Administration system for Aalborg Martial Arts

In the theme: Development of a Database System for a Specific Application

Group: Bi405F13 – Kenneth Korsgaard Gad, Jesper Itkin Lammert,

Christian Friis Lyngbo Andersen, Jeniith Jeyanthisivam

University: Aalborg University, Institute of Computer-Science, 4 th. Semester

05/29 2013

Title:	Administration system for Aalborg Martial Arts
School:	Aalborg University, Institute of Computer-Science, 4 th. Semester
Semester theme:	Development of a Database System for a Specific Application
Project period:	February 1st – May 29th 2013
ECTS:	15
Supervisor:	Tanvir Ahmed
Project group:	Bi405F13

_____
Jesper Itkin Lammert - Informatics

_____
Christian Friis Lyngbo Andersen - BaIT

_____
Jeniith Jeyanthisivam - BaIT

_____
Kenneth Korsgaard Gad - Informatics

SYNOPSIS:
In this project we have made an administration system for Aalborg Martial Arts.
The semester's theme is "*Development of a Database System for a Specific Application".*
The project includes a report with documentation of the system, work process, tools etc. and a CD with the system.

Pages: 86
Appendix: 7

Content

Chapter one: The Administration system

Introduction

This semester project got the title "*Development of a Database System for a Specific Application*"

The project is made in cooperation with Aalborg Martial Arts and the goal is to produce a product that in the end can be used by the gym. Aalborg Martial Arts organizes the largest kickboxing tournament in Scandinavia, called Danish Open, with 270 participants in 2012 and around 500 expected participants in March 2013. This case has been chosen because of our own connection to the organization through a member in the project group and the possibility to make a functional product.

We knew in advance that the organization has huge problems with too much paperwork and their tournament system. These problems will be elaborated further in this paper. But this already gives us a basic idea for implementing a database in the system.

In the next sections, we will look at the case for this project, explain what kickboxing is, look into their old system and define a problem statement, which we can work on.

Case description

As mentioned in the introduction this case is described as "*Development of a Database System for a Specific Application*".

The obvious choice would be to construct a website, where the users could connect to a database, but how this could work and alternatives will be looked at. The dynamic content will be programmed in mainly object oriented PHP code. The database will be developed in MySQL, which is a free alternative to MS SQL, and highly compatible with PHP.

Beside the practical work with coding, we get to work with an actual association and learn how to meet their demands. Our contact person in Aalborg Martial Arts is Nicolai Fedderholt. In this paper he will be referred to as our contact person or just Nicolai. Nicolai is a member of Aalborg Martial Arts and he is the person with the responsibility for organizing their tournaments. He knows everything about the gym, organizing tournaments and their current system(s).

Furthermore we will learn to use a software development method. This method should be picked specifically for our project. This development method will be used for organizing, analysing, implementing and evaluation. We will choose a method that fit our own requirements, so we can have maximum process.

Before moving on some knowledge about kickboxing and the current system is useful to know why we create the system as we do.

Kickboxing

There are three judges, two fighters and one referee in a kickboxing fight. The judges cast their judgment differently in the different disciplines, but they must reach an agreement. If they cannot however, the referee steps in and tell the judges that the fight is a draw, and they have to vote for a winner of the fight, since the outcome of a fight cannot be a draw. The judges sit on the side-line giving points and the referee is in the ring, making sure the fight rules are held, start and stops the fight and makes sure the fight does not come out of control. Kickboxing has three different disciplines, where different rules are applied for each discipline.

Light contact: In light contact, the gyms system is in use, where the judges click on a mouse to give points to either blue or red. A knockout is not allowed and will disqualify the fighter who did it.

Semi contact: In semi contact the referee stops the fight after each hit, and then the judges has to vote which one hit first, and to finally give a point to a fighter.

Full contact: In full contact a knockout is legal and will end the fight, and the fighter who got knocked out has lost. Each hit also gives either one or two points. The judge gives these points and in the end votes are calculated to determine the winner, if no knockout has occurred.

Furthermore, the fighters only fight against other fighters in their own weight class and gender. Each fighter also has a coach with on tournament days. His job is to make sure the fighters are ready and give water in the ring. He does not have any influence on the actual fights.

This gives us some key points, or attributes, we have to remember when creating the system; Fighters, judges, referees, coaches, disciplines, weight classes and gender.

Current system

Aalborg Martial Arts are currently using an internet-system called OKBR.net (Online Kickboxing Registration) to manage the tournament the gym is hosting, as well as maintain information on the fighters.

It is written in PHP, and the system is using CSS to control the styling.

The system is written in a way, such that everyone can create a new gym and thus register the fighters in the newly registered gym. After the gym has been added to the system, the person that registered the gym can start adding fighters to OKBR, in order for the fighters to join tournaments and events conducted through OKBR, such as Danish Open 2013.

If the gym has forgotten their current information, there is no way to restore the information, other than to sign up again and thus they have to re-enter all the information previously added into the system, making the gym appear twice in the system.

After a gym has been signed up, the gym can assign its fighters to the gym, by adding the fighters one by one. Each fighter contains the following information: Name, birthday as well as birth year and gender.

When entering the fighters into an event, the gym also has to add age-class (Junior, etc.), Weight Class (75 - 90, etc.) as well as discipline (Full contact, etc.). After the fighter has been added to an event, he or she cannot be edited, which means if you happen to enter the wrong information, the fighter has to be deleted and entered from scratch again.

If a fighter changes gym, the new gym has to enter the fighter's information, and the fighter has joined the new gym. The fighter is still registered in the old gym, unless the old gym actively deletes this fighter. This means that the system can contain an infinite amount of replica users, which could cause the database to work slower than it would normally.

It also makes requirements towards the gyms trivial, because they can just re-enter the data they previously entered, and they are in no way harmed. This can cause a serious problem concerning registration towards events, as one could imagine gyms join the same event several times – both by not being aware if the persons have been set to participate in the event, as well as how easy it is to

sign up and enter, which means that two or more persons from the same gym can make the fighters join the event.

At the event there is another big problem, especially with hardware. The judges use mice to click and give points. The mice disconnect a lot and the match has to start over. These points operate in another system, which is made for the matches. The system shows the fighters name and gym logo on a big screen. The logos don't have a defined size on the screen, so sometimes a big logo will fill 60 % of the screen, making everything move and the idea of the big screen useless. The system doesn't talk together with OKBR.net so everything has to be put in manually afterwards. So fighters and gym information's are put in at OKBR.net. Then printed out to make a tournament schema, then this information is put in a fight system, for the big screen and giving points and then afterwards put back in the fighter books (paper books) and in OKBR.net.

The administrator of any given event on the website, can at all times, print a registered fighters.csv sheet, containing all the information of the fighters that are, at that time, registered to the event. The fighters aren't sorted in any way, however, which means that the administrator must, by manual work, reorganize every fighter as well as make a structure in which the event itself can be held. This leaves the administrator with a lot of work, which the system has no way to help to sort out, meaning that the administrator has to put a lot of work towards organizing the fighters, which could be avoided simply by adding some key features to the system database.

The system is written in a way, such that it is very limiting in how it can help facilitate the administrators with conducting the event, which means that the administrators experience isn't particular good, and their organizing of the event isn't as smooth as one could have hoped it could have been. The administrator gets a list with fighters, listed randomly, often with duplicates and fighters who do not show up. The administrator has to organize this list and then create a tournament schema, from the list. All this takes a lot of time and creates huge frustration every single time.

The screenshot is taken from OKBR.net. Here you can see how people just creates a lot of gyms, to get into the system when they forgot there password or username. Also a lot of duplicates have been created.

Klubber

Klubnavn	Postnr.	Land	Navn	Telefonnummer	E-mail
ADMIN	- -	-	-	-	-
1	1 1	1	1	1	1
A	A A	A	A	A	A
Team Tae-Kibo	1030 Vienna	Austria	Levente Bertalan	+43 6606531984	lisa.koessler@gmail.com
HKV	2270 Herenthout	Belgium	Tommy Harzé	0032478244406	hkv@telenet.be
Liège Point-Fighting Club	4000 Liège	Belgium	Tran Quang	00324485314376	lpfcda@hotmail.com
Contact Sports	2270 Herenthout	België	Tom Harzé	32478244406	hkv@telenet.be
CHIN-GU FULL CONTACT	2450 KØBENHAVN SV.	DENMARK	Bayram KORKMAZ	+45 29934841	chin-gu@mail.dk
Zen-To	1617 Kbh	DK	Lars Krusaa	26373787	larskrusaa@mac.com
fightworld	2500 Valby	DK	Nick M. Christensen	20408313	info@fightworld.dk
WKA (Iran, men kontakt venli.gst deltageren personligt)	5935 Bagenkop Sønenbro	Danmark	Ata Vafakhah	nej	ataa.vafakhah@yahoo.com
Nørrebronx	2200 København N	Danmark	Nina Lund	60812706	nina_b_lund@yahoo.com

Figure 1 - List of gyms in OKBR.net (Online Kickboxing Regristration)

From all the problems we have conducted and the wished requirements Nikolai have given us, we have a foundation for the problem statement.

Problem statement/ Statement of Purpose

Several problem areas in Aalborg Martial Arts gym were found. These can be fixed with new software and hardware. There are three big problem areas in the current system.

First is the big administration problem. There is simply too much paperwork on the day of a tournament. It takes minimum two people to match the fights, from the fighter books with the fighter information, to the competition papers. These papers are going to be changed again, if one of the fighters cannot compete, the fights will be moved around and the affected fighters should be notified and maybe new judges should judge the fight - all of this should of course be documented in fighter and judge books. This paperwork could be fixed with an administration system that contains all information about fighters and judges, and then by command automatically put up the match fixing after weight class, category and matches. This would save a lot of administration time, stress and confusing at the competition.

The second big issue is the competition system itself, more specifically the system that operates under the fights. Here we also see several problems. The system does not work probably in a lot ways as

described earlier. No mouse working, cannot go back in the system, cannot put in reasoning for early stopping etc. The worst problem with this system is when a fight is stopped before time, because of the judges' mice. They each sit with one fragile cable mouse, which often are pulled out. This stops the match and the points have to be given again. Here it is not just a question about new software, but there should also be looked at new hardware instead of a cable-connected mouse.

The third issue is the matchmaking. This is also written on paper out from data in the fighter books. This creates several problems. The fights have to be matched out from several criteria's, such as discipline, weight class and number of fights. When the matching is all done, there will always be changes at the tournament day. A fighter can get hurt, gain or lose weight etc. and the tournament plan has to be changed. After this the affected fighters, judges and coaches have to be notified. Another factor is the break between fights, where each fighter should have a minimum break of 30 minutes. To make the complete system will take a lot of time and is not all relevant for this project. Therefore, a limitation section is appropriate to narrow down what we want to work with.

Project limitations

The scope of this project can be quite huge. The system itself can consist of a lot of different elements, some of which is a calendar, a tournament planner, administration, tournament system (which includes hardware problems). Due to the time limitations, we cannot manage all problem areas.

First of all we have chosen not to work on the tournament system, because of the connection to bad hardware. We do not want to put any costs on the organization. Because of the limited time, resources and the demand for a database in the project, we have chosen to work with the administration part of the gym. This part of the problem area are also the most relevant to the case description - "*Development of a Database System for a Specific Application*", because of all the data we have to work with - the fighter books should be digitalized and the tournament information should be organized in way, that minimizes the paperwork on the tournament day.

Now we have narrowed down the workspace, which means that we can move on to the other sections. The first thing we have to look at is the development approach, because it sets the stage for how we are going to work.

Chapter two: Development approach

Before we start up the actually project, meaning the analysis, design, coding, evaluation we need an approach to manage all this. There are a lot of different criteria's that should be considered, to give us the best tools for organizing, documenting, communicating etc. We will therefore discuss the different approaches and pick the one we find most useful for a long term school project, with a customer to consider.

Waterfall method – theory

The waterfall method is a software development approach, which moves forwards and down like a waterfall. There are 6 phases in the approach: *Requirement specification, analysis, design, Implementation, testing and acceptance.* The main idea with the waterfall method is the use of phases. The phases make the project transparent, so every member of a project knows exactly what is going on.

This approach envisages that the developer uses a lot of time in the early stages to be sure no mistakes are done, and every phase must be complete before moving on to the next phase. It is easier to correct anything that goes wrong earlier and it is cheaper to fix the problems, this way.

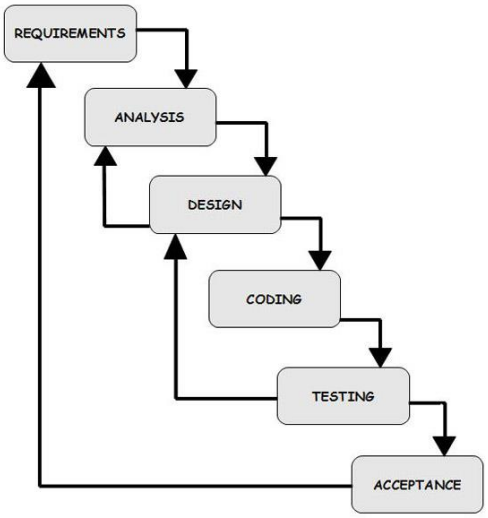


Figure 2- The Waterfall Model - (Satalkar, 2011)

The waterfall method can have different phases to different projects. The method has to fit the exact project. This means it is accepted to put two phases together, re-name a phase or simply take one out, as long as you follow the principles of the method. For example, a maintenance phase is not always required. You can see the model and the phases that we have chosen to use. From the model, you can also see where in which phases there is chance to go back with. For example, if the costumer does not accept the product, we have to go back with new requirements. The six phases will be explained next.

Requirements: In this phase all the systems requirements should be found, listed and described. This can be done in cooperation with the customer or the list can be showed and accepted by these. It is really important that all requirements are found in this phase, because it ensures a smoother development. This way, the programmers do not have to deal with major changes later in the project.

Analysis: In this stage, it is about finding the best solution for the described requirements. Should the software be made as a phone application or web browser solution? In the analysis phase, we will also try to think about our systems future. We do not need to analyse who is using the system; we already know that from the requirement phase.

Design: In this phase the flowchart of the system is created with the requirements in mind. This creates the architecture of the system and makes the implementation phase a lot easier and hopefully without errors. A graphical design is also created, or at least discussed.

Implementation: In this phase the actually coding are carried out. In this phase the flowchart gets physically created. If the other phases have been done right, this phase should go smoothly, but there is also a chance for new functions to be made. We are still in a learning process, with a lot of different coding skills.

Testing: In this phase the system gets tested. This is done to make sure that the software has all the requirements implemented, but also that the system is easy to use for the user. If any errors are detected, the requirement phase to describe new functions that can help this error, or go back to the design phase, to change the user interface if that is needed, and then to the implementation phase again. So when we go back in the waterfall model, we follow the phases again.

Acceptance: This phase is all about pleasing the customer and perhaps helping with implementing the system into the organization. This phase can be critical. Is the customer satisfied, the whole project can be finished (if no support or maintenance deal have been agreed upon), but is the customer not satisfied, the waterfall method have to be started from scratch again. New requirements come in and the team starts from phase one. (Parekh, 2013)

Change management

In the waterfall model the documentation is an important part, as you will have to document everything in every stage, so all the knowledge gained through this process is documented. This way, even if other teams can get the information they need to, by looking at the documentation.

Even though the approach sets the stage for not moving backwards, it is allowed anyways. It is still the waterfall method but it is called modified models, it is basically the waterfall method, but with small changes which makes the model more flexible and addressed some of the critique of the "pure" waterfall method.

According to the paper; "*A Rational Design Process: How and Why to Fake It*" (Lorge Parnas & Clements, 1986) it is actually impossible to make a rational design process, where all your choices are well documented, analysed and discussed. A human brain can simply not overview all the details in a system, there is simply too much - from graphical design, to the backend programming. It can also be the lack of skills, maybe the programmer does not know how to make a something smart in flash, Photoshop or maybe the designer cannot program the functions, so they match the design. Therefore, all developers should be careful to put too much work into phases that have a big chance for being changes and furthermore have a plan for changes.

We will give the change management task, to one member in the group. When a new function is added, perhaps to make the system even better, this management person will add this to the documentation. If we complete each phase 100 %, this should not happen, but all experience says it will. Therefore, it is important with a plan and a change manager, to make sure the documentation is correct.

Waterfall method in our project

The development approach chosen in this project is the waterfall model, which is explained above. This approach is chosen because it fits and it is flexible in regard to commitment levels. The group consists of five (in the end four) members, who have work and other unrelated interests, which conflicts with time that could have been spent on the project. It is important to choose a method that gives the opportunity to prioritize our resources. The waterfall method gives us the opportunity to organize, and not just the project, but every phase in the project. We can still work together in teams,

but home assignments are also acceptable. With the waterfall method we don't have to spend a lot time on iterative work. The errors should be found early in every phase.

The waterfall model is ideal if the developers have all the requirements of the customer needs. Getting the requirements and design out of the way first also improves quality; it is much easier to catch and correct possible flaws in the design stage, rather than in the testing stage after all the components have been integrated and tracking down specific errors is more complex. Finally, because the first two phases end in the production of a formal specification, the waterfall model can aid efficient knowledge transfer when team members are located at different locations.

The emphasis on requirements and design before writing a single line of code ensures minimal waste of time and effort and reduces the risk of schedule slippage, or of customer expectations not being met.

The waterfall method also requires documentation for analysis of requirements, design and code, which is essential for a university project, due to the report. The documentation also helps us in communicating with our partner in the kickboxing club and communicating internally in the group. Of course Nikolai does not need all the documentation, but requirements and graphical design could be some essential places to get his input. The documentation happens during each phase, as well as when the phase has ended. This helps us in organizing and having an idea of, how far in the process we are.

In some phases during the project, there is a bigger chance for changes than others. These can be the function list, which furthermore can change the database and even the graphical design. Therefore, during our process these documentation papers should be "open", meaning they are not done yet, but more can be added. It is almost impossible to get all the requirements and functions down on paper and not add any in the later phases. Even though Nikolai has a good idea of the system he wants, maybe we will discover a smarter or simpler way to do this during the project, and why not implement it then? This is also why we don't put pictures and diagrams in the report during the process. They will be implemented in the report when they are 100 % done. This is actually faking the rational process in the waterfall method, but it is necessary to be able to add more functions later on.

