Online WebSolution

Lasse Trier

Kongens Lyngby 2006
IMM-B-ENG-2006-58

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Resumé

I denne rapport beskrives udviklingen af et Content Management System (CMS). Jeg beskriver udviklingen, fra ide over analyse og design til implementering og test af det endelige produkt. Det færdige produkt er blevet et program, hvor man, via browseren, kan oprette og designe en webløsning, hvori der kan indsættes og redigeres i sider, tekst og billeder. En færdig opbygget webløsning hvor mit program er benyttet kan ses på www.teaterkatalysator.dk.

Medfølgende denne rapport vil være en CD med kildekoden, en installationsfil, standard databaser, samt billeder og denne rapport i pdf-format.

Summary

This thesis is describing the development of a Content Management System (CMS). I am describing the development from idea, through analysis and design, to the implementation and test of the final product. The product is a program, in which a user, through the browser, can build and design a websolution. In this solution it is possible to insert and edit pages, text and images. A complete websolution build with my program is accessable at www.teaterkatalysator.dk.

This thesis is accompanied by a CD including the complete sourcecode, an install file, default databases, images and the thesis in pdf-format.

Forord

Denne rapport udgør dokumentationen af et IT-Diplom-eksamensprojekt.

Vedlagt rapporten er en CD hvorpå det udviklede program findes både som kildekode og som en installerbar fil.

Fokus i denne rapport vil først være at beskrive arbejdsgang, metoder og teknikker brugt til løsning af opgaven. Derefter vil der fokuseres på programmelets opbygning, design og testfase.

Lyngby, Juli 2006

Lasse Trier

Tak til...

Jeg vil gerne takke teaterforeningen TeaterKatalysator og i særdeleshed foreningens formand Nina Bundgaard, der har været meget interesserede i min ide. Derfor har jeg også haft løbende kontakt med Nina gennem hele forløbet, hvor en stor del af designet, er blevet testet af hende. Dette har udmyntet sig i en endnu bedre brugerflade og desuden i websiden www.teaterkatalysator.dk, der er opbygget med mit endelige produkt. Det har været dejligt at mærke at TeaterKatalysator har kunne bruge mit produkt og se det i funktion. Især med henblik på at netop de, er en del af den målgruppe jeg gerne ville have fat i.

Indholdsfortegnelse

Kapitel 1

Introduktion

1.1 Introduktion

I internettets barndom, var der mange forskellige metoder til at dele information med hinanden. Men først efter at Http-protokollen og Netscape-browseren fandt indpas begyndte internettet og World Wide Web at blive allemandseje. Herefter er det gået meget stærkt, og nu er der intet større firma med respekt for sig selv, der ikke har en webløsning[1] tilknyttet. Derudover er der flere og flere privat-personer, der har adgang til internettet hjemme.

Webløsninger

Til at starte med, var det fortrinsvis firmaer med egen it-afdeling, der udviklede deres egne webløsninger, men efterhånden som internettet blev mere udbredt, og interessen for at eksponere sig på dette medie blev større, åbnede markedet sig for web-baserede reklamevirksomheder, hvis formål var at udvikle webløsninger til firmaer. Samtidig begyndte mange private personer at lave hjemmesider, især efter at Microsoft lancerede frontpage, i deres office-pakke, der gjorde det meget

[1] Se bilag F for en liste med de begrebsforklaringer jeg har valgt at benytte rapporten igennem.

simpelt at lave en webløsning. Herefter er der kommet et utal af programmer til at lave webløsninger. Mange web-hoteller har f.eks deres egne programmer, hvor man med meget enkle midler kan sætte sin egen løsning op.

Teknikken bag

Som udgangspunkt består en webside af en statisk fil i sproget HTML[a]. Dette sprog blev udviklet for at publisere rapporter mellem universiteter. Efterhånden som websider blev adopteret af den private sektor til at reklamere, opstod hurtigt et ønske om at gøre websiderne mere interessante for kunden. Desuden opstod behovet for at kunden kunne interagere med websiden for at kunne købe eller bestille produkter. Disse behov er forsøgt opfyldt ved udviklingen af bl.a. PHP, ASP, JavaScript, Flash og senest ASP.Net. Samtidig er databaser blevet en stor del af webløsninger for at gemme kundeinformationer, håndtere produktkartotek osv.

[a] HyperText Markup Language

Som beskrevet i ovenstående afsnit, har tekniske tiltag betydet at webudvikling er blevet en del mere indviklet og kræver et dybere indblik i programmering, end tilfældet var med statiske HTML-sider. I takt med at kompleksiteten i webløsninger blev større, og ønsket om at programmeringsmæssigt ukyndige stadig havde mulighed for at rette dele af deres webløsningen, såsom tekst og billeder, opstod de såkaldte Content Management Systems (CMS).

1.2 Formål

Formålet med dette projekt er at udvikle et CMS. Som beskrevet ovenfor, er CMS'er lavet, for lettere at styre indholdet af en webløsning. Derfor vil jeg fokusere meget på, at systemet bliver så intuitivt og let tilgængeligt som muligt. Af samme grund vil jeg designe systemet, så der ikke er behov for at installere ekstra programmer på brugerens computer. Derfor skal al funktionalitet kunne administreres direkte i browseren. Systemet skal give brugeren mulighed for, at kunne oprette en webløsning fra en tom side og derefter kunne redigere i indholdet, herunder menustrukturen, tekst og billeder. Et vigtigt komponent i systemet, vil være at kunne gøre løsningen så personlig som muligt. Derfor skal det være muligt, at bestemme udseendet af webløsningen, på så mange parametre som muligt, mens der stadig er tanke på tilgængeligheden og kompleksiteten, set fra brugerens side.

1.3 Målgruppe

Målgruppen til systemet er brugere, uden kendskab til webprogrammering. Brugeren er interesseret i selv at kunne designe sin webløsning ved brug af egne billeder og grafik. Brugeren vil have mulighed for let at kunne redigere i indholdet på løsningen. Brugeren ønsker fleksibilitet i form af tilgang til systemet uanset hvor han befinder sig og selv at bestemme hvor webløsningen skal hostes fra.

1.4 Baggrund og Motivation

Under min praktikperiode begyndte jeg at overveje hvordan denne sidste fase af mit ingeniør-studie, eksamensprojektet, skulle forløbe. Jeg overvejede bl.a. hvilken form mit afsluttende eksamensprojekt skulle have. Det første spørgsmål, der opstod, var om mit projekt skulle være fortrinsvis teoretisk eller mere praktisk orienteret. Jeg besluttede mig for den mere praktiske tilgang, da jeg gerne ville lave et program, der kunne bruges. Og da de fleste programmeringssprog i dag, er meget hurtige at udvikle i, i forhold til tidligere sprog, er det muligt at udvikle meget funktionalitet, i den begrænsede periode projektet løber over. Derefter skulle jeg beslutte mig for hvilken programmeringsplatform jeg ville benytte mig af. I praktikperioden programmerede jeg for første gang i ASP.Net, hvilket gjorde, at interessen for at lave projekt inden for .Net platformen opstod. Jeg syntes det kunne være spændende, at udvikle et system, der kunne udnytte de muligheder for brugerinput, på websider .Net-platformen tilbyder.

Baggrunden for at lave et CMS er opstået, ved at jeg gentagne gange er blevet bedt om at opdatere folks webløsninger. Stort set alle firmaer har efterhånden en webside og mange mindre virksomheder (enkeltmands virksomheder) vælger en standard-løsning, der er lavet af "en ven" for et par tusinde kr. Disse sider er , for størstedelens vedkommene, lavet i frontpage. Det er statiske løsninger udfærdiget af genbrug fra tidligere sådanne opgaver. Hver gang siden skal opdateres skal vennen kontaktes, og siden genopbygges. Og da denne ven allerede har fået sin løn bliver han/hun længere og længere tid om at få tingene gjort. Dette ender med et hav af mere eller mindre forældede websider. Større virksomheder går som regel "i byen" for at få lavet deres sider. Disse sider koster i nærheden af 30.000 kr og opefter, men til gengæld følger der et CMS med. Disse CMS'er er der efterhånden næsten ligeså mange af, som der er www-baserede reklame-virksomheder. Problemet med mange CMS'er er at prisen er høj. Derudover kræves der ofte en del baggrundsviden for at benytte disse, og "almindelige mennesker" kan ikke koble dem på deres eksisterende webløsning.

1.5 Hovedproblemstilling

På baggrund af ovenstående afsnit har jeg besluttet mig for denne grundlæggende opgaveformulering.

> **Opgaveformulering:**
> *Opbygning af et programmel, der gør det muligt for en amindelig bruger, uden kendskab til HTML eller programmering, at oprette en webløsning og redigere menustruktur og indhold på denne, uden andet værktøj end brugerens browser.*

Dette vil jeg se som mit ovenstående mål, hvorfra jeg vil udvikle systemet.

Herunder vil jeg liste de problematikker, som systemet skal udarbejdes udfra. Jeg har valgt at dele listen op i 3 dele.

Design: *Ved design, menes udseendet af webløsningen.* Her skal det være muligt at bestemme størrelser, farver og billeder på rammen af websiden. Derudover skal der være mulighed for at bestemme udseendet af menustrukturen og hvordan indholdet af løsningen skal presenteres.

Indhold: *Ved indhold, menes sider/menu'er, tekst og billeder tilknyttet webløsningen.* Der skal være mulighed for at indsætte, slette, flytte og redigere disse parametre.

Administration: *Ved Administration, menes brugertilgang til webløsningen.* Her skal det være muligt at oprette brugere. Disse skal bestå af brugernavn, password samt roller, der derved bestemmer og begrænser adgangen til systemet.

Alle disse muligheder, skal som sagt være tilgængelige ved hjælp af et brugerinterface, der benyttes direkte fra browseren. Dette skal gøres ved et intuitivt, simpelt og overskueligt system, som brugeren kan logge sig på.

1.6 Udviklingsmetode

Ved udviklingen af dette projekt, har jeg valgt at tage udgangspunkt i to udviklingsmetoder.

Unified Proces (UP) er en metode, jeg bruger til at administrere tidsforbrug og hvilke problematikker og fokus jeg har gennemløbet igennem hele forløbet. Det er en metode, der har givet mig et godt overblik gennem alle faserne i

projektforløbet, og derfor været en hjælp til rapportskrivningen. UP er et gennemarbejdet metodesæt, som næsten alle der har lavet softwareprojekter, har været i kontakt med, og jeg ser det som den mest basale projektadministration, hvorfra de fleste udviklingsmetoder stammer. UP bruger en faseopdeling, som jeg i store træk har benyttet mig af. Grunden til at jeg ikke har fulgt den slavisk, er at jeg er har været ene om projektet, hvilket har givet en større frihed arbejdsmæssigt.

Object Oriented Analysis and Design (OOAD) er den anden metode jeg har benyttet. OOAD har jeg brugt ved udviklingen af programmet. Da jeg har valgt .Net, som udviklingssprog, er det naturligt at bruge OOAD, da .Net lægger meget op til objektorienteret programmering. OOAD har desuden givet et godt overblik over funktionaliteterne, hvilket har gjort udviklingen klarere og mere stringent, samt testforløbet overskueligt. Det har også gjort det let at have fokus på Usability[2] og derved mindske kodemængden.

1.7 Kapitelgennemgang

I kapitel 2 vil jeg vise analyse-fasen af mit projekt. Heri vil blive beskrevet de teknologier jeg har benyttet, Derefter vises min kravspecifikation, hvorefter jeg definerer UseCases, konseptuelle klasser samt en domænemodel.

I Kapitel 3 viser jeg min design-fase, hvor jeg videredefinere klasser fra analysekapitlet. Her bliver den grundlæggende arkitektur beskrevet.

I Kapitel 4 præsenteres implementeringsfasen, hvor jeg beskriver mere detaljeret klasseinteraktioner. Her bliver de færdigt udviklede klasser og databasemodeller forklaret.

Kapitel 5 omhandler testfasen.

Til sidst viser jeg min endelige konklusion i Kapitel 6.

[2] Mulighed for genbrug af kode

Kapitel 2

Analyse

I dette kapitel vil jeg først vise hvordan jeg har tilrettelagt mit projektforløb. Herefter vil jeg beskrive de teknologier jeg har brugt ved udviklingen, samt grundlaget for mine valg. Så definerer jeg aktører og en grundig kravspecifikation, hvorfra jeg opbygger UseCases. Til sidst præsenteres konseptuelle klasser og deres interaktion i en domænemodel.

2.1 Planlægning

I dette afsnit vil jeg vise hvordan jeg, ved hjælp af UP, har planlagt mit projektforløb.

Uge nr.	Beskrivelse
1	**Inceptionsfase** Herunder markedsanalyse, teknologianalyse samt grundlæggende kravspecifikation
2	**1. Itteration** Analysefase Herunder markedsanalyse teknologianalyse samt kravspecifikation gjort færdig
3	**1. Itteration** Analysefase Herunder Aktører og Usecases
4	**1. Itteration** Analysefase Herunder Aktører og Usecases færdig samt domænedefinition
5	**1. Itteration** Designfase påbegyndes Analysefase færdiggøres.
6	**2. Itteration** Designfase fortsættes Implementation påbegyndes
7	**2. Itteration** Designfase fortsættes Implementation fortsættes Test af det implementerede påbegyndes
8	**2. Itteration** Designfase fortsættes Implementation fortsættes Test af det implementerede fortsættes
9	**2. Itteration** Designfase afsluttes Implementation afsluttes Test af det implementerede fortsættes
10	**2. Itteration** Test af det implementerede afsluttes Færdiggørelse af rapport

2.2 Teknologianalyse

Jeg har, som beskrevet i problemformuleringen, valgt at programmet skal give mulighed for at kunne redigere hjemmeside kun ved brug af browseren. Dette

betyder at jeg kun har 3 muligheder hvad udviklingsplatform angår: JSP, PHP eller ASP. Herfra kan PHP hurtigt afvises, da et system med den ønskede funktionalitet, ville tage for lang tid at udvikle (om muligt), med det efterhånden forældede sprog. Ved valget mellem JSP og ASP, har jeg valgt ASP. Dette er gjort af flere forskellige årsager. Blandt dem kan nævnes, at jobmarkedet er større inden for ASP-udviklere, hvilket gør det relevant at have kompetencer inden for dette. Derudover har jeg arbejdet i ASP under min praktik-periode, hvilket betyder at jeg har kunnet opbygge flere funktionaliteter i systemet, da jeg havde et kendskab til det i forvejen. Herefter skulle det besluttes hvilken version af ASP, jeg skulle benytte. Jeg valgte den gamle ASP fra, så valget stod mellem ASP.Net version 1.1 eller version 2. Jeg har valgt at bruge den ældre version 1.1, da jeg som beskrevet i afsnittet ovenfor er interesseret i "Uafhængighed af udvikler". Dette betyder at kunden selv skal finde et sted at hoste sin side. Og da version 2 stadig er meget ny, findes der ingen webhoteller i den billigere ende der understøtter den.

Desuden har jeg valgt at benytte C# og ikke VB som baggrundskode. Dette skyldes bl.a. at dette er hovedsproget i .Net. Desuden er kompetencer inden for C# også bedre rent job-mæssigt.

Ved valg af database, besluttede jeg, til at starte med, at benytte MS Access. Dette skyldes bl.a at jeg ville have en simpel database, så jeg kunne fokusere på funktionalitet frem for hastighed og optimering. Desuden er Access-databasen også simpel at sikkerhedskopiere, da den består af én fil. Dette valg har jeg været nødt til at revurdere, da de fleste webhoteller tager udgangspunkt i Microsofts anvisninger omkring sikkerhedsinstillinger på serverne. Disse, af Microsoft anbefalede, sikkerhedsinstillinger betyder at den eneste database man kan benytte er Microsofts egen MS SQL. Derfor har jeg været nødsaget til at udvikle en udgave til denne database også. Ud over de ovenfor nævnte værktøjer, har jeg benyttet en del JavaScript, da der ikke findes noget tilsvarende til client-side-scripts i .Net pakken. Jeg har desuden benyttet mig af CSS (Cascading Style Sheets) da denne metode er defacto inden for web-opbygning. I de følgende afsnit vil jeg give en kort beskrivelse af de ovennævnte teknologier samt de værktøjer og metoder jeg har brugt til at udvikle projektet og til at skrive denne rapport.

