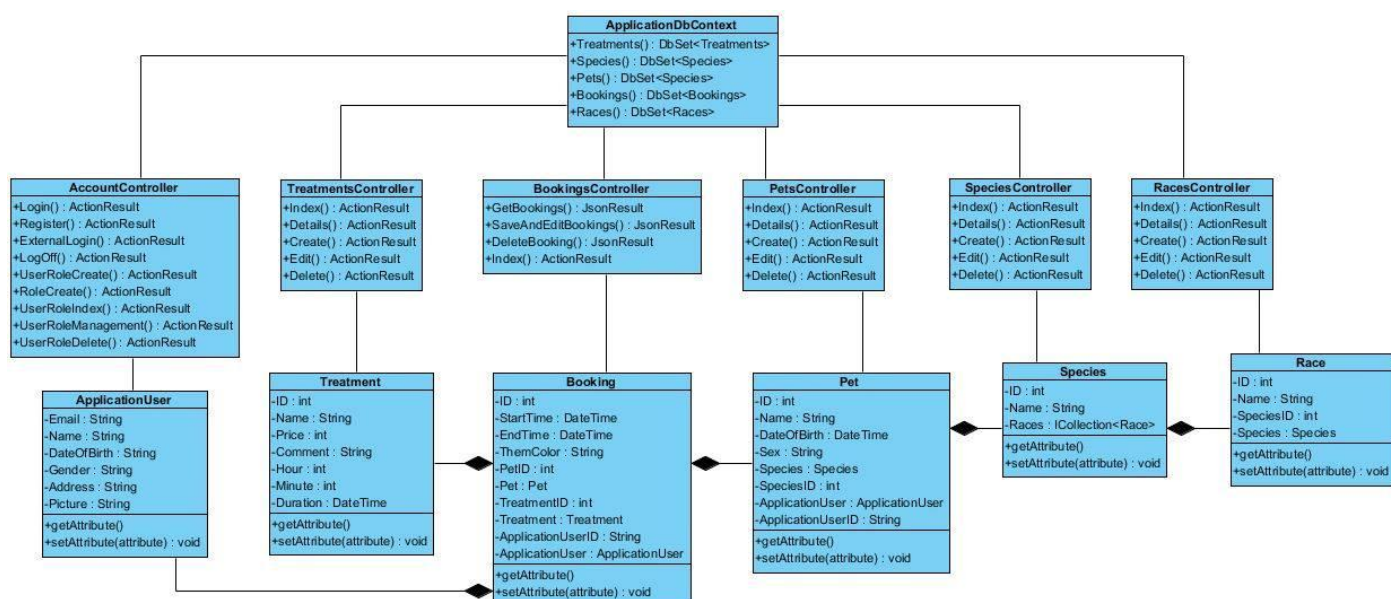
In this chapter we will design the software architecture and visualize which components that are going to be put together. We will therefor also design the database, and how the different tables will be connected with each other.

### 5.1 Design Class Diagram *(MI & ES)*

A design class diagram is a static diagram that describes and visualize the structure of the system by showing how the system's classes has been put together. We have used the diagram as documentation, where it gives us a good overview of the system, additionally we also used it in the beginning phase of our project, that way everyone in the team had a good basis of what/how to build the system. We have throughout the project updated the diagram as our way of solving requirements/problems are constantly changing.

A class diagram consists of software classes that contain attributes and operation (methods), in addition there are also visibility marks which tells if a class is private (-), public (+) or protected (#). Furthermore, there are also connections between the classes, which are: **Association** is a simple connection (normal line), **Generalization** if one class inherits from another class (one arrow), **Aggregation** as a less powerful connection (white diamond) and **Composition** that is the strong connection (black diamond).

In our class diagram we have avoided adding view classes as it will give total chaos to the diagram, furthermore we also avoided some of the classes that are autogenerated by the framework.

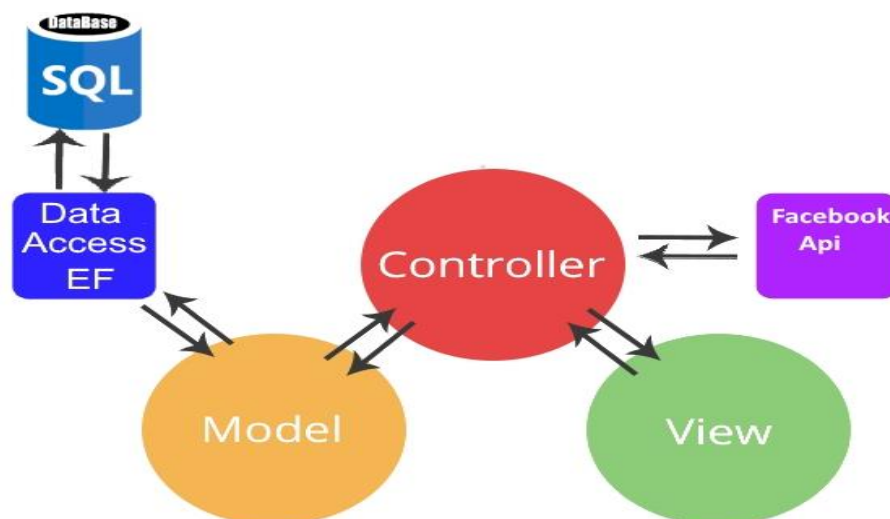


## 5.2 Software Architecture (BN)

In this architecture we will strive for Low Coupling and High Cohesion, we will do this by applying the Model, View and Controller (*the MVC architecture*). We are going to use Structured Query Language (SQL) for storing data and C# for coding. Moreover, we will use visual studio framework and Entity Framework for mapping the database.

We made our architecture with the approach to achieve low coupling and high cohesion. The way we are going to achieve that, is by implementing the design principle “separation of concerns”<sup>3</sup> which will make it easier for us to work while developing the application, but also if any problem arises or changes happens it can then easily be located. Another reason why it is good is because every part of the application can be replaced by another component, for example we as software developers have little to no web development and graphic design experience. For us to make the design of the application look good would require a lot of time because of our inexperience, but by applying this design principle we could easily replace the view with something that other developers have made without having to mess with the logic of the application. Last but not least, testing the system can be easier achieved since the other components won’t be affecting each other.

The figure below shows an overview of the entire system, all the involved parts there is in the system and the flow of data.



<sup>3</sup> [https://en.wikipedia.org/wiki/Separation\\_of\\_concerns](https://en.wikipedia.org/wiki/Separation_of_concerns)

### 5.2.1 ASP.NET MVC

The majority part of our system architecture is based on ASP.NET MVC which is a web development framework from Microsoft that combines the efficiency and clarity of model-view controller (MVC) architecture. ASP.NET MVC is used to build dynamic web pages and web applications, using HTML, CSS, JavaScript, server scripting and C # language.

When using ASP.NET, you use something called Razer Engine, which allows us to create dynamic content on the website, in addition to that, makes the front end easier for the back-end programmer.

#### **Model**

The Model represent the data structure and nothing more it's here we define the data type and it's from here we define the database structure because we are using code first approach to generate the database using Entity framework. If the business logic change or a new feature needs to be added, it can easily be implemented by just adding new model classes. The model class will have the data type and relationships which is used to auto generate the database via entity framework and have the business logic structured. This would not affect any other component and the expanding of the application can be approached without restructuring the whole application.

#### **View**

The view is where all HTLM get rendered for the client-web browser and we have stick to that approach by not having any backend coding on this component. Doing this can help simplify the process of replacing the view with another HTLM view that is designed and developed by a professional graphic designer. Another thing is this could also be replaced by a mobile application because it is not high coupled with any other component.

#### **Controller**

The Controller is the data manipulator in the MVC, this is where data get gathered and get manipulated to be disputed to the requester. Each model have a controller assign to it by doing this we can easily find the functions and manipulate the data and send it to the view or the requester. Having this structure can help us to apply test cases that is narrowed to the specific component and functions. Reusability of code that we have on this class is important in a case that in a future we would like to build a mobile application.

#### **Summary**

The goal of our software architecture is just like Lego all our component are small part that interact with each other to structure a bigger architecture, but unlike Lego we can replace any component at any given time even if it is already build without restructuring or demolishing the architecture. This approach will help us make developing and maintaining the application much easier.

Overall the ability to expend the application in the future or making any changes on the business logic or the view design like making a mobile App is much easier with this approach. Making different kind of test to a specified component can be a big help. If new developers were to get involved in the future this would make it easy to get an overview and get acquainted with the application.

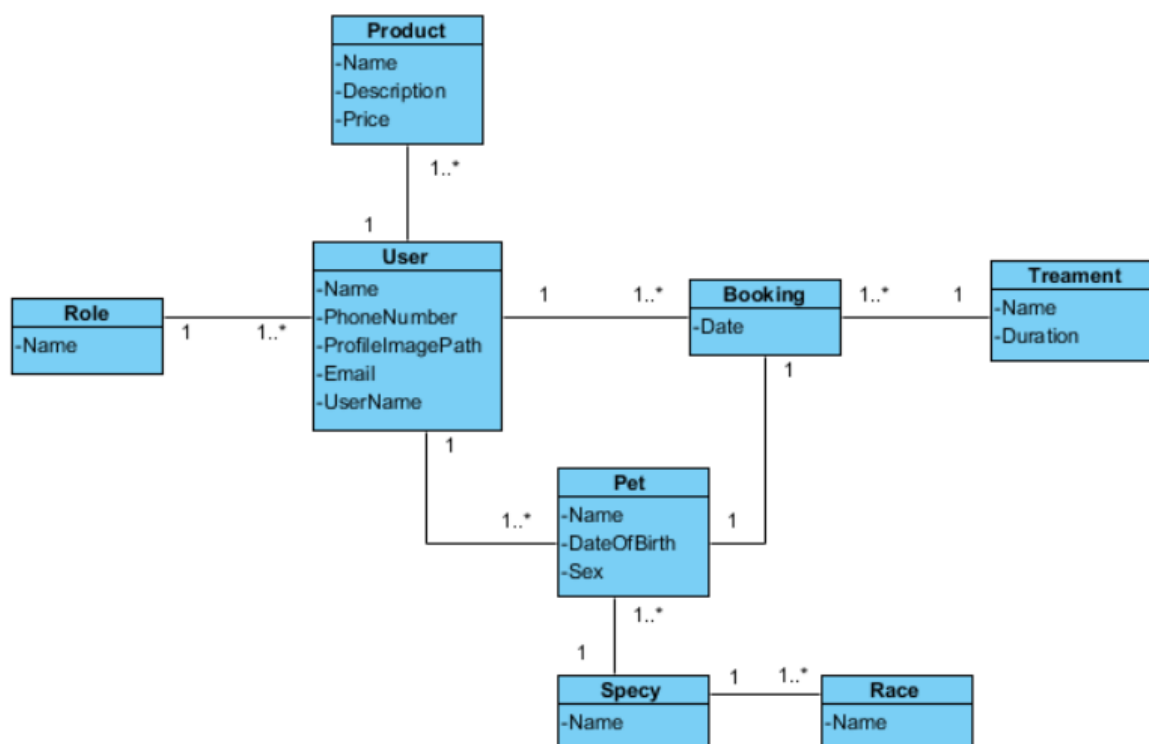
## 5.3 Database (MI & ES)

In this part we will create the design for our database, first by making a conceptual data model, then a logical and at last make the final physical data model.

### 5.3.1 Conceptual Design

After we have gathered and analysed all of the requirements, we can begin to create the conceptual design model for our database. The purpose of the conceptual design is to describe the entities and the relations they have to each other (*type of the relationship*).<sup>4</sup> Then you describe all the attributes each entity contains. The first step for us was to identify all the entities and afterwards which attributes they should have. We find the entities and the attributes by analysing the requirements for all the nouns and put them in a list. From the list we agree on the entities and the attributes, when that is done we decide their relationship with each other and add associations between them and give them multiplicities.<sup>5</sup>

One of the reasons to make a conceptual design, is that it gives the opportunity for non-technical stakeholders to understand the business knowledge from a data perspective, without diving into details about each entity. The conceptual data model/design can with other words be described as a representation of the data requirements. It's also possible to use the conceptual design as inspiration for the logical data model/design.