Implementation of development approach in our project

The waterfall lifecycle will be implemented in our project as an overall guideline for our development approach. We will try to complete each phase, before starting on a new, even though this can be really hard to comply. As mentioned earlier; if a new requirement or function is found in the implementation phase, of course we will add this new feature. This means that during the project the development approach will be modified to accommodate our needs. We will make an implementation plan, try to follow it and evaluate this in the end of the project. This implementation plan is created to make an easy overview of what every phase should include.

Requirements

The requirements, both functional as well as non-functional, will be established in cooperation with our contact person in Aalborg Martial Arts - Nikolai. The result of this cooperation is a list of functions that will be implemented in the system. The contact person has a very clear idea of what he needs, since the current system has some limitations and gives more work for him due to incompatibilities between the systems in use. However, not all the wishes of the contact person will be met, due to the limited timeframe and limited programming experience, the group as a whole possesses. But the project limitation will be created by us and accepted by the gym, before the work continues.

Analysis

This phase is used to get an understanding of the problem domain. The main product of the analysis is use case diagrams and the analysis of the environment. The use cases will be used in the design phase to create flowcharts and furthermore in the testing phase to make a usability tests.

We do not need to analyse our target audience. We know who will use it, where it will be used etc.

Design

The user interface will first be created in an abstract view, using flowcharts. These flowcharts have been created from use cases, which have base in our stakeholders. This way gives an abstract idea of the interface, with the real stakeholders in mind will be created when looking at how the user interacts with the system.

After creating this abstract design, this will be used during the coding phase. Here the graphical design will slowly come together, but the functions in the system are the biggest priority. Therefor we will use simple design rules to get the best possible user interface. The design rules will be explained and followed as much as possible in the coding phase.

Coding

The design phase is dedicated to user interface, but we also need to the database, but to create a better flow in the report, this will be presented in the coding phase. The data associations will be drawn in an entity relational diagram. Furthermore the database structure will be exhausted in a database model diagram that represents the final database structure which will be implemented. The design of the database will be considered to be more important than the user interface design and the program structure, hence the learning goals of this semester.

The programming will have basis in the products of the design phase. However, each feature will be written on a post-it note, and will be in a pool of undone features. The developer, who is responsible for that feature, will be written on the post-it note and the note will be moved to the section called “in progress”. When the feature has been developed, the post-it note will be moved to the section “done”. Unlike the agile approach the planning game will not be applied, since it is considered a waste of time, due to the varied programming skills.

When starting on this phase, we should know everything we need to implement and how this should be done. All the features should be found and defined, the database should be designed and an abstract design should be ready.

Testing

In this phase the use cases will be re-used. Every use case can be used as a task in a usability test. By making usability test on our use cases, we will look at the most essential functions in the system and how the user actually uses and finds these functions in the system. Why have smart email functions if no one knows where it is? This should not happen on a site with good usability, which is pretty essential in the system, because of the many users and all their interaction with the system.

The data from the usability test will be analysed; solutions will be found and implemented.

Acceptance

In this phase we will correct the mistakes and ambiguities that we found in conjunction with Nicolai. When we get acceptance for the corrections we make them fit to the requirements. The system will be used in real life, at the next tournament, but there will be no support from us as the project is done by the next tournament. We have had a great working relationship with Nicolai, and we already have a deal on getting feedback on the system, after we are done with the project, just to confirm how it works. The system will be implemented before the project is finished, for Nicolai's evaluation.

Now we have clarified how we will move on in the project, so naturally the requirements section is the next phase.

Chapter three: Requirements

Now this project will take a turn and focus more on our product. We have decided on our project limitations. We had our first meeting with our partner. And we have chosen a developing method - The waterfall method. This paper will therefore now start on the first of six developing phases. The first one being requirements, this will be the third phase in the paper.

Requirements are found to understand the customer's needs. They are listed and described, so there is no doubt about what each function satisfy for the customer. If this phase is done correctly, there should not be any errors further or add-ons further on.

For collecting the requirements we had two semi-structured open interviews with our contact, Nicolai, from the gym. He was able to provide us with an overview of their existing systems, his work with planning and administrating a tournament and of course his idea for making this better. As told, the interviews were very open and a lively discussion with Nicolai, about options and wishes. Nicolai has been working with their old system for too long, in his own words. Because of this, he has a really good idea of how a new system should be and a lot of features that the system should have. This gave us some requirements, but even more important, it gave us some stakeholders. Stakeholders, or actors, are the people who are going to use the system.

Actors

The actors will now be described, so that they can be used to look at who will use the systems. This is essential for especially the database. We need to know who use it and what permissions, functions etc. they should have.

Stakeholder: Administrator

Permissions: Access to admin functions, can change own and others profiles. Can retrieve a list of participants of a tournament.

Description: The administrator is the person who can update the homepage, create events and change the information in other users' profile. Basically the administrator has access to view and change everything in the system. The most important feature of the administrator is to access the database and get needed information. This could an email list for a specific gym or event. Most important

though is the information on an event. The day of the event, the administrator can access the database and get a full list of fighters in every discipline, with their coach, weight class, gym and gender.

Stakeholder: Fighter

Permissions: Access to edit in own profile and update his own role.

Description: The fighter has few rights in the system; he can update his own information, read his fight book and sign up for a gym. It is the coach job to enrol him in a tournament, but the fighter should have a possibility to withdraw himself from a tournament.

Stakeholder: Judge

Permissions: Access to edit in own profile and update his own role.

Description: The judges are part of a three man judging team. Each judge gives out points in a fight, to find a winner and a loser. The judges should be able to see which fights they have been judging. A judge is also able to participate as a coach or a fighter. But only the administrator can upgrade users to a judge. This is because to be a judge you need training, lectures and approval.

Stakeholder: Referee

Permissions: Access to edit in own profile and update his own role.

Description: The referee is almost the same as a judge. At the tournament, he is in the ring with the fighters, making sure the rules are held. In the system, he does not have any particular roles, but it is important to remember the referee, so that if a tournament system is implemented in the future, there is already a referee role in the system. As with the judges, it is only the administrator who can upgrade users to this role.

Stakeholder: Coach

Permissions: Access to edit in fighters and own profile and update his own role. He also got permission to enroll fighters into tournaments.

Description: The coach is connected to a gym, and has the responsibility to enroll his fighter to a tournament. A coach is able to compete as a fighter too.

Stakeholder: Member

Permissions: Access to the same functions as fighter, but cannot be enrolled to a tournament.

Description: A member is basically the same as a fighter, but cannot be enrolled to a tournament. This person is not member of the gym in the system, but he is registered. It is actually a fighter, that has not joined a gym yet.

Stakeholder: Guest.

Permissions: Access to view profiles and news.

Description: A person who just wants to look at the website for news or checking out fighters. The guest is not logged into the system.

Functions

Creating a function list will make you able to identify all functions that the system should have, and will also give us the possibility to check the consistency with the use case. A function is activated, performed and gives a result (Mathiassen, 2000, s. 136). All three phases depends on which type of function this is. There are four types of functions and when categorizing in this way, it helps us understand what the function actually do.

Update: Update is a function which is activated by an event in the system and it then changes the systems state.

Signal: The signal function is activated by a change in the systems state. This will result in a reaction to the actor, for example by providing them with information, or by directly change or manipulate the back end of the system.

Read: These function types are activated by the actors need for information. When the actor asks for information, the system will show the some or all of the information, depending on the actors' permission to see information.

Compute: These function types are also activated by the actors need for information. The difference here is that the system also needs some information from the actor. When this information is received the system will make a calculation and show the new information for the actor.

Function list

This list with functions is created from the interviews with our customer and some additional functions have been added from the creation of flowcharts. This creation found place later in the project, but as described earlier, even in a rational developing method, it is impossible to get all functions from the beginning.

The yellow marked functions are the ones that we dropped later on in project. Why these got dropped will be explained in the function descriptions. As argued earlier it is almost impossible not to take some functions out and new ones in. That happened for us as well. The new functions can be seen in the bottom of the function list and they will also be explained afterwards.

Feature	**Complexity**	**Condition**
Register member	Medium	Update
Update role	Easy	Update
Update permissions	Easy	Update
Create tournament	Medium	Update
Read news	Easy	Read
Send invitations	Medium	Signal
Create news	Easy	Update
Enroll fighters	Hard	Signal
Get fighter list in tournament	Medium	Calculate
Update fighter book	Hard	Signal
withdraw from tournament	Easy	Update
Update Judge book	Hard	Signal
Edit functions	Easy	Update

View profile	Hard	Read
Edit profile	Medium	Update
View fighterbook	Hard	Read
View judgebook	Hard	Read
Tournament deadline	Medium	Update
Retrieve information	Easy	Read
View gym info	Easy	Read
Change gym logo	Easy	Update
Create gym	Medium	Signal
Ekstra functions		
Create new discipline	Easy	Signal
Create new weight class	Easy	Signal
Create new fight	Medium	Signal
Declare winner	Easy	Update
Search members	Medium	Read

Description of functions

For a better understanding of each single function, a description has been made. This will give an overview of our intentions with each function.

Register member: This is a function for creating an account in the system.

Update role: Members can update their own account to fighter or coach. This gives different permissions, but it is the same update function. This also makes it possible to have more roles. A fighter can also have taken the referee education, and the administrator needs to know this for planning tournaments. Only the administrator can update a member's role to judge, referee or administrator.

Update permissions: Administrators can update any member's permissions to fit what he or she decides, at any time, for any reason. This function is kind of still in the system. The administrator can

change everything in the system, but the original thought was that he could really specify what each member could see and do in the system.

Create new tournament: The administrator can create a new tournament, where coaches can enroll their fighters.

Read news: On the front-page, news will be presented in a short version. For the full news, users can click on 'More…' and read the full news.

Create news: The administrator is able to put new news on the web page.

Send invitations: Administrator can send invitations for tournaments to anyone. This function is dropped, but instead the administrator can send mass emails to all members in the system.

Get fighter list in tournament: The list of fighters for a tournament can be generated and printed by an administrator.

Update fighter book: After a fight has been fought the participating fighters fight book will be updated with the winner and loser by the administrator or judge.

Update judge book: After a judge has judged a fight the winner and loser is noted in the judge's judge book.

Edit functions: Members of the system can get email notifications, this should be optional. This was also dropped, due to our time resources. It's not that needed.

Withdraw from tournament: Fighters, judges and referees can withdraw themselves from a tournament.

View Profile: Anybody can view profiles for every member in the system.

Edit Profile: Update personal information.

View Fighter book: This function is activated automatically, when a member becomes registered as a fighter. It enables other people to see the fighters fight history, in the fight book.

View Fudge book: This function is activated when a member becomes a judge and then works like the fight book.

Enroll fighters: Coaches and administrators can enroll fighters to a tournament.

Tournament deadline: An administrator-function that makes it possible to set a deadline for registration for a tournament.

Retrieve information: Coaches can retrieve specific information on members in their gym. This was one of the functions, we saw as "nice to have", and therefor due to our resources, this function was not made. It is still possible for the coach to get information, he just needs to create his own list and take the information from each profile. We have been concentrating on the administrator, before the coaches functions.

View gyms: Guests and members can view the information on gyms.

Change gym logo: Administrator can change the logo of a gym. This function is not implemented in the system, but is still possible to do this from the database.

Create gym: Administrator can create a new gym.

And now for the functions that have been added later on in the project.

Create new weight class: This should not really happen. But it's made if the weight classes' change or a new discipline gets in with new weight classes.

Create new discipline: This is made so the system can be used for more disciplines or even other sports.

Create new fight: The administrator can create new fights in a tournament. He can do this before the tournament, so everyone can see their fights. This also creates a tournament system, which was not a part of our limitation, but a wish from Nikolai.

Declare winner: The administrator, judge or referee puts in the winner of the fight and the fight books will be updated.

Search members: This is made, so that users can find each other easily.

Actors and functions overview

We have many actors and they almost use the system the same way, but some functions should only be available to some actors. To get an overview of which actors should have permissions to which functions, we have created a table. This will help us remember our thoughts about the system and will be a great help in the implementing phase.

Functions / Actors	Guest	Member	Fighter	Coach	Judge	Referee	Admin
Register member	X						
Update role		X	X	X	X	X	X
Update permissions							X
Create new tournament							X
Read news	X	X	X	X	X	X	X
Create news							X
Send invitations							X
Get fighter list in tournament							X
Update fighter book					X	X	X
Update judge book					X	X	X
Withdraw from tournament			X				
View profile	X	X	X	X	X	X	X
Edit profile		X	X	X	X	X	X
View fight book	X	X	X	X	X	X	X
View judge book	X	X	X	X	X	X	X

Enroll fighters				X			X
Tournament deadline							X
Retrieve information				X			
View gyms	X	X	X	X	X	X	X
Change gym logo							X

The requirement phase is now done and we now know which actors should use the system, which functions should be in it and who should have permission to use it. Therefore, we are ready for the next phase in the project - Analysis.

Chapter four: Analysis

In this chapter we analyse the systems environment, meaning it's hard- and software requirements. We have to know our limits from these, because Aalborg Martial Arts do not have money to upgrade. We will also do a SWOT-analysis, mainly to force ourselves to look differently at the system, but especially to think about the systems future. Furthermore, we will analyse what the actors should navigate in the system, by creating small use-case scenarios. These will used to create flowcharts, which will give us an abstract idea of the system design.

Analyses of hardware and software

In the introduction of this project, we looked into the kickboxing club, a bit their software and hardware problems and decided on a limited problem area - the administration part. However, even in the smallest part of a problem area, software and hardware decisions have to be made. In this part, we will explain our thoughts behind these decisions. These are made before the design and implementation, so that this should not be a problem further on.

Our partner, Aalborg Martial Arts, cannot afford new hardware, so the system has to work on a computer, even though an application for a pad could have been a solution. What they do have of available hardware, are computers connected to the internet. A browser-based system connected to a database for storing all the information, would therefore be essential.

The system has to be flexible. It cannot depend on correct data. Meaning that fighters can be sick or even cheat with their information, like weight class. The system should therefore just create a list, with the information, which the administrator needs.

The information stored in the database should always be visible and changeable for the administrator, so that he is able to change these data, even in the last minute. The administrator should also be able to get exactly the data he needs. It should therefore be easy to view data everywhere with an internet connection, and the downloaded list should be created in a way, that makes it easy to edit the information, in case of sickness, withdrawals and cheats.

SWOT-analysis

A SWOT-analysis is a technique to outline **S**trengths, **W**eaknesses, **O**pportunities and **T**hreats in a system and/or business. The SWOT-analysis have been chosen, so that we can look at the system weaknesses and strengths and in this way force ourselves to think about the system in a new way. The SWOT-analysis will also provide us with thoughts about the systems future. How it should be made, so other systems can be implemented, but also thoughts about how to make high usability, so the users want to use the system. First, the SWOT will be made and then our thoughts will be explained.

	Strengths	Weaknesses
	• Online/open fighter books with info • Less paperwork • Online mailsystem • Structured enrollment	• No cooperation with the point system • Needs engagement from all actors • Administrator has too much work • Very open system • No tournament system
Opportunities • Can be implemented to other gyms • Complete database with info from all danish gyms • Implement other systems • Forum	• Earning money by selling the system. • Point system can be implemented • A tournament system can be implemented	• Needs better security • Have to work 100 % • Needs good server
Threats • Users don't want to use the system	• User input to make system better	• Everyone have to use the system

The strength and weaknesses in the SWOT-analysis kind of explains themselves, like less paperwork or no tournament system. What we have been using SWOT for is to look at opportunities and threats, and then the weaknesses and strengths in these opportunities and threats to especially make us think about the future of the product. Therefore, this will be a discussion about the main points in our SWOT-analysis.

One of the first opportunities that came to our minds is that other gyms can use the system as well. Of course, all gyms should be able to create themselves in the system, but not necessarily use the system to create tournaments, disciplines etcetera. Strength in this angle could be to sell this right for a small amount of money to other gyms. We now from a fact that right now, all gyms have a lot of paperwork for each tournament and they hate it. However, when selling a system it has to work 100 %. Of course, that is what we are going for, but we are not going to make ongoing support, so if they want a new feature in a month or have found something frustrating, this will not be changed by us. We will make a usability test to find errors, but some may first occur when system is used for an actually tournament.

Another angle could simply be to just pass the system on to other gyms, get it implemented and get all fighters from all gyms to fill in their information. This would create a more united martial arts community in Denmark and a forum could then be a good idea. A weakness here could be the security. The bigger the system gets, the more likely is it to be attacked. It could even be from a member, who was mentioned in a bad way on the forum or have anger towards the martial arts community in another way. Also with more information, pictures etcetera, the server has to be with a good standard.

By simply giving the system away to other gyms, we also offer them the opportunity to continue working on it. If they have the competences, they can implement other systems, like a tournament system. By thinking like this, we have to make sure that the code- and database design is good and easy to understand.

The biggest threat to the system is that the users will not use it. We have taken this into consideration, by giving the coach the job to enroll his fighters. This actually means that the fighters only have to create themselves in the system and then the coach can enroll them to tournaments, without their acceptance. They of course talk regular together and train for a tournament, so it should not come as a surprise to a fighter if he is enrolled in a tournament. But if the users will not use the system, an idea could be to gather data about why? Get some user inputs and implement them to the system. A weakness is of course that you cannot be enrolled to a tournament if you are not a user in the system. It is therefore important to communicate clearly, why fighters have to go into this website and put in their information. But then again, it is a small community and they have all seen how bad the current paper system works.

Use cases

In the requirement phase we found our stakeholders, or actors, we will use these actors to create use-cases. With these use cases, use case scenarios will be created. Out from these different scenarios it is possible to create flowchart diagrams of how the user is interactive with the system. This will give us an overview of how the actors will use the system and most important, it will give us an abstract idea of the homepage design.

But before creating use-cases, the use-case theory will be explained.

Use Case Theory

Use cases can as written before be used for finding requirements, but they are also useful for user documentation, UI design and for understanding the system without the technical aspect. The definition of a use case can be done pretty simple; A use case describe how an actor uses a system to achieve a goal. For example: *A student needs to reserve a meeting room by using the schools intranet.* The actor is the student, the system is the intranet and the goal is a reserved meeting room. As in the example a use case is not a technical description of the system but a description of the interaction with the system, how the actor use the system by pressing buttons, scrolling or some third interaction to reach a goal of with value for the actor. In the example there is no value, for the student when entering the intranet, the value for this actor is when a room is booked. An actor can also be another system, which the system in the use case has to communicate with, even though this is not the case in our project.