2.2.0.1 ASP.Net 1.1

ASP.NET er en videreudvikling af ASP[1] . ASP var Microsofts svar på danskeren Rasmus Lerdorf's populære PHP[2] fra 1995. Grundlaget for disse programmer var at skabe dynamik på de statiske HTML sider. Dette gøres ved at benytte

[1] Active Server Page fra 1996
[2] Først Personal Home Page, 1997 ændret til PHP: Hypertext Preprocessor

en Server til at generere HTML sider ud fra klientens input.

ASP.NET Web applikation eksekveres gennem en række forespøgelser og svar via HTTP-protokollen mellem klientens browser og web-serveren. Dette sker på følgende måde:

1. Klienten forespørger resourcer fra serveren ved at skrive en URL i browseren. Browseren sender forespørgelsen til web-serveren.
2. Web-serveren analyserer forespørgelsen og søger efter en process til eksekvering af forespørgelsen.
3. Resultatet sendes tilbage til klientens browser.
4. Browseren læser svaret og viser det som en webside til klienten.

Numrene henviser til figur:2.1

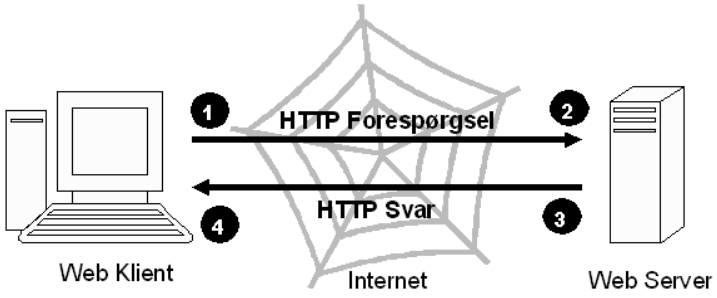

Figure 2.1: ASP.Net eksekvering 1

Web-serveren benytter Microsofts Internet Information Services (IIS) ved forespørgelser vedrørende ASP.NET sider. For at kunne benytte dette projekt er det derfor nødvendigt at serveren har IIS kørende og .NET framework installeret. Serverens eksekvering (ovenstående pkt. 2) fungerer således:

1. IIS modtager, som det er vist ovenfor, en HTTP-forespørgsel. Når denne forespørgsel er efter en ASP.NET side (.aspx) sendes den videre til aspnet_isapi.dll.
2. ISAPI'en[3] sender forespørgelsen videre til ASP.NET worker process (aspnet_wp.exe).

[3]Internet Server Application Programming Interface (ISAPI)

3. ASP.NET worker process kompilerer den forespurgte aspx fil til en assembly, opretter applikationsdomæne og sætter CLR[4] til at eksekvere assembly'en i domænet.

4. Når denne assembly indeholder kode tilhørende en ASP.NET side benytter den sig af klasser i FCL[5] for at generere det korrekte svar til klientforespørgelsen.

5. ASP.NET worker process modtager svaret og sender det tilbage som en response packet til ISAPI'en.

6. ISAPI sender svaret videre til ISS'en der sender pakken videre til klienten.

Numrene henviser til figur:2.2

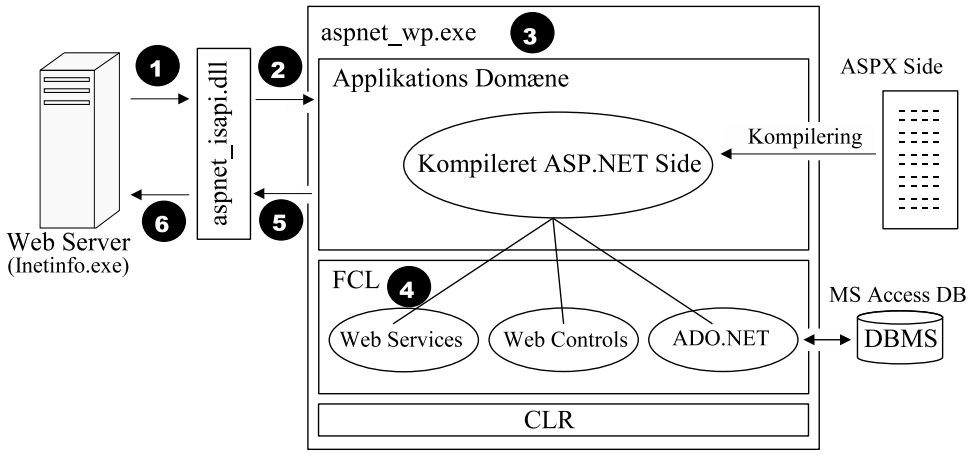

Figure 2.2: ASP.Net eksekvering 2

En af de største fordele ved at benytte ASP.NET er muligheden for at skrive funktionalitetskoden i en bagvedliggende fil. Denne metode kaldes "Code-behind".

2.2.0.2 C#.Net

C# er et programmeringssprog udviklet af danskeren Anders Hejlsberg. Dette sprog er det der ligger tættest op af .NET framework. Det er udviklet for at samle fordele fra andre sprog, som Delphi, Java og C++, og har mange ligheder

[4]The Common Language Runtime (CLR)
[5]The Framework Class Library (FCL)

med disse. Det er desuden dette sprog der er grundlaget for resten af .Net-pakken. Dette sprog benytter jeg som "Code-behind" til ASPX siderne.

2.2.0.3 JavaScript

JavaScript er et Scripting Programming Language udviklet af Netscape. Det blev implementeret for første gang i Netscape version 2 browseren sidst i 1995. JavaScript er mest brugt som et client-side script og det er også sådan jeg har brugt det. Fordelen ved at bruge dette, er at man undgår en masse server-side forespøgelser, da disse scripts sendes til klienten, og køres der.

2.2.0.4 CSS

Cascading Style Sheets (CSS) bliver administreret af W3C[6] og bruges til at definere udseendet som f.eks. farver og fonte på en webside. Fordelen ved at bruge CSS er, at man kun behøver at ændre i css.-filen i stedet for at sætte f.eks. fonttyper på alle tekster i webløsningen.

2.2.0.5 MS Access

MS Access er et relational database management system fra Microsoft. Det er nok den simpleste database på markedet, da den minder mere om et regneark end en reel database. Dette program er brugt til at oprette de to databaser jeg bruger i projektet.

2.2.0.6 MS SQL

MS SQL Server har jeg brugt af nød, ikke af lyst. Dette skyldes ikke funktionaliteten eller hastigheden, da de langt overgår Access. Det skyldes derimod at Access er meget lettere at tage backup af, da man bare henter en fil. Som skrevet ovenfor er dette den eneste database, der kan benyttes hvis man benytter Microsofts anvisninger omkring sikkerhedsinstillinger på severen. Jeg har valgt 2000 versionen, da denne er gennemtestet og mere stabil end den nyeste version.

[6] World Wide Web Consortium (W3C)

2.2.0.7 Projektværktøjer

Af projektværktøjer har jeg brugt MS Visio til modulering af UML og reverse-ingeniering. Til denne rapport har jeg brugt LaTeXversionen fra MikTek, med LaTeXEditor (LEd) som brugergrænseflade. Jeg har desuden benyttet StarOffice 7.0, bl.a. til billedbehandling.

2.2.1 Markedsanalyse

Umiddelbart burde markedet være mættet med hensyn til CMS'er. Der findes som sagt et utal af forskellige udgaver, hvoraf nogle endda er gratis. På trods af det, ser jeg stadig "et hul" i markedet mellem de 1-2000 kr's statiske løsninger, de større virksomheders løsninger, som beskrevet ovenfor, og de gratis udgaver. Jeg mener at der er et marked for privatpersoner, mindre virksomheder og foreninger. Disse målgrupper har ikke økonomi til de dyre løsninger. De har ikke ekspertise/tid/lyst til at sætte sig ind i brugen af de gratis CMS-systemer. De har lyst til selv at bestemme hvordan deres side skal se ud. De har selv mulighed for at lave billeder og logo'er eller behøver ingen. De vil have mulighed for selv at redigere indholdet på siden. Løsningen som jeg laver, retter sig mod disse målgrupper.

I figur:2.3 har jeg listet de vigtigste forskelle mellem de forskellige typer systemer.

2.3 Aktører

Herunder defineres aktørerne i projektet. Aktørernes rettigheder arves opad, hvilket betyder at øverst listet aktør har rettigheder som alle øvrige aktører. Næstøverste aktør har ikke rettigheder som øverste aktør, men har alle nedenstående aktørers rettigheder osv.

2.3.1 Administrator

Administratoren er personen der administrerer tilgang til brugere af systemet, herunder brugernavn og password, samt rollefordeling.

	Professionelle Systemer	*Gratis Systemer*	*1-2000 kr's systemer*	*Mit system*
Økonomisk aspekt				
Anskaffelsespris	Kr 20-30.000 +	kr 0,00	kr 1-2000	2-10.000 kr
Løbenden udgifter/ Driftomkostningner	100-1000 kr/md	5-100 kr/md	5-100 kr/md	5-100 kr/md
Drift aspekt				
Udviklingstid	Ca. 1 md	Brugerstyret, 1 dag	1-2 uger afh. af udvikler	Brugerstyret, 1 dag
Opdateringstid	Brugerstyret,1 dag	Brugerstyret, 1 dag	1 uge-1 md+ afh. af udvikler	Brugerstyret, 1 dag
Afhængighed af udvikler	Design og ofte hosting	Intet	Fuld afhængighed af udvikler	Intet
Design aspekt				
Design	Professionelt design, ofte merpris	Tema'er, eller html-programmering	Afhængig af udvikler.	Brugerstyret, modulopbygget
Kompleksitet	Komplekst system. Undervisning medfølger	Komplekst system. Hjælp fra nettet.	Ingen kompleksitet for bruger.	Simpelt. Intuitivt. Mulighed for kompleksitet
Dynamik	CMS indbygget	CMS indbygget, samt tema'er til udseende	Statisk, kræver ombygning af systemet	CMS indbygget, samt dynamisk designopbygning.
Specielle Funktioner	Mange muligheder, ofte merpris	Mange muligheder, ofte programmeret af private.	Afhængig af udvikler. Ofte ingen.	Ingen (forberedt til flere sprog, samt mailservice)

Figure 2.3: Forskelle mellem gængse systemer og mit

2.3.2 Designer

Designeren har mulighed for at bestemme designet/udseendet af webløsningen.

2.3.3 Editor

Editoren har mulighed for at indsætte sider og moduler[7] til webløsningen.

2.3.4 Writer

Writeren har mulighed for at oprette, skrive og rette i moduler der ikke er tilknyttet sider.

[7] Se bilag F

2.3.5 Client

Kunden har kun mulighed for at se den færdige publicede webløsning.

2.4 Kravspecifikation

Ved udarbejdelsen af kravspecifikationen, har jeg taget udgangspunkt i hovedproblemstillingen, men også i markedsanalysen. Jeg har valgt at lave en opdeling af krav, for dermed at vise hvilke krav, jeg har har ment var de vigtigste, og dermed fokuseret mest på. Under udviklingen af kravspecifikationen, har jeg lagt meget vægt på at systemets brugerinterface skal være intuitivt. Det vigtigste krav, og grundlaget for projektet, er at systemet skal kunne betjenes af personer uden kendskab til programmering. Derfor har jeg, igennem hele projektforløbet, haft kontakt med formanden for teaterforeningen "TeaterKatalysator" Nina Leman Bundgaard. Dette samarbejde har mest vedrørt designet[8], men har også haft indvirkning på kravspecifikationen.

2.4.1 Primære krav

De primære krav, er de vigtigste krav. Disse krav er også grundpillerne i de gængse CMS'er.

- **Redigering af tekst:** Dette punkt er efter min vurdering det vigtigste punkt i en webløsning. Det er også dette punkt, der er grunlaget for alle CMS'er. Det skal være muligt via et interface at ændre i tekster på websiden.
- **Indsætning/Sletning af tekst:** Dette kan ses som et underpunkt til redigeringen. Men da webløsningen som udgangspunkt er tom, bliver dette punkt mere specifikt og får derfor sin berettigelse. Dette punkt omhandler både tekst og billede. Jeg har valgt at betegne en samling af tekst og billeder som "et modul".
- **Redigering af sider** Redigering af sider omhandler indsættelse, sletning, ændring af navn og flytning af sider. Punktet retter sig mod menustrukturen på webløsningen.

[8] Læs mere om samarbejdet med "TeaterKatalysater" under Design

2.4.2 Sekundære krav

De sekundære krav, omhandler design og administration af webløsningen.

- **Administration:** Under administration, skal det være muligt at tilføje/ slette brugere på systemt. Desse brugere skal tilknyttes en rollemodel, svarende til aktørerne.
- **Design af rammen:** Ved "rammen", mener jeg det statiske på webløsningen, der ikke ændrer sig efter hvilken indholdsside der vises. Det skal være muligt at indsætte top/bund/side-billeder samt rammer, farver og størrelse. Desuden skal der være mulighed for at indsætte en tekst i top og bund.
- **Design af menu:** Navigationsmenuen skal kunne vises på forskellige måder. Jeg har valgt at begrænse det til 3 forskellige typer: Dropdown vandret, Dropdown lodret og Knapper vandret med undermenuer i venstre side. Menuerne skal kunne ændres i størrelse, farve og rammer.
- **Design af indhold:** Der skal kunne bestemmes standard teksttyper og farver til siden. Derudover skal der kunne laves opdelinger af siden. Jeg har valgt at begrænse opdelingen til 3 typer: 1 kolonne, 2 kolonner og 3 kolonner.

Begrænsningerne i designet har jeg lavet primært for at gøre systemet så let bruge som muligt. Desuden mener jeg at jeg har taget de vigtigste funktioner med.

2.4.3 Tertiære krav

De tertiære krav har jeg listet, for at gøre det muligt at opfylde disse krav senere hen. Det er krav, jeg ikke har set vigtige nok til fuldt at implementere i dette projektforløb, da de ville betyde en overskridelse af tidsplanen. Jeg har dog lavet forberedelser, så de kan implementeres fuldt ud senere.

- **Sprog:** Der skal være mulighed for at systemet understøtter flere sprog.
- **Specielle funktioner:** Ved specielle funktioner, kan f.eks. nævnes Galleri, Blog, Forum og E-handel. Den eneste specielle funktion, jeg vil understøtte, indenfor projektforløbet, er en mailklient til brug ved tilmelding af f.eks. nyhedsbreve.

2.5 UseCases

2.5.1 UC1: Login

Primær aktør	Bruger
Interesser	• Bruger: Mulighed for at benytte funktioner i systemet • Administrator: Sikkerhed for at der kun tildeles adgang til det tiladte i systemet i forhold til brugerens tildelte rolle.
Prebetingelser	Ingen
Postbetingelser	Brugeren er logget på systemet
Succes Scenarie	1. Brugeren indtaster sit domænenavn + "admin.aspx" i sin browser, for at komme til loginsiden. 2. Brugeren indtaster sit brugernavn og adgangskode i de dertil beregnede felter. 3. Brugeren trykker "Login" for at logge sig på systemet. 4. Brugeren Logges ind på systemet, hvor hans muligheder bliver presenteret.
Alternativ Scenarie	4a Brugeren genkendes ikke af systemet pga. forkert brugernavn eller adgangskode og lukkes derfor ikke ind.

2.5.2 UC2: Logout

Primær aktør	Bruger
Interesser	• Bruger: Sikkerhed for at brugeren er logget af systemet.
Prebetingelser	Brugeren er logget på systemet
Postbetingelser	Brugeren er logget af systemet
Succes Scenarie	1. Brugeren går til login-siden. 2. Brugeren trykker "Logout" for at logge sig ud af systemet. 3. Brugeren logges af systemet.
Alternativ Scenarie	1a Brugeren lukker sin browser, hvorved han logges ud af systemet.

