

Billetbestillingssystem

Nordisk Film Biografer

Roskilde EDB skole
August-november 1998

Morten Selmer
Nikolaj Norman
Sandie Pedersen
Peter Theill



Indholdsfortegnelse

1. PROJEKTGRUNDLAG	4
1.1 RAMMER	4
1.2 RESSOURCER	4
<i>Biografi, Sandie Pedersen</i>	4
<i>Biografi, Peter Theill</i>	5
<i>Biografi, Morten Selmer</i>	5
<i>Biografi, Nikolaj Norman</i>	5
1.3 VEJLEDER	6
1.4 KONTAKTPERSON	6
1.5 PROJEKTORGANISATIONEN	6
1.6 PROJEKTGRUPPEN	6
1.7 PLANLÆGNING	7
1.8 PROJEKTMØDER	7
1.9 DOKUMENTATIONSFORM	7
1.10 KRITISKE FORUDSÆTNINGER	8
1.11 BESLUTNINGSPROCES	8
1.12 SIDEEFFEKTER	8
2. INTERN KONTRAKT	9
3. EKSTERN KONTRAKT	10
4. VALG AF VÆRKTØJER	12
4.1 TEKSTBEHANDLING	12
4.2 DIAGRAMTEGNING	12
4.3 PROJEKTPLANLÆGNING	12
4.4 IMPLEMENTERING AF HTML	12
4.5 GRAFISK LAYOUT	12
4.6 PROGRAMMERING: JAVA BUILDER 2.0	13
4.7 PROGRAMMERING: JAVA DEVELOPMENT KIT	13
4.8 PROGRAMMERING: SERVLETS	13
4.9 PROGRAMMERING: JDBC	13
4.10 ANSVAR	13
5. VALG AF METODE	14
5.1 METODER	14
5.1.1 Systemudviklingsmetoder	14
5.1.2 Objektorienteret metode [OO]	14
5.1.3 Analysemetode	14
5.1.4 Designmetode	14
5.1.5 Notationsform	15
5.1.6 Risikostyring	15
5.1.7 Implementeringsmetode	15
5.2 PROJEKTSTYRINGSMODEL	16
6. PROJEKTPLANLÆGNING	17
6.1 KVALITETSSTYRING	17
6.2 REFERENCELINIE 1: FORANALYSE	18
6.2.1 Mellemprodukter og ønskede tilstande	18
6.2.2 Kriterier for vurdering af mellemprodukter	18
6.2.3 Procedurer for vurdering af mellemprodukter	18
6.3 REFERENCELINIE 2: ANALYSE	19
6.3.1 Mellemprodukter og ønskede tilstande	19
6.3.2 Kriterier for vurdering af mellemprodukter	19



6.3.3 Procedurer for vurdering af mellemprodukter	20
6.4 REFERENCELINIE 3: DESIGN	20
6.4.1 Mellemprodukter og ønskede tilstande	20
6.4.2 Kriterier for vurdering af mellemprodukter	20
6.4.3 Procedure for vurdering af mellemprodukter	20
6.5 REFERENCELINIE 4: IMPLEMENTERING	21
6.5.1 Mellemprodukter og ønskede tilstande	21
6.5.2 Kriterier for vurdering af mellemprodukter	21
6.5.3 Procedurer for vurdering af mellemprodukter	21
6.6 REFERENCELINIE 5: TEST	21
6.6.1 Mellemprodukter og ønskede tilstande	21
6.6.2 Kriterier for vurdering af mellemprodukter	22
6.6.3 Procedure for vurdering af mellemprodukter	22
6.7 REFERENCELINIE 6: AFLEVERING	22



Professionel systemudvikling kræver aktiv og kompetent projektleje gennem hele forløbet. Projektgruppen bør derfor bruge tid på at gennemføre en systematisk projektetablering i starten af forløbet.

Mange grupper og enkeltpersoner har noget på spil i forhold til et systemudviklingsprojekt, og de vil hver på deres måde præge organisationen og forløbet af projektet.

Nedenfor gives fem gode grunde¹ til at gennemføre en systematisk projektetablering:

Projektetableringen kan bidrage til at gøre udgangspunktet mere realistisk.

På den ene side er projektet givet ved en mere eller mindre præcist beskrevet opgave og eventuelt ved en række øvrige målsætninger. På den anden side står projektgruppen som regel over for en række organisatoriske, økonomiske og tekniske betingelser.

Projektetableringen kan øge projektgruppens forståelse og accept af udgangspunktet for forløbet.