Use Case scenarios

The use case scenarios are created with a goal for all the actors, something they want to do in the system. We have chosen some of the most essential functions for each stakeholder. These goals have been found in a lively discussion in the group, to get our minds and ideas more united. After this, the use cases will be used again to create flowcharts, to specify even more clearly how the stakeholders will use and navigate in the system.

Stakeholder: Administrator

Use case example 1: The administrator wishes to send out emails with invitations to all coaches in a specific gym.

Goal 1: Get a list with a specific gym coaches emails.

Use case example 2: It is the day of a tournament. The administrator needs a list of fighters in the full contact discipline. He wants the list to be divided into gender and furthermore into weight classes.

Goal 2: Get a list of fighters in the full contact discipline, divided into gender and weight class.

Use case example 3: A gym have sent a new logo to the administrator. They want it as their gyms new logo on the website.

Goal 3: Change the gyms logo.

Stakeholder: Fighter

Use case example 1: The fighter wants to check his last fight by logging in to his profile and look at his last fight.

Goal 1: Check his last fight.

Use case example 2: The fighter wants to search in his fights for a specific fight, using the search function.

Goal 2: Look for a specific fight.

Stakeholder: Judge

Use case example 1: The judge wants to add or remove more roles to his profile.

Goal 1: Register as more than one role.

Use case example 2: The judge wants to look at fights he has been judging in the last tournament.

Goal 2: Check which fights he judged in the last tournament.

Stakeholder: Coach

Use case example 1: Coach logs into his profile, and then he enrolls four fighters to a tournament.

Goal 1: Assign four fighters to a tournament.

Use case example 2: The coach wants to send out an email to all members of his gym.

Goal 2: Get a complete email list of members in a gym.

Stakeholder: Member

Use case example 1: The member wants to compete in a tournament, so he wants to upgrade himself to a fighter.

Goal 1: Upgrade own profile to fighter.

Stakeholder: Guest.

Use case example 1: The guest wants to find his brothers profile, to see his fight book.

Goal 1: Finding a specific profile.

After all the analysis we have done we are ready for the design phase. In this phase, we will keep in mind what the earlier phases set the stage for and design the system by those criteria.

Chapter five: Design

The architecture of the design will be created using flowcharts. This should make the implementation phase go a lot smoother. We need to have the same system in mind, when starting on the coding.

Of course, no system is complete without a user-friendly graphical design, something that the kickboxing homepages have never thought about. Our thoughts about the graphical design will be explained and ideas will be documented, this is made out from modern design- theory and principles.

Flowcharts

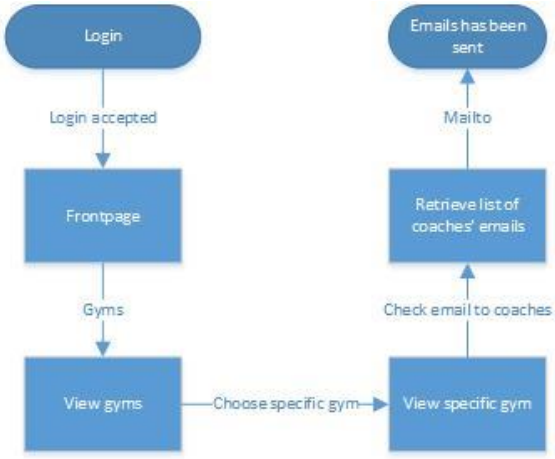
From each use case, flowcharts will be created. These flowcharts will show the actors process to reach his goal. They have been created to give us overview, of how the system should be connected, which will give us a picture of an abstract design. We will know how the actors are supposed to find the specific functions, under which menu etcetera.

The flowchart uses different shapes to describe the process. The ovals symbolize the start and the end of the process, the rump symbolizes a decision and the squares describe what the actor sees and where in the system he is. The arrows are the process itself, and tell us what the actor does.

Stakeholder: Administrator

Use case example 1: The administrator wishes to send out emails with invitations to all coaches in a specific gym.

Description of flowchart: The administrator logs into the system and when his information is accepted, he will be presented to the front-page. He then presses 'Gyms' and get presented with pictures of all the registered gyms. After that, he presses on a specific gym. Here he can see information about the gym but it is also here he can retrieve information about the gyms members. He checks off coaches and emails, and presses 'retrieve info'. He will now be provided with the email list and a box to write an



Figur 3 - Administrator Flowchart 1

email in. When he presses 'Mail to' the email will sent out to the coaches in the gym

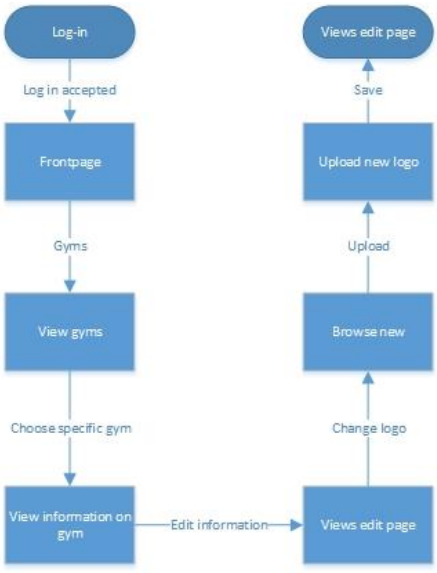
Use case example 2: It is the day of a tournament. The administrator needs a list of fighters in the full contact discipline. He wants the list to be divided into gender and furthermore into weight classes.

Description of flowchart: When the administrator logs into the system and his information is accepted, he will be presented to the front-page. He then presses 'Tournaments' and can now view all the upcoming tournaments. He chooses a specific tournament and can now view information about this one tournament. Because he is administrator, he has a 'View enrolled fighters' button. When he presses this, a list with the enrolled fighters will appear on the homepage. This list will automatically be divided into gender and furthermore weight classes. The administrator can now press 'Print' to get the list in a printable version.

Figur 4 - Administrator Flowchart 2

Use case example 3: A gym has sent a new logo to the administrator. They want it as their gyms new logo on the website.

Description of flowchart: When the administrator logs into the system and his information is accepted, he will be presented to the front-page. After this he presses 'Gyms' in the menu and he can now see all the gyms in the system. The administrator presses on a specific gym. Here he can see information about the gym and he presses 'Edit information' Now he can see the same side, but just in a "edit layout" which means he can change information, but also change the gyms logo. The administrator presses 'change logo' and he can now browse on his computer for the new logo. When the new logo have been found he presses 'Upload', the logo uploads and when the administrator presses 'save' the logo



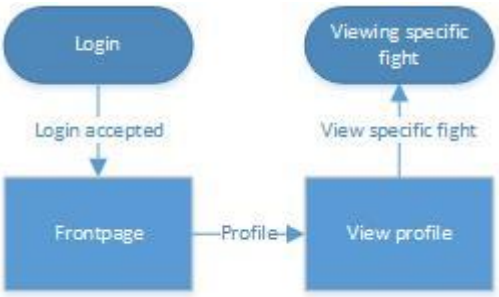
Figur 5 - Administrator Flowchart 3

will be updated. The administrator is still in the edit page.

Stakeholder: Fighter

Use case example 1: The fighter wants to check his last fight by logging in to his profile and look at his last fight.

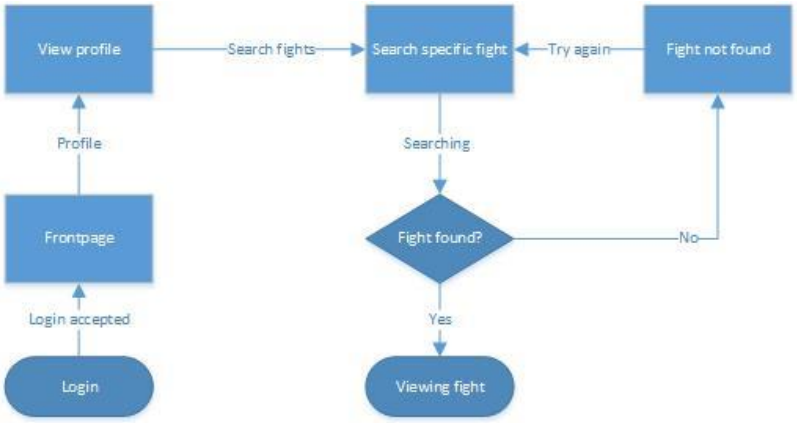
Description of flowchart: When the fighter logs into the system and log in are accepted, he gets to the front-page. He then presses 'Profile' and can now view his own information page. Here there will also be a list with his last five fights. (If he wants to see them all, the fight book should be opened). He presses on the last fight and can now view information about this particular fight.



Figur 6 - Fighter Flowchart 1

Use case example 2: The fighter wants to search in his fights for a specific fight, using the search function.

Description of flowchart: When the fighter logs into the system and log in are accepted, he gets to the front-page. He then presses 'Profile' and can now view his own information page. There will be a 'search' box on his profile page. This search box will only search for fights in, which the fighter has participated. The fighter search for a competitive fighters name and all the fights they have competed in against each other will show up in a list. The fighter picks the specific fight and can now view information about this.

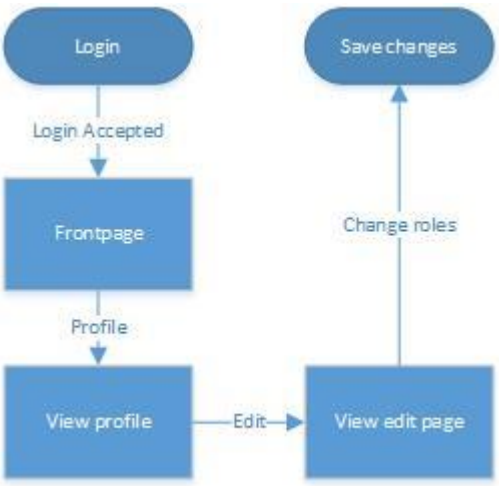


Figur 7 - Fighter Flowchart 2

Stakeholder: Judge

Use case example 1: The judge wants to add more roles to his profile.

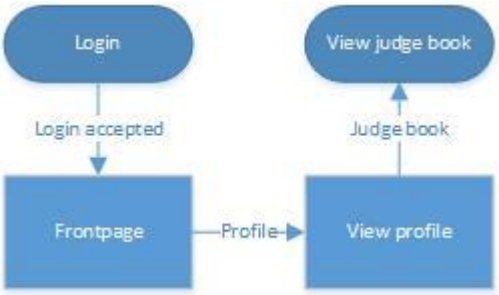
Description of flowchart: The judge logs into the system and if his information is correct, he will be directed to the front-page. He presses 'profile' in the menu and gets to his own profile. The judge presses 'edit' and he can now check off more roles to add these. He ends this up by pressing 'save' and the new roles have now been added to his profile.



Figur 8 - Judge Flowchart 1

Use case example 2: The judge wants to look at fights he has been judging in the last tournament.

Description of flowchart: The judge logs into the system and if his information is correct, he will be directed to the front-page. He presses 'profile' in the menu and gets to his own profile. He now have a new menu in the top and there he presses 'Judge Book'. He can now see the last fights that he has been judging.

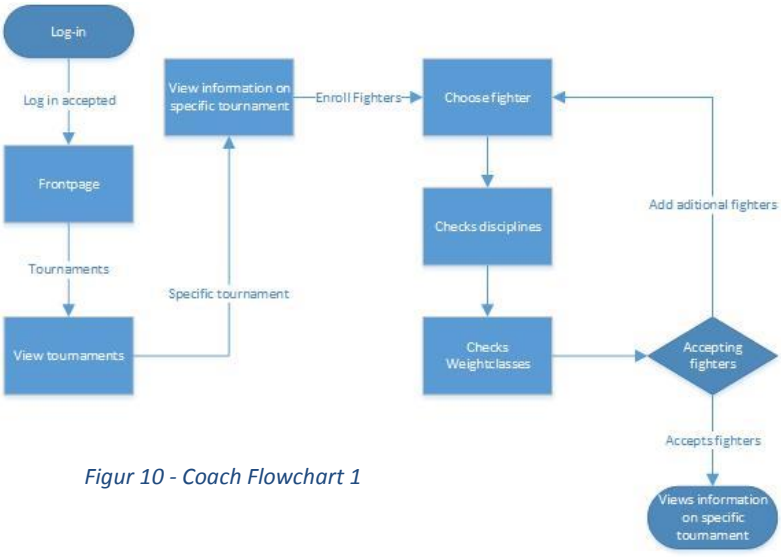


Figur 9 - Judge Flowchart 2

Stakeholder: Coach

Use case example 1: Coach logs into his profile, and then he enrolls four fighters to a tournament.

Description of flowchart: A coach logs in and is directed to the front-page. He then presses 'Tournaments' in the menu and can view a list of upcoming tournaments. He then presses on a specific tournament and can now view

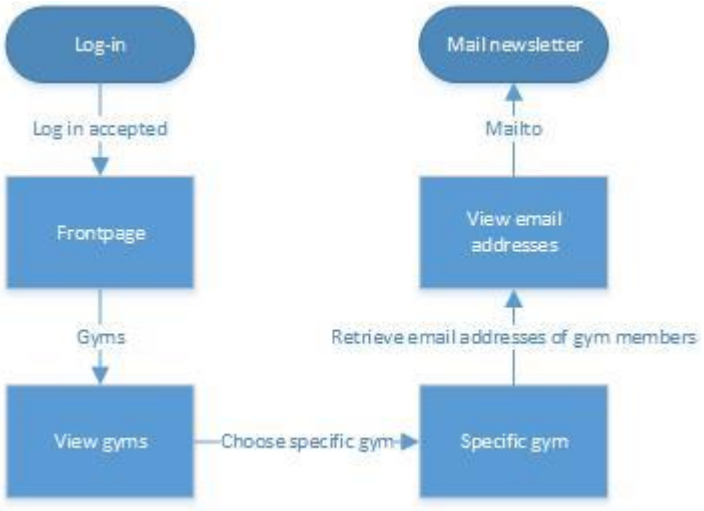


Figur 10 - Coach Flowchart 1

the information about this tournament. Because he is a coach, there will be an 'enroll' button, which he presses. He can now choose a fighter from a dropdown list, and then check off which disciplines and weight class the fighter should participate in. This can be done again and again until all the fighters, the coach wishes to enroll, have been enrolled. When the coach presses 'accept fighters' these will be added to the tournament.

Use case example 2: The coach wants to send out an email to all members of his gym.

Description of flowchart: A coach logs in and is directed to the front-page. He then presses 'Gyms' in the system and can view a list of gyms in the system. He then presses on a specific gym and can now view information about this gym. It is also here that he can retrieve information about the members in the gym. He can choose which information he wants for specific actors or them all. He chooses emails and all members and gets a list of emails and a box where he can write the message. He presses 'send' and the mail gets send to all members.

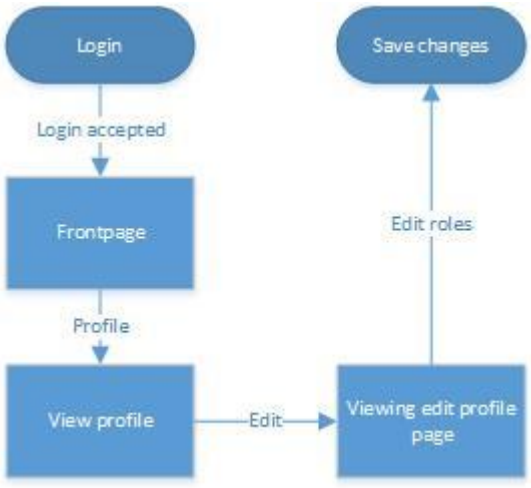


Figur 11 - Coach Flowchart 2

Stakeholder: Member

Use case example 1: The member wants to compete in a tournament, so he wants to upgrade himself to a fighter.

Description of flowchart: A member logs into the system and if his information is correct he will be directed to the frontpage. He presses 'profile' in the menu and gets to his own profile. The judge presses 'edit' and he can now check off more roles, to add these. He ends this up by pressing 'save' and the new roles have now been added to his profile.

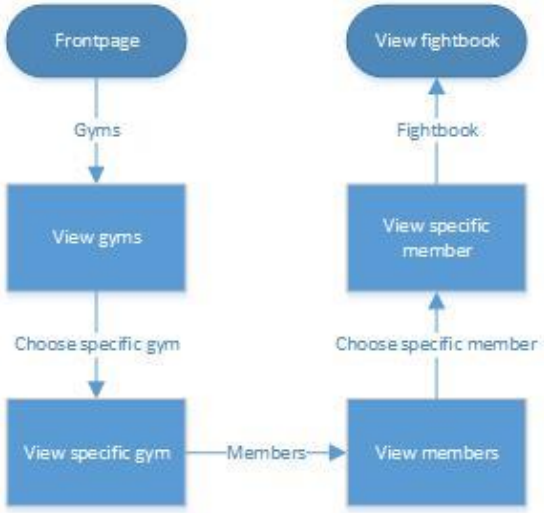


Figur 12 - Member Flowchart 1

Stakeholder: Guest.

Use case example 1: The guest wants to find his brothers profile, to see his fight book.

Description of flowchart: The guest does not log into the system, so he starts directly on the front-page. He then presses 'Gyms' and chooses his brothers gym and can now view information about this gym. He then presses 'Members' and a list with member's shows. He then finds his brother and presses on his profile. He can now view his brothers' profile and presses 'Fight Book'. Now he can see his last fight.