2.5.3 UC3: Opret bruger

Primær aktør	Administrator
Interesser	• Administrator: Mulighed for at tilføje en bruger til systemet.
Prebetingelser	Administratoren er logget på systemet
Postbetingelser	En bruger er tilføjet systemet
Succes Scenarie	1. Administratoren går til Administrationen. 2. Administratoren trykker tilføj bruger. 3. Administratoren indtaster brugernavn og adgangskode for den nye bruger. 4. Administratoren vælger hvilken rolle den nye bruger skal tildeles. 5. Administratoren kan se den nye bruger i listen over brugere.
Alternativ Scenarie	

2.5.4 UC4: Rediger/Slet bruger

Primær aktør	Administrator
Interesser	• Administrator: Mulighed for at rette/slette en bruger til systemet.
Prebetingelser	Administratoren er logget på systemet
Postbetingelser	En bruger er rettet/slettet i systemet
Succes Scenarie	1. Administratoren går til Administrationen. 2. Administratoren vælger brugeren i listen over brugere. 3. Administratoren retter brugernavn og adgangskode for den nye bruger. 4. Administratoren retter hvilken rolle den nye bruger skal tildeles. 5. Administratoren kan se den nye bruger i listen over brugere.
Alternativ Scenarie	3-5a Administratoren vælger "slet bruger", for at fjerne brugeren fra systemet, hvorefter denne bruger ikke kan ses på listen over brugere.

2.5.5 UC5: Design ramme

Primær aktør	Designer
Interesser	• Designer: Mulighed for at oprette og rette de ydre rammer på sin webløsning.
Prebetingelser	Designeren er logget på systemet
Postbetingelser	De ønskede ændringer er gemt i systemet
Succes Scenarie	1. Designeren går til menupunktet "Set Frames". 2. Designeren bestemmer sidebredde og yderste ramme på webløsningnen. 3. Designeren bestemmer top- og bundbillede samt disses rammer på webløsningen 4. Desigeneren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	4a Designeren vælger at se ændringer uden at gemme. Dette præsenteres i et nyt browservindue. Herefter submittes ændringerne, der gemmes i systemet, eller der vælges ikke at submittes og ændringerne gemmes ikke.

2.5.6 UC6: Design menu

Primær aktør	Designer
Interesser	• Designer: Mulighed for at indstille præsentationen af menustrukturen på webløsningen.
Prebetingelser	Designeren er logget på systemet
Postbetingelser	De ønskede ændringer er gemt i systemet
Succes Scenarie	1. Designeren går til menupunktet "Set PageStyle". 2. Designeren bestemmer hvilken menutype der ønskes på webløsningen. 3. Designeren bestemmer standardbillede, i henhold til valget af menutypen, samt disses rammer og størrelser, på webløsningen. 4. Desigeneren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	4a Som UC5

2.5.7 UC7: Design menubuttons

Primær aktør	Designer
Interesser	• Designer: Mulighed for at indstille knapperne på menustrukturen i webløsningen.
Prebetingelser	Designeren er logget på systemet
Postbetingelser	De ønskede ændringer er gemt i systemet
Succes Scenarie	1. Designeren går til menupunktet "Set MenuButtons". 2. Designeren bestemmer knapperne, i henhold til valget af menutypen, herunder disses rammer, størrelser og farver. 3. Desigeneren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	3a Som UC5

2.5.8 UC8: Design standards

Primær aktør	Designer
Interesser	• Designer: Mulighed for at bestemme standard-instillingeren til indholdet på webløsningen.
Prebetingelser	Designeren er logget på systemet
Postbetingelser	De ønskede ændringer er gemt i systemet
Succes Scenarie	1. Designeren går til menupunktet "Set Standards". 2. Designeren bestemmer kolonnebredde samt rammer, farver og baggrundsbilleder tilknyttet kolonnen. 3. Designeren bestemmer standard teksttyper. 4. Desigeneren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	

2.5.9 UC9: Editor NewPage

Primær aktør	Editor
Interesser	• Editor: Mulighed for at oprette en ny side/nyt menupunkt til webløsningen
Prebetingelser	Editoren er logget på systemet
Postbetingelser	Siden er oprettet i systemet
Succes Scenarie	1. Editoren går til menupunktet "Set Menus". 2. Editoren opretter en ny side. 3. Editoren bestemmer navn, placering og type til siden. 4. Editoren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	

2.5.10 UC10: Editor DeletePage

Primær aktør	Editor
Interesser	• Editor: Mulighed for at slette en side i webløsningen
Prebetingelser	Editoren er logget på systemet Siden er oprettet i systemet
Postbetingelser	Siden er slettet fra systemet
Succes Scenarie	1. Editoren går til menupunktet "Set Menus". 2. Editoren vælger en side. 3. Editoren vælger at slette siden. 4. Editoren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	

2.5.11 UC11: Editor MovePage

Primær aktør	Editor
Interesser	• Editor: Mulighed for at flytte en side i webløsningen
Prebetingelser	Editoren er logget på systemet Siden er oprettet i systemet
Postbetingelser	Siden er korrekt flyttet i systemet
Succes Scenarie	1. Editoren går til menupunktet "Set Menus". 2. Editoren vælger en side. 3. Editoren vælger en ny placering til siden. 4. Editoren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	

2.5.12 UC12: Editor InsertModule

Primær aktør	Editor
Interesser	• Editor: Mulighed for at indsætte et modul på en side i webløsningen
Prebetingelser	Editoren er logget på systemet Siden er oprettet i systemet Modulet er oprettet i systemet Modulet er ikke knyttet til en side i systemet
Postbetingelser	Modulet er indsat på siden i systemet
Succes Scenarie	1. Editoren går til menupunktet "Set Menus". 2. Editoren vælger en side. 3. Editoren vælger et frit modul. 4. Editoren indsætter modulet til siden. 5. Editoren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	

2.5.13 UC13: Editor FreeModule

Primær aktør	Editor
Interesser	• Editor: Mulighed for at fjerne et modul fra en side i webløsningen
Prebetingelser	Editoren er logget på systemet Siden er oprettet i systemet Modulet er oprettet i systemet Modulet er knyttet til en side i systemet
Postbetingelser	Modulet er fjernet fra siden i systemet Modulet er ikke slettet fra systemet
Succes Scenarie	1. Editoren går til menupunktet "Set Menus". 2. Editoren vælger en side. 3. Editoren vælger et modul tilknyttet siden. 4. Editoren frigør modulet fra siden. 5. Editoren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	

2.5.14 UC14: Editor MoveModule

Primær aktør	Editor
Interesser	• Editor: Mulighed for at flytte et modul på en side i webløsningen.
Prebetingelser	Editoren er logget på systemet Siden er oprettet i systemet Modulet er oprettet i systemet Modulet er knyttet til en side i systemet Der findes mere end et modul tilknyttet siden
Postbetingelser	Modulet er flyttet på siden i systemet
Succes Scenarie	1. Editoren går til menupunktet "Set Menus". 2. Editoren vælger en side. 3. Editoren vælger et modul tilknyttet siden. 4. Editoren flytter modulet op eller ned på siden. 5. Editoren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	

2.5.15 UC15: Writer NewModule

Primær aktør	Writer
Interesser	• Writer: Mulighed for at oprette et modul i webløsningen.
Prebetingelser	Writeren er logget på systemet
Postbetingelser	Modulet er oprettet i systemet
Succes Scenarie	1. Writeren går til menupunktet "Set Modules". 2. Writeren vælger at oprette et nyt modul. 3. Writeren vælger at indsætte tekst eller billede i modulet. 4. Writeren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	

2.5.16 UC16: Writer DeleteModule

Primær aktør	Writer
Interesser	• Writer: Mulighed for at slette et frit modul i webløsningen.
Prebetingelser	Writeren er logget på systemet Modulet er oprettet i systemet Modulet er ikke knyttet til en side i systemet
Postbetingelser	Modulet er slettet i systemet
Succes Scenarie	1. Writeren går til menupunktet "Set Modules". 2. Writeren vælger et frit modul. 3. Writeren vælger at slette modulet. 4. Writeren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	

2.5.17 UC17: Editor/Writer EditModule

Primær aktør	Writer/Editor
Interesser	• Writer: Mulighed for at ændre i et frit modul i webløsningen. • Editor:Mulighed for at ændre ethvert modul i webløsningen.
Prebetingelser	Writeren/Editoren er logget på systemet Modulet er oprettet i systemet
Postbetingelser	Modulet er ændret og gemt i systemet
Succes Scenarie	1. Writeren/Editoren går til menupunktet "Set Modules". 2. Writeren vælger et frit modul. Editoren vælger et modul. 3. Writeren/Editoren vælger at ændre, slette eller tilføje tekst/billede i modulet. 4. Writeren/Editoren submitter ændringerne, der gemmes i systemet.
Alternativ Scenarie	

2.6 Domæne

Da dette projekt udelukkende består i opbygning af en webløsning, er en klassisk domænemodel faktisk ikke mulig. Det skyldes at alle konceptuelle klasser er abstrakte. Dette system, er ikke en løsning til en fysisk problemstilling, hvor det er vigtigt at definere fysiske objekter, men et værktøj til at præsentere abstrakte ting i et vituelt miljø (internettet). Derfor har jeg valgt at bruge to metoder til at definere domænet på. Den ene er, at definere de konseptuelle klasser, som en slags "kasser" eller byggeblokke, der kan samles i større beholdere og endelig til en færdig "figur". Den anden metode, er at se klasserne som dele af et dokument, der kan samles til et helt dokument.

2.6.1 Konceptuelle klasser

Beholdere	Modul/Artikel Menu/Side Menuknap Felt Ramme
I beholdere	Tekst Billede
Definitionsobjekter	Størrelse TextType Farve MenuEffekt Sprog
Roller	Administrator Designer Editor Writer Client
Strukturelle Opdelinger	Forside, ramme Forside, opsætning & design Menudesign Standardinstillinger Menu/side oprettelse Modul/artikel oprettelse Administration Webløsningen

2.6.2 Domænemodel

I figur 2.4 viser jeg relationerne mellem de konceptuelle klasser fra ovenstående afsnit i en domænemodel. Disse klasser og referencer bliver brugt, direkte designmæssigt, som sådanne klasser. Men også ved design af administrationsinterfacet, vil jeg benytte disse opdelinger, til at skabe kontroller til håndtering af brugerinput til klasserne.

Jeg har desuden vist en opdeling mellem "Design" og "Content", der vil blive gældende igennem design og implementering. Dette har jeg valgt af flere grunde. En grund er at en opdeling i to dele vil gøre opbygningen og test af programmet lettere og mere overskueligt. En anden grund er, at der vil blive ændret jævnligt i "Content"-delen, mens "Design"-delen vil være forhåldsvis statisk når den først er opbygget.

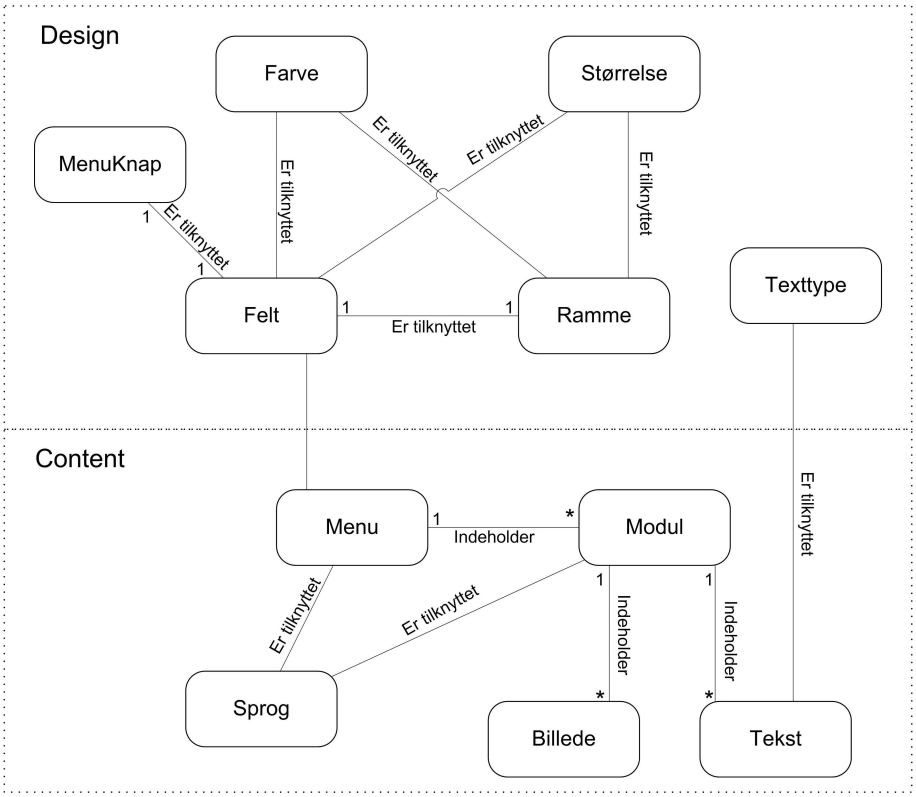

Figure 2.4: Domænemodel

2.7 Delkonklusion (Analyse)

I dette kapitel har jeg oparbejdet en analysemodel til brug ved design af mit system. Jeg har lavet en konkret kravspecifikation og defineret aktører i systemet, hvorfra jeg har udviklet en gruppe UseCases. Ud fra disse, har jeg defineret konceptuelle klasser og oprettet en domænemodel.

Herved har jeg fået et specifikt udgangspunkt til næste kapitel, hvor designet af systemet bliver udviklet. Især UseCases og domænemodelen er grundpillerne i designet. De er også de eneste dele af dette kapitel, der er blevet ændret i, under design og implementering. Ændringerne har ikke været funktionsmæssige, men jeg har opdateret navne, for at give en tættere tilknytning mellem analysen og designet.

Jeg har desuden lavet forskellige opdelinger i analysefasen, der vil være gennemgående og mere markant i design- og implementeringsfasen. Den første er opdelingen i de to begreber "System" og "Webløsning"[9]. Den anden opdeling er mellem "Design" og "Content", hvor Content ligger som den primære kravspecifikation, og Design, som den sekundære.

[9] Se ordbogen i bilag

Kapitel 3

Design

I dette kapitel, vil jeg først vise, hvordan jeg har valgt at opbygge programmets overordnede arkitektur. Herefter vil jeg gå i dybden med de forskellige delelementer samt interfaces, databaseopbygning og -tilgang.

3.1 Arkitektur

Arkitekturen på figur 3.1 skal ses som et oversigtskort over de følgende afsnit, hvor de forskellige dele vil blive behandlet.

3.2 Interfaces

Som beskrevet i Analysekapitlet, har jeg lavet en opdeling i system og webløsning. Dette betyder at jeg har designet to forskelige interfaces. Et interface til systemet, der skal administrere og opbygge det andet interfacet til webløsningen.

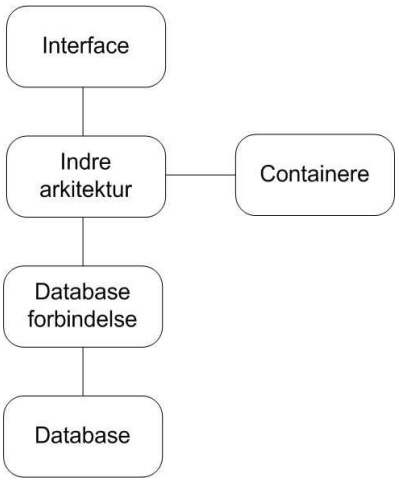

Figure 3.1: Overordnet oversigt af klassetyper

3.2.1 System-interface

Som jeg har beskrevet tidligere, er dette interface grundlaget for mit projekt. Derfor har jeg lagt en stor energi i at skabe et så intuitivt design som muligt. Under udviklingen af dette design, har jeg haft tæt kontakt til Nina Bundgaard, der har givet en uvurderlig feedback. Dette samarbejde, har bl.a. medført en opdeling[1] af interfacet i en slags dobbelte faneblade, hvor de strukturelle opdelinger udgør hoved-fanebladene. Dette har jeg gjort, for at undgå en for stor informationsmængde, der kan forvirre brugeren. Det har været en balancegang, at begrænse informationmængden, og samtidig at opretholde overblikket. En anden grund til at jeg har valgt at bruge faneblade er, at jeg derved kunne opbygge interfacet, så det både kan benyttes, som en step-by-step løsning, men også giver mulighed for at gå ind hvor som helst i systemet og rette i en enkelt del. I figur A.1 i bilag A kan det ses hvordan systeminterfacet er opbygget rent strukturelt. Her indsættes de strukturelle opdelinger fra analysefasen, som "TopLevel-faneblade". Derunder vises i "Overmenuerne", de forskellige elementer, tilknyttet det valgte TopLevel-faneblad, der kan ændres i. I "Undermenu-knapperne" vises de muligheder der er for ændring af det valgte element. I figur 3.2 og 3.3 kan ses de interfaces, jeg benytter, samt disses interaktion og lagdeling i forhold til figur A.1. Grunden til opdeling i "Design" og "Content" her, er at jeg har afviget lidt fra ovenstående opbygning i "Content"-delen (se afsnittet om "Brugerinput-elementer" herunder).