Opgaven bliver måske formuleret klarere og mere eksplicit. Deltagerne får mulighed for at sætte deres præg på projektet. Betingelserne kan gøres tydeligere og måske ændres. En forudsætning for at projektdeltagerne føler et ansvar overfor projektet er, at de kender opgaven og målene i det omfang, de kan formuleres.

Projektetableringen kan klarlægge og påvirke forholdet mellem projektgruppen og omgivelserne.

Projektgruppen skal forstå de krav og forventninger, omgivelserne stiller til projektet. Tilsvarende skal omgivelserne forstå de krav og forventninger, som projektet stiller. Omgivelserne er først og fremmest brugerne og brugerorganisationen, udviklingsorganisationens ledelse, interesseorganisationer og eventuelt andre projekter.

Projektetableringen kan styrke selve projektgruppen.

Arbejdsformen skal tilrettelægges og forløbet skal planlægges, og alle deltagernes individuelle forventninger og krav bør diskuteres.

Projektetableringen giver nogle holdepunkter i forhold til styringen af projektets videre forløb.

Et af de væsentlige produkter af etableringsaktiviteten er projektgrundlaget, der bl.a. præciserer opgaven, de kritiske forudsætninger for projektets gennemførelse, de overordnede planer samt de arbejdsformer, der tænkes anvendt.

I etableringsaktiviteten kommer også vigtigheden af at skrive kontrakter – både internt i gruppen og eksternt med kundevirksomheden – så alle parter ved, hvad de kan forvente.

¹ Hentet fra Andersen, Niels Erik et al.: Professionel Systemudvikling.



1. Projektgrundlag

Et væsentligt resultat af projektetableringen skal være et projektgrundlag, der som minimum fastholder de vigtigste beslutninger om projektets overordnede udformning og gruppens arbejdsform. Ideelt set bør projektgrundlaget afspejle konklusionerne og konsekvenserne af de mange analyser og diskussioner under projektetableringen.

1.1 Rammer

Præsentation af samarbejdspartner: Nordisk Film Biografer [NFB] er et søsterselskab af Nordisk Film, som ejes af Egmont-koncernen. NFB består af en lang række danske biografer både i provinsen og i hovedstadsområdet. Sammenslutningen består hovedsagelig af et fælles administrationssystem samt et fælles billetbestillingssystem.

Hovedkontoret ligger centralt placeret i Palads-biografen i København. Her foregår administrationen af biograferne, mens selve telefoncentralen og databasen ligger placeret i Imperial biografen. Tillige har NFB haft en hjemmeside, der beskrev de film der blev vist i de forskellige biografer landet over. Denne service er imidlertid overtaget af EON Film, som er produceret af Egmont Online A/S, der ligeledes er et søsterselskab af Egmont-koncernen.

Formål: At projektgruppen udvikler en prototype på et billetbestillingssystem, som skal anvendes over Internettet samt udvikler en rapport, der dokumenterer hele udviklingsforløbet.

Billetbestillingssystemet er et alternativ til deres nuværende billetsystem. Vores hjemmeside skal fungere som en skal, der kontakter deres billetbestillingssystem, når man bestiller en billet på Internettet.

1.2 Ressourcer

Personressourcer: Projektgruppen består af fire deltagere, som hver især i projektperioden må forventes af bruge 40 mandtimer pr. uge. Den samlede projektkapacitet består dermed af ca. 1920 mandtimer. Projektgruppen anvender flekstid som arbejdsform, og der påregnes ca. to timers hjemmearbejde. Mødetiden er kl. 9⁰⁰ og der arbejdes til omkring kl. 15⁰⁰. Alle projektdeltagere må forventes at yde deres optimale i hele perioden. Projektgruppen består af følgende personer:

Biografi, Sandie Pedersen

Sandie er 22 år og efter gymnasiet fortsatte hun på tjeneruddannelsen, hvorefter hun fortsatte på datamatikeruddannelsen. Kan bidrage ved OOA og OOD samt som den, der skal samle trådende i projektet. Specialefag: Objektorienteret systemudvikling og systemadministration.

Stamdata

Sandie Pedersen
C/O Camilla Jansson
Neergårdsparken 35 1. Mf.



4000 Roskilde
Tlf.: 4632 4237
Email: starwarsboots@hotmail.com

Biografi, Peter Theill

Peter er 21 år og efter gymnasiet, fortsatte han direkte til datamatikeruddannelsen. Hovedsageligt kan han bidrage mest på den programmeringsmæssige side, da han har erfaringer med mange forskellige sprog, fx Java, C/C++, Pascal/Delphi, JavaScript, VBScript, Svendsk, Basic og Assembler. Specialefag: Oversættertechnik & konvertering og objektorienteret programmering.