Figur 13 - Guest Flowchart 1

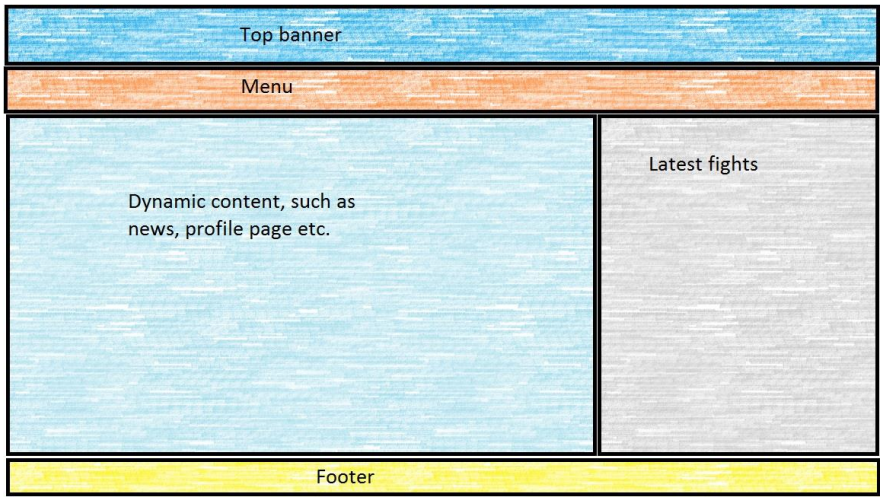
Sketches of user interface

This part of the project has been done more than once, to see each other's ideas of how the web page should look like. The first design workshop was made kind of early and before the created flowcharts. This was done to get a visual on each other's thoughts. This turned out to be a really good idea. We all had different ideas of how the system should look and how the user should operate in the system. Even though we thought we were talking about the same system, this was far away.

That is why we used our use cases, to create flowcharts. The flowcharts show exactly how a user will use the system, which creates some "working frames" for creating a design, meaning that now we all have the same knowledge about how the systems flow should be.

After the flowcharts, we now knew what should be in the design. A log-in on the front-page, a menu with minimum four pages; front-page, gyms, tournaments and profile. Other decisions were also made in creating of the flowcharts, the front-page should be with news and latest fights. Another really important discussing was made creating the flowcharts, regarding where the different functions should be. Here it is all about what would be most natural for the users. For example the 'edit' for editing your profile, should be under your profile. At first thought, it could be an idea to collect all functions in one place, but this would simply not be natural for the user.

After the discussions and the creation of flowcharts, we could now create a simple design. We have on purpose not made the design in details, because of possible changes, but also because of the focus on the database. The complete design will actually be made during the implementation phase, where we will follow some of the basic rules for graphical design on a webpage. These design principles will be explained, and after it should be possible to actually see these on the web page design.



Figur 14 Abstract design of system front page

Design principles

When designing user interfaces on websites there are some specific rules, or more guidelines, for doing so. There have been created a lot of these throughout time, but some of the most respected and useful are Jakob Nielsens ten usability heuristics (Nielsen) and the gestalt laws (Gamborg, 2013). If the designers have this in mind when designing the interface, the usability will be way better. First the 10 heuristics will be explained and then the gestalt laws. We will further on use these for designing the interface.

Jakob Nielsen ten usability heuristics

1. Visibility of system status

The user should always know what is going on in the system and where the users are in the system. This means that the system should give feedback to the user, especially if the user should wait. For example, when the system is loading or sending data. The user should not be trapped in the system. A visible menu on all sites or “goes back” buttons can help here.

2. Match system and real world language, logic etc.

The systems language should be the same as the users. It does not mean that a website for Danes cannot be in English; this could perhaps be a startup language. But it means that a sport website should use sport terms and a gamer website should use gamer terms. This also makes the website feel more professional for the users. Information should also appear in a logical way and be presented with real world conventions, like a book and not just two long lines that needs scrolling. The point is to use real world conventions as much as possible.

3. User control and freedom

A good system should always give the users an opportunity for undo and redo. Users often try functions “for fun” or by mistake. Therefore the system should support an edit function or ask the user “are you sure you want this?” before making the change. It is naturally that the users want to explore an often-used system, and there for making mistakes, so undo and redo can help the users fix their own mistakes.

4. Consistency and standards

The system should use the web standards users are use too. For example a house icon means home and get you to the front-page. The user should not wonder what a button, word or anything on the website means or do.

5. Error prevention

It is always better to prevent an error then give an error message. This can be done by a good design and testing before the release. Again, a confirmation for users can also prevent many errors from happening. Tell the user what they are changing and if they are sure.

6. Recognition rather than recall

The user should not have to remember information, instructions, actions etc. This can be prevented by making these visible or easily retrievable everywhere on the website. For example, a user should not have to remember if he is logged-in or not. This should always be visible.

7. Flexible and efficiency to use

The system should be built for both experienced and inexperienced users. Instructions should be visible, but optional. The system should also guide the users to speed up the interaction. This way the user gets a better flow and a better experience using the system. But the system should never talk down to people, especially not to experienced users. Another way to make a system flexible and efficiency is to make it possible to tailor often-used functions, so they fit the users' needs. For example, the system could give the user the opportunity to choose which information they want, instead of just give them all information.

8. Aesthetic and minimalist design

Text on the website should not have irrelevant information, stick to the main points. This also counts for the graphical design. A website should not have confusing design with lot of animations, sounds, shapes etc. only if there is some bigger meaning. For example, on an artist-, museum- or music website, where the user should discover and experience the page. So just because you made something cool in Photoshop, it is not sure it will work on the website, remember - kill your darlings.

9. Help users recognize, diagnose and recover from errors

I wrote earlier that a good system should always give error messages, when an error happens. But error messages can be really different. A good error message explains what happened and not in code language, explains where and why it happened and then suggests a solution.

10. Help and documentation

Of course, a system is best when the user does not need help to use it, but sometimes this can be necessary, especially in expert systems. Help and documentation should always be visible, explained in a language that is understandable for the user, made in short terms and broken into steps. Often users will use the search function to find help, so good search terms for documentation should be created.

The Gestalt laws

The gestalt laws describe how we humans perceive context and relationships in our brains. In other words, they tell us about how we look logical on the world and by implementing these rules to our web design; we will have better-organized information, data, pictures etcetera presented to the user.

When we see a web page, our subconscious will automatically try to organize, arrange and decide which objects belong together. Therefore, we will use the gestalt laws to make sure that the functionality will not lower the usability in the web design. The laws will be presented and explained, even though they can seem somewhat logical, it is fundamental for a good web design, especially one like ours, where people will look for specific information.

1. The law of proximity

This law tells us that symbols arranged close to each other will be perceived as belonging together. This can be dealt with by using air in the design. Less air between objects that belongs together, and more air between those that does not. This law is especially good when the objects are two different kinds, like text and picture.

2. The law of similarity

Now this law tells us that symbols looking alike will be perceived as belonging together. This law will be used on buttons. If the buttons look alike, the users have no doubt where to press to start an action. This law can also be used to make objects belong together, but with a lot of air between them. For example, if there is a red information box on the top of the web page and in the button, we will expect these to belong together, perhaps with some of the same information. This law also covers web conventions. If we see an underlines text on a web page, we expect it to be a link, because it will be similar to all other links on the internet.

3. The law of closure

Symbols in the same frame, will be perceived as belonging together. This law is extremely good on websites with a lot of information. By framing information, data, symbols, yes many different objects together, we can organize these and make the overview for the user much better. This law is not just seen on websites, but also other medias like newspapers etcetera because its logical way to organize lots of different symbols together.

4. The law of connectedness

Symbols that are connected will be perceived as belonging together. Colours or directly lines connecting symbols will make us think these belong together, even though the shape or type of object

can be different. This tells us to think about colours. If we use the same colour on different objects, these will be perceived as belonging together, even though this is not the case. Lines can be used to put objects together, even though they are of different type, colour, shape etcetera. A strong tool if the other laws can be hard to implement.

5. The law of figures and background

This law is really important. It is rarely that it is not followed but when that happens, it can really confuse the user. This is because our brain sees the smallest figure as a part of the front and not background. There should be no doubt in our mind, what the background is and what the front is. One way to make this clearly is by the contrast in colours. A background colour with a contrast colour for text will make the text much more readable. Maybe we can read a text on a background with different colours, but our mind will use a lot of energy on this, making the user tired and maybe even frustrated. This can ruined to best functional systems.

Chapter six: Coding

In this phase, we will start up the actually coding of the system. Our main focus is to get the database to be 100% functional, but we also need a running website for Aalborg Martial Arts. Of course, this website would not matter without a database. The website will be an administration system for all the gyms and their members, making the fighter book digital. But it also provides the possibility for creating tournaments, where people can be enrolled and then provide the administrator with a logical list of enrolled fighters, coaches, judges and referees, making it easy to create a tournament plan. Due to our resources, as explained early in the project, we will not make a tournament planner.

First, we will explain the different programming languages that we have been used, what they can and why we have chosen them. After this, we explain our database and then we have some bits of code that we are extra proud of having produced. These will also be explained.

Standards, conventions and language explanations

The purpose of having standards and conventions is to have common rules for writing code, hence the different styles each team member uses. Common standards and conventions are applied to the code in order to enhance readability and thus the understanding of the code, when reading through a code snippet that is written by another team member.

The standards and conventions will be explained for each language respectively.

PHP

PHP is used for the dynamic content of the website, which includes, but is not limited to: arithmetic, security, communication to and from the database.

PHP stands for PHP: Hypertext Preprocessor, but it used to stand for Personal Home Page Tool. It is used to developing homepage and is popular alternative to ASP.

PHP is cross platform, which is one of the main reasons why we chose to develop in PHP. The other reason why we chose this language is because we had a course call project supported programming, which was in PHP language.

Object Oriented

PHP has the ability to function as both imperative and object oriented code, the later will be applied, hence it is easier to adapt to and from other programming languages that are popular such as Java, C# and C++. These programming languages are fundamentally object oriented. Enhanced readability, easier refactoring and easy reuse of code are other reasons to use object oriented PHP code.

Structure

In PHP, there are different ways of structuring the code. The purpose of having a common standard of structure is to have a relatively uniform way of code structure. Our code structure should look like this this example:

```
class Class_name
{
   function function_name($parameter_name)
   {
     /*
      * Comments.
      */
     for($i = 0; $parameter_name > $i; $i++)
     {
         /*
          * do something
          */
     }
   }
}
```

Note that the curly brackets are used to indicate that the code that belongs to that block will be inside the curly brackets. PHP is open to not to have curly brackets where the following code in a block only contains a single statement. I our code the curly brackets will be used in either case, hence the statements below.

for($i = 0; $parameter_name > $i; $i++)
{
echo $parameter_name += $i;
}
	as opposed to:	for($i = 0; $parameter_name > $i; $i++)
		echo $parameter_name += $i;

Class names will be written with the first letter in capital and the remaining letters in lowercase. Function, variable and parameter names will be written in lowercase letters only. Underscore will be used instead of spaces between words. This rule applies to all non-PHP keywords.

HTML

HTML is the foundation of a homepage. It stands for HyperText Markup Language. Its primary used to structure homepages.

HTML will be used to represent the GUI to the user via a browser. Furthermore, the some of the PHP code will be implemented in the HTML to ensure dynamic content. The HTML will be written in a way that will ensure good readability. The generic structure will look something like this.

```
<html>
  <head>
    <title>Some title</title>
  </head>
  <body>
    <h1>Some headline</h1>
  </body>
</html>
```

In order to keep the HTML code nice, clean and readable spaces are used to indicate which child tags are member of which parent tags. For example, the h1 tag has been moved four spaces further than the parent body tag. All HTML pages will have a *.PHP extension to enable PHP code within to be rendered.

CSS

CSS is used to style the homepage, thereof the name cascading style sheets. This controls how HTML will be shown in the browser. If used correctly the CSS language can make the homepage aesthetic. We used CSS to style our homepage so that it looks more professionally; it was not necessary in this project, because the main focus was on the database. Therefore, it was possible just to use HTML for the foundation and PHP to connect to the database.

CSS is in other words charge of the styling of the HTML page. To enforce readability the generic code will look like this.

```
h1
{
   font-size: 30px;
}
```

The purpose is this format is to use a similar format as in the PHP code to ensure good readability and this formatting should give a better overview as the code grows in size.

MySQL

MySQL is the most used open source relational database management system. It runs as a server, which can be accessed by multi users to more than one database. The SQL stands for Structured Query Language. We chose MySQL because it's the most popular SQL management system, and it was used in our course; Database Development. The alternatives as postgreSQL were not even a consideration because of the clear advantages we had using MySQL besides that it is very compatible with PHP. The scripts such as insert, select, update, delete, create statements will be written in a certain way. The generic script structure will look like this:

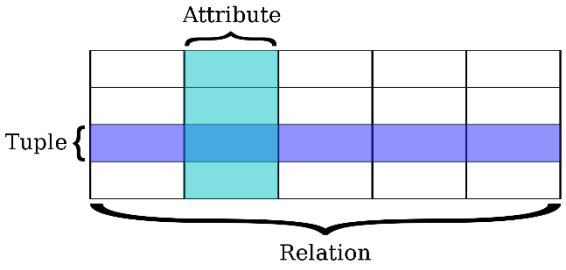
```
SELECT column1
FROM table1
WHERE column1 > 5 AND column1 < 10;
```

For each keyword, except for keywords in the WHERE clause, there will be a new line. This rule is will enforce improved readability in contrast to when the SQL query is written on one line only. Like the PHP code, the convention is to use underscore to separate column, database and table names that normally would have been written in two words.

Database theory

Tables

A database consists of tables amongst other items. But the tables are used to contain all the data that are being stored within the database. A database can have zero or more tables, which can have zero, or more attributes. The tables are used to organize the data in a considerate manor in regards to performance, storage space and the cohesion of the data. A table is defined as a set of tuples that have some attributes in common. The tuple often represent an object or a concept, which information is organized and persisted within the table.



Figur 15 Image from (wikimedia)

MyISAM vs InnoDB

MyISAM and InnoDB are storage engines MyISAM is the default storage engine but it has some disadvantages. It lacks the ability to support transactions and in our case, it is not possible to make relations between tables. InnoDB is newer and better for our needs, because it supports foreign keys. However, the MyISAM engine has better performance than InnoDB.

We chose InnoDB because we have relations between tables in our database and it default in the newer version.

Primary keys

Primary keys are used to identify table by assigning one or more attributes as a primary key. The primary key's primary task is to ensure uniqueness of each tuple and the primary key ought to have a relation to the other attributes in the same table.

A primary key can consist of one attribute or more than one attribute which is known as a composite primary key. The primary key can also have foreign key attributes as a part of the key or consists of foreign keys entirely.

Foreign keys

Foreign keys are attributes that are from another table, but imported to a table in order to indicate a relationship between two tables. The relationships between tables are normally as follows:

One and only one.
One or more.
Zero or more.
Zero or one.

Normalization

Normalization is used to normalize the data, in other words to organize the data in a way so the database is optimized for data retrieval, performance and cohesion. The normalization is a rule set with which the relations can comply or not.

1NF

1NF is all about the data contained within the attributes. The rule is that the data within the attributes must be atomic. In other words multivalued tuples are not allowed, only indivisible values can comply with 1NF (Elmasri & Navathe, 2007, s. 348-352). Consider this employee table:

SSN	**Name**	**Address**	**Project**	**PNumber**	**Salary**
123456789	Stan Smith	302 Walker Road, Newcastle	Alpha	213	25000
987456123	Jeremy Miller	74 Danum Road, Scunthorpe	World conquest	248	24500
115547892	Jill Stevenson	445 Shawfield Street, London	Echelon	498	26000

The table above does not fulfil the rule of 1NF by only containing atomic attributes, hence the name and address attributes have multivalued data. However, it is rather easy to make the table comply with 1NF.

SSN	**First_name**	**Last_name**	**Number**	**Street**	**City**	**Project**	**Pnumber**	**Salary**
123456789	Stan	Smith	302	Walker Road	Newcastle	Alpha	213	25000
987456123	Jeremy	Miller	74	Danum Road	Scuthorpe	World conquest	248	24500
115547892	Jill	Stevenson	445	Shawfield Street	London	Echelon	498	26000

The table now complies with 1NF, new attributes have been created, the table contains the same information as before, but it is much better organized.

2NF

The second normal form rule is based on the primary key. It states that all non-primary key attributes have to be fully functional dependable on the primary key of the relation that they belong to (Elmasri & Navathe, 2007, s. 352). If we look at the table above, not all attributes are fully dependable on the primary key. However, it is possible to comply with 2NF by dividing the one table into more tables. We consider the salary to be negotiable and individual for each employee.

SSN	**First_name**	**Last_name**	**Number**	**Street**	**City**	**Salary**
123456789	Stan	Smith	302	Walker Road	Newcastle	25000
987456123	Jeremy	Miller	74	Danum Road	Scuthorpe	24500

115547892	Jill	Stevenson	445	Shawfield Street	London	26000

Pnumber	**Project**
213	Alpha
248	World conquest
498	Echelon

Now all the attributes are fully dependable on the primary key. First_name, Last_name, Number, Street, City and Salary are all dependable on SSN. However, the trivia about how dependable the address is on SSN is debateable, but in this instance, the person holding the SSN lives at a particular address.

Boyce-Codd normal form

There is a saying: “Each attribute must represent a fact about the key, the whole key, and nothing but the key, so help me Codd.” (High Arch, 2009) Now let us break it down:

Each attribute

Each attribute must be atomic independent attributes, such as SSN, name etc.

Must represent a fact

Each attribute in a relation is a truth about the relation and is by itself a fact.

About the key

The key needs to be absolutely unique and not reusable and must be an unchangeable representative of the relation. An incremental identifier is a good example of a unique unchangeable key.

The whole key

The key must be sufficient to identify the values of the non-key attributes and the same data cannot be represented twice with different keys. An example: In an address book two people lives in the same city, on the same street, in the same house, the address will be represented twice, one time for each person. However, this is a violation of the Boyce-Codd normal form. Then we have to decompose the relation into more relations.

And nothing but the key

If there is a another street in another city, but with the same name, it is not the same street, and they only share the name, then it is okay to let the same name appear twice, but the key has to be different indicating that it is not the same street at all. The relation examples below are the product of the decomposition of one single relation in order to comply with the Boyce-Codd normal form. However, these examples do not allow duplets to appear. For example if there is a road in London called Walker Road, then there will be a reference to Street_ID 1, instead of making a new tuple.

ID	**SSN**	**First_name**	**Last_name**	**Address_ID**	**City_ID**	**PNumber**	**Salary**
1	123456789	Stan	Smith	1	1	213	25000
2	987456123	Jeremy	Miller	2	2	248	24500
3	115547892	Jill	Stevenson	3	3	498	26000

City_ID	**City**
1	Newcastle
2	Scuthorpe
3	London

Address_ID	**Number**	**Street_ID**
1	302	1
2	74	2
3	445	3

PNumber	**Project**
213	Alpha
248	World conquest
498	Echelon

Street_ID	**Street**
1	Walker Road
2	Danum Road
3	Shawfield Street

So help me Codd

When a database fulfils the Boyce-Codd normal form it automatically also complies with the third normal form. But the other way around is not necessarily true.