<sup>4</sup> [https://technet.microsoft.com/en-us/library/ms190651\(v=sql.105\).aspx](https://technet.microsoft.com/en-us/library/ms190651(v=sql.105).aspx)

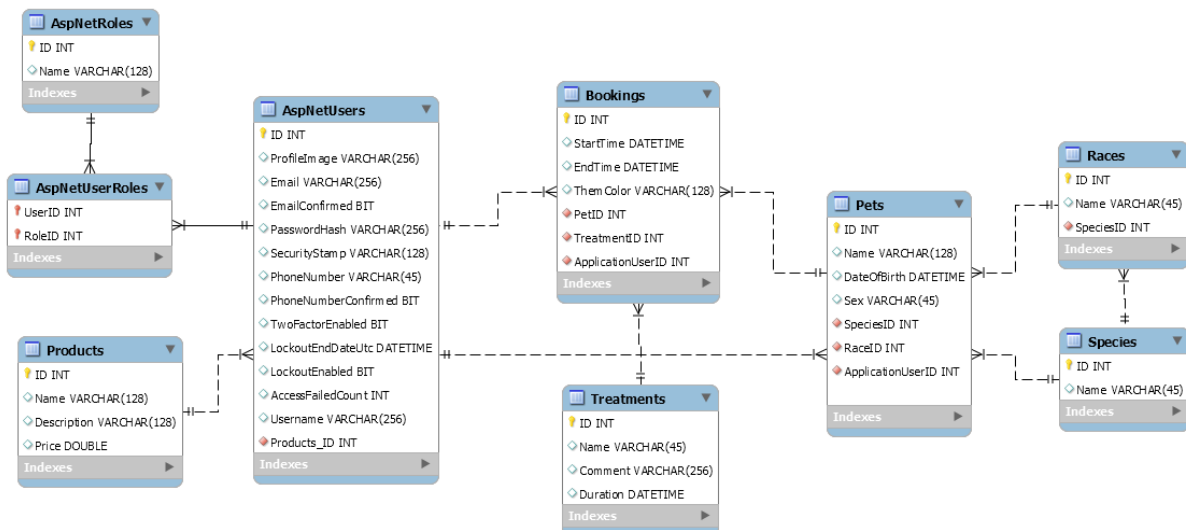
<sup>5</sup> <http://www.tomjewett.com/dbdesign/dbdesign.php?page=association.php>



### 5.3.2 Logical Design

Logical data model or the logical design describes the database and the tables with as much detail as possible. This design will include all the entities and the relationships between them. The attributes for the entities are also specified and the keys for the entities are chosen. Which means that the primary key and the foreign keys are specified for each entity. Some people also adds the datatypes for the attributes, which we also intend to do.

In this phase we also normalize the database<sup>6</sup> to ensure that we have eliminated any redundancies there might be. At this stage our logical data model looks like:



We have in our logical data model started with creating the entity names, and afterwards added the different columns/attributes. Then we selected the different keys, such as the primary key (*the yellow keys*) and the foreign keys (*the red keys/diamonds*) for each table. Thereafter we added the relationships and the multiplicities. Every entity has an ID attribute that will be a primary key and therefor unique. Our idea with this logical data model is to describe the data requirements as of now in a detailed way.

<sup>6</sup> <http://searchsqlserver.techtarget.com/definition/normalization>

### 5.3.3. Physical Design

The physical design is the actual design/blueprint of the database, how it is looking when the database is in its physical form or when we have released it. However, we will not make another diagram in this phase, as our database hasn't changed since the logical design (*since it would look exactly as our logical design*) except one table that we didn't make because of the missing implementation of a user story (*Products*).

What we have done instead, is that we estimated and calculated what we expect the size of the database is going to be in the next 5 years (*How big the database is*):

#### Bookings

ID - int	4 Bytes
StartTime - datetime	8 Bytes
EndTime - datetime	8 Bytes
ThemColor - nvarchar(50)	100 Bytes
PetID - int	4 Bytes
TreatmentID - int	4 Bytes
ApplicationUserID - nvarchar(128)	256 Bytes
<b>Total</b>	<b>384 Bytes</b>

The "Bookings" is one of the tables that we expect will have the most records, we expect that there's going to be approximately 20.000 bookings in the database in 5 years. We can now calculate the expected size of the database by saying; tables total bytes (384) \* expected bookings (20.000) = 7.680.000 Bytes. Which means the size will be 7,32 MB.

Since we are not able to show all the calculated tables, we have only chosen to visualise the Bookings table in the report. You can find the rest of the calculations in the appendix "*Estimated storage size*".

We have calculated and expect the size of our database to be 56 MB. We have chosen to launch the application on Azure server and we have found out what it is going to cost for a storage space on Azure if Grundtvigs Dyreklinik want to publish it there. It's going to cost 464,52kr<sup>7</sup> pr. month to have a storage space of 5GB which will be enough for Grundtvigs Dyreklinik.

---

<sup>7</sup> <https://azure.microsoft.com/da-dk/pricing/details/sql-database/>

## 6. SCRUM *(ES)*

When developing a software system, it is important to choose the right development process. A software development process must be perceived as a guideline for how activities are to be performed and planned.

The typical activities are identification of customer needs and requirements, delegation of tasks and responsibilities, testing of the system and documentation of the project.

Therefore, in this project we have chosen to use the development framework SCRUM. We have chosen to work with SCRUM for two reasons. Firstly, because we already have experience working with this development method. Secondly, since this project is in its alpha stage of planning and the people that wants the system are not experienced or have any heavy tech background. This would mean, that in the course of the project, there will probably be many changes during the development. Which is why we believe, that SCRUM is the right fit for this project, as SCRUM is one of the most agile development method.

So, what is SCRUM? SCRUM is a well-known agile development method that you use to build systems. Here you and your development team, work together as a unit to reach a common goal. The benefit of using Scrum is that it is very flexible and because of that, it is very easy to respond to changes. It is also good at showing us how things are progressing throughout the development, which makes it easier for us, to get a better understanding of the bigger picture.

The way we have used SCRUM, is by first choosing how we will allocate the roles between team members. As you now, In Scrum, there are different roles that needs to be filled. There is;

- **The Scrum Master**, who makes sure the whole project process run smoothly so that the team can reach their goal. He also organizes and facilitates critical meetings, and he makes sure that the team comply with the rules.
- **The Product Owner**, represents the customer's voice. He is responsible for Product Backlog where he writes the backlog items (*which are the requirements of the system like User Stories*) and he also prioritize them.
- **The Team**, comprises of analysts, designers, developers, testers, etc.

Here we have chosen to change the role of each team members, on each sprint. That way, we all get the chance to try out each SCRUM role.

We're also going to have sprints, and here we have decided that each sprint is going to be planned differently. Because when we start the project, we (the team) are still going to have other responsibilities like work, so our schedule is going to be chaotic. Moreover, people also have other stuff that makes it impossible for us to work consistently each sprint. Therefore, the length of each sprint is going to be different.

To get a better track of our Scrum board we have chosen to put all our user stories/tasks in an online program called Trello, that way it will be easier for us to maintain and have access to our Product Backlog and Sprint Backlog.

In order to follow the development of the system, we have used Burndown Chart. Here we would at the end of each working day, write down how many hours we had spent on the different user stories and calculate it into our burndown chart. This way we could quickly get an overview of the project and see if we were on track. In addition, we have also created sprint retrospectives each sprint where we would evaluate the sprint about what went wrong, what could we do better and what steps are to be taken next. Lastly, we are also going to follow the different meeting ceremonies that SCRUM provides.

## Sprint 1

We have planned to complete

### Plan:

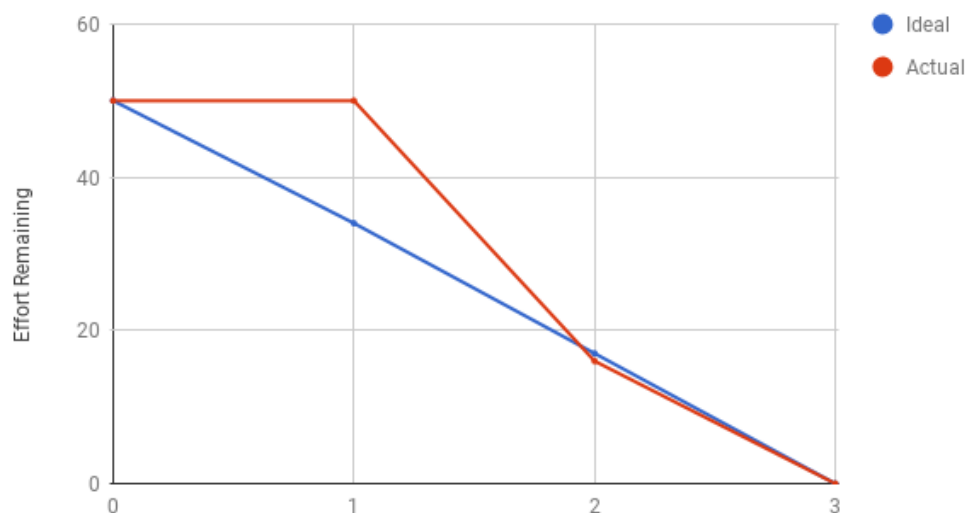
- Setting up/Building the skeleton of the system (10hr)
- US 01 (8hr) - CRUD Treatments
- US 12 (8hr) - *CRUD Species*
- US 13 (8hr) - *CRUD animal Race*
- US 07 (8hr) - *User should be able to register themselves*
- US 10 (4hr) - Login/Logoff to the system
- Test Case (4hr)

### Evaluation:

In the first sprint, we planned that we were going to work **50 hours** and it would last for one week. We started by creating a Visual Studio Team Services (VSTS) repository for our Project (*which includes - code, database, user stories and testcases*) and made sure that every team member was invited. This took us some time, since it was the first time we had used VSTS. So, before we began creating the Visual Studio Team Services Account and the team project itself, we made a test project first in which we used to learn how to use VSTS, as you can imagine we made many mistakes and learned a lot (*like create repository for the code, pull, push and rollback etc*). We thereafter scrapped the project made a new one and began to develop the user stories.

Overall, it went well and we finished the sprint in time despite the complication in the beginning, this was due to that we had experience in developing a ASP.NET project, which made us finish the user stories faster than we had calculated. The last thing we did was integrate the different features together and thereafter ran our test cases on VSTS for the user stories we had completed, which they all passed.

Burndown Chart 1



## Sprint 2

We have planned to complete

### Plan:

- US 06 (16hr) - CRUD Pets
- US 08 (30hr) - Book a Treatment
- Test Case (4hr)

### Evaluation:

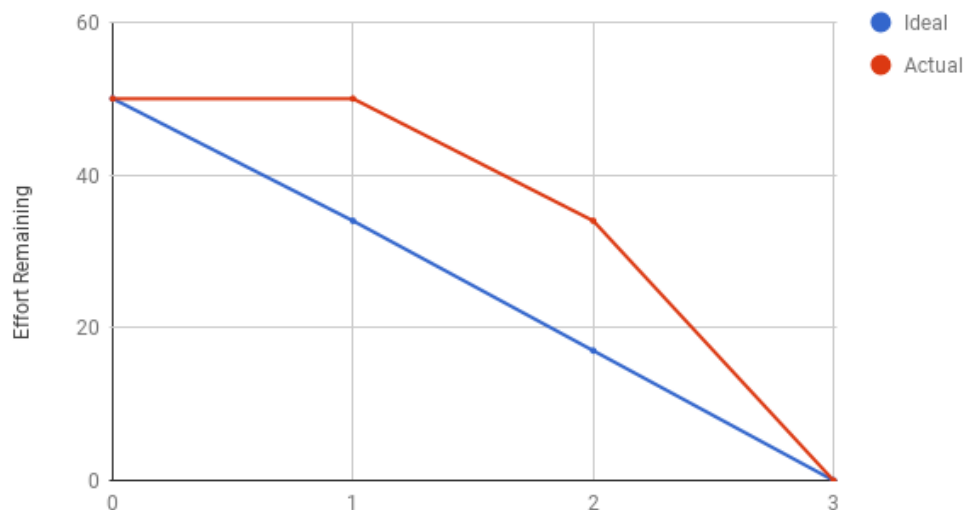
In this sprint we planned to work 50 hours which should result in coding two user stories. The user stories mentioned were crucial to the system but could not be coded any earlier due to them being dependent on other user stories/features which we developed in an earlier sprint.

In this sprint, we also decided to do pair-programming on US 08, because we knew that this specific user story was going to be very challenging and because we had two user stories to complete.

In the burndown chart you can see that in the beginning we had a slow start because we had to do a lot of research to the US 08 as we needed to make a schedule which none of us had tried. We found out that instead of making the calendar system from scratch we could download the jquery-ui and fullcalendar script, which would help us develop the user story. We also found out that we needed to redo a small part of user story 01 (CRUD treatments) because the duration variable was in "int" and we needed it to be a specific datetime, this meant we needed to change Durations model, controller and views.