Stamdata

Peter Theill
Kalundborgvej 173
4300 Holbæk
Tlf 5944 3314
Email: peter@conquerware.dk

Biografi, Morten Selmer

Morten er 27 år og har for en del år siden taget Højere Handelseksamen, hvorefter han tilbragte 3 år i flyvevåbnet, hhv. som hundefører, raket operationsassistent og administrations assistent. Disse år har givet ham en del erhvervserfaring og dette kan være til stor fordel, der kombineret med hans altid gode humør, kan hjælpe gruppen i mange situationer. Et andet plus er hans erfaringer i Access, HTML, JavaScript samt grafisk layout. Specialefag: Objektorienteret systemudvikling og systemadministration.

Stamdata

Morten Selmer
Rolighedsstræde 16, st. tv.
4300 Holbæk
Tlf.: 5944 6071
Email: helio@post8.tele.dk

Biografi, Nikolaj Norman

Nikolaj er 22 år, og har inden han blev optaget på Roskilde EDB-skole, taget Højere Handelseksamen på Holbæk Handelsskole. Desuden har han stor erfaring indenfor brugen af Access, systemudviklingsmetoder samt netværkopsætning. Specialefag: Objektorienteret systemudvikling og systemadministration.

Stamdata

Nikolaj Norman
Slotshaven 3D, 40
4300 Holbæk
Tlf.: 5943 1214



Email: norman@post10.tele.dk

1.3 Vejleder

Lek. Cand. Scien. Michael Claudius

Email: claudius@roskildebc.dk

1.4 Kontaktperson

Lars Jørgensen

Axeltorv 4

1609 København V.

ljnfb@ibm.net

1.5 Projektorganisationen

Styregruppen består primært af virksomheden og projektets vejleder, og sekundært af projektets egne deltagere.

1.6 Projektgruppen

Projektarbejdsform: Ledelsen er demokratisk, da flertallet bestemmer. Der afholdes daglige projektmøder, hvor dagens opgaver diskuteres ud fra den tidsplan der bliver udarbejdet i starten af projektforsløbet. I beslutningsprocessen er det flertallet der bestemmer, men så vidt det er muligt undgås ændringer i projektmålene.

Roller: Da vi har valgt at køre projektet demokratisk, har vi samtidig valgt at lade rollerne i projektet være lidt flydende. Vi er fuldt ud klar over, hvad konsekvenserne kan blive ved denne beslutning, der kunne fx opstå uenighed om, hvem der skal udføre en given opgave. For at opnå en styring af projektet samt opnå optimal arbejdsfordeling ved projektmøder, gruppemøder og eksterne møder, har vi fastslået rollerne som følgende:

Sandie Pedersen:	Projektleder
Peter Theill:	Bibliotekar
Morten Selmer:	Kontaktperson, referent
Nikolaj Norman:	Sekretær

1.7 Planlægning

I starten af projektforsløbet skal der udarbejdes en overordnet tidsplan. Denne udarbejdes i forhold til den viden, som projektdeltagerne besidder ved projektets start, men kan revideres efterhånden som ny viden kommer til. Planlægningen skal indeholde referencelinier, og disse skal være beskrevet med vurderingskriterier- og procedurer for hver referencelinie.

1.8 Projektmøder

Der skal afholdes møder internt i gruppen – om morgenen en kort gennemgang af det manglende samt fordeling af opgaver, og eventuelt et eftermiddagsmøde efter behov.



Der afholdes om muligt møde én gang ugentligt med projektvejlederen. Denne skal have skriftligt materiale i god tid, hvis noget sådant skal vurderes.

1.9 Dokumentationsform

Ved projektetablering er der vedtaget en fælles standard for opsætningen af dokumenter. Denne standard skal bidrage til en højere gennemsigtighed i projektet og samtidig gøre det nemmere for projektdeltagerne at samle dokumenter. En kort gennemgang af standarden ses nedenfor.

Rapporten er opdelt i dele, som indeholder de overordnede aktiviteter såsom foranalyse, analyse, design, implementering, projektetablering m.m.

Deloverskrift	Garamond, punkt 56 med kapitæler
Deltitel	Times New Roman, punkt 24 med kapitæler
Overskrift 1	Arial fed, punkt 18 med kapitæler indrammet i sort boks
Overskrift 2	Arial fed, kursiv punkt 16
Overskrift 3	Arial fed, punkt 12
Citatblok	Garamond kursiv, punkt 12
Billedtekst	Tahoma fed, punkt 8
Normal	Garamond, punkt 12

I rapporten anvendes deloverskrift og deltitel til begyndelsen på en rapportdel. Derefter følger en ny side med delbeskrivelse. Hvert kapitel begynder med overskrift 1, som indeholder kapitelnummeret samt kapitlets titel. Overskrift 2 anvendes til nummererede afsnit som fx 1.1 og overskrift 3 er også nummererede underafsnit af formen 1.1.1. Citater markeres med en mørkegrå firkant til venstre for citatteksten. Brødteksten er en normal skrifttype, som er læsevenlig fordi den har seriffer (fødder).