[1] Se Analysekapitlet under konceptuelle klasser

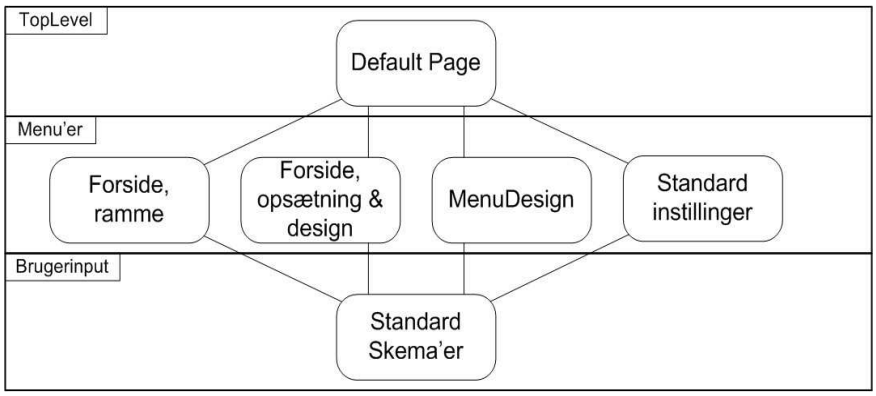

Figure 3.2: Interface-del (Design) klasser

Efter fastlæggelsen af denne lagdelte menustruktur, defineres brugerinputet (3. level Placeholder i figur A.1). Her har jeg taget udgangspunkt i UseCases fra analysedelen, da jeg der har defineret hvilke funktioner og input, der skal være tilgængelige. Disse dele beskriver jeg i nedenstående afsnit ("Brugerinput-elementer").

3.2.2 Webløsning-interface

Dette interface hænger meget sammen med systeminterfacet, da det dette design skal kunne defineres af det ovenstående. Det sværeste har været at begrænse mig, da jeg både ville have mulighed for at gøre designet af webløsningen så dynamisk som muligt, men samtidig gøre systemet let at bruge. Jeg har valgt at definere et grundlæggende "skema" som kan ses på figur A.2i bilag A. I denne figur viser jeg de "felter" som webløsningen består af[2].

Dette interface har ingen input-elementer. Den eneste interaktion der kan forekomme, er valg af side i menu-feltet. Derfor er "indhold"-feltet i figur A.2 det eneste, der skal loades dynamisk. Dette felt kan opdeles i op til tre kolonner. I disse kolonner, kan et predefineret modul-skema bruges til indsættelse af samlinger af tekst og billeder. Til dette interface benyttes kun to klasser som vist på figur 3.4

[2]Denne del af interfacet bygger på UseCases 5 og 6

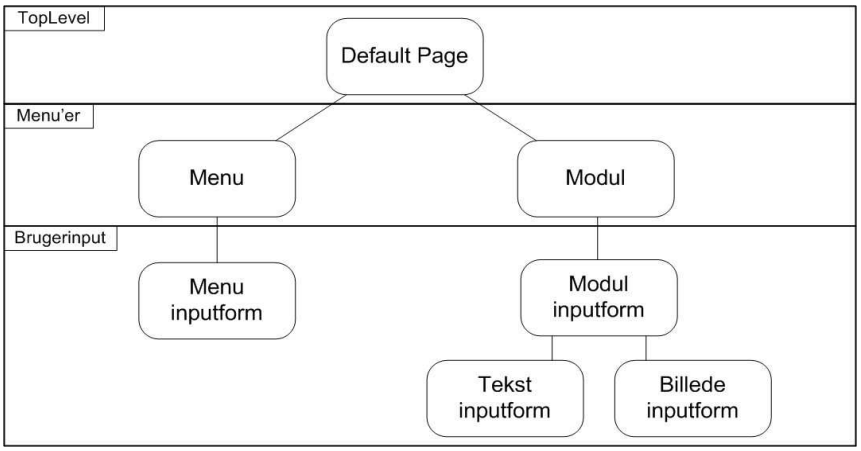

Figure 3.3: Interface-del (Content) klasser

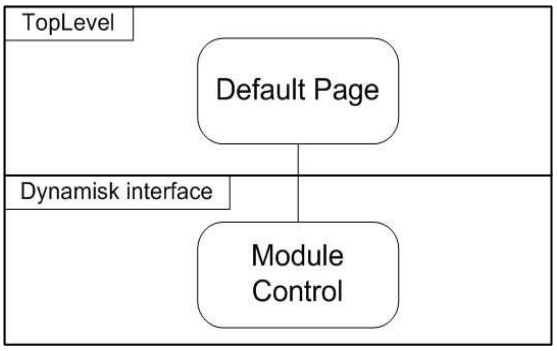

Figure 3.4: Interface-del (Webløsning) klasser

3.2.3 Brugerinput-elementer

Ved dette interfacedesign, har jeg startet med skitsetegninger, som jeg i samråd med Nina Bundgaard har gennemarbejdet og videreudviklet og endeligt givet det færdige design. Jeg har valgt ikke at tage disse skitser med i rapporten, da de som råskitser, ikke er tydelige nok. Desuden er det færdige resultat, stort set identisk med de sidste tegninger.

Under "Design"-delen af dette interface, har jeg holdt mig stringent til ovenstående menustruktur, da der her er mange sammenfald i brugerinput, såsom farvevalg og størrelser. Her er det vigtigst for brugeren, at han tydeligt kan se, hvad der ændres i, mens selve input-delen er enkel. Derfor har jeg defineret nogle

standard-elementer hertil. Disse elemeter er:

- Farveskema, hvor farver kan indsættes, enten ved valg i skemaet, eller ved at skrive farvekode.
- Rammeskema, hvor rammens tykkelse og type bestemmes.
- Størrelse, hvor en bredde og højde på feltet kan defineres.
- Billedskema, hvor et billede kan indsættes i et felt.

Under "Content"-delen har jeg valgt at lave to definerede inputskemaer, der henholdsvis administrerer sider og moduler[3]. Dette har jeg gjort, først og fremmest fordi disse to dele er de mest komplekse. Det betyder at overskueligheden og intuitiviteten er vigtigst her. Jeg har valgt at benytte en step-by-step metode, hvor mulighederne bliver vist gradvist. Dette betyder, at der ikke skabes forvirring med for meget information, når man starter med at opbygge en side eller et modul.

3.3 Indre Arkitektur & Containere

Denne del af programmet er forbindelsen mellem input/output til brugeren/kunden og databaseforbindelsen. Dette kan også betegnes som programmets backbone. Som nævnt ovenfor, er der to interfaces; et til brugeren og et til kunden. Denne opdeling beholdes også i denne del af programmet. Desuden bibeholdes opdelingen af "Design" og "Content".

3.3.1 System-del (Design)

Jeg har valgt at tage udgangspunkt i et XML-dataset, som container til denne del. Grunden til dette valg er, at der er et fast antal variable inputs. Disse input er desuden meget ens, som beskrevet under interface-afsnittet i dette kapitel. Interfacet til denne del, kræver at data går begge veje, hvorfor jeg har valgt at definere 2 typer klasser. Den ene klasse er til at opdatere datasettet, mens den anden klasse er til opdatering af interfacet. Opdateringen af interfacet, opdeles i en klasse til hvert interface-skema. En oversigt, kan ses på figur 3.5

[3] Se UseCase 9-17

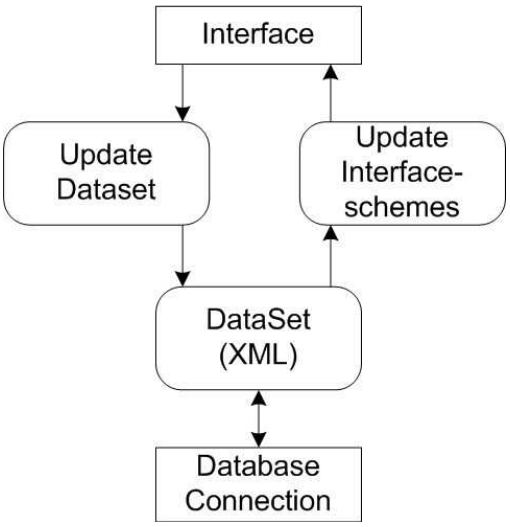

Figure 3.5: System-del (Design) klasser

3.3.2 System-del (Content)

Jeg har valgt at benytte klasser, som containere i denne del. Det skyldes at mængden af data ikke er konsistent, da det er muligt at tilføje menuer/sider og moduler/artikler til systemet. Derfor har jeg valgt at benytte en nedarvning af klasser. I figur 3.6 vises de klasser jeg benytter. Her ses de to øverste -Items-klasser der fungerer som Creator[4] for de nedenstående -Item-klasser. Som beskrevet i analysen, kan moduler indholde både tekst og billede. Derfor har jeg indsat en TextImage Collection, som Creator, der kan indeholde begge typer.

3.3.3 Webløsning-del (Design)

I stedet for at benytte en forbindelse til databasen, har jeg her valgt at oprette et stylesheet. Da disse data er statiske i forhold til kunden, er dette at foretrække, fremfor at skulle kontakte databasen. StyleSheetet er oprettet som en enkel Css-fil, der bliver dynamisk oprettet i systemet, ved at benytte samme dataset som i system-delen. Det gøres ved en enkel klasse, der skriver direkte til filen som en almindeligt tekstfil.

[4] Pattern

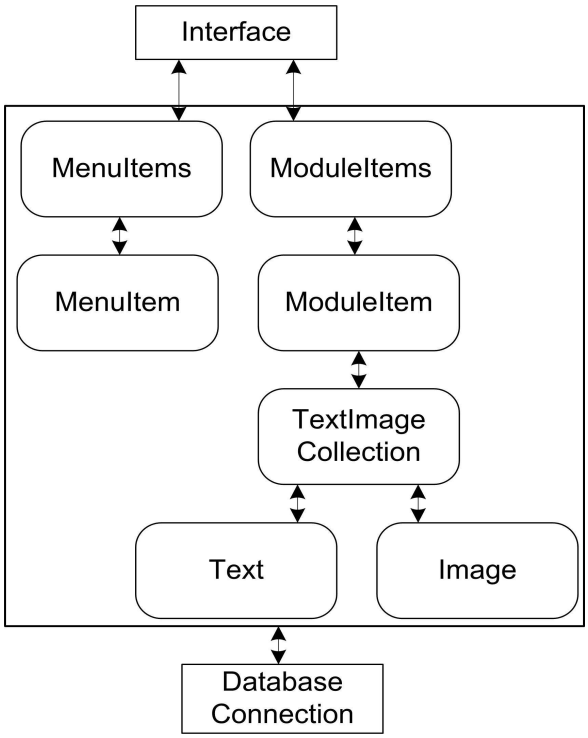

Figure 3.6: System-del (Content) klasser

3.3.4 Webløsning-del (Content)

Her benyttes de sammme klasser som under System-delen. Herved bibeholdes strukturen, så containere og forbindelsen til databasen kan genbruges.

3.4 Database Forbindelse

Databaseforbindelsen er, som navnet antyder, forbindelsen mellem den indre arkitektur og selve databasen. Dette kan ses som en Facade/footnotePattern, der betyder, at resten af programmet, ikke er fastlagt på en bestemt database. Så hvis man vælger at skifte til en anden database, er det kunne i dette lag, der skal laves ændringer.

I webløsnings-delen, skal der kun læses fra databasen, da der ikke skal gemmes data fra kunden. Som beskrevet i ovenstående afsnit, er denne del den samme som i system-delen. Derfor vil jeg tage udgangspunkt i system-delen, hvor jeg har fortsat opdelingen i "Design" og "Content".

3.4.1 DB-forbindelse (Design)

Som beskrevet i ovenstående afsnit (Indre Arkitektur), har jeg valgt at benytte et dataset som container. Dette betyder at datasettet kan opdateres direkte fra databasen. Det samme er gældende fra dataset til database. Da disse funktioner er simple, har jeg valgt at samle dem i en klasse. Denne kan både fylde datasettet fra databasen, og opdatere databasen fra datasettet, som vist i figur 3.7

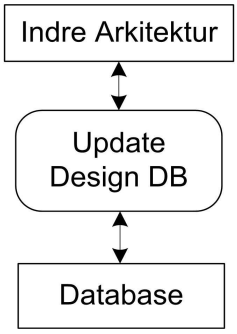

Figure 3.7: DB forbindelse (Design) klasse

3.4.2 DB-forbindelse (Content)

Ved "Content"-delen, har jeg valgt at opdele forbindelsen i to klasser. En til at læse fra databasen, og en til at skrive til den. Dette skyldes at der her er tale om en mere kompliceret forbindelse, da der ikke, som under "Design", er tale om en

fast mængde data, men derimod om både tilføjelse og sletning i databasen. Derfor har jeg også valgt at oprette to klasser ovenover selve database-forbindelsen. Disse to klasser er lavet som forberedelse til at skrive henholdsvis moduler og menu'er til databasen. Jeg har valgt at lave disse forberedelses-klasser for sig, for at simplicifere den grundlæggende database-adgang. Disse klasser bruges til at oprette referencer og forbindelser mellem menu'er, moduler og deres underliggende klasser. Se klasserne i figur 3.8.

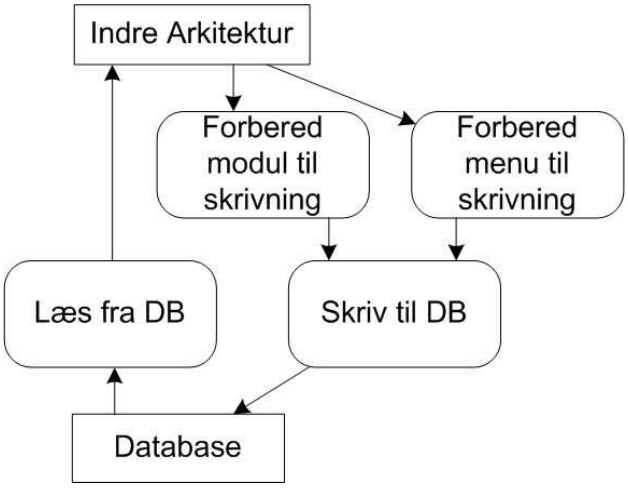

Figure 3.8: DB forbindelse (Content) klasser

3.5 Database

Databasen er, som resten af programmet, delt op i to. Dette skyldes flere ting. For det første, er der, som beskrevet i de foregående afnit, brugt et DataSet i "Design"-delen, hvorfor jeg har valgt at oprette en specifik database, der svarer til datasettet. Dette er gjort for at gøre dataforbindelsen simplere. Desuden går jeg ud fra, at denne del af programmet, vil blive opdateret meget sjældent. Derfor er det besparende rent backup-mæssigt at have denne opdeling.

3.6 Delkonklusion (Design)

Jeg har valgt at beskrive designet at programmet uden at gå i detaljer, men kun vise de store linier. Jeg har i stedet valgt at beskrive detaljer i næste kapitel

"Implementering", da jeg der kan vise de færdige klasser i helhed. Dette er gjort til dels fordi jeg mener, at dette giver et større overblik over mine overvejelser og opbygningen af programmet. Desuden har jeg, pga. den korte tidsramme, valgt at begynde implementeringen efter dette grundlæggende design. Dette har kun kunnet lade sig gøre, fordi jeg har været alene om projektet, hvor jeg har benyttet en masse grovskitser (som ingen andre kan læse) ved tilblivelsen af programmet.