Overall this sprint went well, pair programming was a success and even though we had a slow start in the beginning we still manage to complete all our user stories, integrate the new features and ran our test cases on VSTS in time.

Burndown Chart 2



## Sprint 3

We have planned to complete

### Plan:

- US 05 (16hr) - Register others in Employee roles
- US 11 (8hr) - Login and Register via Facebook or Google
- Usability Test (6hr)
- Test Case (4hr)
- Create/Run Automated Testing Script (12hr)

### Evaluation:

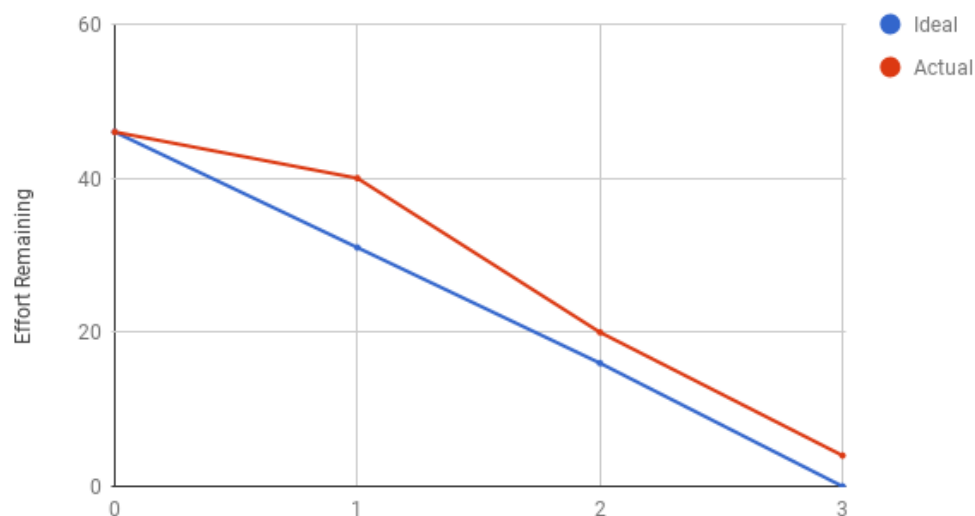
In this sprint we have planned to work 46 hours which have result in coding two user stories, creating an automated test script and perform a usability test.

Oure Performance in the usability test went well. We had found some people at our school, who gave us some crucial feedback about out application, which we needed to change in the next sprint. To create our automated test script, we had to make a license in the program Hp Unified Functional Testing. Here we used the language visual basic to create a script, that runs through every requirement in the application. This will run every time a new build comes, which is at the end of every sprint when we integrate the completed US. Finding the right tool and creating the automated script went smoothly, because we had a person from the group, that has experience working with it at his student job.

Implementing the US 05 went smoothly throughout the project, whereas the US 11 didn't. The first problem that occurred in US 11 was the version you got when creating the Facebook app, which is an API that allows you to login through Facebook was not compatible with our VS. So, to fix it, we had to find a way to downgrade the version which thereafter worked as it should. Because of this delay we did not finish the login with Google, so we had a talk with our customer/product owner, where we decided to scrap the Google part.

Overall, we didn't finish the sprint because we missed 4 hours of work, which was the Google login. Another reason why we didn't finish on time was because we spent a lot of time arguing about how to delegate the work.

Burndown Chart 3



## Sprint 4

We have planned to complete

### Plan:

- US 14 (14hr) - Retrieve information from Facebook
- Make changes after response from usability test (8hr)
- Validation (8hr)
- Change Interface to Danish (8hr)
- Publish in Azure server (8hr)
- Run Automated Testing Script (2hr)
- Unit Test (4hr)

### Evaluation:

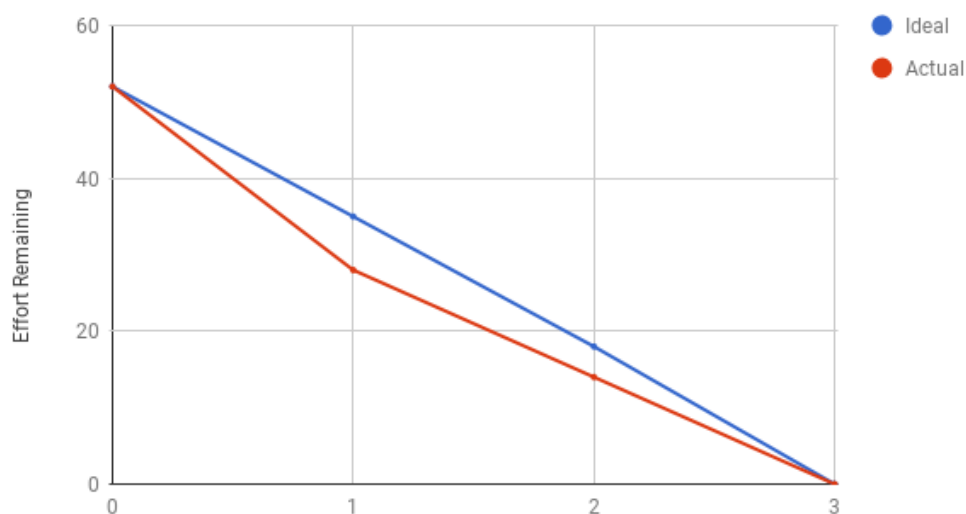
In this sprint we have planned to work 52 hours which was mostly spent on working with different aspects of refactoring and making small changes to the application/system. Besides that, we also found out from the last sprint that instead of just logging in through Facebook, we would also like to get specific information from Facebook. So, we had made a new user story for this sprint.

We also finished with changing the interface from English to Danish, after we had got some response from our usability test.

Later on, we published the system on Azure, but ran into some problems with the connection string. We got some insight from our previous projects, and then remembered how the connection string should look like and changed it. We also completed the validation for the different fields, so that you for instance can't leave a field empty. At the end of our sprint we ran our automated test script, which was successfully. In addition, we also made a Unit Test, where we test if we get the data from the database and if the validation on the properties works.

Lastly, we have agreed that this will be our last sprint and therefore we would not code any more functionalities/US. That way, we could spend the last of our time to write our bachelor report for this project.

Burndown Chart 4



## 7. Implementation

In this chapter we will show how we implemented the different parts of the system and document how we have done it.

### 7.1 Database and Entity Framework (ES)

To create the database in our system we have chosen to use Entity Framework code first, here the entities and attributes is made based on the code. Entity Framework is a tool you get when using Visual Studio and is an **object-relational mapper** that enables .NET developers to work with relational data using domain-specific objects<sup>8</sup>. It is also used when you want to connect to SQL Server and gain access to the database.

The way we have used it, is by using a feature called *code-first*; the idea is that we can define classes in the model (C# objects) and then generate a database from those classes. In addition, Visual Studio (ASP .NET) is integrated with the Microsoft SQL Server, which makes it easier for us to alter or change some in the database/tables later on.

The reason why we have chosen to use Entity Framework's code-first approach is because; by using it you can build a database up much faster, in addition it is very easy to maintain and modify the database, and it can all be done with just a few lines of code. It is also a tool that the entire team have experience in using and therefore gives us confidence in tackling any challenges that may arrive.

#### Creating a table:

The way we have created a table via entity framework is by first creating a model class; here we have created Treatment which will act as the entity/table in the database. Thereafter this class should contain some attributes and these attributes will then represent data fields in the database table.

```
public class Treatment
{
    public int ID { get; set; }
    public String Name { get; set; }
    private int price { get; set; }
    public int Duration { get; set; }
    public String Comment { get; set; }
}
```

Then you go to the ApplicationDbContext class, which is located in IdentityModels.cs and here you paste this line of code:

```
"public DbSet<Treatment> Treatments { get; set; }".
```

What this does is that it assigns the class model to the database.

Then the last step is to type "update database" in the console, by doing that the Framework will generate a table with the name Treatment and the attributes of the class will represent database fields.

---

<sup>8</sup> <https://docs.microsoft.com/en-us/aspnet/entity-framework>



Another way to use entity framework is by using a method called "Database-First", as the name suggests, it is about you first create the database, and thereafter auto generates the code from the database.

### 7.1.1 Relations between tables

To create relation between tables via code first - entity framework you need to use something called navigation properties, which provide a way to navigate an association between two entity types via classes. In our database, we are going to have three different relationships, which I will explain in this segment.

**One-to-Many:** The first relationship that I will explain is the One-to-Many relation. Imagine you have Species entity and a Race entity, the relation between them is; one Species can have many races (*so a One-to-Many relation*). To code that, so the entity framework understands it, one must first go over to the Race class and add these lines;

```
public int SpeciesID { get; set; }
public virtual Species Species { get; set; }
```

By making a variable and defining the type as `Species`, the framework will understand that the Race class want to create a relation to the Species class. This way, the `SpeciesID` becomes a foreign key in the Race table, which means when you create a Race you can refer the Race to a Species (*The int SpeciesID variable is not necessary, but if you don't have it, the framework will give you a default name to the foreign key*).

Additionally, in the Species class you must add a `ICollection`. By adding a `ICollection` variable you can hold many instances of that specific class. You do it like this;

```
public virtual ICollection<Race> Races { get; set; }
```

Doing the steps above you will have a "One-to-Many relation" in the database, which makes it possible that a Species can have more than one Race element.

**One-to-One:** A One-to-One relation means that two entities A and B in which one element of A may only be linked to one element of B, and vice versa. When creating a One-to-One relationship, the primary key also acts as a foreign key, and there is no separate foreign key column for either tables. An example of how you would do it in code first, would be If we had a HomeAddress and a ApplicationUser class, and we wanted to create a one to one relation between them it would then look like this:

In the ApplicationUser class we would have;

```
public string ApplicationUserID { get; set; }
public virtual HomeAddress HomeAddress { get; set; }
```

And In the HomeAdress class we must have a primary key which also is a foreign key;

```
[Key, ForeignKey("ApplicationUser")]
public string ApplicationUserID { get; set; } }
public virtual ApplicationUser ApplicationUser { get; set; }
```

By doing it this way, the Entity Framework understands that you want a One-to-One relation.

**Many-to-Many:** In a Many-to-Many relation we want it to be possible for `AspNetRoles` to have many `AspNetUsers` and vice versa. To create a Many-to-Many relation, you are going to need 3 classes. To give an example; from our system we're going to talk about `AspNetRole` and `AspNetUser`, because the relation between them is a Many-to-Many.

First, you'll need to add this line of code to the `AspNetRole` and `AspNetUser` class;

```
public virtual ICollection<AspNetUserRole> AspNetUsersRoles { get; set; }
```

Then you make a class called `AspNetUserRole` and you add these variables

```
public int AspNetRoleID { get; set; }
public virtual AspNetRole AspNetRole { get; set; }

public int AspNetUserID { get; set; }
public virtual AspNetUser AspNetUser { get; set; }
```

By doing these steps and updating the database from the console, the framework will understand that you have made a Many-to-Many relation and therefore the `AspNetRoleID` and `AspNetUserID` will act as foreign keys. *(Just to point out, we didn't make a Many-to-Many relation between these classes this way, because these classes are AspNet classes and are autogenerated when you create the project. But this is an example of how to do it when using entity framework code-first)*

### 7.1.2 SQL and ORM (MI)

As earlier told, we have used object relational mapping where we convert our data and objects from the code into a relational database. But what if we had made our database through DBMS by using plain SQL which separates the database from the application code. How would it look and what difference would it make? So, in this chapter we want to discuss the differences between using ORM and SQL, and how we have use DML (*Data Manipulation Language*) to populate the database.

#### Data Entry

The first thing we will show, is how we have populated the database before going on with discussing ORM and SQL. To populate the database, we have used DML queries to insert data into the different tables.

The first thing to do is to have a DBMS installed and from there create a new query. In this query we will first specify the database that we want to use, and afterwards make a "insert into" statement where we will select the exact table and its columns that we want to insert into. Then we will add the values.

```
USE [GrundtvigsDyreklinik]
GO

INSERT INTO [dbo].[Species]([Name])VALUES ('Hund')
INSERT INTO [dbo].[Species]([Name])VALUES ('Kat')
INSERT INTO [dbo].[Species]([Name])VALUES ('Slange')
INSERT INTO [dbo].[Species]([Name])VALUES ('Fugl')

GO

USE [GrundtvigsDyreklinik]
GO

INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Golden Retriever', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Husky', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Mastiff', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Beagle', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Chihuahua', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Golden Retriever', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Dobermann', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Bulldog', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Greyhound', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Labrador', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Maltese', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Rottweiler', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Shih Tzu', 1)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Perser', 2)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Angora', 2)
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Sphinx', 2)
TRUNCATE TABLE [dbo].[Races]
INSERT INTO [dbo].[Races]([Name],[SpeciesID])VALUES('Runeilla', 2)
```

This way we will not be going through the "longer" process of inserting data into the database from the web-application/system like the end-user.