Der anvendes to forskellige slags punktopstillinger: en med tal og en med firkantet punkt. Begge er rykket ind fra venstremargen.

Til programkode anvendes en fixed pitch font, nemlig Courier New, punkt 9.

Al tekst skrives med lige margen, da det gør teksten mere overskuelig.

I en sides øverste venstre hjørne angives, hvilken rapportdel man befinder sig i, og i øverste højre hjørne findes Nordisk Film Biografers isbjørne-logo. I nederste højre hjørne angives, hvilken del, man befinder sig i samt hvilken side man befinder sig på. På forsiden til hver del findes NFB-logoet som vandmærke.

1.10 Kritiske forudsætninger

At alle projektdeltagere er til rådighed i hele projektperioden

At projektvejleder er til rådighed i hele projektperioden

At virksomheden er til rådighed i hele projektperioden

At den gensidige respekt bevares internt i gruppen – såvel fagligt som socialt

At projektgruppen mangler praktisk kendskab til implementeringsværktøjerne



Hvis kritiske forudsætninger ikke opfyldes: Hvis alle fire projektdeltagere ikke er til rådighed i hele projektperioden, falder bemanningen og projektet vil derfor have svært ved at kunne gennemføres.

Hvis projektvejleder ikke er til rådighed, vil projektgruppen søge hjælp hos andre vejledere.

Hvis virksomheden ikke er til rådighed, vil projektgruppen fortsætte sit arbejde, men med en mere fiktiv virksomhed som samarbejdspartner.

Hvis den gensidige respekt ikke bevares internt i gruppen, vil projektgruppen drøfte sagen med vejlederen og prøve at finde en løsning.

Hvis projektgruppen mangler kendskab til implementeringsværktøjerne, kan det resultere i, at et enkelt gruppemedlem kommer til at programmere hele implementationen.

1.11 Beslutningsproces

Med fire projektdeltagere skal beslutninger træffes demokratisk. En god argumentation er derfor i fokus, men hvis der er tale om uenighed og komplicerede beslutninger kan vejleder inddrages, og beslutningen kan eventuelt træffes i samråd med virksomheden. Ændringer i projektmålene vil ikke blive ændret, så vidt det er muligt.

1.12 Sideeffekter

Projektgruppen ønsker at specialisere sig i den objektorienteret analyse, objektorienteret design og objektorienteret programmering. Desuden skal samarbejdet med NFB styrke gruppens indblik og erfaring med et virkeligt virksomhedssamarbejde samt hvordan man udvikler en web applikation, som kan samarbejde med en database ved hjælp af JDBC. Det er gruppens ønske at højne dette kendskab samt at tilegne sig erfaringer ved at arbejde i et projektmiljø.

2. Intern kontrakt

Den interne kontrakt skal fungere som et hjælpemiddel i projektet. Hvis der skulle opstå uenighed mellem projektdeltagerne eller hvis der skulle opstå tænkte problemer, kan kontrakten anvendes som en slags rettesnor.

Den enkelte projektdeltager forpligter sig til at bidrage med minimum 40 mandtimer pr. uge i hele projektperioden.

Projektdeltageren skal være positivt indstillet på et aktivt projektforsøb og deltage aktivt i alle projektets aktiviteter, såfremt det er muligt mht. projektdeltagerens individuelle viden.

Diskussion m.m. bør være præget af et fagligt indhold, og bør have relevans for projektet.

Den enkelte projektdeltager relaterer sin indsats i forhold til den pågældendes viden og hvis muligt også på andre områder.

Projektdeltageren skal arbejde i fællesskab for at indfri projektets mål.

I tilfælde af konflikt mellem projektdeltagerne, skal en udenfor stående konsulteres.



For at sikre gennemsigtighed i projektet, skal alle projektdeltagere arbejde ud fra et fælles udgangspunkt.

I tilfælde af sygdom hos den enkelte projektdeltager, forpligter denne sig til at give besked herom.

Alle beslutninger tages i fællesskab.

Undertegnede er enige heri.