Med hensyn til den valgte opdeling i "Design" og "Content", har jeg valgt den af flere årsager. Som nævnt tidligere, skaber denne opdeling en mere overskuelig opbygning, men grunden til de forskellige database-tilgange er også, at jeg har valgt, at "Design"-delen kun skal understøtte en bruger ad gangen, hvorfor et dataset kan benyttes som længerevarende container. Derimod er "Content"-delen meget afhængig af at understøtte flere tilgange på samme tid, da den bliver brugt både ved system-delen og i kundernes webløsning. Derfor kræves en direkte tilgang, hvor databasen lukkes hurtigt igen efter læsning.

Kapitel 4

Implementering

I dette kapitel, vil jeg vise de implementerede klasser og gøre rede for deres parametre, funktioner og interaktion med hinanden. Derefter vil jeg vise de endelige databasestrukturer. Jeg har i dette afsnit valgt at ændre begreberne fra system/webløsning til henholdsvis admin og default. Det har jeg gjort på baggrund af implementeringen, hvor jeg har oprettet to websider; Admin.aspx og Default.aspx, som eneste tilgange til programmet.

4.1 Klasserelationer

Herunder vil jeg vise klasserelationerne, som jeg har implementeret dem. Jeg har taget udgangspunkt i mit design fra foregående kapitel. Grunden til at jeg vælger at vise dem i dette kapitel i stedet for under design, er at jeg her benytter filnavne fra programmet. Jeg har valgt at vise klasserelationerne opdelt i to grupper. Den ene er interfaces og den anden er den indre arkitektur samt databaseforbindelse. Desuden har jeg i den grafiske fremstilling valg at opdele i "Design" og "Content". Dette betyder ikke at det er to forskellige default- og adminsider.

4.1.1 Interface (Default)

I figur 4.1 vises opbygningen af interfacet i default-delen. I "Design"-delen kan ses, at jeg benytter BuildCss.cs via Default.aspx til at oprette stylesheetet DynamicCss.css. I "Content"-delen, opretter Default.aspx en MenuList.cs som container for menustrukturen, hvorefter den opretter en ModuleList.cs som container for moduler tilknyttet den valgte menu. Herefter benyttes Module.ascx til at vise modulerne på siden.

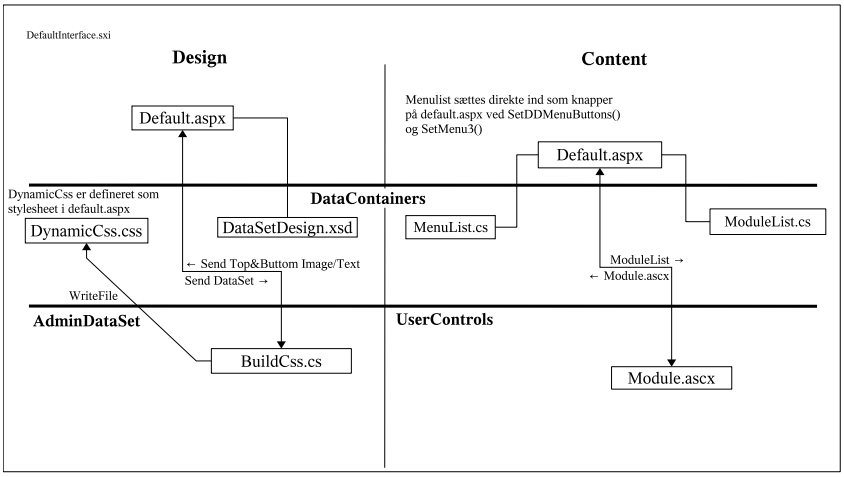

Figure 4.1: Klasserelationer default-interface

4.1.1.1 DropDownMenu

Grunden til dette seperate afsnit er, at jeg har benyttet mig af et særskilt javascript-fil, cbddm.js. til denne feature. Som udgangspunkt for denne kode har jeg hentet en dropdownmenu fra http://www.webdevtips.co.uk/dropmenu/index.html. Det system var statisk, så jeg har måtte ændre en del i tilgangen til koden, men funktionaliteten er lånt derfra.

4.1.2 Interface (Admin)

I figur 4.2 vises opbygningen af interfacet i admin-delen. I "Design"-delen heri, kan ses at jeg fra Admin.aspx benytter 4 faste kontroller, der svarer til de øverste faneblade[1] der er defineret i Admin.aspx. Disse kontroller benytter herefter faste skemaer (WebCostumControls defineret i en særskilt dll-fil) til brugerinput. I "Content"-delen benyttes ModuleControl.ascx og MenuControl.ascx, der igen bliver styret af de øverste faneblade. MenuControl.ascx benytter kontrollen MenuForm.ascx til brugerinput, samt TranslateMenu.ascx til at oversætte menuen til et andet sprog. ModuleControl.ascx loader ModuleTableForm ind til brugerinput, hvori Text.ascx og Image.ascx kan indsættes til henholdsvis at indsætte tekst og billede i modulet. Her er igen en klasse til oversættelse af modulet. Denne klasse benytter de samme Text.ascx og Image.ascx klasser, hvori oversættelsen kan indsættes. Derudover benyttes Module.ascx, som beskrevet i ovenstående afsnit, til preview af modulet.

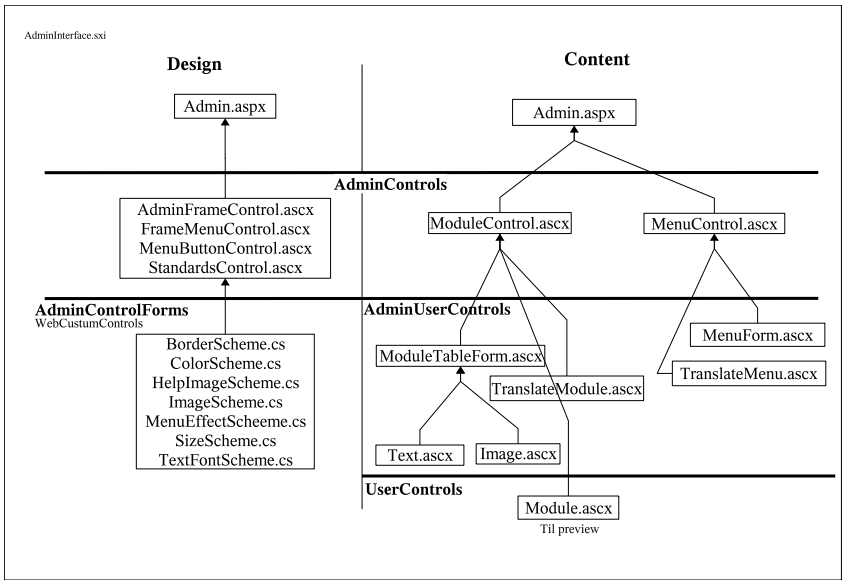

Figure 4.2: Klasserelationer admin-interface

[1] Se bilag A billede A.1 TopLevel faneblade

4.1.3 Data fra database (Default)

I figur 4.3 vises hvordan Default.aspx henter data fra databasen. I "Design"-delen benyttes UpdateDesign.cs til at opdatere datasettet DataSetDesign.xsd fra databasen Design.mdb. Dette er en midlertidig løsning, jeg har lavet for at kunne hente tekst og billeder til topfelt og bundfelt[2]. Dette er ikke en ønskelig løsning, da datasettet ikke bør benyttes til denne del af programmet. I stedet skulle stylesheetet være den eneste datatilgang. I "Content"-delen vises hvordan MenuList.cs og ModuleList.cs opbygges ved brug af underklasser. Her benyttes DBRead.cs som facade for de overliggende objekters tilgang til data i databasen Content.mdb.

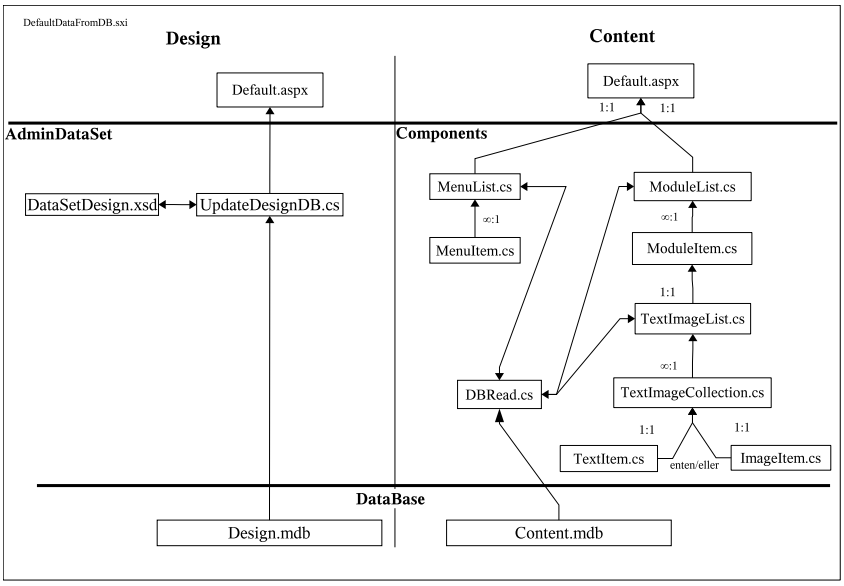

Figure 4.3: Klasserelationer defaut-læsning fra database

4.1.4 Data fra database (Admin)

I figur 4.4 vises hvordan data fra databasen bliver hentet og vist i Admin.aspx. I "Design"-delen ses at admin.aspx benytter AdminControls og WebCostum-Controls[3]. Data fra databasen bliver opdateret i datasettet DataSetDesign.xsd

[2]Se bilag A billede A.2
[3]Se figur 4.2 for beskrivelse af disse

via UpdateDesignDB, hvorefter datasettet benyttes som contaner for Updatekontrollerne. Disse bliver brugt af WebCostumControls til at vise data fra databasen. I "Content"-delen benyttes DBRead af ModuleTableForm.ascx og MenuTableForm.ascx til at hente data fra databasen. Disse forms loades ind i AdminControls[4], der igen bliver loadet ind i Admin.aspx.

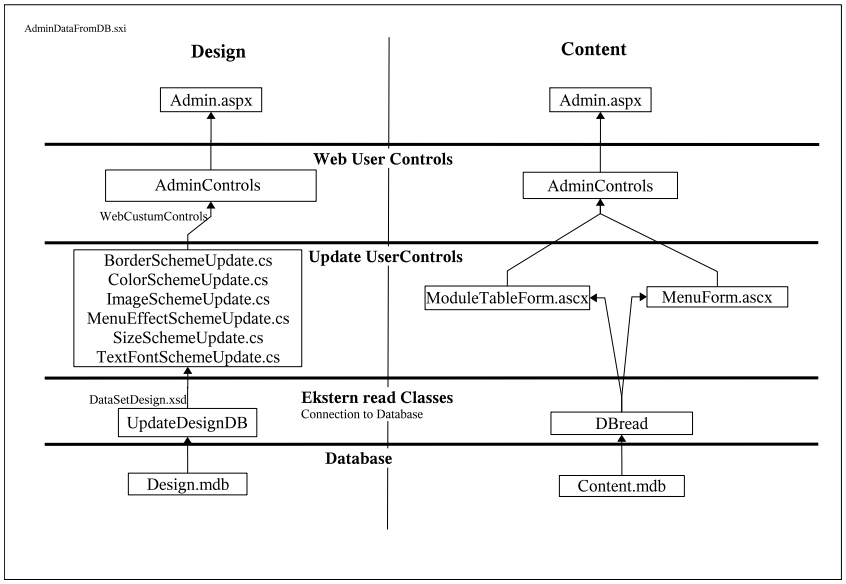

Figure 4.4: Klasserelationer admin-læsning fra database

4.1.5 Data til database (Admin)

I figur 4.5 vises hvordan data fra brugerinterfacet gemmes i databaserne. I "Design"-delen benyttes UpdateDataSet.cs af AdminControls. Dette gøres ved at sende WebCostumControls til UpdateDataSet.cs, der derefter udtrækker de nye værdier indsat af brugeren og indsætter dem i datasettet. Derefter benyttes UpdateDesignDB til at skrive hele datasettet ind i databasen. I "Content"-delen benytter ModuleTableForm.ascx og MenuForm.ascx henholdsvis WriteModule.cs og WriteMenu.cs, der forbereder moduler og menuer til skrivning. Herefter benyttes DBWrite til at indsætte moduler og menuer i databasen.

[4]Se igen figur 4.2

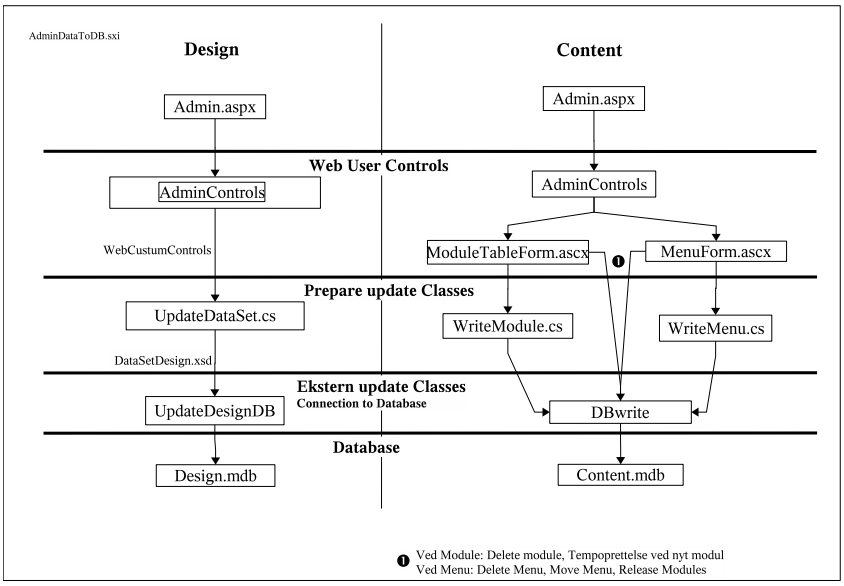

Figure 4.5: Klasserelationer admin-skrivning til database

4.1.6 Login

Jeg har valgt ikke at lægge så meget vægt på sikkerheden. Login-proceduren består i at brugeren, når sessionen starter, får vist Login.ascx, hvor der indtastes brugernavn og password. Ud fra det indtastede hentes brugeren fra Content.mdb via DBread. Herefter gemmes brugertypen i en sessions-variabel, og alt efter hvilken brugertype det er vises knapper til de tilhørende AdminControls. Der kan til enhver tid logges ud igen, ved at brugeren benytter Login.ascx igen, og trykker Logout, hvorefter sessions-variablen slettes.

4.1.7 Administration

Ved Administrationen benytter jeg en særskilt kontrol, AdminControl.ascx. Her gives der mulighed for Administratoren for at administere brugere til webløsningen. Her er det muligt at se brugernavn og password, samt hvilken type brugeren er. Desuden kan der oprettes nye brugere. Brugerne skrives ind i databasen Content.mdb via DBwrite. Derudover kan der gemmes en sikkerhedskopi af databaserne, der uploades til brugeren. Denne sikkerhedskopi kan så igen indsættes i systemet og derved overskrive den nuværende. Denne sidste del, virker

kun i Access-udgaven af programmet, da SQL serveren ikke benytter en enkel fil.

4.2 SystemSekvensDiagrammer

Jeg har valgt ikke at gøre så meget ud af netop dette aspekt af projektet. Dette skyldes først og fremmest tidspres, at lave systemsekvensdiagrammer tydelige nok til at andre kan læse dem. Det skal ikke forstås sådan at jeg ikke har lavet systemsekvensdiagrammer, men at jeg har lavet dem i hånden som skitser under design og implementeringen af programmet. Jeg har trods alt lavet to eksempler på systemsekensdiagrammer, hvor jeg har taget udgangspunkt i henholdsvis UseCase 5 og 15. Disse kan ses i bilag B.

4.3 Systemklasser

Jeg har, ved implementeringen af klasserne og deres funktioner, gjort meget for at de har beskrivende navne. Nogle af disse er vist i bilag C. Jeg har valgt ikke at sætte dem alle sammen ind, men henviser i stedet til CD'en, hvor de alle er at finde som jpeg-filer under mappen Klasser. Dette skyldes at de ikke er til at læse, hvis de skal tilpasses en side. Jeg har ikke set det nødvendigt at gennemgå klasserne trin for trin, men ment at de udmærket kan representere sig selv, hvis man sammenholder dem med "Klasserelations"-afsnittet.