## Plain SQL vs ORM

Using Object Relational Mapping (ORM) instead of plain SQL in a DBMS has its advantages and disadvantages.

ORM is mostly used for model-first approaches where you create your database and tables from the objects/classes. With ORM it is much easier to maintain the code and for the most it is self-protected. ORM offers strong mapping, so it is very hard to make stupid mistakes, such as comparing double to string as it can happen with SQL. It is also much more time consuming to write for instance CRUD operations with SQL. Another advantage of using ORM is that the security for preventing SQL-injection is taken care of by the ORM framework. Compared to SQL where it is required that you do manual intervention to prevent from an SQL-injection attack.

Whereas plain/raw SQL-code are generally much better in performance than ORM and you get a better control over the SQL queries and database. With plain SQL it is also much easier to make changes because of the data independence.

### 7.1.3 Trigger

Trigger is an SQL query that is triggered after Insert or Delete, most of the time triggers are used for auditing but in general their use is most suitable for database manipulation that can happen in the background.

We have made a trigger where it gives a “Role” to users, every time they register themselves in the system. The reason we chose this was because it would automatically give a role to users. This step would otherwise take too much time for the administrator to manually complete.

We have also in our trigger hard-coded which emails gets to be administrators. Here it checks and sees if the inserted email matches with the admin emails in the trigger. If not, the user will be given the role as a regular user.

```
CREATE TRIGGER [dbo].[TriggerRoles]
ON [dbo].[AspNetUsers]
AFTER INSERT
AS
BEGIN
    declare @userId varchar(200)
    declare @roleId varchar(200)
    declare @admin varchar(200)

    select @admin = Email from inserted

    select @userId = Id from inserted

    if (@admin = 'benyamneamen@gmail.com' Or @admin = 'elsa@hotmail.dk' Or @admin = 'ince@live.dk')
    select @roleId = Id from [dbo].[AspNetRoles] where Name = 'Admin'
    else
    select @roleId = Id from [dbo].[AspNetRoles] where Name = 'User'

    Insert into AspNetUserRoles (UserId , RoleId)
    values (@userId, @roleId)
```

The way the trigger works is that it runs every time that there is an insert in the “AspNetUser” table. The if statement checks if the 3 emails (our emails) matches and if it does, it gives us the administrator role from the “AspNetRoles” table. Otherwise it goes to the else statement the role and give the role “User”.

If this functionality was on the application side it would take a lot of coding, here we would have to register the user first. Then take the Id from that user and the roleId from the table “AspNetRoles”. Lastly, we would then have to assign those two values in the “AspNetUserRoles” table. By having it on the database side, we make it more simple and efficient.

## 7.2 Security

Security is one of the most needed subjects to have in consideration when building a webpage. We have thought about three different security subjects, which is database, validation and user management. By following basic coding conventions, database management, the identity management for managing different users with different responsibility and using the ASP.NET platform is how we have secured our application. In the coming sub categories, we will explain briefly how we configured the application so that it is safe from being misused and alteration of the database.

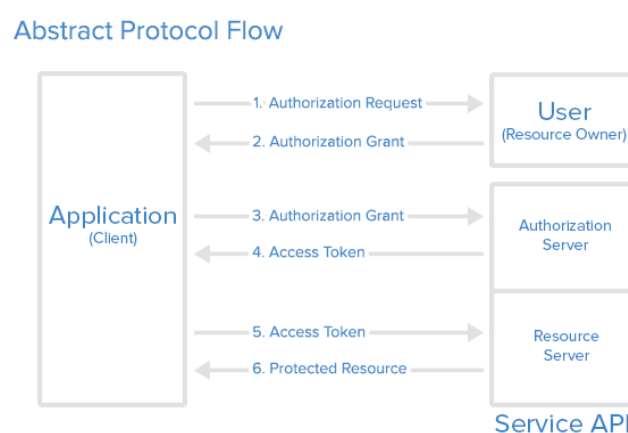
### 7.2.1 Identity management (BN)

We chose to use the Microsoft Asp.Net Identity framework for identifying and authenticating application users. What makes this Identity management different from other identity managements or self-made management systems is that it comes with a build-in functionality. The developers can use this to restructure the system, so that it fits their desire. Some of the core functionalities that are included are user's database structure, third party login, hashing of password, token and role base login to mention some of them.

#### 7.2.1.1 Login-OAuth (BN)

OAuth 2.0 is a standard access protocol for authorization of users. A user that want to make a booking in our system have to login to the application. By either registering through our systems user-registration or use his Facebook account. Sensitive information such as password is hashed to protect identity theft. To enable the third part authorization for our application, we used OAuth 2 authorization framework. OAuth uses HTTP to communicate and get access to users account. One of the benefit of using OAuth is that a developer doesn't have to think about the security for the user information. The reason being if the request is trusted by e.g. Facebook then most probably the user is who he said he is.

The following steps are how OAuth works.<sup>9</sup>



1. The application requests authorization to access service resources from the user
2. If the user authorized the request, the application receives an authorization grant

<sup>9</sup> Picture and step one to six explanation are from this: link <https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>

3. The application requests an access token from the authorization server (*API*) by presenting authentication of its own identity, and the authorization grant
4. If the application identity is authenticated and the authorization grant is valid, the authorization server (*API*) issues an access token to the application. Authorization is complete.
5. The application requests the resource from the resource server (*API*) and presents the access token for authentication
6. If the access token is valid, the resource server (*API*) serves the resource to the application

#### 7.2.1.2 Roles (ES)

ASP.Net Framework has a powerful feature called Identity roles which handles authentication and authorization in the application. The way to use authorization, is by having these .NET attributes, that you can apply to any action method or a controller class which then adds additional logic when a request is processed. There are different types of filters, but the one I want to talk about is authorization filters that we have used in our application.

Our Application have three different users; a passive user (*a guest*), active end-user (*a member*) and an administrative user (*an employee*). Each role has separate responsibility/ability to interact with the application and to handle these roles we have used the Identities roles management.

In our database we have called the roles a "User", a "Admin" and if you are "Null" you are a guest.

- A **Admin** have the highest hierarchy and his responsibility is to manage and maintain all the data in the application. And if necessary also has the ability to handle other users.
- A **User** that have registered to the system and can therefore manage his personal data (*like add pets, book time etc*).
- A **Guest** is the lowest in the hierarchy and is only able to view some of the information and content. He has no possible way of changing or adding any data other than buying products.

Once we had defined the roles it was time to implement it in the code by restricting the access to the functionality, so that anyone without authorization can't modify or add data. We did this by adding annotation like this;

```
[Authorize]
public class ManageController : Controller
{
    All the method in the class...
}
```

By adding `[Authorize]` over a controller class, you get access to all the methods in the class if you are an "authorized user". This means, you can tell the system that you can only access this controller class and its functionality if you are logged into the application. You can also be more specific and insert `[Authorize]` over a method, this means only the method is authorized.

Additionally, you can also give access to a specific role, that way different roles can have access to different features. To apply this, you first add the different roles into AspNetRoles database table. Afterwards you associate AspNetRoles with the AspNetUser table, by using the Many-to-Many table AspNetUserRoles. This tells the system that a specific user, can have access to a particular method or a whole class.

When delegating roles to a user, one can either hardcode it in the database or make an interface like we have. The last thing you do is to choose which role should have access to the different functions in the application by writing the code below;

```
[Authorize(Roles = "Admin")].
```

And anyone trying to use a feature that they don't have access to, will be redirected to the login page with a message that they don't have access to enter the specific section.

Adding authorization in one's application is really important and without it, people would be able to cheat and create chaos.

### 7.2.1.3 Validation (ES)

Validating your application and protecting it from the user is an important part of security. This is because a user's interaction with the application can lead to wrong data entering the database.

That's why it's important to protect the server from receiving any harmful data and guide the user, by specifying which data he/she can expect from the client.

By using MVC Framework in Visual Studio, you can use something called "Model Validation/Data Annotation", which we have used. Data Annotation means that you can get all your variables in your classes to follow some kind of rule, that are then enforced throughout the system. This enforcer is standard built in the Asp.Net project, which you can find in the model class *"DefaultModelBinder"*.

Validation logic-executing in the browser is a must in today's applications, because it gives an instant feedback to the user. The reason we do this is because the client doesn't have to check if the data is right by sending to the server, hence minimizing the request and response time.

So what does this mean to the user? - This means that the user cannot enter incorrect data in the database and it can also help him insert the correct data by giving him an error message. These error messages are also supported by Data Annotations, and by setting a validation attribute and an Error Message property in front of a variable, the system will then help the user if they enter wrong data. We apply this on our project by first adding it in the view (*html*) class which displays the validation error message, by writing this code;

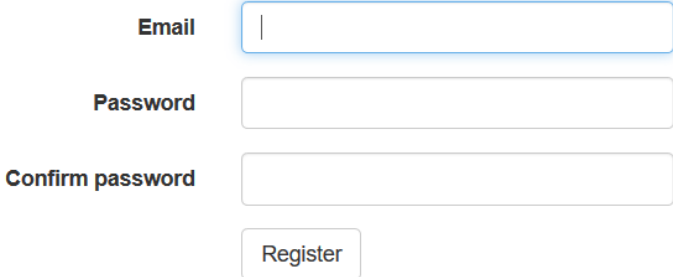
```
@Html.ValidationMessageFor(model => model.Password, "", new { @class = "text-danger" })
```

Thereafter in the model class we add data annotations on the attributes. These annotations act like rules which we defined, that the data coming back from the server must follow. This is done by placing the require annotation on top of the variable;

- [Required]
- [StringLength(60, MinimumLength = 4, ErrorMessage = "{0} should be between 4 to 60 Char")]
- [DataType(DataType.Password)]

By applying these validations, we have been able to give a user a richer experience and a smoother interaction with the application. An example of this would be - when a user registers and maybe forgets to type Email or Password, they would then get an error message before submitting the form to apply the changes like the picture below.

- Feltet Email skal udfyldes.
- Feltet Password skal udfyldes.



The form consists of three input fields stacked vertically. The first field is labeled 'Email' and has a blue border. The second field is labeled 'Password' and the third is labeled 'Confirm password'. Below these fields is a button labeled 'Register'.

We have in our application made all different kinds of validation on our fields (*attributes*).

#### 7.2.1.4 Database Security (BN)

Database is the heartbeat of the application, anything that can happen without thoughtful action can jeopardize the whole application, for this reason we didn't take any risk for securing the database.

There are two sides to this. The first being which end-user have the right to manipulate the data in the database (*such as insert, deleting and updating data to a table*). Here we have done whatever necessary to secure our application using Identity Management for any user's misbehavior as described above.

The other side is the development/administration side of the database where e.g. a developer can create, update and delete the database or tables. Therefore it's important that everyone doesn't have full access.

We haven't focused much on the security of the database, as we only had one admin account that had full access to the Azure database.

If there were more people involved in the development of the system, then securing the database would have had a higher priority. What we could have done was to implement roles, that way you can specify which advanced users have access to modify the database and tables.

By having no roles at all, that would mean that everyone has access to the entire database. This could mean that any developer making a mistake in the code can jeopardize the whole database.

### 7.3 Integration *(BN)*

We have in our application integrated Facebook login and FullCalendar. The Facebook login is an API where users can login to get basic information. The FullCalendar is a JQuery library for displaying bookings in a calendar. The core principle of Integration in software, is to make two or more sub systems work together as a single component. Often when we as software developers talk about integration there are two instance that normally comes to mind, which are; if the component or the feature to be integrated is a legacy system or not. Fortunately for us, we didn't have to integrate a legacy system. The following segment is about how we integrated the Facebook API and the FullCalendar library.

When integrating the Facebook API, you have to think about what kinds of problems that can occur in the future. Some examples of this, could be if Facebook decides to change their API or if we chose to use another API. Ideally this change should be applied without disrupting or restructuring the whole system, in our case it's only a matter of replacing the API source for getting data. We are able to do this because our system really doesn't know where or how it's getting the data it's only using data that it gets. By structuring the system this way, it will be more manageable when we build and restructure the system in the future.

The way we have integrated the FullCalendar Library is that we, just like the whole application architecture separated the classes in MVC structure where the View, Model and Controller have their own responsibility. We made a Javascript file where we make an Ajax call to the controller to manipulate the data. Then we define the data types in the model and display the View in the HTML file.