So what is all the fuzz about? The main reason for normalize the database is to get an elegant database design that reduces redundancy and organize the data in neat way. But the real discussion is when to go all the way with normal forms, stop half way or not normalizing at all. Going all the way with normalization is good if you need to organize data, reduce redundancy or just need a good guideline to good database practices. Everything is well and good, but why would you not choose to normalize your database? There are certain drawbacks with normalization that need to be considered when the database is being designed.

When you are normalizing your database, you often end up having more relations than you had initially. Which means that the need for join statements increases and making the database slower since join statements require more performance than a normal select statement. In the example above the single relation was split into five different relations requiring five join statements to recreate the initial relation, resulting in a higher response time. Even though the database design is made with normalization in mind, the design process can be rather time-consuming due to the increasing complexity of the database and the data integrity. In some cases, rapid development is essential and more important than an elegant designed database. However, there are some risks when making a quick and dirty database. As the project evolves and code is written, the need for better-structured data may become apparent. The real discussion whether to normalize or not and to which normal form is basically found in the problem: performance versus data integrity. (Chapple, 2013)

ERD theory

An ERD are used to model the problem area in question, which can be used to build the structure of the database. However, the ERD does not concern any kind of data types, since those are not relevant at the point in time, where ERDs are applied. ERDs are written in a friendly way, so that database novices can get a grasp of what kind of structure, the database will have at the time of design. However, the ERD are not exhaustive in regard of all the technicalities, but it leaves, nonetheless, a very good overview of the database tables, keys, relations etcetera. The key elements of ERD are described in the subsections.

Entities

An entity in an ERD can be a real-world object such as a person, but it can also cover something that has been conceptualized such as a university course. Every entity can have a number of attributes that describes the properties of that entity. This combination of an entity and one or more attributes is the most fundamental concepts of ERD. In an ERD the entity is displayed with a rectangle with the name of the entity within it. (Elmasri & Navathe, 2007, s. 61-62)

However, where the normal entity has a key of its own, the weak entity has no key. Nonetheless, a weak entity can have a partial key though, which consists of more than one attribute. The weak entity is presented in the ERD as a regular entity, but it has two rectangles instead of just one. The partial key

is presented as a normal key attribute, but has a dotted line instead of a solid line. (Elmasri & Navathe, 2007, s. 76-77)

Attributes

Attributes are used to describe the properties of an entity as mentioned earlier. The attributes in an ERD are displayed by an ellipse with the name of the attribute within, which is attached to the entity via a line (Elmasri & Navathe, 2007, s. 62-63). It is generally speaking a good idea to keep the attributes atomic for flexibility purposes for later usage. However, this design advice may be disregarded, if it is considered to be irrelevant for the database use.

Composite Attributes

Composite attributes differs from normal attributes in a distinct way, which is when an attribute can be generalized such as a name. A name can be divided into first name, middle name and last name. The composite attributes are displayed as a tree of attributes, but only one attribute is attached to the actual entity. In the example with names, the attribute name would be attached to the entity, while the first name, middle name and last name attributes would be attached to the name attribute. (Elmasri & Navathe, 2007, s. 63)

Multivalued Attributes

Multivalued attributes are attributes that can have more than one single value. For example, a customer in a bank can have multiple accounts. The multivalued attribute is displayed in the ERD by having an extra ellipse around it, so there are two ellipses in total. (Elmasri & Navathe, 2007, s. 63-64)

Derived Attribute

Derived attributes are attributes that are calculated based on other attributes. For example if the birthday of a person is stored in the database and we want, the age of the person stored as well. The age attribute needs to be updated once a year to keep the value of attribute correct. The calculations would be (age = current date - birthday). The derived attribute is presented in the ERD as a normal attribute, but the ellipse is dotted instead of solid. (Elmasri & Navathe, 2007, s. 64)

Key Attributes

The key attribute is defined by its ability to uniquely identify each entity. For example a person has a social security number that uniquely identifies the person. However, one entity may have multiple key attributes, but the combination of the key attributes must uniquely identify the entity. The key attribute is presented in the ERD as a normal attribute, but the name of attribute is underlined. (Elmasri & Navathe, 2007, s. 66)

Relationship

Relationships are represented as a rhombus that is connected to two or more entities. The relationship shows how the entities are related and it is a good indication of, where a foreign key is needed, when the ERD is to be transitioned into a working database. (Elmasri & Navathe, 2007, s. 70-76)

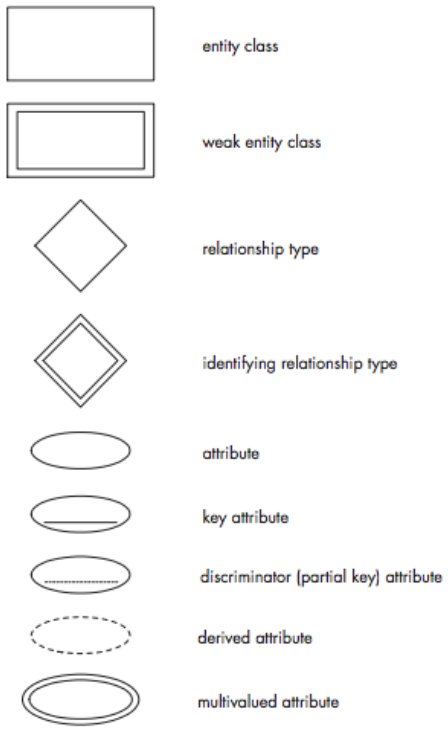
Participation

There are two types of participation. The one is partial participation and the other is total participation. Total participation means that the total set of entities in one entity must be related to another entity in its entirety. The total participation constraint is shown in the ERD by a double line between the two entities in question. A partial participation is as the name suggests only a partial constraint between two entities and is presented in the ERD with a single line between the entities. The partial participation indicates that not all sets one entity has a relation towards the other entity. (Elmasri & Navathe, 2007, s. 75)

The syntax for ER-diagram takes practice to remember, but the main things are what we used in our ER-diagram. Beside the symbols we have used there is a lot more.

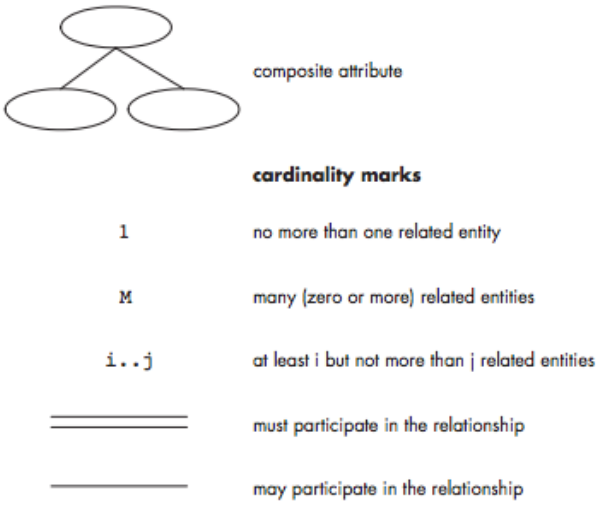
To the right we have the whole syntax symbol list.

As you can see we did not use cardinality in our ER-diagram, it is because we were advised by our supervisor to omit it, as it is not important for the client.



The cons for the ER-diagram are that it does not specify the data we want in the database or how it is stores. That is one of the reasons why we have DMD.

The reason why we made the ER-diagram is to get an overview of the database also this is a learning process for us, where crating an ER-diagram is an important part of it. (Riccardi, 2002)



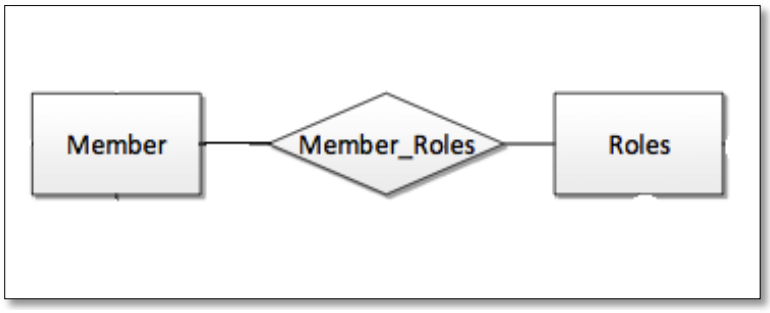
Figur 16 – Complete symbol list (Riccardi, 2002)

ERD in practice

In this chapter, the main focus is to enlighten any given client how our ERD works, and the purpose of the diagram briefly as we went through this in the ERD theory.

Entity in practice

ERD is short for entity relation diagram; it is a high-level abstract model, which describes data and their relationship by using 3 terms: entity, attribute and relationship. So basically it is a way to describe a database. It shows how entities and attributes are related with other entities and attributes. Before moving on, we have to know what an entity is, as Peter Chen explains it an entity can be thought of as nouns. As in the picture underneath: a member, a role and so on. By the way, Peter Chen developed the ER-model in 1976 (Addbot & et. al., 2013). The picture shows an example of how entities are related to each other in our diagram.



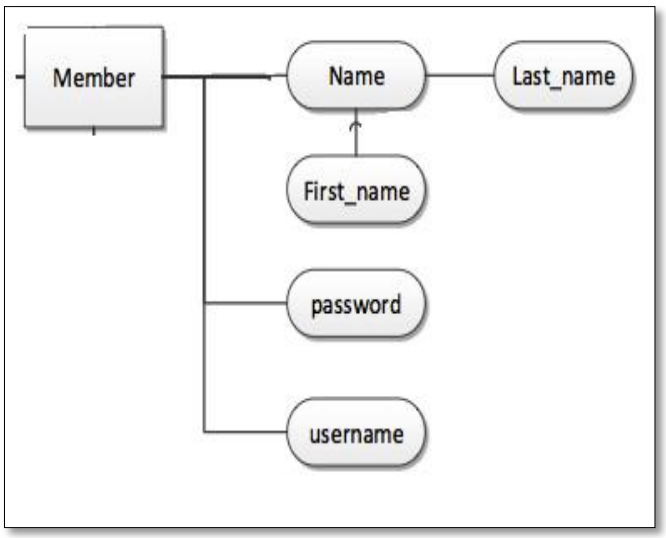
Figur 17 - Relation between member and roles

Relationship in practice

If lack of fitting relationship name the name is called both entities name with an underscore, as in this picture. Otherwise, it is allowed to give the relationship a fitting name, as long as the name is unique. The relationship can be thought as a verb, if there is a fitting name to the relationship. The purpose of a relationship is to connect a number of entities together.

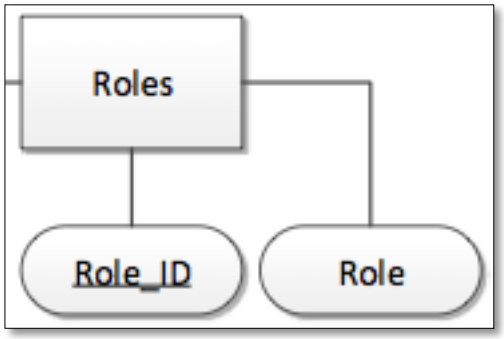
Attributes in practice

Beside the relationship between two interties there is attributes, which are pretty important as well. Attribute tells us what is inside the entity or the relationship. In the picture below, there is an example of how an entity has attributes.



Figur 18 - Attributes of member

Everything that's connected to Member is attributes Last_name is also an attribute, but because it is composite to name it's still an attribute to Member. What is important about attributes is the primary key. Every entity has to have a primary key, which has to be unique. If it is not possible to have only one attribute, which can be unique, then the only option is to combine to attributes, which, in combination becomes unique. Even though it is possible to find a unique attribute, often it is still recommended to create an ID attribute to easily identify an entity.



Figur 19 - Roles with primary key

As you can see in the picture above there is a line under Role_ID that's the symbol for the primary key which you also can see in the in the list of symbols above.

Visualising the relations between the objects and concepts

Now that we have a good grasp on what an ERD is and how it is used, we are ready to sculpt our ERD. First, we need to look at the objects and conceptualisations that are available to us. The content of the table below is based on our interviews with Nicolai.

Fighter	Judge	Referee	Administrator	Invitation to tournament	Roles
Email	Fight	Tournament	Points	Winner	Loser
Fight book	Matchmaking	Delegating rights	Right	Lists of member information	Member
Coach	Gym	Weight class	Discipline	News	Age group
Participate in tournaments	Withdrawal				

Many objects are usable in the ERD. The following are going to become entities due to the need for them to have properties and data within them. Bear in mind that we have limited us from making a tournament and matchmaking system, which means that already now, “points” has been taken out of the list. “Tournament” should also have been taken out of the list, but it seems necessary to make the fight book.

Roles	Fight	Tournament	Weight class	Discipline	News
Age group	Gym	Rights	Roles	Member	

Administrator, coach, fighter, referee and judge are considered to be roles that a member can have none or more of. Which means that for flexibility “Roles” will have two attributes: “Role_ID” and “Role”. This way it is possible to have more roles that the ones mentioned. The final ERD can be found in the appendix section of this rapport. It is obvious that the ERD is made as a representation of reality and the various objects and concepts are ordered in a considerate manner in order to conceptualise the entities, their attributes and the relationships between them.

Database model diagram

Furthermore, we have made a DMD, which we created with the ERD in mind. The DMD was very important to have, before the database is created. The DMD has everything we need that is missing in the ER-diagram. By looking at our DMD, we know that tables we have to create, how they relate to each other. In our DMD, we can also see the foreign key that is important, when we have to make the relations in the database. In the DMD, we also made sure to create a new table every time there was a many to many relationship, which we in the ERD could not see. It is not important for the client to know everything about the tables and how they relate to each other, therefore often only the ERD is presented to the client. Underneath you can see just an example of how a table in our DMD looks like. The complete DMD can be found in the appendix section at the end of this rapport.

Figur 20 - DMD entity of member

The Member is the entity and below is all the attributes. In front of the last two attributes, it says FK. FK is short for Foreign Key they both are child from parent in other tables. The first attribute “Member_ID” is marked “PK”, which means “Primary Key” and, as discussed in a previous section, the primary key defines the collection of data that the Member entity contains. The attributes in member are necessary to keep track of who the members are, but also has information that are relevant to the generation of the fight book, such as “Day_of_Birth” in order to calculate the age of member at any given time. The relation between the member and the gym is also mentioned as a foreign key. The member can only be affiliated with one gym at a time, but he can change gym, if he moves to another city. However, this construction of gym affiliation has proven to be inconsiderate towards storing information of which gym a member was member of, at a specific time. This produces a problem, when generating the fight book, since Nicolai wants to keep track of gym affiliation as the fight book is populated. (Riccardi, 2002)

Presentation of code

We start with the CSS code, and then we move on to PHP. The code will first be presented and then explained.

CSS code

```
/* Columns */
#columns
{
		overflow: hidden;
		width: 1040px;
		margin: 0px auto;
		padding: 0px 80px 0px 80px;
}

#columns .content
{
		overflow: hidden;
		padding: 0px 0px 50px 0px;
}

#columns h2
 {
		padding: 0px 0px 20px 0px;
		color: #000000;
}

#columns #column1
{
		float: left;
		width: 680px;
		margin-right: 40px;
}

#columns #column2
{
		float: right;
		width: 300px;
}
```

In the picture above a bit of the CSS code is shown. It is used to make the layout for the system. As described before, CSS is used in HTML to help designing a web page. This CSS code creates a new ID, which can be used in HTML's DIV tags. The ID name is "#columns", which are used on this site to show the content. The ID is 1040 pixels wide, in which the content can be put.

When being inside the "#columns" ID the h2, which is used to make a header, or headline. The font size has been changed from standard h2 to a customized one, which will only be used while inside the "#columns" ID.

Next two new ID's has been made inside the "#columns" ID. This is for placing the content different places. This CSS code is used almost on all pages, except for the profile pages. The ID "#column2" is used for the "Latest fights" column to determine where inside the "#columns" ID it should be placed. The ID "#column1" is used for all the content on the specific page of the system you are looking at. This could be the news on the front page.

PHP code

Generate fightbook

```
private function generateBook($fights)
{
        $fightbook = new ArrayObject();
if($fights->count() > 0)
{
foreach ($fights as $fight)
{
        $fightbookitem = new FightBookItem();
        $fightbookitem->fight_id = $fight->fight_id;
        $fightbookitem->fighter_1 = $this->membercontroller->getMemberByID($fight-
>fighter_1_id);
$fightbookitem->fighter_2 = $this->membercontroller->getMemberByID($fight-
>fighter_2_id);
$fightbookitem->winner = $this->membercontroller->getMemberByID($fight-
>winner_id);
$fightbookitem->discipline = $this->disciplinecontroller->getDiscipline($fight-
>discipline_id);
$fightbookitem->tournament = $this->tournamentcontroller-
>getTournamentByID($fight-                >tournament_id);
$age1 = new DateTime($fightbookitem->fighter_1->day_of_birth);
$tdate = new DateTime($fightbookitem->tournament->date);
$fightbookitem->fighter1_age = $age1->diff($tdate);
$age2 = new DateTime($fightbookitem->fighter_2->day_of_birth);
$fightbookitem->fighter2_age = $age2->diff($tdate);
$fightbookitem->fighter1_weight_class = $this->weightclasscontroller-
>getWeightClass($fightbookitem->fighter_1->weight_class_id);
```

```
                $fightbookitem->fighter2_weight_class = $this->weightclasscontroller-
                >getWeightClass($fightbookitem->fighter_2->weight_class_id);
                $fightbookitem->fighter_1->gym_id = $this->gymcontroller->getGym($fightbookitem-
                >fighter_1->gym_id);
                $fightbookitem->fighter_2->gym_id = $this->gymcontroller->getGym($fightbookitem-
                >fighter_2->gym_id);
                 $fightbook->append($fightbookitem);
}
}
else
{
        echo 'No fights in exists';
}
       return $fightbook;
}
```

The code from Fight.php is used to generate a fight book by populate a new object of FightBookItem, which contains all the necessary information. The method takes a parameter of ArrayObject, which contains the list of fights to be evaluated in order to generate to fight book. The first thing that happens is that a new ArrayObject called $fightbook is initialized, this variable will contain all the FightBookItems, which in turn represent a fight, with opponent, winner, tournament information and so forth. Next a check is performed to ensure that the parameter $fights contains any fights at all, if it does, then the foreach loop will be executed, if not the else statement will be executed, which simply writes, “No fights exist”. The foreach loop loops through the elements in $fights and name each object $fight. In the beginning of the loop, a new variable of the object FightBookItem is initialized. Then the fight_id is stored in $fightbookitem for later use. In the next two lines the fighter’s, his opponents’ and the winner’s details from php/Member.php is being stored as an object of Member in $fightbookitem. After that the discipline is retrieved and stored in $fightbookitem as well as an object of tournament, weightclass, gyms and dateinterval for later calculation of age. In the last line of the foreach loop the $fightbookitem object, containing all the information regarding one fight, is stored in the $fightbook ArrayObject which is later returned and presented to the user as the fight book, judge book or referee book depending on the contents of the parameter $fights.