4.4 Databaser

Databasernes opbygning er vist i bilag D. Jeg har, som tidligere nævnt, startet med to access-databaser, og herefter lavet en version til MS SQL-server. Heri har jeg taget udgangspunkt i Access-databasen, men SQL-serverens databaser er identiske med disse.

4.5 Delkonklusion (Implementering)

Implementeringen og designet af programmet har været meget tæt knyttet sammen. Det skyldes både den korte tidshorisont, men også det faktum at jeg har arbejdet alene med det. Derfor har det ikke været helt så nødvendigt at lægge mig helt fast på hvordan klasserne skulle agere i forhold til hinanden. Hvis der havde været flere om projektet, havde det været nødvendigt, for at kunne arbejde uafhængigt af hinanden. Der har ikke været nogen væsentlige problemer i forhold til implementeringen. Jeg har ikke nået at oprette en tilfredsstillende fejlhåndtering. Dette skyldes tidspres, men jeg forventer at få den del implementeret hurtigst mulig. Jeg har valgt denne del fra, da jeg mener at det vigtigste var at få lavet et stabilt system med de ønskede funktioner.

Kapitel 5

Test

Jeg har igenmme hele implementeringsfasen løbende testet klasser og deres funktioner. Jeg har ikke benyttet mig af et specielt testprogram, men brugt breakpoints til at undersøge hvorvidt det ønskede resultat fremkom. Denne del har jeg ikke gjort meget ud af at dokumentere, da jeg ikke er stødt på fejl, der ikke hurtigt kunne rettes.

Kapitel 6

Konklusion

Jeg mener at have lavet et seriøst alternativ til de gængse CMS'er, der i forvejen er på markedet. Systemet har også vist sig stabilt, da det er at finde på www.teaterkatalysator.dk. Jeg har, som beskrevet tidligere, valgt at fokusere på at lave et færdigt brugbart system, der er simpelt at bruge. Det har desværre betydet at fejlkontrol og testdelen ikke har fået nok tid. Jeg mener dog at den ultimative test er gjort, ved at lade teaterforeningen TeaterKatalysator benytte programmet. En skrivelse fra TeaterKatalysators formand, mener jeg er bevis for at jeg har lykkedes med mit projekt. Denne kan ses i bilag E

Der er stadig mange ting, der kunne forbedres ved programmet. Bl.a. har jeg ikke gjort mig speciel umage for at optimere koden til tusinder af kunder. Dette har været et bevidst valg, da min målgruppe ikke har dette ønske. Jeg ville desuden gerne have givet mulighed for fritekst-søgning og mulighed for indsættelse af meta-data, men det har der desværre ikke været tid til. Jeg regner dog med at udbygge systemet efter denne projektperiode, så de tertiære krav i kravspecifikationen også bliver opfyldt.

Bilag A

Interface-design

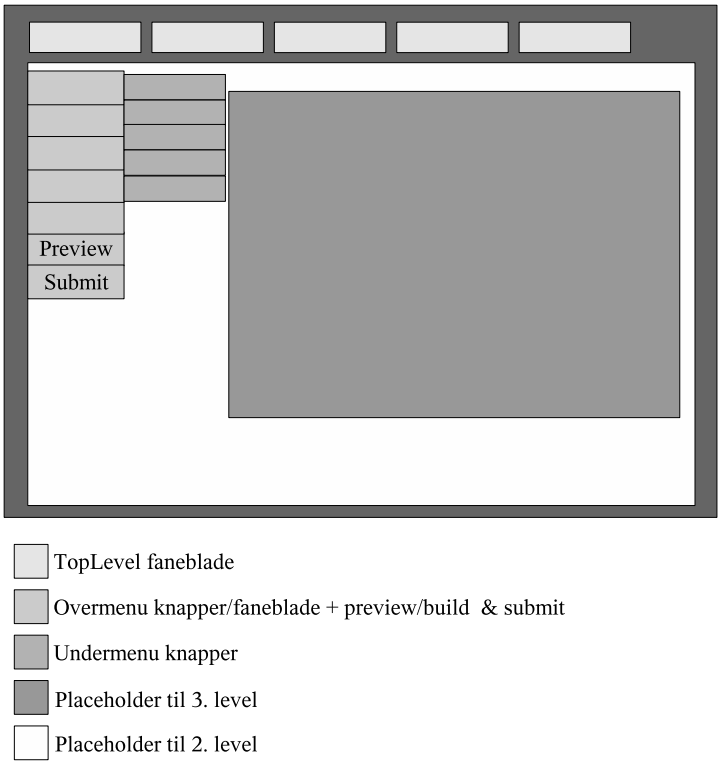

Figure A.1: System-interface

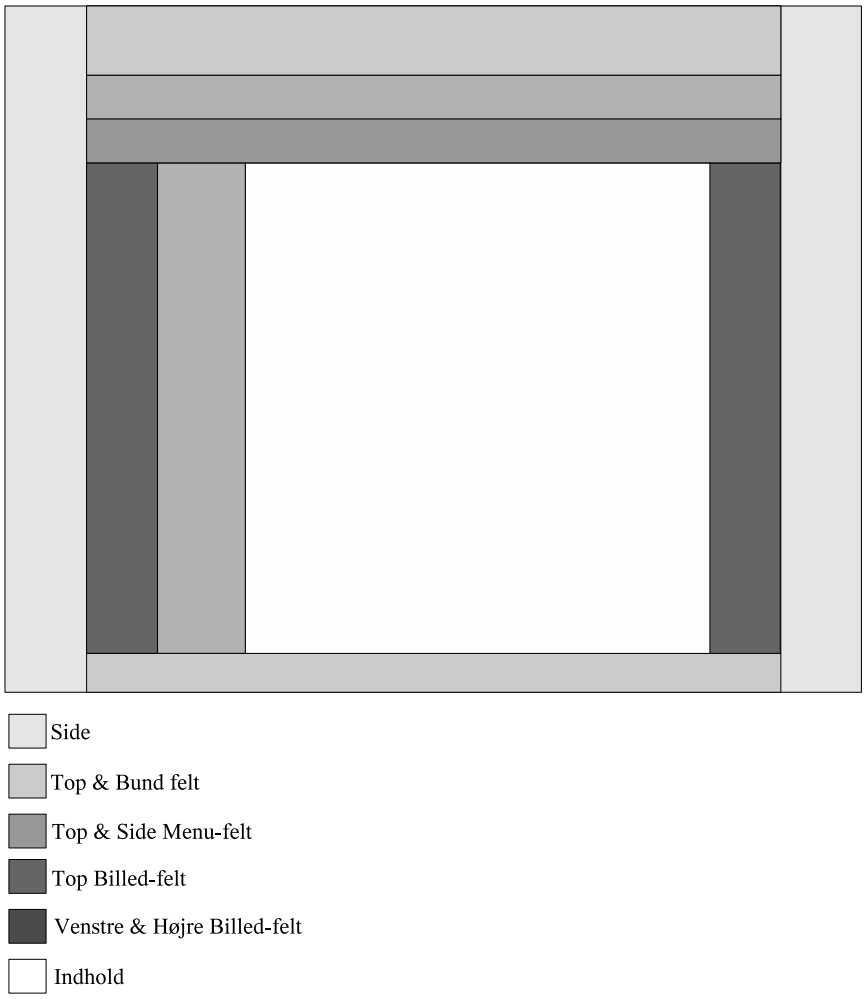

Figure A.2: Webløsning-interface

Bilag B

System-sekvens-diagram

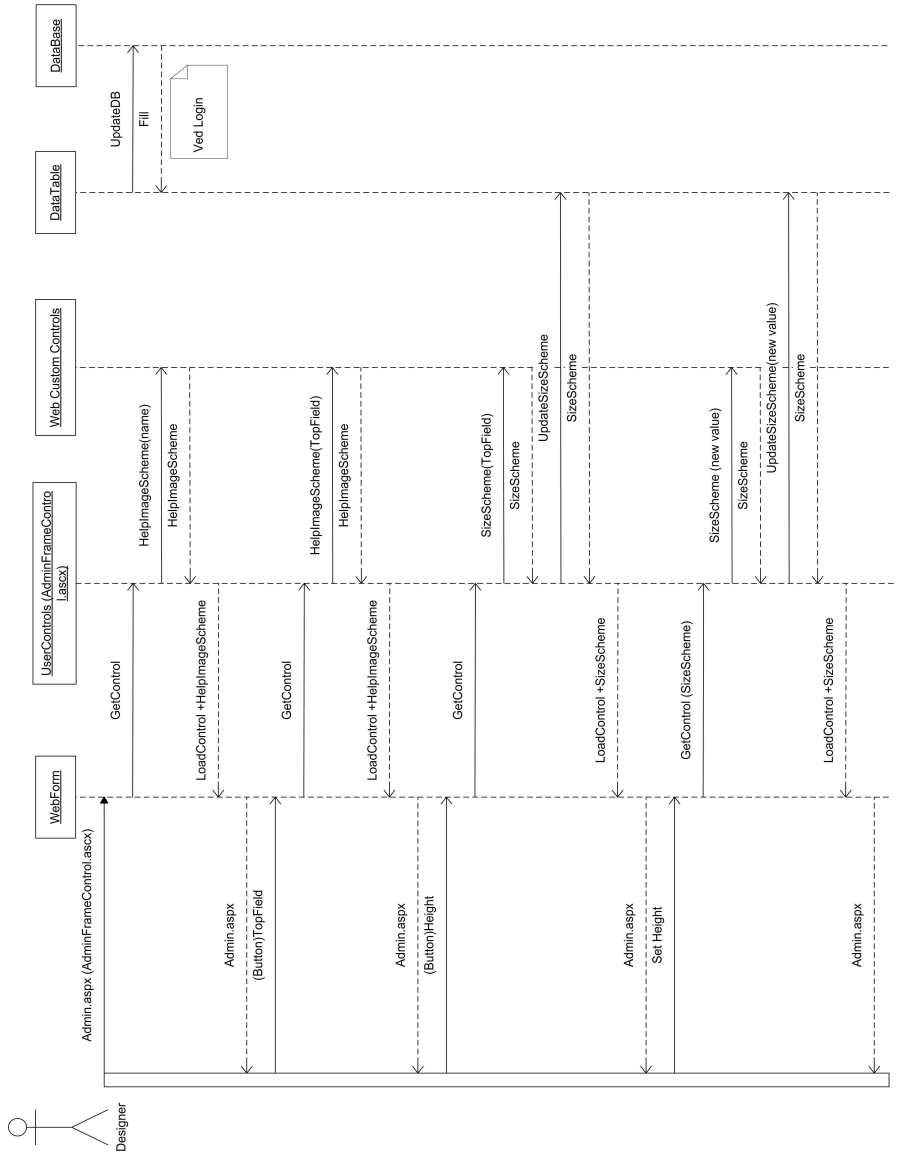

Figure B.1: Systemsekvensdiagram "Design"

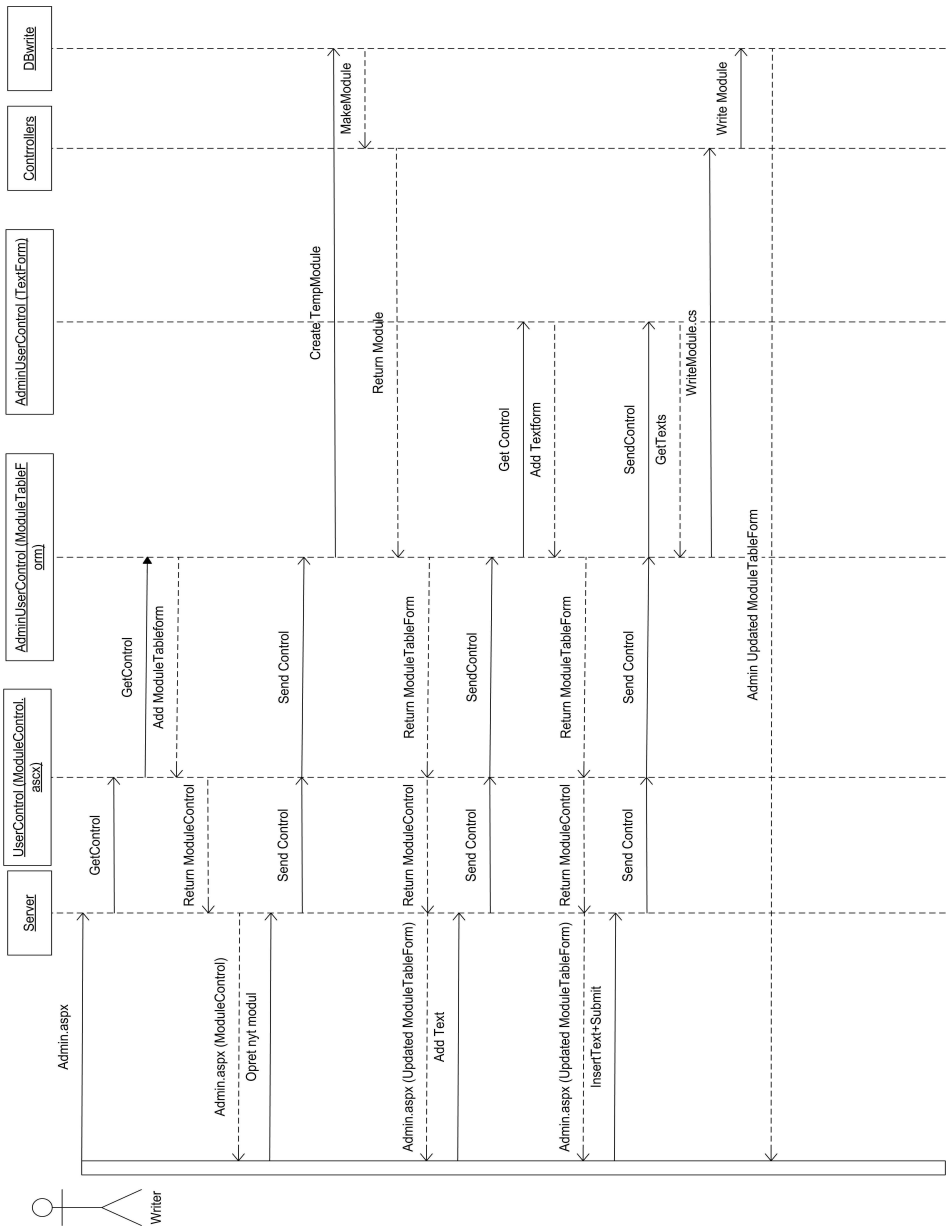

Figure B.2: Systemsekvensdiagram "Content"

Bilag C

SystemKlasser

Admin
#btnFrame : Button #btnFramMenu : Button #btnMenuBtns : Button #btnStandards : Button #btnMenus : Button #btnModules : Button #btnAdmin : Button #btnLogout : Button #phMain : PlaceHolder
-Page_Load(ind sender : object, ind e : EventArgs) -ShowButtons(ind Type : int) #OnInit(ind e : EventArgs) -InitializeComponent() -btnFrame_Click(ind sender : object, ind e : EventArgs) -btnFramMenu_Click(ind sender : object, ind e : EventArgs) -btnMenuBtns_Click(ind sender : object, ind e : EventArgs) -btnStandards_Click(ind sender : object, ind e : EventArgs) -btnMenus_Click(ind sender : object, ind e : EventArgs) -btnModules_Click(ind sender : object, ind e : EventArgs) -btnAdmin_Click(ind sender : object, ind e : EventArgs) -btnLogout_Click(ind sender : object, ind e : EventArgs)