Later on, if the FullCalendar get outdated or the customers want to use another calendar like Googles or a self-made Calendar, we would then just replace the javascript file. This would mean that the controller, model and view in our system would not change. Hence the changes will not affect the system structure and flow.

Using external API like we have used have both Advantage and disadvantage. The advantage is that we don't have to create the whole feature from scratch, as the saying; Why reinvent the wheel?

Some of the disadvantage that are common when using external API are what if the API get out of service or change its core structure. This would affect our system severely because we are dependent on the API that gives us data. Therefor we have thought about these problems when we designed the system. We have also followed the MVC pattern so that every component and API in the system can be replaced without damaging the core system structure.



### 7.3.1 Facebook

Facebook just like many other applications are embracing the open source way of thinking, because of this reason the Facebook API offer different services to be used by other applications. We have chosen to use the Facebook API service which provides us a way to both sign in to our application, and in addition allows us to retrieve user's information - like Name, Profile Picture and Email.

We choose this because: firstly, we could display users profile pictures without having to save the images in the database, which usually is a costly business in terms of database space. By using this API, we saved the company of paying unnecessary database cost. Secondly, instead of the user registering via our own registering system, the user can register by just pressing the Facebook login button, this allows them to sign into our system, which will make the process easier for the user.

#### 7.3.1.1 External login (ES)

For the sign in part, we used ASP.Net own built-in functionality, which enables developers to rather easily implement a couple external/third-party API logins such as Google, Facebook and Twitter. This built-in functionality is called OAuth 2.0 which is a tool you can get when using Visual Studio.

The first thing we did, that would allow users to log in to our application with Facebook, was to go to the Visual Studio setting and enable SSL in our Grundtvigs Dyreklinik project. This allows us to secure the application on the browser level, by keeping your username and password cookies a secret, without it, you're just sending in clear-text across the wire. Thereafter we set the project URL to the new https-address in the project properties, which we are going to use when creating the Facebook app/API (The https-address *may change if we decide to host the solution on Azure*).

The next step to allow users to actually sign in via Facebook, is to create the Facebook app (API), which you do at Facebook developers home site. After creating the app, you get an AppId and an AppSecret which are two strings. Back to the application code, you go to the Startup.Auth class and write;

```
var facebookAuthOptions = new FacebookAuthenticationOptions()
{
    AppId = ConfigurationManager.AppSettings["Facebook.AppId"],
    AppSecret = ConfigurationManager.AppSettings["Facebook.AppSecret"],

    Provider = new FacebookAuthenticationProvider()
    {
        OnAuthenticated = (context) =>
        {
            context.Identity.AddClaim(new Claim("urn:facebook:access_token", context.AccessToken, ClaimValueTypes.String, "Facebook"));

            return Task.FromResult(0);
        }
    }
};
facebookAuthOptions.Scope.Add("email");
facebookAuthOptions.Scope.Add("public_profile");
app.UseFacebookAuthentication(facebookAuthOptions);
```

Thereafter you take the two strings and write them in the web.config like in the code snippet below

```
<add key="Facebook.AppId" value="1685636048407295" />
<add key="Facebook.AppSecret" value="3f93091237a5d2065cd713b21bc83f43" />
```

It is worth noticing that it is not recommended to store sensitive data in a regular project-file due to security risks. There seems to be some debate about how to actually store sensitive data. Do you place it in the code or in the web.config file (*like we did*) or do you leave it out of the project entirely and reference it from an external file? We believe Microsoft advice<sup>10</sup> is to use the third option, which we didn't do, mostly because we didn't know how and this application isn't a high target for hackers so security isn't going to be problem for us.

### 7.3.1.2 Retrieve information (BN)

Once the user chose to log in with Facebook using OAuth it was natural to use the access token that is already generated to retrieve information. The access token we get from Facebook is kind of a key that allows our system to ask Facebook to provide additional information. We save it, or in ASP.NET term add the access token to a claim that we can reuse on the whole application. As seen in the picture above it shows how we add the Facebook access token to a claim. This is where we add some scope like email and public profile that we want the Facebook API to give us.

When this is done we can use the access token to ask Facebook for any information the user have accepted to be retrieved by our application.

In our case we first get the access token from the application claim, with this access token we ask Facebook to give us user's public information like name, picture and email. Since Facebook API is a restful service that uses the HTTP method of GET, POST, PUT and DELETE. We therefore have to specify the method name and fields of data that we want to retrieve. By using the stateless protocol, it gives us a fast and reliable communication between different APIs. We then get the data in JSON format that have a key and value pair.

In the picture below, we use the GET method to get some of the user's information from the Facebook API. Once this is done, each fields name will hold the data that we can use to populate the data for the model.

```
//getting facebook access and anfo start Benyam
var fbClient = new FacebookClient(loginInfo.ExternalIdentity.Claims.FirstOrDefault(x => x.Type == "urn:facebook:access_token")
    .Value);
dynamic FbInfo = fbClient.Get("me", new
{
    fields = "first_name,last_name,gender,id,name,email,picture,location,birthday"
});

var picture = (((JsonObject)(JsonObject)fbClient.Get("/me?fields=picture.type(large)"))["picture"]);
var urlPic = (string)((JsonObject)((JsonObject)picture)["data"])[0]["url"];
Session["prp"] = urlPic.ToString();
//getting facebook access and anfo end Benyam

return View("ExternalLoginConfirmation", new ExternalLoginConfirmationViewModel {
    Email = FbInfo.email,
    Name = FbInfo.name,
    Gender = FbInfo.gender,
    Picture = urlPic,
    DOB = FbInfo.birthday,
    Address = FbInfo.location
});
```

---

<sup>10</sup> <https://docs.microsoft.com/en-us/azure/architecture/multitenant-identity/key-vault>

### 7.3.2 Booking with Full Calendar (BN)

Grundtvigs Dyreklinik wanted a calendar to display the booked and un-booked appointment-schedule that both the customers and admin can use to handle appointments. If we were to code this requirement from scratch, it would be very time consuming. Instead we used the FullCalendar which is a JQuery library. The FullCalendar is a JavaScript and CSS file package that can be downloaded into a project. The downside of this kind of library is, if you just want to use some aspects of the package, you would have to download all of it into the project, which can result in a slow server and it also takes some space.

Implementing FullCalendar was easy because we already knew where all the components was going to be placed. The hardest thing was that this was a library we had no knowledge of, therefore it took some time to read about the documentation and implementing the requirement. We started by downloading the JQuery library called FullCalendar, this include some CSS and Javascript file that we put on the application folder named Content and Script.

In ASP.NET there is something called Bundling, which reduces the amount of server request the browser has to make, hence making the application faster. Basically, what we did was instead of letting each CSS or Javascript file be requested by the browser. We included it all in the bundle file that would be send in one request. By doing this we have reduced the response time of the webpage almost by half. See the picture below for creating a bundle of CSS and Javascript file for FullCalendar and datepicker.

```
//jQuery fullcalendar and datepicker plugin css
bundles.Add(new StyleBundle("~/Content/fullcalendar").Include(
    "~/Content/fullcalendar.css",
    "~/Content/themes/base/jquery-ui.css",
    "~/Content/jquery-ui-timepicker-addon.css"));

//jQuery fullcalendar and datepicker plugin js
bundles.Add(new ScriptBundle("~/bundles/fullcalendar").Include(
    "~/Scripts/moment.js",
    "~/Scripts/fullcalendar.js",
    "~/Scripts/jquery-ui-{version}.js",
    "~/Scripts/jquery-ui-timepicker-addon.min.js"));
```

Now that the plugin has been downloaded and the files was in place in the application, it was now time to configure and use the plugins. There are some parameters and javascript code that need to be coded so that the FullCalendar can display and manipulate data. To do this we made a custom javascript file where we specify our configuration and data for the FullCalendar. Once the Javascript file was ready to display some data specification on our view, we simply made a reference in the HTML file. We made these separations because we wanted to have the option of replacing the plugin without having to restructure the whole application, if this was something to be done all we need to do is replace the javascript file.

As it can be seen in the picture below, the FullCalendar need an event of data that it can show on the calendar in our case we got the data from our method called GetBookings which is in the Booking controller class. Other than that, we also specify the business hours and days. Below is a picture of how we display and retrieve the event data.

```
$('#calendar').fullCalendar({
  //theme: true,
  contentHeight: 400,
  defaultDate: new Date(),
  header: {
    left: 'prev,next, today',
    center: 'title',
    right: 'month, agenda, basicWeek, basicDay,'
  },
  defaultView: 'month',
  timeFormat: 'H(:mm)',
  allDayText: 'GrundtvigsDyreklinik',
  displayEventEnd: userName,
  title: true,
  eventLimit: 4,
  firstDay: 1,
  slotLabelFormat: 'HH:mm',
  slotEventOverlap: false,
  maxTime: '18:00:00',
  minTime: '09:00:00',

  businessHours: [
    {
      dow: [1, 2, 3, 4, 5],
      start: '09:00',
      end: '18:00'
    }
  ],
},
```

Getting the data for the event happens on the controller, it is also here that all the data manipulation takes places. What we have in the view are only the raw HTML structure and the javascript file is an API caller.

## 7.4 Version Control *(MI)*

As we told in the earlier chapters, VSTS was a very new tool for us, but as soon as we read a little bit of documentation and tried making a test repository, it gave us an insight in how to use this version control tool.

We had created the test repository where we tested how to connect to the source control from Visual Studio and how to commit, push and pull changes. During our tests we encountered many errors due to inexperience. This resulted in the repository being destroyed and therefore we had to create a new repository. After we had gained confidence in using the version control tool in VSTS, we could then start to create the real repository.

The first thing we did was to create a new VSTS project called Grundtvigs Dyreklinik and inside this project we created a new repository for our code.

Changeset	User	Date	Comment
30	Mert Ince	28/11/2017 11:04:54	Slut Sprint 2
29	Binyam Neamen	27/11/2017 19:47:43	layout and identity fix
28	Binyam Neamen	26/11/2017 20:30:09	Some view changes
27	Binyam Neamen	26/11/2017 20:22:59	Fixed buggs US Bookings Edit
26	Elvis Saronjic	24/11/2017 22:56:25	Fixed Treatments Duration(datetime)/view
25	Binyam Neamen	24/11/2017 15:52:06	Coded US08 - Book a treatment
24	Mert Ince	24/11/2017 15:28:52	Changes to edit pet
23	Elvis Saronjic	24/11/2017 14:55:21	...
22	Elvis Saronjic	24/11/2017 13:50:03	Pet - Create and Edit View instead of textfield we have made a Dropdown
21	Elvis Saronjic	24/11/2017 13:08:23	Fixed a bug with the Pets Gender
20	Elvis Saronjic	24/11/2017 11:42:45	Coded US 06 - CRUD Pet
19	Binyam Neamen	21/11/2017 15:15:23	Testing if Booking Calender is compartible with VS 2017
18	Mert Ince	21/11/2017 11:38:30	Start Sprint 2

Look at appendix "Version Control" to see all commits.

After we had configured and connected to VSTS in Visual Studio, we started programming the system. The good thing with using this version control was that we didn't need a third-party software, like GitHub, but could actually use VSTS which was integrated with Visual Studio. You can either from the web browser or inside Visual Studio see the changes made. In Visual Studio you can view the changes in a more detailed way, e.g. if we need to know exactly in which class and where in the code there has been made any changes.

Even though this was the first time using VSTS we quickly learned how to use it with committing, pulling and pushing changes. Some improvements we could have made, were describing the commits more in details.

## 8. Test

This is the chapter where we will show the tests we have performed and documented. We have tried to make many different tests as possible to ensure that the quality of our product (system) was acceptable. By making those tests we also saved time, that made it easier for us to find bugs and errors.

### 8.1 Test Cases *(MI & ES)*

A test case is a way to determine or make sure that a feature or a specific function works as it should on the system. This way test cases can help us validate the system and make sure that the functional requirements are met.

We created the test cases based on our user stories we made in the earlier stage, where a single user story can sometimes contain several test cases. We believe that we have covered all the possible test scenarios. We have created our test cases in Visual Studio Team Services and linked them to the user stories we created. VSTS have made it easier for us to keep track of our test cases and later on test them.