Peter Theill

Morten Selmer

Nikolaj Norman

Sandie Pedersen

3. Ekstern kontrakt

Denne kontrakt er indgået mellem

*Nordisk Film Biografer
Axeltorv 4
1609 København V.*

og

*Peter Theill, Morten Selmer, Sandie Pedersen, Nikolaj Norman
Roskilde Handelsskole, edb-skolen*

Kontrakten er indgået i forbindelse med udarbejdelse af hovedopgaven på datamatikeruddannelsens 5. semester, hvor projektgruppen skal udvikle en prototype på et edb-system til billetbestilling over Internettet. Systemet udvikles og leveres i henhold til følgende vilkår:

Leverance

Virksomheden får ikke leveret dokumentation, der er udfærdiget i forbindelse med udviklingen af edb-systemet, og får derved ikke mulighed for at anvende det til eventuel videreudvikling.

Projektgruppen kan *på ingen måde* holdes ansvarlig for hardware og basisprogrammer, der måtte være nødvendigt for at anvende systemet.

Såfremt NFB ytrer ønske om at se hele eller dele af dokumentationen, vil der i projektgruppen blive taget stilling til dette.

Betaling

Projektgruppen udvikler edb-systemet i forbindelse med et uddannelsesprojekt og



skal derfor ikke modtage betaling for edb-systemet.

Installation

Projektgruppen er ikke ansvarlig for installation af systemet, men kan – såfremt den er villig – være behjælpelig.

Virksomheden har lov til at anvende systemet ubegrænset samt at videreudvikle det, men edb-systemet må ikke videregives eller sælges til tredjepart.

Vedligeholdelse, modernisering, ændringer og support

Projektgruppen kan ikke holdes ansvarlig for vedligeholdelse, modernisering eller senere ændringer i edb-systemet. Der gives ingen service eller support efter edb-systemets levering.



Garanti

Der gives ingen garantier for, at der leveres et fuldt funktionsdygtigt produkt. Projektgruppen kan ikke, hverken økonomisk eller juridisk, holdes ansvarlig for produktets eventuelle fejl og mangler.

Tavshedspligt

Projektgruppen erklærer hermed at være indforstået med, at materiale og information, som erhverves i forbindelse med projektarbejdet for Nordisk Film Biografer, er absolut fortroligt og ikke må oplyses eller videregives til tredjepart – undtaget herfra er dog den nødvendige kontakt til vejleder og censor i studieforløbet.

Undertegnede er enige heri.

Peter Theill

Morten Selmer

Nikolaj Norman

Sandie Pedersen

Lars Jørgensen
Nordisk Film Biografer

4. Valg af værktøjer

Vi vil kort beskrive de valg af værktøjer, som vi mener er passende for vores projekt samt nævne nogle af de alternative produkter der findes på softwaremarkedet.

4.1 Tekstbehandling

Som tidligere i andre projektsammenhænge blev Word fra Microsoft valgt uden megen diskussion, da alle i gruppen allerede har arbejdet meget med dette udbredte og velkendte produkt.

Et alternativ til Word er Word Perfect fra Corel. Dette er også et glimrende produkt, men da ingen har arbejdet med det, mente vi, det ville være spild af tid at sætte sig ind i dette produkts funktioner, genveje m.m.



4.2 Diagramtegning

Vi har valgt Visio, da der ligeledes i gruppen er et godt kendskab til produktet. Det er meget fleksibelt og giver bl.a. mulighed for, at man selv kan skabe sine egne skabelonstandarder.

Som et godt alternativ til Visio er der produktet ABC Flowchart, som bl.a. indeholder flere kendte notationsformer og derfor er nemt og hurtigt at benytte. Visio blev valgt primært pga. to ting: For det første, fordi erfaringer viser, at det er langt nemmere at indsætte Visio-tegninger i et Word-dokument og for det andet, er det hurtigere at producere figurer i Visio end i ABC Flowchart.

4.3 Projektplanlægning

Vi har valgt at bruge Excel fra Microsoft. Selvom Excels hovedformål er, at man kan lave sine egne regneark, så giver Excel også mulighed for at lave en hurtig tidsplan, der efter vores mening er nemmere at lave og ser bedre ud, end fx hvis man skulle benytte MS Project, der er et decideret projektstyringsværktøj.

4.4 Implementering af HTML

Værktøjerne vi vil benytte til at implementere hjemmesiden vil blive HomeSite fra Alliare.

I HomeSite, der er et udbredt shareware program, skal man selv programmere hjemmesiden direkte i HTML kode.

Der findes et hav af andre værktøjer, som automatisk kan generere HTML kode. Her kan nævnes Pagemill og HoTMetaL. Vi har dog valgt at benytte HomeSite, da kendskabet til dette er stort, og giver bedre kontrol over det skrevne HTML.