Default
-tdNavbar : Table = new Table() -trNav : TableRow = new TableRow() -tbSubMenus : Table -trSubMenus : TableRow -tdSubMenus : TableCell -dsDesign : DataSetDesign -iImage : int[] -sImagePath : string -sText : string[] -MenuType : int -MenuEffect : int -pageId : int -TopButtonNr : int -languageId : int = 1 -thisMenuList : MenuList -thisMenuItem : MenuItem = new MenuItem() -ConnString : string -ModuleIDs : ArrayList = new ArrayList() #phMain : PlaceHolder #phLeft : PlaceHolder #phCenter : PlaceHolder #phRight : PlaceHolder
-Page_Load(ind sender : object, ind e : EventArgs) -GetColumnWidths(ind pageType : int) : int[] -SetAlignment(ind td : TableCell, ind ver : int, ind hor : int) : TableCell -SetImage(ind imgBtn : ImageButton, ind imgName : string, ind imgRow : ImagesRow) : ImageButton +SetDDMenuButtonsStd(ind tdMenu : TableCell, ind menuType : int, ind menuEffect : int) : TableCell +SetDDMenuButtonsPre(ind tdMenu : TableCell, ind menuType : int, ind menuEffect : int) : TableCell -SetMenu3Std(ind tdMenus : TableCell[], ind PageID : int) : TableCell[] -SetMenu3Pre(ind tdMenus : TableCell[], ind PageID : int) : TableCell[] -RenderTopMenu(ind btnId : string) -OnButton_Click(ind sender : object, ind e : CommandEventArgs) -OnLink_Click(ind sender : object, ind e : CommandEventArgs) +GetLnkNames(ind pageId : int) : MenuItem[] -GetParentId(ind pageId : int) : int -SetControlDynamic(ind ColumnWidth : int[]) -SetDynamicModuleLeft(ind ColumnWidth : int) -SetDynamicModuleCenter(ind ColumnWidth : int) -SetDynamicModuleRight(ind ColumnWidth : int) #OnInit(ind e : EventArgs) -InitializeComponent()

Figure C.1: Admin.aspx & Default.aspx

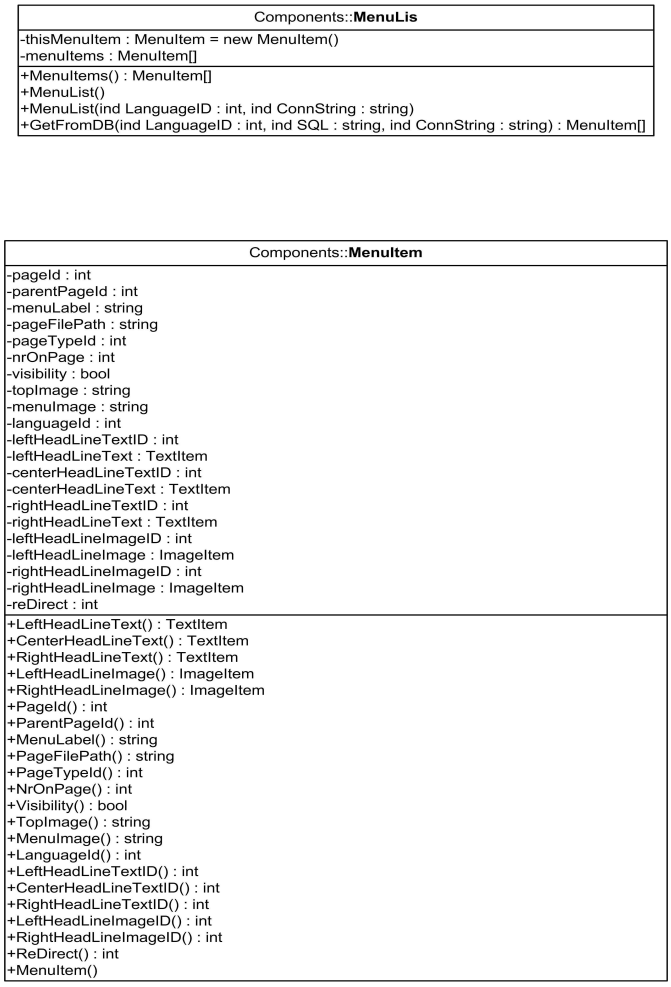

Figure C.2: MenuList.cs & MenuItem.cs

Components::**ModuleList**
-languageID : int -menuID : int -columnID : int -moduleCollection : ArrayList = new ArrayList()
+ModuleCollection() : ArrayList +ModuleList() +ModuleList(ind MenuID : int, ind LanguageID : int, ind ColumnID : int, ind ConnString : string) -GetFromDB(ind LanguageID : int, ind MenuID : int, ind SQL : string, ind ConnString : string, ind news : bool) : ModuleList

Components::**ModuleItem**
-moduleID : int -languageID : int -connString : string -nrOnModule : int[] -newsDate : DateTime -topSpaceT : int -topSpaceB : int -topLine : bool -bottomSpaceT : int -bottomSpaceB : int -bottomLine : bool -columnID : int -columnWidth : int -nrOnPage : int -news : string -header : string -subheader : string -rightImage : ImageItem -txtimgLst : Text_ImageList -rightImageID : int
+NewsDate() : DateTime +RightImage() : ImageItem +TxtImgLst() : Text_ImageList +NrOnModule() : int[] +TopSpaceT() : int +TopSpaceB() : int +TopLine() : bool +BottomSpaceT() : int +BottomSpaceB() : int +BottomLine() : bool +ColumnID() : int +ColumnWidth() : int +NrOnPage() : int +News() : string +Header() : string +SubHeader() : string +ModuleID() : int +RightImageID() : int +ModuleItem() +ModuleItem(ind ModuleID : int, ind LanguageID : int, ind ColumnID : int, ind ConnString : string) +ModuleItem(ind ImageID : int, ind ColumnID : int, ind ConnString : string)

Figure C.3: ModuleList.cs & ModuleItem.cs

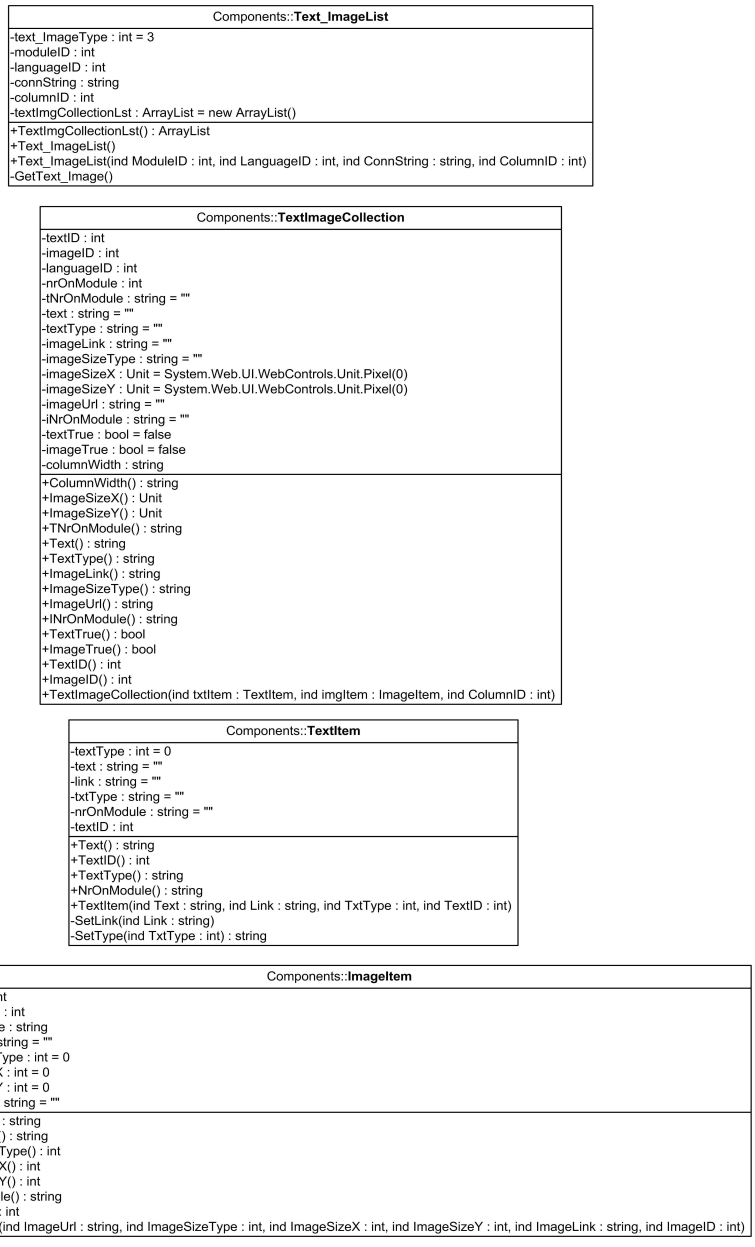

Figure C.4: TextImageList.cs & TextImageCollection.cs & TextItem.cs & ImageItem

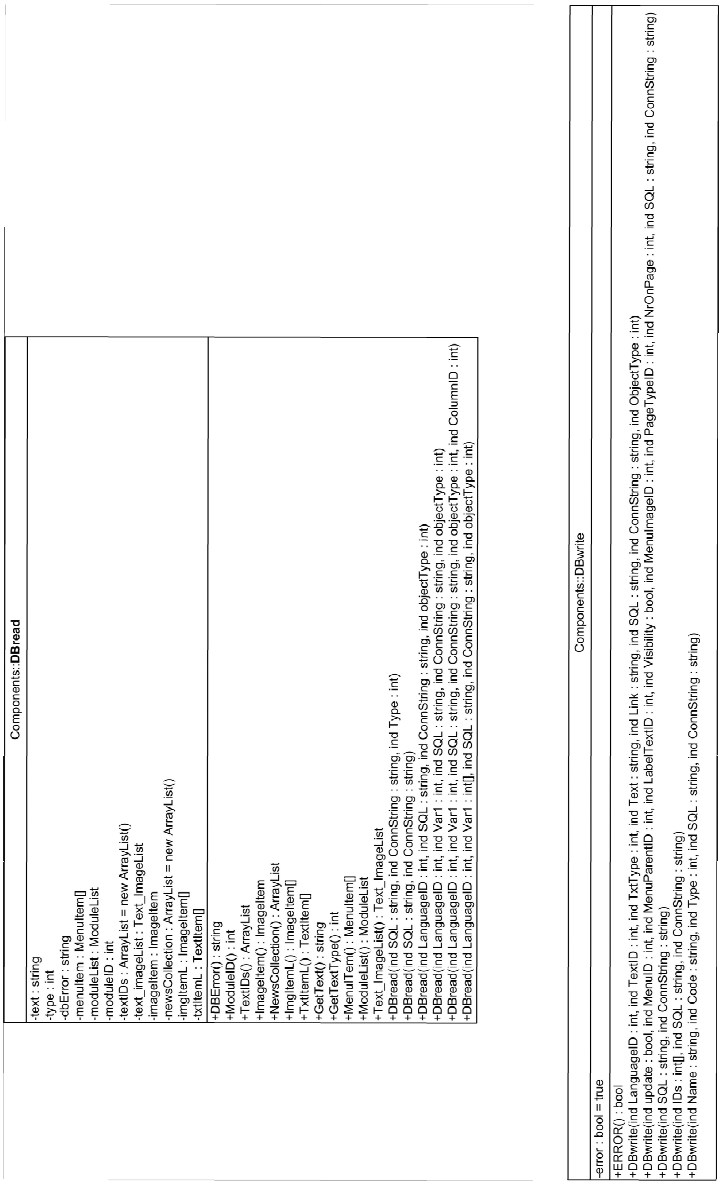

Figure C.5: DBRead.cs & DBWrite.cs

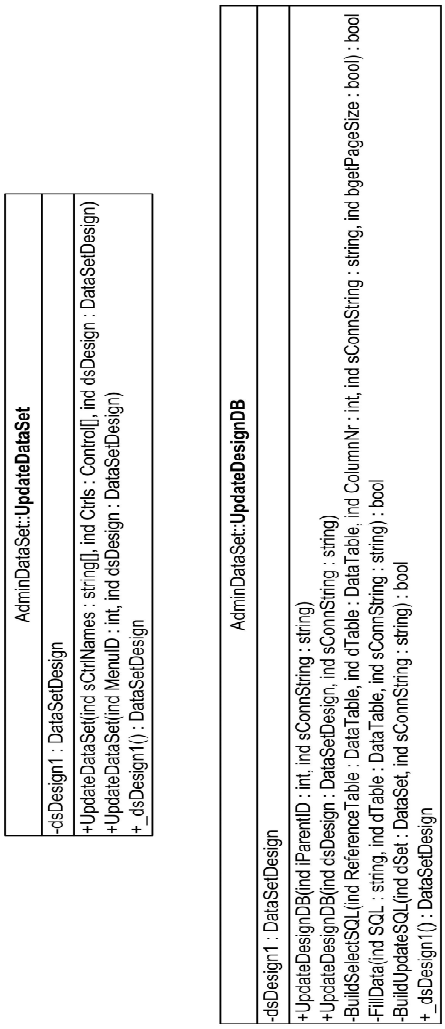

Figure C.6: UpdateDesignDB.cs & UpdateDataSet.cs

Bilag D

Databaser

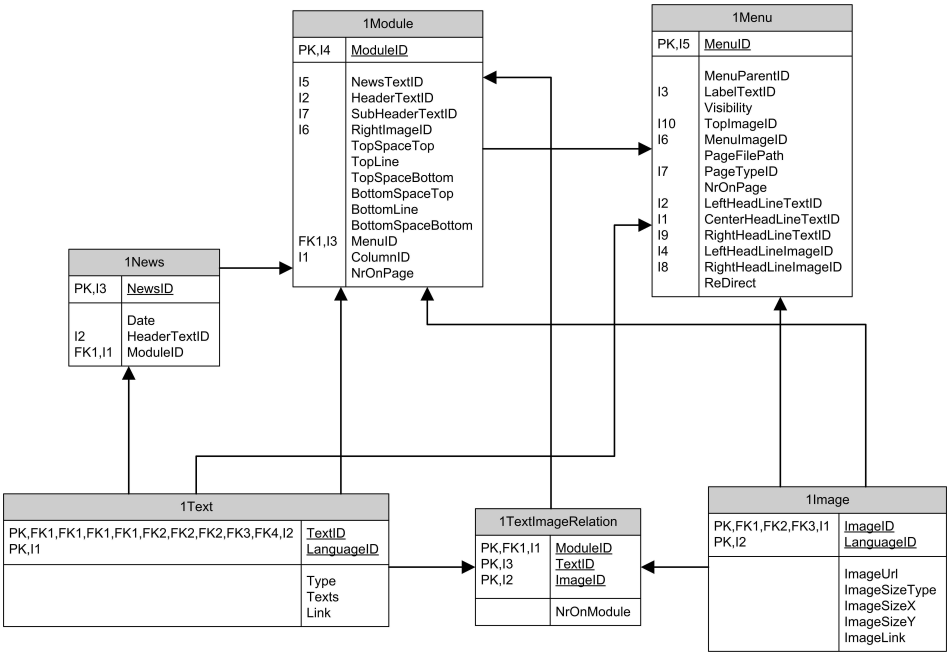

Figure D.1: Content database

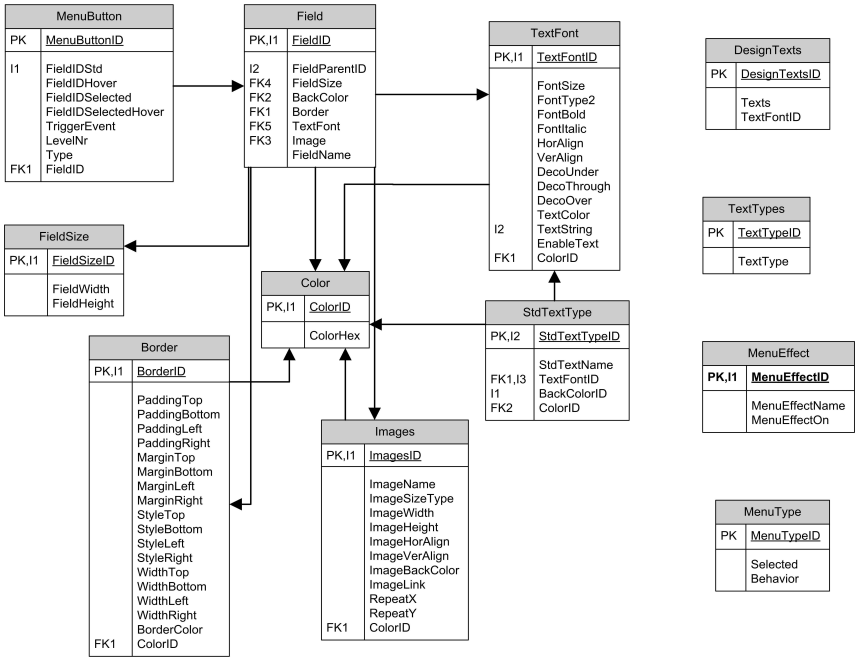

Figure D.2: Design database

Bilag E

TeaterKatalysator

TeaterKatalysator
www.teaterkatalysator.dk

Vanløse juli 2006

Vedr. Samarbejdet mellem Lasse Trier og TeaterKatalysator

TeaterKatalysator er en lille teaterforening på omkring 20 medlemmer under Københavns Kommune. Den startede for godt tre år siden og hovedformålet er at udfordre de etablerede københavnske teaterscener, med noget seriøst og ambitiøst teater for amatører. Det er en forening der drives af frivillige kræfter og målet er at den økonomisk bare skal kunne løbe rundt så vi kan blive ved med at sætte to forestillinger op årligt. Der er således ingen der bliver lønnet i TeaterKatalysator, hverken skuespillere, bestyrelsesmedlemmer, teknikere eller instruktører.