Here is a part of our test cases we made: *(zoom in)*

ID	Title	Summary	Test Steps	Precondition	Expected Result	Actual Outcome	Status
TC01	Employee - Login	Employee has to login to the site in order to administrate the site.	1. On top navigation bar, press 'Log in' 2. In email field enter: 'employee@gd.dk' 3. In password field enter: 'test1234' 4. Press 'Log In' Button	Has to be registered	The site should authorize the employee, and redirect to the homepage.	Login & redirection works	Passed
TC02	Employee - Log off	Employee to be able to log off the system.	1. On top navigation bar, press 'Log off'	Has to be logged in	The site should log off the employee, and redirect to the login page.	Works as expected result.	Passed
TC11	Employee - Create specy	Employee has to create specy in the system for users to be able to create their pets	1. On navigation bar, press 'Specy' 2. Click 'New Specy' 3. Write specy name: 'Dog' 4. Click 'Save' button	Has to have admin rights and be logged in	The newly added specy should be added to the list of species	Works as expected	Passed
TC12	Employee - Update specy	Employee has to update specy in the system if there is any changes.	1. On navigation bar, press 'Specy' 2. Click 'Edit' for the specy that needs to be updated 3. Write specy name: 'Cat' 4. Click 'Save' button	Has to have admin rights and be logged in	The specy should be updated to in the list	Works as expected	Passed
TC20	User/Customer - Register	Customer should be able to register himself to in the system.	1. On navigation bar, press 'Register' 2. Fill in the information name: 'Mert' 2.1. Phonenumber: '30254896' 2.2. Email-address: 'nce@live.dk' 2.3. Password: 'testpassword' 2.4. Confirm password: 'testpassword'	None	The user should created in the system and be redirected to the account page.	Works as expected	Passed
TC17	User/Customer - Create pet	Customer should be able to create their pets in the system.	1. On navigation bar press 'My Account' 2. Click 'Create Pet' 3. Fill in the information with name: 'Test' 3.1. Choose specy: 'Hund' 3.2. Choose race: 'Golden retriever' 4. Click the 'Save' button	Has to be logged in.	The pet should be created and visible for the users account.	Works as expected	Passed

See the full list of test cases: <https://goo.gl/XUNwvw>

#### Examination of a test case:

In one of our test cases (**TC17**) we see when a customer/user creates a pet in our system. We can see that this test case has a precondition, where the user/customer has to be registered and logged in the system, in order to create his/her pet. In this test case we expect (expected result) that the users pet will be created and be visible in users account page. After we went through all the test steps, the actual outcome was as we expected it to be, so there for the status for this test case is; passed.

23 17 - Create Pet

Mert Ince 0 comments Add tag

Steps

On navigation bar press 'My Account'

Click 'Create Pet'

Fill in the information with name: 'Test'

Choose specy: 'Hund'

Choose race: 'Golden retriever'

Click the 'Save' button

The pet should be created and visible for the users account.

## 8.2 Traceability Matrix (MI)

Traceability matrix is a way to trace requirements, where you document the links between the requirements and the product being developed. Traceability matrix is usually in the form of a table to show the relation between requirements (*in our case user stories*) and the test cases. This means that by making the table we ensure that all requirements are covered by minimum one test case.

A traceability matrix is a very useful way for you to get an overview of the functional requirements and which user story to test. By using traceability matrix, it will also get much easier to track the test cases of a specific requirement, if for instance there have been any changes in the requirement that affects the test cases.

### Traceability matrix for Grundtvigs Dyrekliniks system

	User story 01	User story 02	User story 03	User story 04	User story 05	User story 06	User story 07	User story 08	User story 09	User story 10	User story 11	User story 12	User story 13
Test case 01													
Test case 02													
Test case 03													
Test case 04													
Test case 05													
Test case 06													
Test case 07													
Test case 08													
Test case 09													
Test case 10													
Test case 11													
Test case 12													
Test case 13													
Test case 14													
Test case 15													
Test case 16													
Test case 17													
Test case 18													
Test case 19													
Test case 20													
Test case 21													
Test case 22													
Test case 23													
Test case 24													

■ = Passed 
 ■ = Failed/not implemented

## 8.3 Usability Testing (MI & ES)

Usability testing is known to be a black box testing technique where the functionalities are in the focus for the test. We will use hallway usability testing, to test the GUI and the flow in general. The hallway usability test is where we find a random person on “the hallway” and have them test the system. The test-persons aren’t by themselves important because we will not choose them specifically for who they are, as the intention for us with this usability test, is to find how user friendly our system is.

It is also our intention to be sure that the flow of the system appears naturally to our users and that the GUI is understandable from the users’ perspective.

We will make a task list that we will give to our testers and let them try the different steps/functionalities in the system. We will observe and analyze the amount of time that it takes for the users/testers to complete the different task, which will in the end give us an understanding of what needs to be changed or modified in the GUI. We will also talk with the testers about their experience with the system and if they have any suggestions.

Tasks list:

1. Register to Grundtvigs Dyreklinik
  - a. Login
2. Create Species
  - a. Fill in required fields
3. Create Race
  - a. Fill in required fields
4. Create a new pet
  - a. Fill in required fields
5. Create a treatment
6. Edit treatment
7. Create a new booking in the system
8. Delete a booking

**While doing the usability test with our testers we have noticed:**

Task 1 (*Register to Grundtvigs Dyreklinik*): That the registration for the users was easy, and they could find the register button fast enough.

Task 2 (*Create Pet*): All the testers was satisfied with the function, even though they were happy with the simple design that this part wasn't complicated.

Task 3 (*Create Species*): The response was positive and it felt natural for our users when creating a new specie.

Task 4 (*Create Race*): This task was also very simply achieved by our testers. Therefor we were satisfied with their overall response.

Task 5 (*Create Treatment*): The treatment was like the other tasks (*Task 2, Task 3 and Task 4*) pretty straight forward for the testers.

Task 6 (*Edit Treatment*): The edit button was to one tester, misplaced and was took a little bit longer time, than it should have.

Task 7 (*Create Booking*): This task was easy for the most of the testers, but there were a few testers that found the time-picker a bit annoying and old school. There was another suggestion from one of the testers, that it should be possible to book a date, by just clicking on the calendar. During this task we actually found a bug in the booking where you actually could book another time in between a booked interval. So, for example if you booked a time from 9 to 12 you could book another one from 10 which is an issue.

Task 8 (*Delete Booking*): Delete booking was a bit confusing for one of the testers, mainly because he wanted an extra GUI for the bookings. Other than that, it was an easy task for most of them.



**Response from some of our testers:**

Adem Ökmen (*Test-person 1*): “Overall the app has a nice GUI and flow, but are missing some design, with colour and effects. The booking is a bit confusing that you can’t pick the date and time from the calendar but have to write it in the date picker. The interface language was also mixed with English and Danish, the whole system should be changed to Danish.”

Sadek Hamed Al-Emara (*Test-person 2*): “The system has some good functionalities. The navigation and flow were easy to follow with. There should maybe be some changes to the GUI, by adding some more colour and changing some of the names for the buttons, so it’s easier to understand.”

Karen Tjørnelund (*Test-person 3*): “The application need some changes with the language, and the formulation of the different errors, pop-ups and labels. Otherwise the flow of the app is fine.”

**Changes made after the usability test:**

After we have tested and got all the response we needed from our testers, we decided to make some small changes to the GUI, in order to make the app a bit simpler, user friendly and with a better flow. We changed some of the placements of the buttons after the response we got from our testers, and changed the interface language to Danish. We didn’t notice that there was a mix up with languages until we had our testers test the app. We also fixed the issue that you could book a time in an already booked interval.

Even though there was some critics to the design/colour, we will not be focusing on that (*as we told in the earlier chapter*). Our main purpose with the GUI was to make it simple for our different users and make a good flow in the application that was easy to follow. We believe that we have achieved our objective by making the testers evaluate the system and made sure that the app became user friendly.

## 8.4 Unit Test (MI)

Unit testing is a software testing method, where you test the single units within a module, which can also be called a module test. The purpose with unit test is to isolate each separate part of the program, and test that the individual components/functions work. This way unit testing ensures that the units fulfils its purpose, before integrating.

The programmers usually code the unit test from scratch, but you can also autogenerate or use some tools/frameworks. There are a few different frameworks when it comes to coding unit tests, such as; Nunit, Flexunit and many more. We chose to use Nunit which is integrated in Visual Studio.

We didn't make unit test for every method/feature we had, because of the lack of time and the closing deadline. So, we choose a few essential parts, which had the highest value for us and therefor made unit test only for those.

The majority of our tests are based on our methods that have a connection to our database. By testing that we can make sure that our code works and that we have connection to the database.

These are the outcome of the unit tests we made:

### ▲ Passed Tests (5)

✓ GetPetDetails	3 sec
✓ GetRaceDetails	46 ms
✓ GetTreatmentsListFromDB	45 ms
✓ SpeciesValidationOnCreate	< 1 ms
✓ TreatmentValidationOnCreate	19 ms

The first unit test we made was to test the "Details()" method in the "PetsController" class which gets a specific pet from the database by giving the id of the pet.

```
[TestMethod()]
public void GetPetDetails()
{
    PetsController pets = new PetsController();
    ActionResult result = pets.Details(3);
    Assert.IsInstanceOfType(result, typeof(ViewResult));
}
```

The method above tests if you can get the pet details for the pet with id 3, if we can not get the details for the pet it would either mean that the method doesn't work or that there is no connection to the database. It is also very important when testing such methods that tables in the database are populated, if not it can also result in the test to fail. We will not be throwing an exception in this test method, because of the already coded exception/error handling in the "PetsController" class.

Another interesting unit test we made was to test the validation on our different properties in the “Treatment” model class. This was one of the simple unit tests, to test if the properties validation worked as intended. The “Treatment” model class contains “Name” and “Comment” properties, and these properties have validation, that will give an error if the properties are left empty.

```
[TestMethod()]
public void TreatmentValidationOnCreate()
{
    CheckPropertyValidation cpv = new CheckPropertyValidation();
    var trm = new Models.Treatment
    {
        Name = "",
        Comment = "Dette er en behandling for dyr med laser ",
    };
    var errorcount = cpv.myValidation(trm).Count();
    Assert.AreNotEqual(0, errorcount);
}
```

With this method we actually had to make a new class in the unit test project called “CheckPropertyValidation”, which represents a container for the validation result. Here we are testing if the validation works, by counting the errors in the “container”. If the error count stays at 0 even though the “Name” is empty, it means that the validation doesn’t work.

Unit test is a good way for us to check the different methods and units, to see if they work as intended, and therefore we can conclude that the code behaves as designed.

Unit testing is very useful when it comes to saving time on finding bugs and errors, but as said earlier we couldn’t make unit test for everything and had to be satisfied with testing the most core functions.

We made our unit test very late in the project, both because of the unpredictable events and we didn’t prioritize it as high as the other activities. But the thought behind it, is that we are going to use these unit tests if we are going to continue with coding the system. That way we can make sure that the new code doesn’t affect the system.

## 8.5 Automated Functional Testing *(ES)*

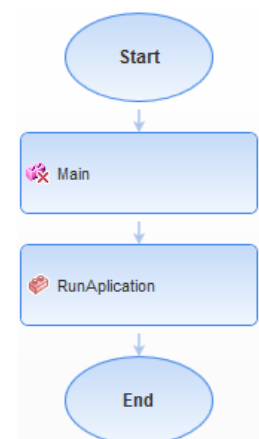
Automated testing can be an important part of software development and by doing it; you allow teams to detect problems early, you avoid repeated manual work and you save time on running tests over and over again, this helps the people in the project save time and money. You also avoid factors such as human error where a person could miss an important step and thereby fail to identify a bug in the application. This philosophy is also used when implementing Continuous Integration, which we have used some part of.

What automated functional testing does, is that it runs a script, that automatically go through the systems interface and checks if the functionalities works properly. That also means it only test what the application does and it is not concerned with the application's internal details.

The way we have used automated functional testing is like the development practice Continuous Integration, where every time we integrate code into our main repository, which we would do at the end of every sprint. We would then verify it by an automated test and if it passed that would mean that the integration was successful.

To build the script we have used a tool called HP Unify Functional Testing, which uses the scripting language visual basic. Here we have made a script called "Grundtvigs\_Dyreklunik" that goes through all of the applications features.

*(Pic To the right represent the overall view)*



The first thing the script does, is choosing which web browser you are going to run the script with and which web address to run it on *(This is all done in the "Main" class).*

```
Setting("DefaultTimeout") = 4000 'set in milliseconds

a = InputBox("Hvilken browser vil du teste med?" &
vbCrLf & "IE = Internet Explorer" & vbCrLf & "ff = Firefox" & vbCrLf & "c = Chrome", "Vælg Browser", "c")

Select Case LCase(a)
    Case "ie" SystemUtil.Run "iexplore", "http://grundtvigsdyreklunik.azurewebsites.net/"
    Case "ff" SystemUtil.Run "firefox", "http://grundtvigsdyreklunik.azurewebsites.net/"
    Case "c" SystemUtil.Run "chrome", "http://grundtvigsdyreklunik.azurewebsites.net/"
    Case "" ExitTest
end Select

'Kør script
RunAction "RunApplication", allIterations

'Afslut test
ExitTest'
```

Testing the application on different web browsers to see if it is compatible is an important step, as we do not know which web browser the customers will use. Here we code the script to run in Internet Explorer, Firefox and Chrome. And thereafter it goes to the class "RunApplication" and when it has gone through that it ends the test.