However, performance wise, this algorithm uses an excessive amount of database connections in order to produce one line of the fight book. The total amount of connections can be calculated by this formula: $Total\ amount\ of\ connections = (number\ of\ fights\ x\ 10) + 1$. Keep in mind that getMemberByID, getDiscipline etc. require a connection each. This means that if a fight book has 10

entries. 101 connections are made and if it has 100 entries, then 1001 connections are made. A new improved version of generateBook has been made, but is not functional at the time of writing.

We are proud of this code because it creates the fight book, which was a big wish from Nikolai and it is a good example of the advantage of choosing object oriented PHP.

Logon

```
<div id="columns">
<div class="content">
<!-- post news -->
<div id="column1">
<?php
		include_once 'php/DBConnector.php';
		include_once 'php/Member.php';
		$username = $_POST['username'];
		$password = md5($_POST['password']);
		$mc = new MemberController();
		$member = $mc->recognizeMember($username);
		if ( $password === $member->password)
		{
			$_SESSION['member_id'] = $member->member_ID;
			$_SESSION['gender'] = $member->gender;
			$_SESSION['email'] = $member->email;
			$_SESSION['country'] = $member->country;
			$_SESSION['last_name'] = $member->last_name;
			$_SESSION['first_name'] = $member->first_name;
			$_SESSION['day_of_birth'] = $member->day_of_birth;
			$_SESSION['sign_in_date'] = $member->sign_in_date;
			$_SESSION['username'] = $member->username;
			$_SESSION['password'] = $member->password;
			$_SESSION['profile_image'] = $member->profile_image;
			$_SESSION['weight_class_id'] = $member->weight_class_id;
			$_SESSION['gym_id'] = $member->gym_id;

			printf("<script>location.href='index.php'</script>");
		}
```

This code is used to check if the username and password entered by the user on logon is correct or not. It starts with HTML, where the before mentioned CSS ID ”#columns” is being used for content. Then the class named “content” from the CSS is used, and at last, the ID “#column1” is used for displaying the content. PHP then opens, because now it has to check if the password was right for the username entered. At first, the DBConnector.php has to be included, since that file is connecting to the

database. Member.php is also included, because it needs to get the password of the entered username. Next is created two new variables which hold the entered username and the entered password. The password is md5 hashed, which is why md5 hashing on the entered password is required. The next variable goes inside the Member.php and hold the information inside the class MemberController. Next, a new variable is created to hold all the information on the member. This is done by calling the function recognizeMember with the parameter username in the MemberController. The Member.php file does the rest of the work. Now all that is left is checking if the password entered is the same as the password for stored in the member variable. If it succeeds then the session started at the top of the PHP file can continue and the different variables can be put into various session variables which can be used later on when the member is logged in. At the end of the if condition a printf is executed with a java script. This java script redirects the user to the file index.php.

Logon failure

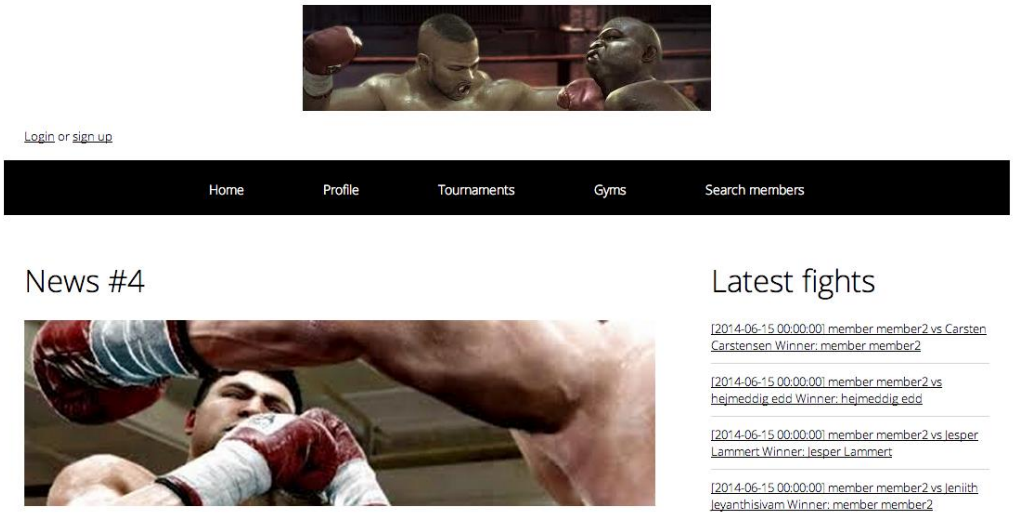
```
else
{
		session_destroy();
		echo "Wrong username or password";
}
```

This next code is the code that is executed if the condition on the last piece of code fails. At first the session that was started at the very top of the PHP file is destroyed, because a session is not needed if it doesn't hold any data. Next, an echo is executed, telling the user that either the username or the password was wrong.

The final system

This will be a description of how the system ended up. We will go through every page and explain what's on the page and which functions on every page.

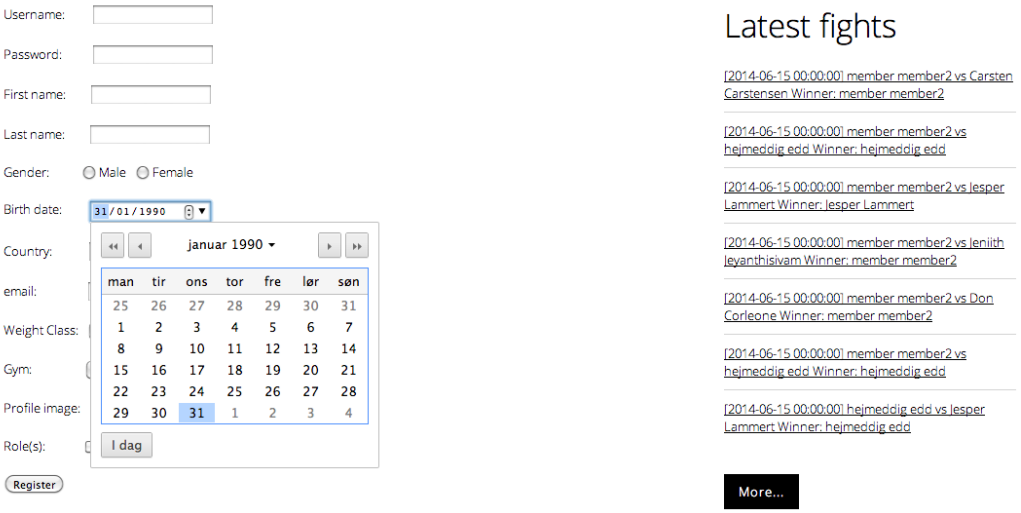
First we have the front page, which consists of a menu, a banner, news and the last fights.



Figur 21 - Screenshot of the systems front page

As you can see, there are login and sign up options in the left corner on the page. If you already have an account, you can log in and do whatever, you intended to do on the page. However, if you do not have an account, you will have to create an account first by clicking on the sign up link. The side is build up from normal design principles as described earlier. For example; log-in and sign up in the left corner. A menu with contrast colours. News headline, picture and description put close together. Air between non-related objects and so on.

This is the page you'll be directed to when signing up.

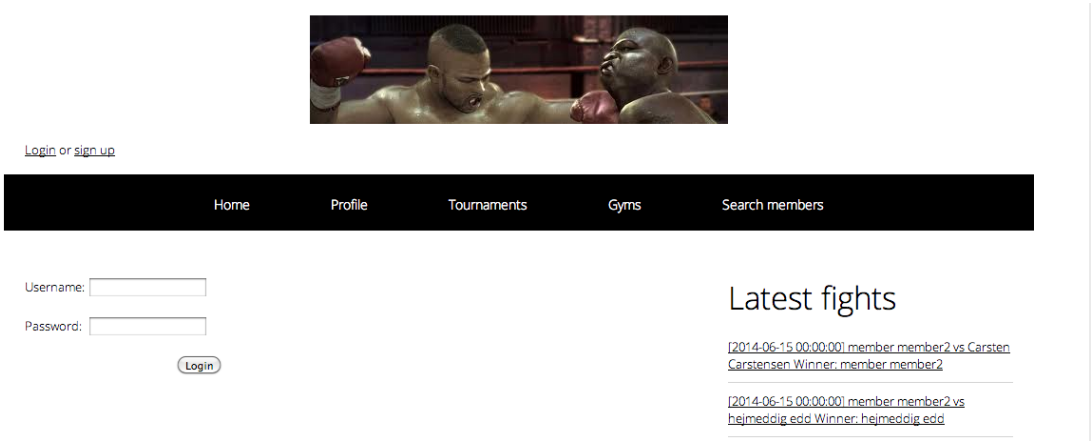


Figur 22 - Screenshot of creating a user in the system

When you have entered all the required information your profile will be created and you will now have access to more functions depending on which role you have chosen. If you chose coach you will be

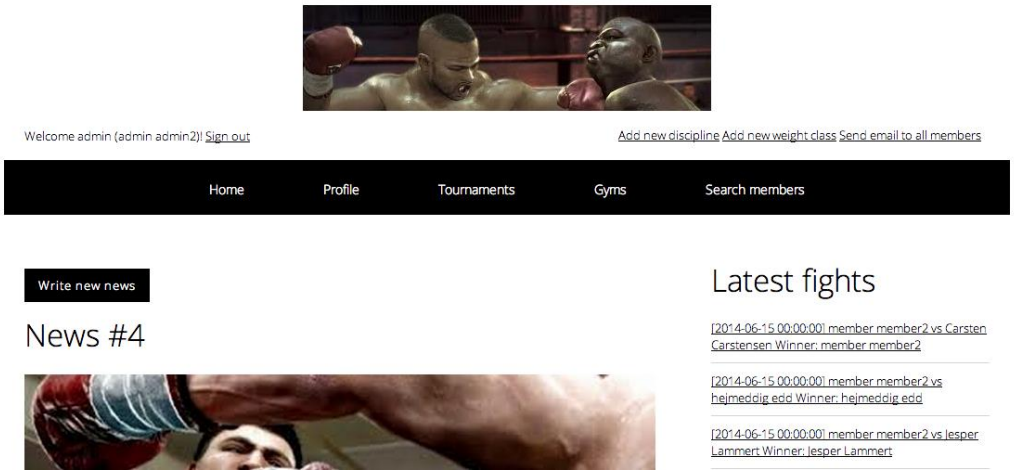
able to enroll fighters to a tournament. On birth date we have made a calendar to make it easier to find the right date, and there is a default date, to browsers that doesn't support this function, just so they can see the syntax.

When you have registered a user you will have to log in.



Figur 23 - Screenshot of log-in page in the system

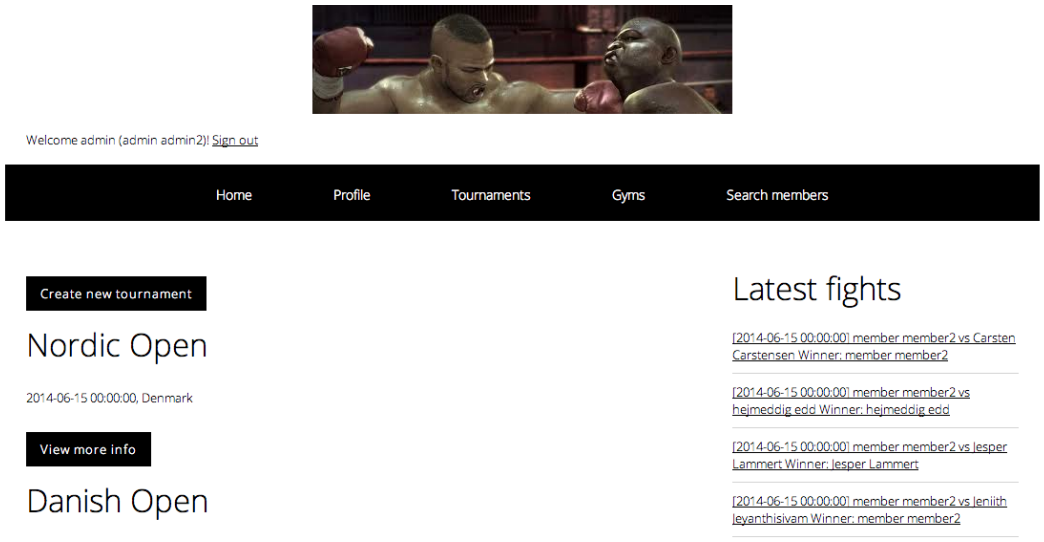
This page is simple, you can log in and that is it.



Figur 24 - Screenshot of admin functions in the system

As admin you have some options that fighter and coach do not have. In the right corner under the banner there is add discipline, add new weight class and send email to all members. These functions are only available for the administrator; the links do not even appear, if you are not logged in as an administrator.

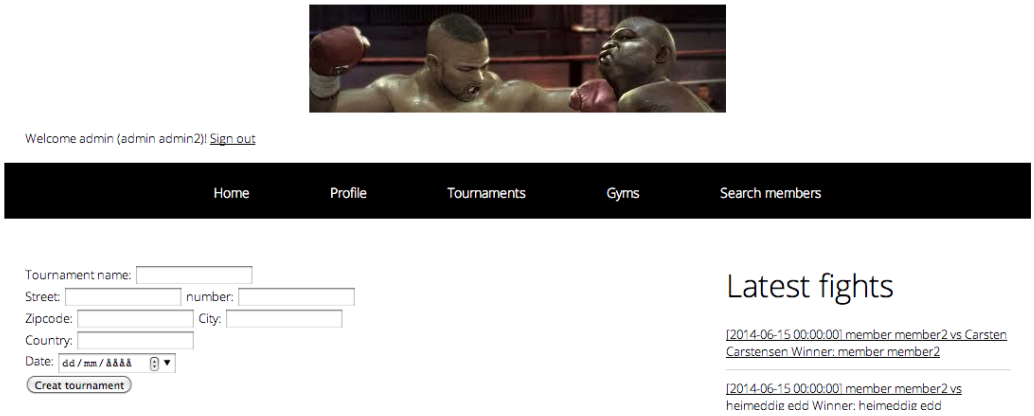
When you click on tournament, you are directed to this page.



Figur 25 - Screenshot of the tournament page in the system

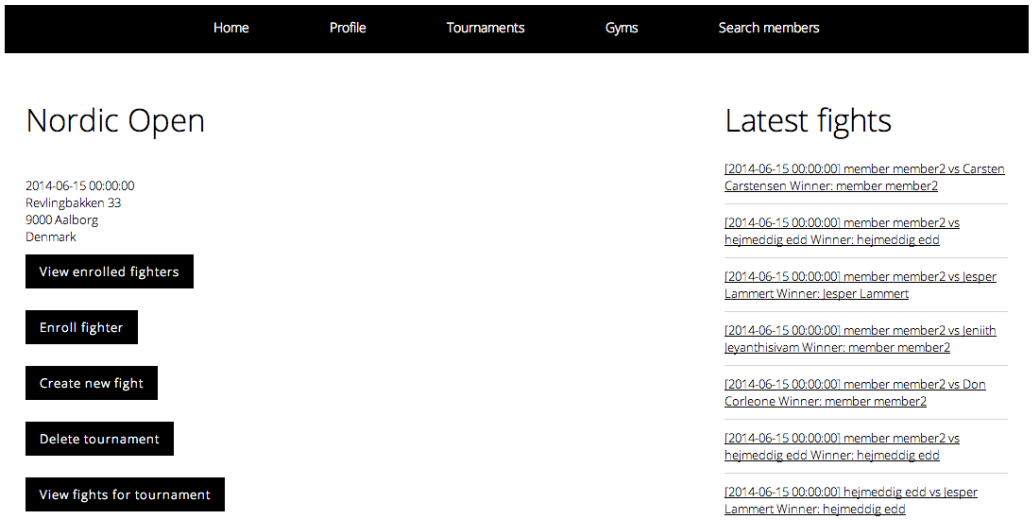
On this page you can create a tournament, which is also only possible for the administrator.

The page looks like this.



Figur 26 - Screenshot of create tournament in the system

If you clicked on view more info instead of create tournament you are directed to a page with a few more options.



Figur 27 - Screenshot of a specefic tournament in the system

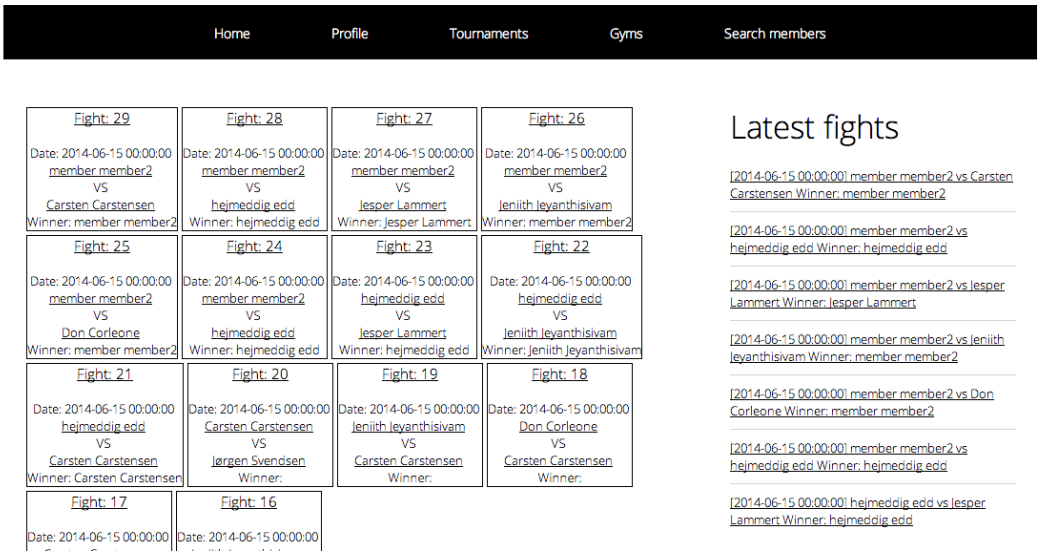
The options are mostly only available for the administrator. The “Enroll fighter” option is also available for coach, so he can enroll his fighter to a tournament. This is an important page for the administrator; this is where the administrator can create fight to the tournament, delete tournament and he gets the list of the enrolled fighters to match make.