For en teaterforening som denne, er det essentielt at have en web-side der er funktionel og let at opdatere. Internettet er den billigste eksponeringskilde, og har man ikke en webside i denne tidsalder, skal man ikke forvente at man med en forening som denne, kan overleve ret længe.

Vi var derfor meget interesserede i at finde en webløsning som kunne opfylde vores primære krav:

1. At vi selv kan opdatere teksterne på siden så vi ikke er hverken tidsmæssigt eller økonomisk afhængige af en webudvikler.
2. At vi selv kunne designe den, så vi ikke var nødt til at købe en dyr og ikke-tilpasset designet færdigløsning.
3. At vi selv kan indsætte nye menu-punkter i takt med nye tiltag og overståede forestillinger.
4. At vores publikum kan bestille billetter direkte på vores hjemmeside
5. At vi ikke er afhængige af at have en It-kyndig fast tilknyttet vores forening.

Vi indledte derfor et samarbejde med Lasse Trier, der netop ville lave en webløsning der kunne opfylde disse krav.

TeaterKatalysator
v / Nina Bundgaard
Email: nina@teaterkatalysator.dk

Figure E.1: Brev fra TeaterKatalysator side 1

TeaterKatalysator
www.teaterkatalysator.dk

Vi har løbende været med til at teste programmet i takt med at det blev udviklet, og derigennem har det været muligt for begge parter at diskutere og udvikle programmet således at det opfyldte disse krav. Resultatet af dette samarbejde kan ses på www.teaterkatalysator.dk, hvor vi nu har en hjemmeside der opfylder alle de krav vi havde til den.

At designe siden tog ikke lang tid, og programmet var meget brugervenligt at arbejde med. Det var en stor hjælp at der var forklarende billeder, samt mulighed for hele tiden at generere en test-side hvor man kunne se resultatet af ens valg.
Den færdige side er funktionel og let at opdatere, og det er kun i tilfælde af at man vil indsætte interaktive sider at en IT-kyndig er nødvendig. Da vi skulle bruge en billetbestilling ved vores sidste forestilling fik vi lavet en side der kunne dette, og nu hvor den er lavet, kan vi bare skjule den, og gøre den synlig igen, når vi næste gang skal sælge billetter.
En anden vigtig funktion ved siden er desuden at det er muligt for vores medlemmer at lave indlæg til siden. Det er nemt at indsætte og redigere både moduler og menuer.

Vi har været meget glade for samarbejdet og det endelige resultat!

Med Venlig Hilsen

Nina Bundgaard
Formand, TeaterKatalysator

TeaterKatalysator
v / Nina Bundgaard
Email: nina@teaterkatalysator.dk

Figure E.2: Brev fra TeaterKatalysator side 2

Bilag F

Begrebsliste

Bruger: En bruger er en fællesbetegnelse for alle roller der kan benytte administrationsdelen af systemet. Altså en bruger, der er blevet tildelt rettigheder i systemet.

Webløsning: Webløsningen, er det færdige udviklede produkt, som kunden kan se. Dette kunne også kaldes et WebSite.

System: Systemet definerer jeg som den del af programmet, der bruges til at opbygge webløsningen.

"Design": Ved Design menes den del af webløsningen, der bliver defineret ved opbygningen af sitet. Design er præsentationen af sitet, f.eks. et topbillede på sitet, eller hvilken teksttype der skal anvendes. Design skal ses i forhold til "Content", der er indholdet af sitet.

"Content": ved Content menes den del af webløsningen, der er indholdet. Altså tekst, billeder og sider, der skal vises til kunden. I forhold til "Design" er Content, det der skal vises, mens "Design" er måden hvorpå det bliver vist.

Modul: En samling tekst og billeder. Kan også betegnes som en artikel.

Menu: En menu kan ses som en side i webløsningen.

Bilag G

Vejledninger

G.1 Installationsvejledning

IIS skal være installeret og kørende på maskinen. Mapperne WebSolution og AdminControlForms skal kopieres fra CD'en til din www-root. Derudover skal der sikres at mappen "Data" + undermapper og filer i WebSolution er tilgængelig for editering. Derefter åbnes WebSolution.sln i Visual Studio .Net 2003, hvorfra programmet køres, enten med Admin.aspx eller Default.aspx som startside. Der vil desuden være en mappe med "tomme" databaser, der kan indsættes, såfremt man ønsker at starte derfra. De installerede databaser er allerede udfyldt med data fra TeaterKatalysator. Dette har jeg valgt, for at kunne vise lidt mere, uden at man behøver at opbygge en hel webløsning selv.

En anden mulighed er at benytte installationsfilen Websolution.msi i "Release"-mappen på cd'en, der installerer systemet. Her vælges "WebSolution" som mappe. Igen skal der sikres at mappen "Data" + undermapper er tilgængelige for redigering.

Såfremt du ikke ønsker at installere systemet lokalt, er du velkommen til at ringe til mig på 26224956 eller maile på trier@post.com, med et tidspunkt hvor du ønsker at teste systemet på min maskine. Min adresse er: http://217.157.203.215

G.2 Brugervejledning (Set Modules)

G.2.1 Opret modul

Oprettelse af modul:

1. Skriv www ."domænenavn"/Admin.aspx
2. Skriv brugernavn og adgangskode og tryk login.
3. Tryk "Set Modules".
4. Vælg "Opret nyt modul".
5. Derefter kan der vælges om modulet skal have separatorlinier.
6. Derefter kan der vælges at indsætte tekst eller billede i modulet.
   (a) Hvis Tekst er valgt, er der mulighed for at indsætte tekst, vælge teksttype og vælge om teksten skal være et link, ved at indtaste en URL i Link-feltet.
   (b) Hvis Billede er valgt, er der mulighed for at vælge fra en liste af billeder. Herefter kan størrelsen defineres, enten ved pixel-størrelse, eller ved at vælge en af de predefinerede størrelser. Også her er det muligt at gøre billedet til et link ved indtastnig af URL
7. Herefter trykkes submit, og trin 6 kan gentages.

G.2.2 Rediger modul

1. Skriv www ."domænenavn"/Admin.aspx
2. Skriv brugernavn og adgangskode og tryk login.
3. Tryk "Set Modules".
4. Vælg "Rediger modul".
5. Herefter vælges den side, modulet der skal rettes, ligger på. Hvis modulet ikke er tilknyttet en side, vælges frie moduler. Det er desuden muligt at liste alle moduler.
6. Under vælg modul vælger man det modul, der skal rettes, fra listen.

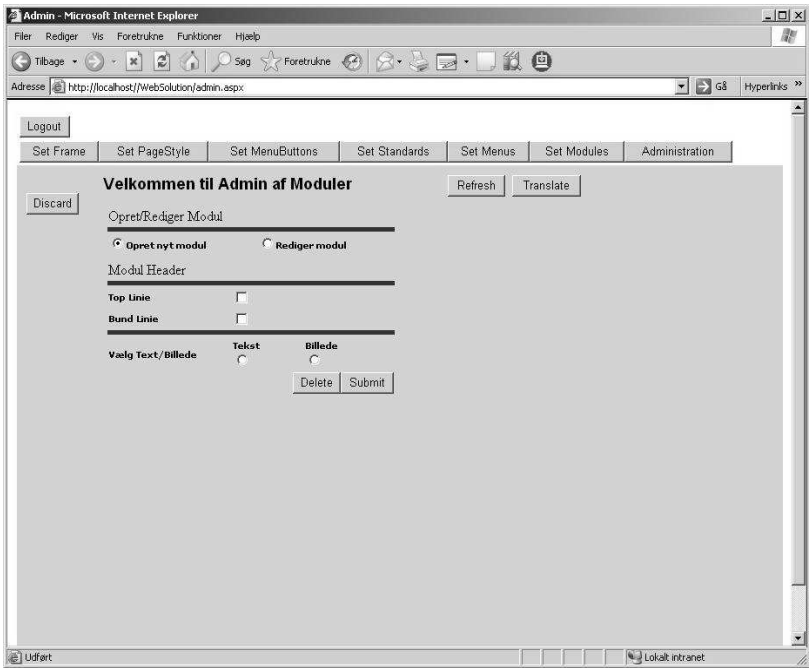

Figure G.1: Opret modul

7. Herefter kommer modulet frem i samme form, som under "Opret modul" og der kan ændres ved tekst/billede.

8. Hvis man ønsker at fjerne en tekst eller billede, gøres det ved at gøre "Tekst"/"Billede" feltet blankt.

9. Når man er færdig, trykkes "Submit" for at gemme. Det er muligt at få et preview af modulet, ved at trykke "Refresh".

G.3 Brugervejledning (Set Menus)

G.3.1 Opret menu

1. Skriv www ." domænenavn "/Admin.aspx

2. Skriv brugernavn og adgangskode og tryk login

3. Tryk "Set Menus"

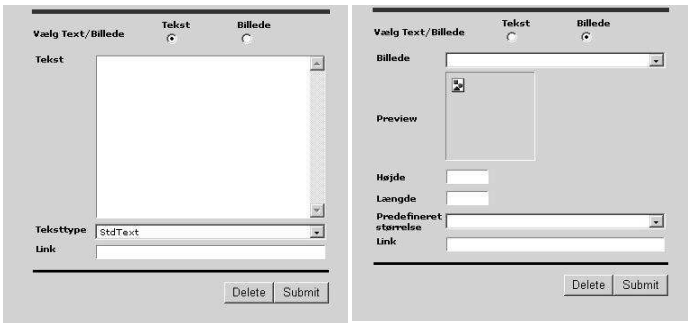

Figure G.2: Opret modul henholdsvis tekst og billede

4. Vælg "Opret nyt menu"
5. Ved "Overmenu" vælges hvor den nye menu skal placeres i menustrukturen. Hvis det skal være en hovedmenu vælges "Root"
6. Ved "Menunavn" indsættes navnet på menuen.
7. Sidetypen bestemmer hvor mange kolonner siden skal have.
8. Det er muligt at "hide/gemme" menuer, hvis de f.ex. ikke er aktuelle, men ikke skal slettes. Dette gøres ved at fravælge "Synlig".
9. Når man er færdig, trykkes submit, for at gemme.
10. Der trykkes Discard for at komme ud af MenuAdministrationen.

G.3.2 Rediger menu

1. Skriv www ." domænenavn "/Admin.aspx
2. Skriv brugernavn og adgangskode og tryk login
3. Tryk "Set Menus"
4. Fravælg "Opret ny menu"
5. Herefter er der to muligheder.
   (a) 1:
      i. Hvis man vil flytte rundt i menustrukturen, benyttes "Overmenu", hvor der vælges hvilken hovedmenu hvorunder menuerne skal flyttes.

ii. Herefter kan feltet position benyttes ved at flytte menuer op/ned eller slette dem. (Det er kun muligt at slette menuen, såfremt denne ikke har undermenuer.)

(b) 2:

i. Hvis man vil ændre i en specifik menu benyttes "Vælg Menu", hvor man vælger den menu man vil ændre.

ii. Derpå kan menuens navn ændres.

iii. Desuden er det også muligt at flytte menuen ved denne metode. Dette gøres ved at ændre på "Overmenu"en og derefter trykke submit. (Hvis menuen skal ændres fra et underpunkt til en hovedmenu, vælges "Root" som overmenu).

6. Herefter trykkes "Submit".

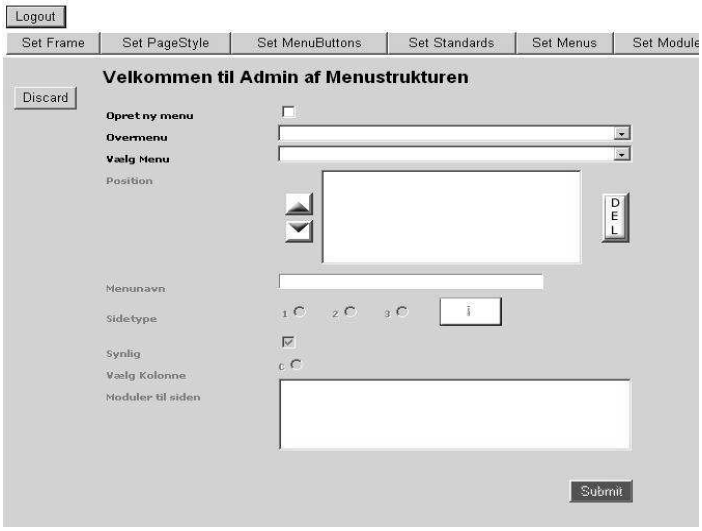

Figure G.3: Rediger i menustrukturen

G.4 Brugervejledning (Design)

Jeg har valgt at lægge brugervejledning til designet ind i systemet. I stedet for en skriftlig vejledning er der billeder for hver hovedmenu, der viser hvilken del af designet man nu beskæftiger sig med. Et eksempel herpå kan ses i figur G.4

Ved login:

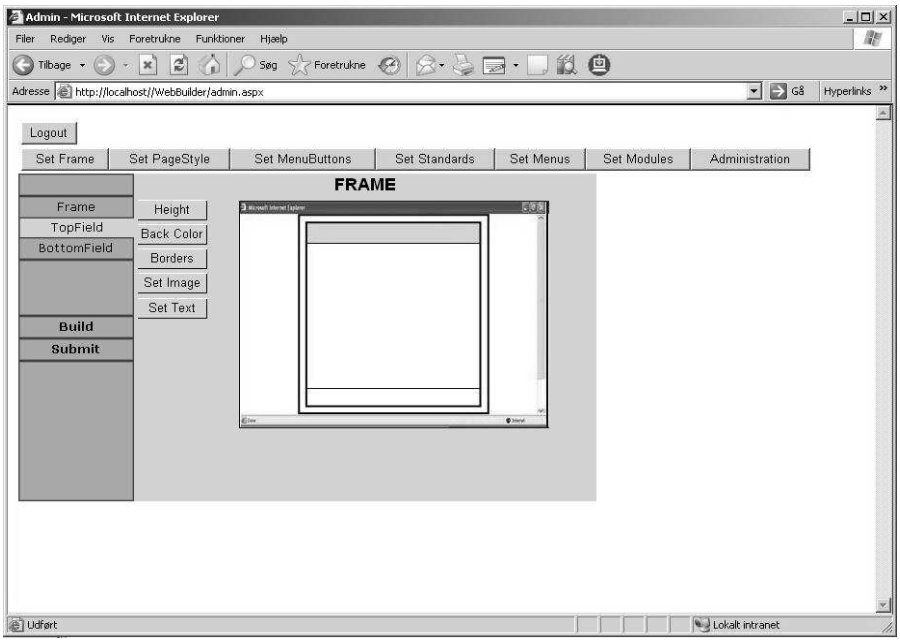

Figure G.4: Eksempel på design-interfacet med hjælpebillede

USERNAME: Admin

PASSWORD: Admin

Bilag H

Litteraturliste

Larman, Craig; "Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design and the Unified Process". 2nd edition. 2001. Publiseret af Prentice Hall Ptr. USA.

Stiefel, Michael & Oberg, Robert J.; "Application Development. Using C# and .Net". 2002. Publiseret af Prentice Hall PTR. USA.

Tittel, Ed (Editor); "MCAD/MCSD. Developing and Implementing Web Applications with Visual C# .Net and Visual Studio .Net". 2003. Publiseret af Que Publishing. USA.