The "RunApplication" is where it goes through all of the application features and since it takes a lot of space we are only going to show a section of it (Look at Appendix "Automated Functional Testing" to see all the

code).

The code snippet below shows how we have scripted the features of logging in to the system and how to create, edit and delete a species.

```
|'***** - Log in *****|
Setting("DefaultTimeout") = 4000 'set in milliseconds

Browser("Grundtvigs Dyreklinik").Link("Log in").Click
Wait(1)
Browser("Grundtvigs Dyreklinik").WebButton("Facebook").Click
Wait(1)

'***** Create Species *****|
Browser("Grundtvigs Dyreklinik").Link("Species").Click
Wait(1)

'***** Create Species *****|
Browser("Grundtvigs Dyreklinik").Link("Create New").Click

Browser("Grundtvigs Dyreklinik").WebEdit("Name").Set "Hest"

Browser("Grundtvigs Dyreklinik").WebButton("Create").Click
Wait(1)

'***** Edit Species *****^
Browser("Grundtvigs Dyreklinik").Link("Edit_Species").Click

Browser("Grundtvigs Dyreklinik").WebEdit("Name").Set "Aligator"

Browser("Grundtvigs Dyreklinik").WebButton("Save").Click
Wait(1)

'***** Delete Species *****^
Browser("Grundtvigs Dyreklinik").Link("Delete_Species").Click

Browser("Grundtvigs Dyreklinik").WebButton("Delete").Click
Wait(1)
```

UFT is firstly about identifying all the objects in the application and thereafter putting them in the Object Repository. The part where it says “Grundtvigs Dyreklinik” represent the entirety of the web page/browser, it tells which location you are at. Thereafter you go deeper and find what is inside of that browser, in our case here, there are objects like a “Link” that takes you to the **Create New** page, a text field where you input **Name** and a **Save** “WebButton” that creates the Species to the database. The way the Object Repository works is like a hierarchy, where every page would be a sub category of the browser, and these sub categories would then have objects in them. In our script, we have tried to make it as simple as possible and therefor we removed the sub category since it was not needed, whereas normally, you would have a more advance hierarchy order.

Our UFT Script has helped us test the application every time there was a new build for the system, here the script has run and tested to see if the newly integrated components have not damaged the working application. The script has also helped us detect problems and bugs on the different web browsers. Instead of testing it manually on every web browser (*like a normal end-user*), we chose to use UFT automated script that has made the process of testing faster and easier for us.

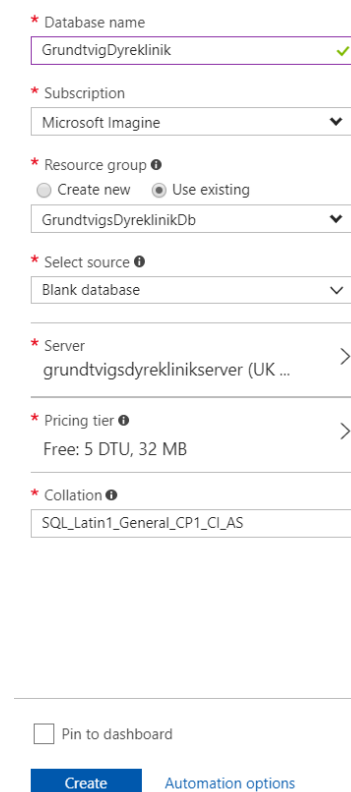
## 8.6 Deploying on test environment (Azure) *(MI)*

After we made the most of the system we wanted to deploy it and test it on a server, but as Grundtvigs Dyreklinik already had their own server, we had to use the resources we had in our hands. That's why we chose to deploy the system and database on an Azure server that we would have access to.

The first thing we did was to create a new server on Azure and create a new database within it, called "GrundtvigsDyreklinik". We first specified the name of the database and the Azure subscription. Normally it costs to have access to Azure, but through our school we have gotten a free subscription. Though the free subscription comes with a few limitations, such as; a storage limit for the database and not being able to use some of the Azure functions, e.g. Azure performance test.

Thereafter we selected a blank database since we already had one locally. The only thing you need now, is to migrate our existing database to Azure.

After we had created the server, database and selected our resource group, we then published our system (*Our Visual Studio project*) to Azure. We did this from Visual Studio where we selected our resource group and added the connection string. The connection string was taken from Azure, which made it possible for us to connect to the database.



The screenshot shows the 'Create new database' form in the Azure portal. The form includes the following fields and options:

- Database name:** GrundtvigDyreklinik (with a green checkmark icon).
- Subscription:** Microsoft Imagine (dropdown menu).
- Resource group:** Create new (radio button) / Use existing (radio button, selected). GrundtvigsDyreklinikDb (dropdown menu).
- Select source:** Blank database (dropdown menu).
- Server:** grundtvigdyreklinikserver (UK ... (dropdown menu).
- Pricing tier:** Free: 5 DTU, 32 MB (dropdown menu).
- Collation:** SQL\_Latin1\_General\_CP1\_CI\_AS (dropdown menu).

At the bottom of the form, there is a checkbox for 'Pin to dashboard' and a blue 'Create' button. To the right of the 'Create' button is a link for 'Automation options'.

Overall it was easy to deploy to Azure and run the system on a real server rather than just locally. By having it on a server you get a more efficient way of testing the system for performance and response time. So even though there might be deviations from the production server, what it does give us, is a good overview on how the system functions/performs on a server.

## 8.7 Performance test (MI)

Performance testing is a way to find the limit of the system and see how the system reacts to different kinds of load. This is a way to test both the responsiveness and usage speed in a system/application.

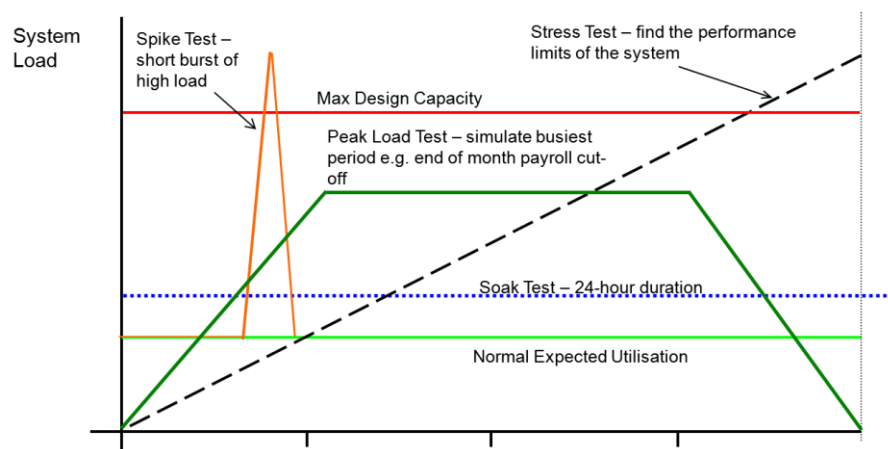
There's four different performance tests, each of them will give an understanding of what your system can handle.

**Spike test:** The spike test is performed by increasing the number of users suddenly by a large amount for a short time. The purpose with doing this is to see if the system is able to sustain the workload, and get back to normal without any problems.

**Soak test:** Soak testing is known as endurance testing, this is done by giving the system continuous load for a longer period. The duration is usually 24 hours, and it is a great idea to monitor the system, to see if there's any memory leaks.

**Load test:** Load testing is a way to understand the systems behavior by simulating the busiest period for the system under a specific load.

**Stress test:** Stress testing is basically to find the performance limits of the system. This test will be done until the system experience slow responses and uses a very high CPU. This can sometimes also result in the system to crash.

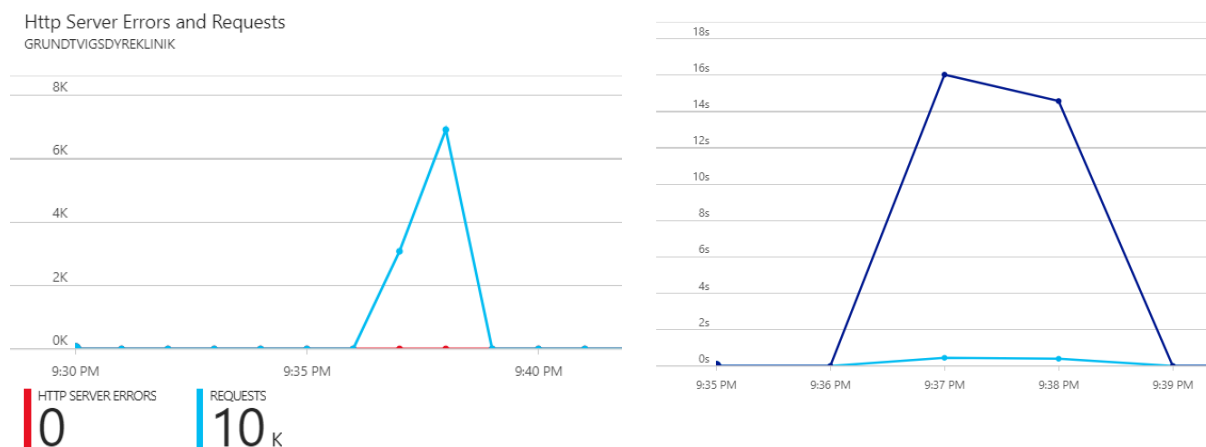


### 8.7.1 Test results

To test our system performance, we have run a couple of different performance tests to see what our system can handle and how it reacts. We wanted to use the tests, to verify and make sure that our system was stable and that we could fulfill our requirements regarding performance. We analyzed the test results in Azure Portal by looking at the different graphs.

#### Our spike test

We started with making a spike test to see how the system would react to 200 users retrieving information from the web application. We did this by looping it 50 times, which is 10.000 request ( $50 \times 200$ ). Our goal was to be able to handle maximum 150 users at the same time without the system crashing. We tried with 200 which was over the maximum design capacity.



The first graph shows that there has been a load of 10.000 requests and none http server errors in 3min. The second graph shows the CPU Time (dark blue) and the response time (light blue) for the same test.

We were satisfied with the results as it turned out that the system could handle short bursts of high load without making a huge effect on the response time and CPU time ("CPU time is the exact amount of time that the CPU has spent processing data for a specific program or process."<sup>11</sup>).

The CPU time was at 16sec at the time we were over the maximum design capacity and the response time was 0,4sec, which is still a very good response time. Now we can determine that the system can handle much more, and we will find out where the limit is.

#### Our load test

In this test we will simulate the busiest time for Grundtvigs Dyreklinik system, where we will simulate 150 users, that will use the system in 3,5 hours, from 11.00 to 14.30, which is the busiest time for Grundtvigs Dyreklinik. After we had run the load test we could conclude that the system could easily handle 150 users in the busiest period of work, with a great response time (0,1sec), and without any errors or issues.

<sup>11</sup> <https://www.techopedia.com/definition/2858/cpu-time>



### Our stress test

We will do a stress test to find the limit of our system. This will give us an overview on how much the system/web application can handle. We will start with simulating 10.000 users doing a http GET (requesting data) in 3min. We will see if the system can handle it or if it will be slowed down by the many requests.

We have stress tested the app by simulating with 10.000 users, which slowed the app a bit down, but didn't crash. The app response time went from 0.1sec to 9sec and the average CPU time was 2,2min. We then tested with 50.000 users at the same time, where after 4min the response time went up to 14sec and afterwards made the application crash.

*See rest of our performance test results in Appendix: "Performance test"*

## 8.8 Acceptance Testing *(BN)*

This is the last test that it is done, for the product acceptance we used the minimum business requirements we made with Grundtvigs Dyreklinik. The min. requirements are; Registering and logging in to the system, CRUD operations for Pet, Species, Race, Treatment and Bookings. What we were looking for was if the customer could fulfil the functions that we were asked to implement with a satisfied result. On this stage we had already performed functionality and usability test where we made some changes and adjustment to the system, so all that is left is, to go through the requirements one at a time with the customer to have them accept the application before handing it over to Grundtvigs Dyreklinik.