The create fight option will direct you to this page.

Figur 28 - Screenshot of creating a fight in the system

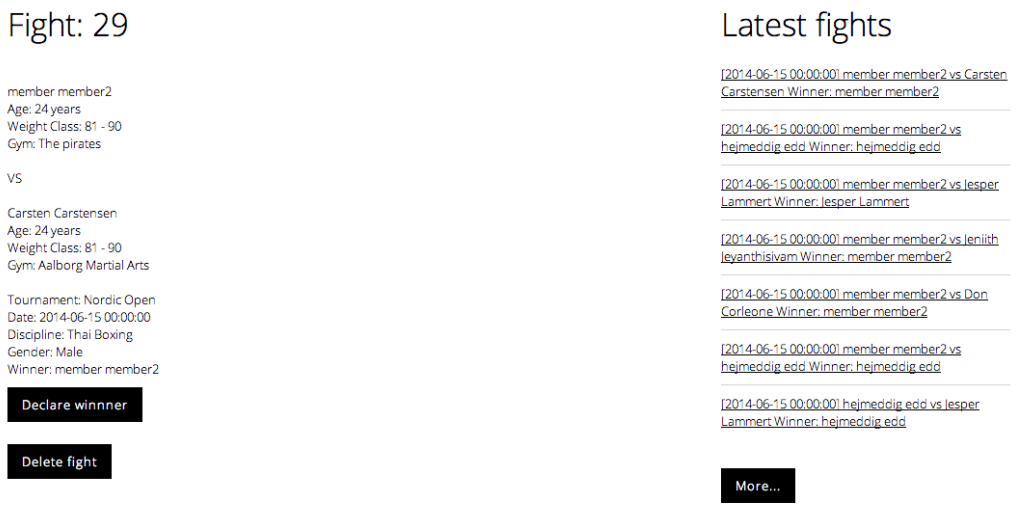
On this page you can chose the fighters and the judges attending to a specific fight.

When the fight is created and the administrator, judge or referee would like to declare a winner, they have to click on the option view fights for tournament under tournament, and they are directed to a page with all the created fights.



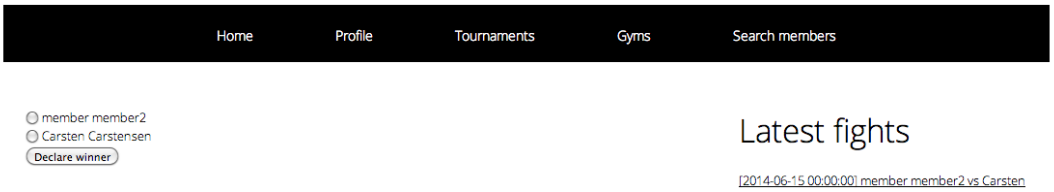
Figur 29 - Screenshot of already created fights in the system

Then they can choose a fight from this list.



Figur 30 - Screenshot of a fight with details in the system

He can now declare a winner.

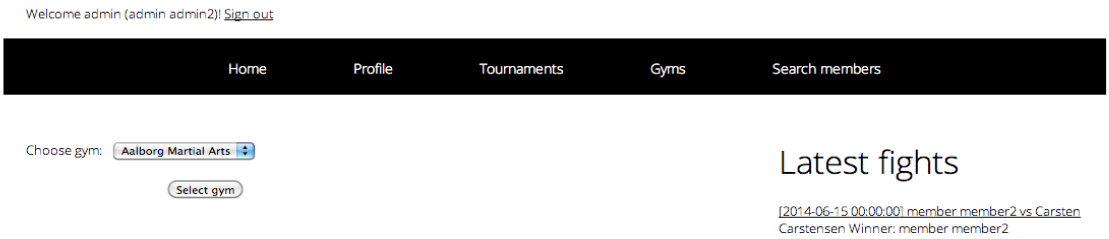


Figur 31 - Screenshot of declaring a winner in the system

We have made radio buttons to make it easy to choose a winner.

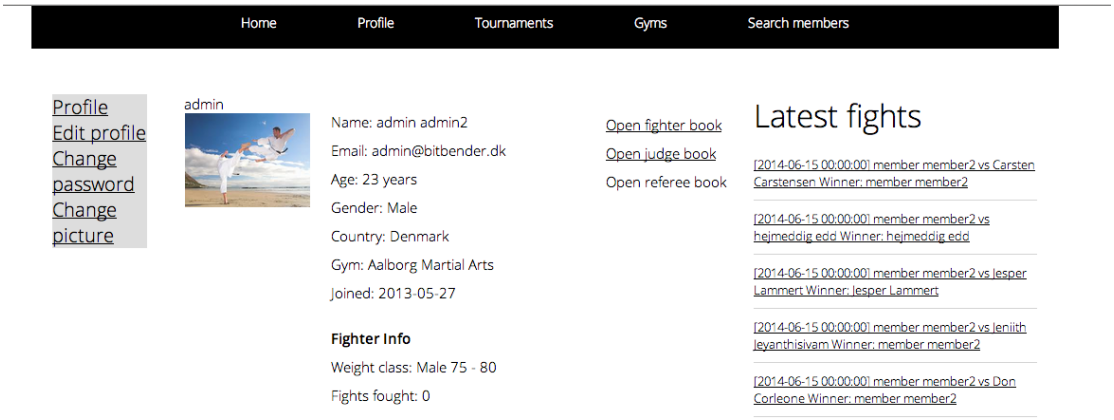
Under tournament we also have the enroll fighter, when you click on that, you can chose a gym from the list.

When you have selected a gym you will be able to choose a fighter from the gym and choose which discipline he will fight in.



Figur 32 - Screenshot of enrolling a fighter to a tournament in the system

When we click on profile to the left for tournament, we will see the profile on the person who is logged in.



Figur 33 - Screenshot of a profile in the system

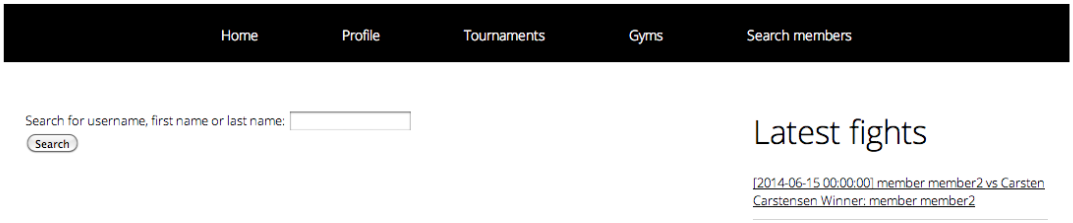
Beside all the obvious information we have the fighter, judge and referee book. This is where all the records are stored.

Date	First Name	Last Name	Age	Weight Class	Gym	VS.	First Name	Last Name	Age	Weight Class	Gym	Tournament	Discipline	Gender	Winner
2014-06-15 00:00:00>	member	member2	24 years	81 - 90	The pirates	VS.	Carsten	Carstensen	24 years	81 - 90	Aalborg Martial Arts	Nordic Open	Thai Boxing	Male	member member2
2014-06-15 00:00:00>	member	member2	24 years	81 - 90	The pirates	VS.	hejmeddig	edd	2014 years	75 - 80	The pirates	Nordic Open	Thai Boxing	Male	hejmeddig edd
2014-06-15 00:00:00>	member	member2	24 years	81 - 90	The pirates	VS.	Don	Corleone	31 years	75 - 80	The pirates	Nordic Open	Full Contact	Male	member member2

Figur 34 - Screenshot of a fighter book in the system

Then we have the gym in the menu, where the administrator can create a new gym, it is only the administrator that can do that.

At last there is the search member function, where you can write the name of a member and find his profile and can look in his fight book, if he or she is a fighter.



Figur 35 - Screenshot of the search page in the system

Now we have got an overview of the menu. However, we need to talk about latest fight, which have been consistent on all pages. This is just the latest fights, where you can see, who won the fight.

And then we have the news on the front page, where the administrator can write news about tournaments or anything he likes.

Chapter seven: Testing

Usability

In this part of the project, the system will be tested. When testing we will be making a usability test, to find errors in the system and see how the actually user interacts with it, by reusing the already made use cases. First some usability theory will be presented, and then our analysing method, the test and of course an analysis of the results.

Usability theory

The following theory is written from; *Rubin, Handbook of Usability Testing*.

You can find usability in every product and system, but it is rarely we think about it, before we use a system with poor usability. Poor usability is never the users fault, but the developers of the system. Poor usability can be fatal, because it makes the user of the system feel frustrated, anger or even stupid, so of course this user will not use the system again, but go the competitors system, even though yours can have better and smarter functions. Therefore, good usability is always important, especially if you want to sell the system or getting users to sign up. But what is good usability? Rubin has set up five criteria's to determine what a system should contain to have high usability, they are;

- Easy to use
- Easy to learn
- Effectively to use
- Satisfying to use
- Flexible and functional

Easy to use means that the users interacting with the system should make sense for the user. For example, buttons should always be visible and logical. If you cannot see it is a button, the system got bad usability.

The next criteria are *easy to learn.* This does not apply for expert systems like flight panels, which actually require an educating before using. Otherwise, a system should always be easy to learn for the users. The users here are the key. Who will use the system? Different users learn different. If the users

are old people, perhaps buttons and font size should be bigger and less choices in the system. This system would not make sense for a younger generation, because they already learned how a web shop normally works. Some systems require good usability for a wide audience, like an ATM.

A system should also be *effectively to use*. The user should not have any problems by achieving his goal in the expected timeframe. Effectively, also means that the system should be fast, in loading pages, data etc. If it takes too long, the user simply gives up. The system should also give the user information about errors and opportunities, to help and guide the user. Finally, the physical environment should be effectively, from the chosen hardware, design but also other criteria's like the place. For example, a screen that stands right in the sun can be hard to read and therefore have bad usability.

A system should furthermore be *satisfying to use.* The user should feel happy and relaxed when using and after using the system. But even better is if a system can surprise the user with an interface, function or something else, that the user hasn't seen coming. For example an even faster or better way to reach the users goal.

Finally a system should be *flexible and functiona*l. Flexible means that the users can use the system differently to achieve the same goals. They should not feel "trapped" in the system. They need opportunities, but at the same time there should be a limit set by the developers. Functional means that all the functions in the system should work every time, if not the users will not be satisfied.

The goal with a usability test is to find those places in the system where the users have problems or spend too much time. This will be done by creating different scenarios or reusing use cases to put up different goals for the test user. This should provide the developers with different kind of errors or problems in the system. These problems can come in a lot of different "sizes" and will be categorized in three categories. These are;

- Critical problems
- Serious problems
- Cosmetic problems

Critical problems will appear when the user cannot finish his given task. These problems will give the user a feeling of frustration and it indicates that the system does not work as intended.

Serious problems will be found when the user interacts with the system in a wrong way. He will achieve his goal, but he has not used the system as intended and spends more time on the given task. This could indicate that the system is not build logical.

The last kind of problems is the *cosmetic problems*. These can be fixed by changing the graphical design in the system. For example this could be a searching box, with almost the same colour as the background and therefore hard to locate for the user.

Before finding the problems in the system, the tasks should be formed. These tasks can have a purpose to test a function, but also the purpose of finding information in the system. When the test user is testing the system, he will be supervised by a test leader, time taker and one or two people taking notes. The test leader's job will be to talk with the user. First of all explain that the test is made to test the system, not to test the particular users IT skills. This is very important so the test person feel comfortable and relax during the test, this will make test more naturally. The test leader will also explain that the test user should think loud. This means the test user should say out loud everything he does and thinks during the test. This can be really hard, and the test leader should be good at reminding about this, in a way that does not frustrate the test user. It is also the test leader who explains the goals, if the test user does not understand these. The test leader may never interfere in the test and help or guide the test user. The test user may only go the next task when the test leader approves. The time taker will note the time for every task and nothing else. The note taker will be taking notes during the test. Write down problems and misunderstandings in the system, during the test. The note taker will also look at the test users' body language, for frustration and other visible feelings.

Analysing method

Before the test is conducted, a choice has to be made for the analysing phase. Should the test be made in an original way with camera setup? If so, this can create enormous amount of data that should be analysed. This is done with one camera on the test user to watch for body language and a recording of the screen where the test is being made. All this video material should be watched, analysed and discussed by the test team, which will take many resources. But when choosing this method it is possible to go back and watch the video again for problems and almost all errors will be found.

Another method is IDA - **I**nstant **D**ata **A**nalysis (*Jesper Kjeldskov, Mikael B. Skov og Jan Stage, "Instant Data Analysis: Conducting Usability Evaluations in a Day").* With this method, there is no video recording. The problems should be identified during the test and analysed and discussed right after the test, when it is still fresh in the memories of the testing team. This will take a lot less resources and still find all the important errors in the system. Some problems may be overlooked, so the test team has to be more focused during the test, to identify all problems. It is really important that the note takers put down *where* the problem is and *why* the problem occurs.

We have decided to go with the IDA method. We do not need a big test set-up for this and we can use our resources elsewhere. We have tried this method before, so we feel we can spot the problems when they occur. With four group members, we also have enough for taking the needed notes during the test.

The usability test

Throughout our project we have wanted to make minimum one usability test. This test should be carried out on Nikolai, so he will be able to evaluate the system as well. This of course requires that Nikolai have not seen the system, and he have not. Otherwise, the test would give us false results. We want to know if the systems usability is good, even the first time a person enters the web page and even more important, we want to see with our own eyes how a new user will navigate in the system. But most important, the usability test will also be a big part of Nikolai's evaluating, so he can give his comments on the system afterwards.

Besides the test on Nikolai, we decided to make two additional tests, to try in see how non-committed users will use the test. This is also done to compare the results to each other and see if any problems go again.

The test questions

We want to use our use cases to test if the different functions work and are logical located. We have located all the functions the way we find it most logical, both from our own discussions but also from internet conventions. Besides the use cases, where they all have a described goal, we will also make some additional tasks for the usability test. We have put in more functions since the use cases, so of course these also have to be tested. The test questions is build up with a story line, so it make more

sense for the test person to reach a goal, but also to make the test more naturally. In all there are 17 tasks in the usability, starting with a fighter, later goes into a coach and in the end the test focuses on the administrator part of the system. Here you can see the three first questions. The rest can be read in the appendix.

---

1. You started to get interested in Martial Arts, but before you can train and can compete in tournaments, you need an account on the web page.
   Create a new user on the web page, join the Pirates gym and check yourself as a fighter
2. On the frontpage you see some news, but it's not the whole newsfeed you can read from the frontpage.
   Read the full news.
3. You remember that it was a really good summer where you put on 8 kilos. Your information in your profile doesn't match this.
   Edit your profiles information to fit your actually weight.

The test persons

It does not matter what persons we use for the test. We need people around the same age as most of the members in the gym and with a minimum to medium IT skills. Aalborg Martial Arts members are used to use an online system, just a really bad one, but this tells us they now how to navigate a home page. Not a competence all people have, just ask people who made usability tests on older people. The gyms members are primary young people. Here you can see the people we used for the test. They vary in some ways like gender, nationality and martial arts knowledge.

Testperson	IT skills	Age	Nationality	Martial arts knowledge
Nikolai Fedderholt	Medium	22	Denmark	High
Keit Matas	Medium	22	Estonia	Low
Jeppe Bach	Medium	21	Denmark	Low

The usability problems

All the usability problems will all be numbered with “P1” etc. It does not matter who found the problem or how many times it occurs, as long as there is a problem we will write it down in a list. After finding all the problems, they will be categorized into cosmetic, serious and critical and in the end a solutions will be discussed.

Listed usability problems:

- P1: Did not see the “more..” button in the bottom of the front page, to read all the news.
- P2: The systems does not update when changing gym. It runs in sessions, meaning that the users have to log out and in again before, they can see the changes, even though it happens in the system.
- P3: Forgot to put in password before making changes in the profile.
- P4: When enrolling fighters, already enrolled fighters also occurs in the list.
- P5: A wrong username was entered. The user gets to a dead end and cannot remember the username.
- P6: When creating a news, the text stops after using the symbol **‘** (apostrophe).
- P7: It is hard to find the mass email function.
- P8: Feeling frustration over “Latest fight”.
- P9: System is a bit slow, which causes frustration.
- P10: Birthdate is unclear.
- P11: Confusing design in tournaments
- P12: Mass email is not on every page

Categorized usability problems:

- Cosmetic problems: P1, P7, P8, P10, P11, P12
- Serious problems: P3, P5, P6, P9, P4
- Critical problems: P2

Suggestions to changes

Let us start whit all the cosmetic problems, then move on to the serious and in the end, the two critical problems. The test persons will be referred to as users in section of the report.