4.5 Grafisk layout

Her vil der blive brugt Adobe Photoshop, da gruppens kendskab også her er langt større end til andre alternativer som fx Corel Draw. Adobe Photoshop er bitmapbaseret og arbejder med layers, dvs. man lægger lag på lag, hvorimod Corel Draw, der er vectorbaseret, ikke har de samme muligheder for lag på lag opbygning. Til gengæld har den en bedre mulighed for at skalere objekter, herunder tekst.

4.6 Programmering: Java Builder 2.0

Vi har valgt at benytte J Builder [JB], selvom den er utrolig langsom (store dele er implementeret vha. Java selv) og indeholder mange uacceptable fejl og mangler. Dog giver den et godt overblik over store programmer, da programmet hjælper med til hurtigt at finde en given metode eller variabel i flere forskellige filer. Derudover indeholder JB en hjælpefil der giver overblik over både Servlets og JDBC.

4.7 Programmering: Java Development Kit

Udover ovenstående Rapid Application Development [RAD] værktøj (JB) har vi også benyttet os af den rå JDK fra JavaSoft. Nogle gange, vil det være nødvendigt at ligge JB fra sig for at undgå alt for mange smarte features.



4.8 Programmering: Servlets

Servlets er en slags applet der eksekveres på en server (deraf navnet). Den kan modtage en request fra en klient og derpå generere HTML kode. En applet er et Java program der kan ses i en browser med en JVM [Java Virtual Machine]. Alternativer til servlets kan fx være ASP eller CGI.

4.9 Programmering: JDBC

Kort fortalt er JDBC en Java API, der kan eksekvere SQL sætninger. Ved hjælp af API'en er det nemt at sende SQL-sætninger til hvilken som helst relationel database.

4.10 Ansvar

Hvert gruppemedlem er selv ansvarlig for sit eget udstyr og evt. anskaffelse de nødvendige programmer.

5. Valg af metode

5.1 Metoder

Nedenfor gives en kort gennemgang af de valgte systemudviklingsmetoder.

5.1.1 Systemudviklingsmetoder

Som grundlag har vi kun to systemudviklingsmetoder at arbejde med, når vi udvikler nye systemer: struktureret og objektorienteret systemudvikling. Ikke dermed sagt at vi er hæmmet i udviklingen, da de to modeller ligger op til vidt forskellige udviklingsmetoder. Hvor den objektorienterede arbejder i objekter og klasser, er den strukturerede mere baseret på funktioner, fx gammel fysisk model og datamodellering, fx E/R diagrammer. Struktureret analyse er baseret på arbejde til store administrative opgaver som f.eks. rationalisering af arbejdsgange og bearbejdning af store, ensartede datamængder. Grunden til at denne metode bliver kaldt den gamle er, at disse systemer efterhånden allerede er opfundet, og derfor må metodens udbredning ses som stoppet.

5.1.2 Objektorienteret metode [OO]

Den objektorienterede metode er derimod i en rivende udvikling, da denne metode hovedsagelig bliver brugt i sammenhæng med case-orienterede opgaver, dvs. sagsbehandling, kommunikation og koordinering. Den slags programmer er der stadig et stort behov for, også langt ud i fremtiden. Forløberne for OO er blevet brugt ganske stift, men i virkeligheden ligger metoderne ikke op til at skulle bruges på denne måde. OO skal i denne sammenhæng både ses som en reaktion på, og en videreudvikling af, de tidligere metoder. OO er ikke specielt bedre end andre analysemetoder, når det gælder udvikling af rene værktøjssystemer (fx tekstbehandling) eller edb-systemer baseret på statiske modeller (fx databaser).



eller statistisk behandling). Det centrale anvendelsesområde for OO er udvikling af alle typer edb-systemer til administration, styring og overvågning.

5.1.3 Analysemetode

Ud fra det materiale vi har fra undervisningen samt erfaringer fra tidligere projekter vil vi vælge at følge Lars Mathiassens OOA, da gruppen har bedst kendskab til denne fremgangsmåde. Gruppen kunne have valgt at følge Coad & Yourdons OOA fremgangsmåde, men da dette ville være for tidskrævende at sætte sig ind i en ny metode, kasserede vi denne.

5.1.4 Designmetode

Vi har dog valgt at anvende Coad & Yourdons OOD metode, da vi ikke har så stor erfaring med Lars Mathiassens fremgangsmåde vedrørende OOD, og har haft gode erfaringer med Coad & Yourdon.

5.1.5 Notationsform

Gruppen vælger ikke at anvende UML notation, da vi ikke har kendskab til denne. Den notationsform som Lars Mathiassen og Coad & Yourdon benytter i henholdsvis OOA samt OOD, er meget ens og derfor vælger vi at benytte denne.