This was not a guided test and it wasn't for us who build the system, for that reason we made the test cases in as non-technical terms as possible, the test cases were a straightforward questions or tasks like; "register user to the system". What we wanted to get from these tests was the acceptance from the product owner, that the given tasks could be performed.

In general, this test helped us to fulfil our deal of the contract we had with Grundtvigs Dyreklinik. After this test there cannot be misunderstanding between us because we had gone through all the business requirement that we had an agreement to fulfil and gotten their acceptance of the product. There were some extra features that we implemented, such as login with Facebook API which wasn't in the minimum requirements.

Test	Test Procedure	Acceptance Requirement	Critical (Yes/No)	Test Result (Accepted/Rejected)	Comments
1	Create User	Yes	Yes	Accepted	
2	Log In User	Yes	Yes	Accepted	
3	Create Pet	Yes	Yes	Accepted	
4	Create Species	Yes	Yes	Accepted	
5	Create Race	Yes	Yes	Accepted	
6	Create Treatment	Yes	Yes	Accepted	
7	Create Booking	Yes	Yes	Accepted	
8	Log In with Facebook	Yes	Yes	Accepted	
9	Edit Pet	Yes	Yes	Accepted	
10	Edit Species	Yes	Yes	Accepted	
11	Edit Race	Yes	Yes	Accepted	
12	Edit Treatment	Yes	Yes	Accepted	
13	Edit Booking	Yes	Yes	Accepted	
14	Delete User	Yes	Yes	Accepted	
15	Delete Pet	Yes	Yes	Accepted	
16	Delete Race	Yes	Yes	Accepted	
17	Delete Treatment	Yes	Yes	Accepted	
18	Delete Booking	Yes	Yes	Accepted	

## 9. Handing over the System *(MI & ES)*

Being a bachelor project, Grundtvigs Dyreklinik is well aware that they will only get a prototype, which includes the minimum requirements. This will though act as a preview of the final outcome, if they want us to continue developing the system.

We will at this point not be able to deliver any maintenance guide for Grundtvigs Dyreklinik as the product isn't in its final shape. We will instead make a support-agreement, so they can call us when they need help or have questions regarding the system. However, we have promised that we will hand over all of the documentation we have made so far and write comments on the code, as to what it does. The plan here has always been to give everything to the wife's spouse, who himself is a developer. As of now, we are done with this project and If they want us to continue with making and deliver the product with all the requirements implemented, it will not be for free, as we are soon to be graduates.

We are handing over the system prototype that has been presented in the acceptance test, with the features that we have made so far. They have showed interest in the system, which gave us the understanding that they want to continue with it, if they have the time and resources. But this will not be clarified until we are done with our bachelor education, where we afterwards will get a final reply whether the husband will continue with it or not.

## 10. Conclusion

We dove into this project in the hope of making a better booking system to Grundtvigs Dyreklinik, so that it didn't have the flaws as their current one. This was an opportunity for us to work in a real project, and to make a product that could solve the problems of our customer, while also demonstrate what we have learned during our education and make use of it.

To manage the project we have used scrum, which is the project management method that we had the most experience and knowledge in, this reduced the risk for the project to scatter. And overall, we found it easier to develop the system when we had guidelines and a plan to follow.

The planning was one of the most essential aspects for us in this project, because if we didn't make a reliable plan we wouldn't have been able to keep track of the process, which could have resulted in us not being able to deliver a product. Using scrum turned out to be a good decision which has helped us to manage the project, but we also felt that some of the techniques in scrum wasn't as relevant for us, since we are a small team.

To implement and manage the database we have used entity framework. This has helped us build the database up much faster and it has been easy to maintain and modify. The reason it turned out to be successful using entity framework is because everyone in the group has experience with it.

In order to ensure the quality of product, we made various tests. The tests were based on both the functional and non-functional requirements, where we made sure that they were fulfilled. Testing a product both during development and when done, is important in order to make the system free of errors/bugs. Creating and applying the different tests helped us in our development of the product, where we saved a lot of time on finding errors. These tests also gave us an insight in the performance of our system and what others think about the interface. This way we knew what changes the product needed, so we could ensure a good quality.

We promised that our system would have a faster process e.g. when creating a user and an overall better flow. We believe that we have accomplished what we promised, where we have built all the core functionalities and made the flow more intuitive. Having in mind that our system is only a prototype, that means it can in the future development be improved.

This has been a very constructive project for us, where we used the knowledge and experience that we have from our bachelor education. Most of the tools and techniques that we used throughout our project, we have learned in our education and it has helped us in our development.

We have in this short period of timeline managed to create a system to our customer, which both parties are satisfied with. This project has made us aware, that we are finally ready to work in real projects for companies.

*Here is a link to our application: <http://grundtvigsdyreklinik.azurewebsites.net/>*

## 11. Reflection and Perspectivation

During this bachelor project we used lot of the methods and techniques that we had learned during our education in KEA. In this chapter we will reflect on the different experiences we had in our project, and talk about what we could have done differently.

### 11.1 Scrum vs Waterfall

In this project we used SCRUM which is an agile development model to manage our project, and the reason we chose scrum was mostly because we already had a lot experience with it. By using this project management framework, it was easier for us to foresee the outcome of the project. If we had e.g. used waterfall, which we don't have much experience with, maybe we wouldn't even be done with the project.

The good thing with waterfall is that the discipline is enforced so that the phases have a start and end-date, which reduce the risk of not being able to keep a deadline. But unlike scrum it's very hard to get back to any phase, if e.g. the development team finds out that there was missing a feature from the requirements phase, it will be expensive and hard to go back to a phase in order to fix it. Another reason we didn't chose such a model, was that we are used to iterative and flexible project management. This wouldn't have been possible with a waterfall approach to the project.

With that said, scrum also had its flaws. Since we are a small team we felt that all the ceremonies weren't necessary. Because when you're a small team as ours, where we constantly sit together and talk about; What we are doing? What we are going to do next? What to improve upon?

### 11.2 Usability Testing

When we decided what kind of usability test we would make, we had in mind that we didn't have much time and therefor had to decide fast. We chose to do a hallway-usability test, as this is the kind of usability test that won't take much preparations. If we had done Guerrilla testing we would have other things in the focus, where you have to ask yourself 4 questions when making this test; "What shall we test? Where will we test? With whom will we test? How will we test? Guerrilla test is also easy and cheap to perform just like the hallway usability test.

David Peter Simon<sup>12</sup> says that *"Guerrilla usability testing is a powerful technique"*, if it's made correctly. We also agree with David when he tells that Guerrilla usability testing can be done with almost anything *"from concepts drawn on the back of napkins to fully functioning prototypes"*. Even though guerrilla testing is pretty easy to make we didn't chose it because, it would take some more preparation compared to the hallway usability test.

---

<sup>12</sup> <http://www.uxbooth.com/articles/the-art-of-guerrilla-usability-testing/>

### 11.3 Version Control

In this project we actually used a new version control tool, that we didn't have much experience with. We started with discussing all the problems we had during our education with using Git, and therefor came up with an alternative tool. One of the group-members had actually used Visual Studio Team Services (VSTS) at their internship, so he persuaded us to make use of it in this project.

We quickly learned that it wasn't easy to use, so we had to read a lot of documentation and ask people that had good knowledge with it. We for instance didn't know how to compare the local code with the one in the repository, right until we asked an expert. VSTS is pretty simple to use if you know the different features and functions in it, but we learned that it will take some adjustment time if the person has never used it before.

Would we use VSTS in another project? Depending on the complexity of a project, it could still be useful, but we would prefer Git over VSTS, because if you know how to use Git, you will have more control over the repository.

### 11.4 Test Driven Development

One of the techniques we thought about using, but didn't do was Test-Driven Development (TDD). TDD is a development process where you make unit test (*test first*) before you begin with coding the system/application. The purpose with TDD is that you start with the test, and the test should fail as there's no code to make it succeed. Then we write code until the test passes, and at last you refactor the code so it is clean.

This process could have benefit us much, if we started with doing test first. Here we would have saved time on testing the different methods, and if they work. The reason we didn't make use of this process was mainly because of our inexperience with TDD, so we were unsure about how long it would take. If we had more time with this project, we would definitely have used test driven development as it is a much faster approach to testing.

### 11.5 Risk Analysis

If we take a closer look at our risks, and how we managed it throughout the project. We can conclude that any of high risks didn't happen, both because of our mitigation and because we have lots of experience in making projects, therefor we didn't change any of the risks. It is always important to make risk analysis, because it make us aware of what could go wrong, and therefore you can prevent it from happening.

## 11.6 Improvements

When developing a system from scratch where you have a short deadline can result in, that the product is not in perfect condition. Therefore, this product can be considered a prototype with all the necessary implementations and parts, but not in its final form/shape.

That's why there can be lots of improvements both to the product and the project. Most of the improvements we can do to the project, is regarding the code and the application.

The improvements we can do is to;

- Improve the quality of the code, by doing it cleaner and adding more comments/explanations if e.g. there's other developers that's going to continue with it.
- Make the application flexible enough that it works from a phone browser, which we could have done if we had a bit more time to focus on the design/interface.
- Create more unit test methods, to test all the features/functions in the application.
- Add more languages to the application, so they can pick which language their interface is going to be.
- Use more time on the sprint retrospective and improve the process of the project management.

There is of course always something that can be improved, but having in mind that we had a limited timeframe, makes us satisfied with reaching our goal of making a fully functioning prototype. Our plan is to keep working on the system, if Grundtvigs Dyreklinik want us to continue with this project.

## 11.7 Final Remarks

The overall outcome of the project was how we expected it to be. We are satisfied with result, where we managed to implement at least the minimum requirements. To do that, we used the different techniques and methods, which we learned during our education.

This wasn't our first project neither was it the last, but every time we participate in a project we learn and can use the knowledge in the next one.

The start of the project was very important for us, when we were collecting the requirements at an early stage. This meant that we could begin as soon as we agreed upon the functionality with the customer. It's hard to get all the requirements right from the start, as the customer almost in every project comes up with new modified requirements. Therefor we are very satisfied that the customer didn't make much changes along the project. This gave us more time on being able to focus on coding the system.

Working on a real project made us realize that it isn't that much different from the projects we have worked on during our education. There is of course the reality of making a customer happy, but it also gave us insight and new knowledge on how to work on a "real" system. The knowledge we have achieved in this project will certainly be used in the future projects to come.

## 12. Bibliography

1. Brian Hambling, SOFTWARE TESTING An ISTQB–BCS Certified Tester Foundation Guide Third edition  
ISBN: 978-1-78017-300-9
2. Database Systems, A Practical Approach to Design, Implementation, and Management Fourth Edition  
ISBN: 0 321 21025 5
3. Pinto, Jeffrey K. Project management: achieving competitive advantage. - Fourth edition.  
ISBN: 978-0-13-379807-4, 0-13-379807-0
4. An Introduction to PRINCE2®, The best possible introduction to PRINCE2, By Frank Turley  
ISBN: 978 0 11 331188 0
5. Applying UML And Patterns - An Introduction To Object-Oriented Analysis And Design And Iterative Development, By Craig Larman, ISBN: 978 0 13 148906 6
6. <https://www.1keydata.com/datawarehousing/data-modeling-levels.html> - Copyright © 2017  
1keydata.com, Visited: 04.11.2017
7. <http://www.uxbooth.com/articles/the-art-of-guerrilla-usability-testing/> - © 2017 UX Booth,  
Visited: 15.11.2017
8. <https://software.microfocus.com/en-us/products/unified-functional-automated-testing/overview> -  
© Copyright 2017 EntIT Software LLC, Visited: 25.11.2017
9. <https://docs.microsoft.com/en-us/azure/vs-azure-tools-publish-azure-application-wizard> - © 2017  
Microsoft, Visited: 04.12.2017
10. <https://martinfowler.com/bliki/UnitTest.html> - © Martin Fowler,  
Visited: 10.12.2017
11. <https://fullcalendar.io/docs/> - © 2017 FullCalendar LLC  
Visited: 27.11.2017
12. <https://msdn.microsoft.com/en-us/library/bb924357.aspx> (Performance test) - © 2017 Microsoft,  
Visited: 12.12.2017
13. <http://jmeter.apache.org/usermanual/get-started.html> - Copyright © 1999 – 2017, Apache Software  
Foundation, Visited: 14.12.2017
14. <https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban> - ©2017. All Rights Reserved  
Smartsheet Inc. Patents and Patents Pending, Visited: 18.12.2017
15. <https://www.prince2.com/eur/prince2-methodology> - © Copyright ILX Group 2017,  
Visited: 18.12.2017