The first problem where the users didn't see the "more.." button could be fixed by moving it up on the page, so that only one news will appear on the front-page. By doing this, the users do not have to look for it but they can see it all the time. The text should also be changed to "read more news..". The next cosmetic problem is to find the mass email function. It only appears on the front-page, so by making it appear on all pages should avoid this problem. One cosmetic problem that did not show up in a traditional way, were the design of latest fights. The users did not use this, but some of them commented that "wow that is confusing". An error has also occurred here. The date shows it like this "[2014-06-15 13:14:15]" simply too much information. One way to fix this could be to just have the date or perhaps nothing before clicking on the fight. After the date and time this information appears "Daniel Danielsen vs. Carsten Carstensen Winner: Carsten Carstensen. This could also be made clearer by highlighting **VS.** and **Winner**. The birthday when registering is also confusing because it is written the MySQL way: 2013-05-23 and not 23-05-2013. By doing this change, this problem should also be fixed. The design in tournaments also confused the users. This could be fixed by putting the text "Nordic open" the date and the "view more info" button in a box together. This way the users know which buttons are for the specific tournament. The last cosmetic problem is again the mass email function. One user knew it were in the top, because he checked them out earlier. When he needed it, suddenly it was gone and it took some time to find it again. This could again be fixed by having it on all pages.

The next problems are the serious problems. The users forget to put in there password before making changes. This function should perhaps be removed, otherwise it should be moved down to the "save changes" button. When a wrong password or username is entered in the system tells the user this. But there is no retrieve password function. This should be made before the system is up running. A faster solution could be that the system tells the user to contact the admin, so that he can change the password. An error has shown when one user created news. The text stops after using symbols or Danish letters. This is an error that of course has to be changed, so these can be used. It is only in the news this error appears. The users complain about the system being a little slow. This is because of all the connections to the database, when the page is loading latest fights. The system makes 10 x

connections for each fight + one. This could actually be done differently, so the page only makes one connection, making the system faster. Another option is to remove latest fights. The last serious problem is that already enrolled fighters appear in the list, when the coach wishes to enroll more fighters. This is because fighters can compete in a tournament in more than one discipline. The system should not make it possible for the coach to check off “full contact” if the fighter is already enrolled in this discipline. This is not a critical problem because when enrolling a fighter two times with the same discipline, he does not appear twice in the tournament information, but still this should not be possible. The user just think he did not do it the first time or an error happened.

There is only one critical problem in the system. That is when the user makes changes in his profile, the system does not update this on the page, but it does in the database. The users have to log out and in again to see the changes. The page runs in sessions and does not update this in the front end. This can easily be fixed by making updated sessions, but it is a critical error from our side. The users really get frustrated when they use the system right, but nothing happens.

Chapter eight: Acceptance

In this chapter we will write about the acceptance process we had with Nikolai after the usability test. He of course had some comments to the system, some smaller changes in some functions. In this process we had a discussing with Nikolai, so we could find the best possible solution to his comments.

In overall he was really happy about the system and our cooperation with him. Therefor we also made a deal to help him put the website up for using and make sure it works as agreed.

Acceptance from Aalborg Martial Arts

After Nikolai tested the system, he had some few comments to make it better. The first change he would like is that fighters should be able to withdraw from tournaments all the time. Right now the system have a deadline on 5 days, Nikolai really like this feature, but it should be changed so he can manage the deadline for enrolling fighters, plus fighters should be able to withdraw after this. This is because fighters can of course get sick or injured and as Nikolai says, “it is unsporting to determine a fight that hasn’t been fought”. Instead, this fight should be cancelled.

Another comment is that it is hard for Nikolai to know who are going to be judges and referee before the day of the tournament. This means that Nikolai cannot create the upcoming fights before the tournament day. We wanted to make him a list with enrolled fighters, we did, but we also made it possible to manage the fights before the tournament day, so this “extra” function we would like to be complete for him. One possibility is to simply remove judge and referee from the fight in the system. Then their judge books will not be updated, so that solution will not work properly. Instead, he should be able to pick a Judge1, 2 and 3. These are not actually judges, but just a name in the system. After the tournament, he should then be able to edit the fights and putting in the real judges. This way he can still plan the fights, without knowing the judges beforehand, but a bit work is required after to update the fights, so all fighter- judge- and referee books are updated.

Nikolai would also like the enrolled list of fighters in a tournament to be sorted after any specific criteria. You should be able to pick “sort after…” and then in a dropdown menu chose “gym” or whatever the user finds fit. This is because fighters and others like to go in and see who is coming from other gyms. A lot of them have some rivals, which they really like to fight.

Furthermore, Nikolai wrote some words about our cooperation, they were written in Danish so this is a translated quote; "*The group have done a great job. They have throughout their project had a good contact with me and holding me updated with the process of the system. It has been a pleasure to work with them and see the final system.*"

Nicolai Fedderholdt
Vice-Chairman Aalborg Martial Arts
Nordkraft, Teglgårds Plads 1, Niveau 5, 9000 Aalborg
Tlf: +45 20 67 38 37 // E-Mail: N.Fedderholdt@gmail.com

Agreement with Aalborg Martial Arts

Aalborg Martial Arts would actually like to use our system, because it is already better then OKBR.net. This we are of course really happy to hear and would of course help with this. We are just happy to see our system being used.

We have made a deal with Aalborg Martial Arts regarding handing over and helping with implement the system to a web server. Jeniith Jeyanthisivam will be the person with this main responsibility. He is the member of our project group, who is also a member of Aalborg Martial Arts. The agreement is to do this after the exam period or in the summertime.

Chapter nine: Conclusion and reflection

Reflection

In this reflection, we will look back at our work process and chosen tools throughout the project.

Throughout this process, we have had four courses. Where three of them were relevant and useful for the project, which was the database course, software engineering course and project supported programming. The last one was on was in another faculty and had nothing to do with our project.

We also wasted some time on one group member, Anders, who was kicked out of the group. He did not show up, and did not do his assignments, but the worst thing was the lack of communication. We could never get in contact whit this group member. The group of course just keeps on working, but after repeating episodes it changes to atmosphere to this feeling "*why should I do his assignments again and show up, when he can get away with it again and again".* After we discontinued the cooperation with him, there was a way better workflow in the group. Documents were not missing all the time, because his lack of commitment.

As we mentioned earlier, we learned about Gantt chart as well as databases. What our main problem what the misunderstanding of the waterfall method. The waterfall method's main focus was that you do not look back when you are done with a phase. We were too hooked on this that we used too much time on the first phases which resulted in less time to the development phase. Perhaps a better structured Gantt chart could have helped here.

When we got through the first phases we looked at our development method again as we had to hand in the development approach to the software engineering course. We discovered that there is a waterfall method with bit flexibility to go back and change some of the models so it fits the system we have created, but this turned out to be one of the biggest problems. We were not prepared for changes in the earlier phases, which created a lot of work, every time a new function had to be implemented.

We tried to get every phase completed before started on a new one. This was also done, but when we came to the implementation phase, we did not have enough time for this. We also discovered how hard it is to follow the instructions we made in the earlier phases. When programming it is really hard to remember everything and get it implemented, even though it is already documented. We had to make an abstract graphical design and describe the design rules we wanted to follow instead of

creating a total design. We did not have the time and needed to start on the programming. In the programming phase, there was more focus on getting the system to work, instead of design and usability.

The waterfall was good for documenting, getting started and creating overview, but we learned how hard it is to actually follow your documents. Especially when we concentrated on getting it to work, perhaps when our programming skills get better, the waterfall method would work better.

In relative to the implementation phase, we intended to help each other with the programming. We have two experienced programmer and two members, who had little experience with programming. Therefore, it was intended that the two experienced programmers should evaluate the programming tasks and give the other two small easy tasks, but even that was too difficult with the limited time, because the two experienced ones had to correct the errors in the code and make it work with the rest of the code. Because of that, it ended up with that the experienced ones used a lot of time on the coding phase, more than expected. The idea of small tasks was good, but the limited time and the level of tasks were not suitable for us. If we had more time, it probably would have been a great idea as doing it is the best way to lean programming.

Conclusion

The purpose of this project was to implement a database system. We found a costumer and then we needed to find a solution to the problem Aalborg Martial Arts had with their administration system.

Now we have the product that was wanted from the client, it does have some errors and omissions and have some features that we at the beginning thought was too big of a challenge considering the limited time we had, but they had to be implemented to get a fully working system.

The cooperation between Aalborg Martial Arts was more or less effortless. Nicolai was very easy to set up a meeting with and was very specific about his wishes. He was helpful en every possible way he could be helpful. We even tested the system on him, to check for usability problems.

We had a few major problems with the system that we need to correct before implementing it for him. However, he seemed satisfied with the cooperation with us. The implementation will happen after we have handed in the project.

We used the waterfall method, which means that the programming phase was one of the last phases so we did not realize how difficult the phase was until we were there. We thought we were ready,

because of the all the documentation, but it's really hard to remember and implement everything from the documentation, when you are really frustrated with the programming.

Overall we made the wanted system and more. Not only an administration system, but it can also make the fights before the tournament, making it a small tournament planner system. Some small changes will though happen, before implementing for Aalborg Martial Arts.

References

Addbot, & et. al. (2013, 2 26). *Peter Chen - Wikipedia, the free encyclopedia*. Retrieved from http://en.wikipedia.org/wiki/Peter_Chen

Chapple, M. (2013). *Should I Normalize My Database? Reasons Not To Normalize*. Retrieved from http://databases.about.com/od/specificproducts/a/Should-I-Normalize-My-Database.htm

Elmasri, R., & Navathe, S. B. (2007). *Fundamentals of Database Dystems* (5. ed.). Pearson Education.

Gamborg, N. (2013, 1 20). *Gestaltlovene*. Retrieved from http://www.nielsgamborg.dk/?p=gestaltlovene

High Arch, D. (2009, February 9). *Stackoverflow.com*. Retrieved from http://stackoverflow.com/questions/539324/what-is-a-good-kiss-description-of-boyce-codd-normal-form

Lorge Parnas, D., & Clements, P. C. (1986, 2). *A Rational Design Process: How and Why to Fake It*. Retrieved from http://sict.moodle.aau.dk/file.php/797/General_Course_Materials/literature/Parnas_Fake.pdf

Mathiassen, L. (2000). Object Oriented Analysis and Design.

Nielsen, J. (n.d.). *Ten Usability Heuristics*. Retrieved from http://zonecours.hec.ca/documents/H2010-1-2357287.portionOK.pdf

Online Kickboxing Regristration. (n.d.). Retrieved 2013, from OKBR.net: www.okbr.net/club_list.php

Parekh, N. (2013, 05 27). *Buzzle.com*. Retrieved 2013, from http://www.buzzle.com/editorials/1-5-2005-63768.asp

Riccardi. (2002, 6 28). *Riccardi - Riccardi_ch4.PDF*. Retrieved from http://www.aw-bc.com/info/riccardi/database/Riccardi_ch4.PDF

Satalkar, B. (2011, 8 26). *Comparison Between Waterfall Model and Spiral Model*. Retrieved from http://www.buzzle.com/articles/comparison-between-waterfall-model-and-spiral-model.html

wikimedia. (n.d.). Retrieved from https://upload.wikimedia.org/wikipedia/commons/thumb/7/7c/Relational_database_terms.svg/2000px-Relational_database_terms.svg.png

Chapter ten: Appendix

The Appendix consists of

- A CD
    - Design mock-up
    - DMD
    - ERD
    - Flowchart
    - Gantt chart
    - Homepage
    - Navigation diagram
    - Pictures of work process
    - SQL code
    - Use case diagram
- ERD in 3A paper
- DMD in 3A paper
- SQL-code

ERD

DMD

SQL-code

```
CREATE DATABASE AMA;
USE AMA;
ALTER SCHEMA AMA DEFAULT CHARACTER SET utf8 ;
SET storage_engine=INNODB;

CREATE TABLE Age_Group(
Age_ID INT AUTO_INCREMENT,
Max_Age INT NOT NULL,
Min_Age INT NOT NULL,
Age_Class_Name VARCHAR(50) NOT NULL,
PRIMARY KEY(Age_ID)
);

CREATE TABLE Weight_Class(
Weight_Class_ID INT AUTO_INCREMENT,
Gender VARCHAR(6) NOT NULL,
Min_Weight DOUBLE NOT NULL,
Max_Weight DOUBLE NOT NULL,
PRIMARY KEY(Weight_Class_ID)
);

CREATE TABLE News(
News_ID INT AUTO_INCREMENT,
Content TEXT NOT NULL,
Headline VARCHAR(255) NOT NULL,
Image VARCHAR(255) NOT NULL,
`Date` DATETIME NOT NULL,
PRIMARY KEY(News_ID)
);

CREATE TABLE Gym(
Gym_ID INT AUTO_INCREMENT,
Gym_Name VARCHAR(255) NOT NULL,
Gym_Logo VARCHAR(255) NOT NULL,
PRIMARY KEY(Gym_ID)
);

CREATE TABLE Discipline(
Discipline_ID INT AUTO_INCREMENT,
Discipline_Name VARCHAR(50),
PRIMARY KEY(Discipline_ID)
);

CREATE TABLE Member(
Member_ID INT AUTO_INCREMENT,
Gender VARCHAR(6) NOT NULL,
Email VARCHAR(255) NOT NULL UNIQUE,
Country VARCHAR(100) NOT NULL,
First_Name VARCHAR(255) NOT NULL,
Last_Name VARCHAR(255) NOT NULL,
```

```
Date_of_Birth DATE NOT NULL,
Sign_in_Date DATE NOT NULL,
Username VARCHAR(255) NOT NULL UNIQUE,
`Password` VARCHAR(255) NOT NULL,
Profile_image VARCHAR(255) NOT NULL,
Weight_Class_ID INT NOT NULL,
Gym_ID INT NOT NULL,
PRIMARY KEY(Member_ID),
CONSTRAINT weight_class_fk FOREIGN KEY (Weight_Class_ID) REFERENCES
Weight_Class(Weight_Class_ID),
CONSTRAINT gym_id_fk FOREIGN KEY (Gym_ID) REFERENCES Gym(Gym_ID)
);

CREATE TABLE Tournament(
Tournament_ID INT AUTO_INCREMENT,
Tournament_Name VARCHAR(255) NOT NULL,
ZipCode INT NOT NULL,
City VARCHAR(100) NOT NULL,
Street VARCHAR(100) NOT NULL,
House_number VARCHAR(100) NOT NULL,
Country VARCHAR(100) NOT NULL,
`Date` DATETIME NOT NULL,
PRIMARY KEY (Tournament_ID)
);

CREATE TABLE MemberDiscipline(
Member_ID INT,
Discipline_ID INT,
Tournament_ID INT,
PRIMARY KEY(Member_ID, Discipline_ID, Tournament_ID),
CONSTRAINT member_id_fk FOREIGN KEY (Member_ID) REFERENCES Member(Member_ID),
CONSTRAINT discipline_id_fk FOREIGN KEY (Discipline_ID) REFERENCES
Discipline(Discipline_ID),
CONSTRAINT tournament_id_fk FOREIGN KEY (Tournament_ID) REFERENCES
Tournament(Tournament_ID)
);

CREATE TABLE Rights(
Right_ID INT AUTO_INCREMENT,
`Right` VARCHAR(255) NOT NULL,
PRIMARY KEY(Right_ID)
);

CREATE TABLE MemberRights(
Member_ID INT,
Right_ID INT,
PRIMARY KEY (Member_ID, Right_ID),
CONSTRAINT member_id_fk_1 FOREIGN KEY (Member_ID) REFERENCES
Member(Member_ID),
CONSTRAINT right_id_fk FOREIGN KEY (Right_ID) REFERENCES Rights(Right_ID)
);

CREATE TABLE Functions(
```

```
Member_ID INT,
ReceiveEmailOnNewTournaments TINYINT(1) NOT NULL,
ReceiveEmailOnNews TINYINT(1) NOT NULL,
ReceiveEmailOnUpdatesOnTournaments TINYINT(1) NOT NULL,
PRIMARY KEY (Member_ID),
CONSTRAINT member_id_fk_2 FOREIGN KEY (Member_ID) REFERENCES
Member(Member_ID)
);

CREATE TABLE Role(
Role_ID INT AUTO_INCREMENT,
Role VARCHAR(50),
PRIMARY KEY (Role_ID)
);

CREATE TABLE MemberRole(
Role_ID INT,
Member_ID INT,
PRIMARY KEY (Member_ID, Role_ID),
CONSTRAINT member_id_fk_3 FOREIGN KEY (Member_ID) REFERENCES
Member(Member_ID),
CONSTRAINT role_id_fk FOREIGN KEY (Role_ID) REFERENCES Role(Role_ID)
);

CREATE TABLE Team(
Team_ID INT AUTO_INCREMENT,
Coach_ID INT NOT NULL,
Fighter_ID INT NOT NULL,
Tournament_ID INT NOT NULL,
PRIMARY KEY (Team_ID),
CONSTRAINT coach_id_fk FOREIGN KEY (Coach_ID) REFERENCES Member(Member_ID),
CONSTRAINT fighter_id_fk FOREIGN KEY (Fighter_ID) REFERENCES Member(Member_ID),
CONSTRAINT tournament_id_fk_1 FOREIGN KEY (Tournament_ID) REFERENCES
Tournament(Tournament_ID)
);

CREATE TABLE Fight(
Fight_ID INT AUTO_INCREMENT,
Fighter_1_ID INT NOT NULL,
Fighter_2_ID INT NOT NULL,
Referee INT NOT NULL,
Judge_1 INT NOT NULL,
Judge_2 INT NOT NULL,
Judge_3 INT NOT NULL,
Tournament_ID INT NOT NULL,
Winner INT,
Discipline_ID INT,
PRIMARY KEY (Fight_ID),
CONSTRAINT tournament_id_fk_2 FOREIGN KEY (Tournament_ID) REFERENCES
Tournament(Tournament_ID),
CONSTRAINT judge_1_id_fk FOREIGN KEY (Judge_1) REFERENCES Member(Member_ID),
CONSTRAINT judge_2_id_fk FOREIGN KEY (Judge_2) REFERENCES Member(Member_ID),
CONSTRAINT judge_3_id_fk FOREIGN KEY (Judge_3) REFERENCES Member(Member_ID),
```

```
CONSTRAINT referee_id_fk FOREIGN KEY (referee) REFERENCES Member(Member_ID),
CONSTRAINT fighter_1_id_fk FOREIGN KEY (Fighter_1_ID) REFERENCES
Member(Member_ID),
CONSTRAINT fighter_2_id_fk FOREIGN KEY (Fighter_2_ID) REFERENCES
Member(Member_ID),
CONSTRAINT winner_id_fk FOREIGN KEY (Winner) REFERENCES Member(Member_ID),
CONSTRAINT discipline_id_1_fk FOREIGN KEY (Discipline_ID) REFERENCES
Discipline(Discipline_ID)
);
```