5.1.6 Risikostyring

Risikostyring er en ledelsesform, som er velegnet til et projekt, hvori der hersker stor usikkerhed. For det første mindsker den sandsynligheden for store overraskelser sent i projektforsløbet og for det andet, er det en effektiv teknik til at holde et højt tempo i projektet.

Risikostyring er ideen om at styre efter hurtigt at mindske projektets samlede risiko. Ved risiko forstås en begivenhed, som har alvorlige konsekvenser.

Risikostyring omfatter to ting: *risikoanalyse* og *risikohåndtering*. Ved risikoanalyse identificerer, beskriver og prioriterer man risici, og ved den efterfølgende risikohåndtering kan man enten beslutte, forhandle, eksperimentere eller analysere:

Beslutning

Man kan beslutte at eliminere, reducere eller acceptere en risiko.

Forhandling

Man kan forhandle med andre parter for at forskyde eller dele risiko og konsekvenser.

Eksperiment

Ved eksperiment kan man skaffe sig information og indsigt, som ikke fra starten var kendt og derved fjerne sådanne risici.

Analyse

Risici kan også fjernes gennem analyse af allerede tilvejebragt information og indsigt.



Denne form for styring finder vi aldeles anvendelig til vores projekt, og derfor vælger vi det.

5.1.7 Implementeringsmetode

Da vi skulle vælge implementeringsmetoden, var vi enige om at bruge så meget Java som muligt, da det bl.a. tilbyder stor sikkerhed, er robust og desuden er platformuafhængig – perfekt til brug på Internettet. En anden grund var, at et af gruppemedlemmerne havde stor erfaring med dette programmeringsprog. For at finde ud af hvad Java tilbyder når man skal have tilgang til en database, besøgte vi JavaSoft (en division af Sun Microsystems) på <http://www.javasoft.com/>. Vi ledte efter en Java version af ODBC, som vi ved giver mulighed for at sende standard SQL-sætninger til en hvilken som helst database med ODBC-support. Vi fandt JDBC.

Ved hjælp af JDBC er det nemt at sende SQL-sætninger til en hvilken som helst relational database.

For at skabe forbindelsen mellem den server vi skal sætte op, og klienten der skal bestille billetter, har vi valgt at bruge servlets efter et tip fra et firma, der netop beskæftiger sig med programmering af netværksbaserede produkter.

Servlets er kort fortalt moduler, der udbygger request/response-orienterede servere, fx Java-baserede servere. Servlets er Javas pendant til CGI-scripts.

5.2 Projektstyringsmodel

Da projektgruppen skulle vælge hvilken model, der skulle styres efter i dette projekt, valgte den at se på tre forskellige styringsmodeller:

- Vandfaldsmodellen
- Spiralmodellen
- Inkrementalmodellen

Med *vandfaldsmodellen* inkluderer et projekt en analyseaktivitet, en designaktivitet og en programmeringsaktivitet, og naturligvis de andre aktiviteter, som man nu måtte ønske at have med. Det vil sige, at det er en meget trinvis model, hvor man afslutter en aktivitet før man begynder på en ny, selvom noget feedback til en tidligere aktivitet er tilladt, om end i et begrænset omfang (primært rettelser af eventuelle fejl fra en tidligere aktivitet).

Med *spiralmodellen* inkluderer et projekt ikke bare analyse-, design- og implementeringsaktiviteter (samt andre aktiviteter, som måtte være til stede i et projekt), men også specifikationer, prototyping og risikostyring i hver aktivitet. Spiralmodellen er velegnet til store projekter, hvor der hersker stor usikkerhed, netop fordi den indebærer især risikohåndtering i hver aktivitet. Spiralmodellen er dog også svær at håndtere.

Inkrementalmodellen inddeler analyse, design, programmering og andre aktiviteter i små trin. Det vil sige, man udfører først lidt analyse, så lidt design, så lidt programmering og disse trin gentages indtil man har det ønskede produkt. Projektgruppen valgte vandfaldsmodellen af følgende grunde: vandfaldsmodellen inddeler de forskellige aktiviteter meget skarpt og forudsætter at én aktivitet er



forholdsvis færdiggjort før den næste påbegyndes. Dette gør inkrementalmodellen ikke. Vandfaldsmodellen blev også valgt fremfor spiralmodellen, da vandfaldsmodellen er nemmere at gå til. Projektgruppen finder spiralmodellen meget interessant, men mener ikke at projektperioden er lang nok til at give sig i kast med en så kompliceret model. Vi har dog inkluderet risikostyring, som skal løbe gennem hele projektperioden, da der hersker nogen usikkerhed – især omkring implementeringsaktiviteten.

Projektgruppen vil ikke være bleg for, at vandfaldsmodellen faktisk kan transformeres over i inkrementalmodellen. Det er en mulighed, som er åben og dette kan gøres hvis projektgruppen finder det nødvendigt.

Gruppen har valgt at styre vandfaldsmodellen ved hjælp af referencelinier, da planlægningen af aktiviteter forbedres gennem en bedre forståelse af de forventninger, der stilles til det samlede resultat til aktiviteterne, og ved at planlægge med referencelinier bibringer man samtidig kvalitetsstyring til projektet.

6. Projektplanlægning

Når man påbegynder et projekt er det vigtigt med en plan, da man i systemudvikling ikke kun designer et produkt, men også en proces. Processen er den, som leder frem til det edb-baserede system. Projektplanlægningen skal ikke tillægges ringe betydning, da den skal bruges som et fælles referencegrundlag ved diskussion og prioritering af projektarbejdet. Samtidig danner planlægningen grundlag for regulering af arbejdsformerne og for vurdering af projektets status. En god plan gør det muligt for de enkelte projektdeltagere at se, hvordan de enkelte aktiviteter bidrager til projektets totale fremdrift, og øger samtidig gennemsigtigheden i processen.

Projektplanen er opdelt i 6 faser med tilhørende referencelinier, og denne findes i appendiks I.

6.1 Kvalitetsstyring

Kvalitetsstyring er under ethvert projektforløb væsentlig. Ved at opdele projektforløbet i fx foranalyse, analyse, design og implementering m.m. defineres faselinier, hvor kvaliteten af mellemprodukterne vurderes. Disse faselinier kalder vi referencelinier. Projektgruppen anvender derfor referencelinier til kvalitetsstyring, og referencelinierne er opdelt i tre dele i overensstemmelse med bogen "Professionel Systemudvikling"². Hver referencelinie er opbygget således:

Mellemprodukter og ønskede tilstande

Hvilke mellemprodukter og hvilken ønsket starttilstand skal de være i?

Kriterier for vurdering af mellemprodukter

Hvilke kriterier skal hvert mellemprodukt vurderes ud fra, for at kunne bestemme om mellemproduktet er i den ønskede tilstand?

Procedure for vurdering af mellemprodukter

² Andersen, Niels Erik et al.: Professionel Systemudvikling.



Hvilke procedurer skal vurderingen af mellemprodukterne ske på?
Hvem skal gennemlæse, teste m.m.?

Ved at anvende denne teknik, som i tidligere projekter har vist sig at være en udbytterig teknik, kan man ved hver referencelinie nøje vurdere og bestemme om man har et produkt, som befinder sig i en ønsket tilstand. Kvalitetstyring går ikke ud på at man har noget færdigt, men at man har nået den tilstand, som man selv har fastlagt.

6.2 Referencelinie 1: Foranalyse

6.2.1 Mellemprodukter og ønskede tilstande

Problemformulering

Færdig og godkendt

Projektplan

Færdig og godkendt

Projektgrundlag

Færdig og godkendt

Foranalyse

Færdig og godkendt

6.2.2 Kriterier for vurdering af mellemprodukter

Problemformulering

Alle projektdeltagere skal være enige om denne og skal være godkendt af vejleder.

Projektplan

Projektgruppen skal være indforstået med planen og denne skal være godkendt af vejleder.

Projektgrundlag, intern og ekstern kontrakt

Projektgrundlaget skal afspejle projektgruppens fælles udgangspunkt, og projektgruppen accepterer den interne kontrakt ved at denne er underskrevet. Den eksterne kontrakt skal være underskrevet af projektgruppen og virksomheden.

Foranalyse

Alle projektdeltagere skal være enige om, at den udarbejdede dokumentation af foranalysen er fyldestgørende nok til at danne grundlag for det videre analysearbejde. Denne skal godkendes af vejleder.

6.2.3 Procedurer for vurdering af mellemprodukter



Problemformulering

Skal gennemlæses af begge projektdeltagere samt afleveres til vejleder.
Dokumentets kvalitet skal herefter diskuteres på et møde med vejleder.



Projektplan

Skal gennemlæses af projektdeltagerne samt afleveres til vejleder. Dokumentets kvalitet diskuteres herefter på et møde med vejleder.

Projektgrundlag, intern og ekstern kontrakt

Skal gennemlæses af begge projektdeltagere og det skal diskuteres, om dokumentet afspejler projektgruppens fælles udgangspunkt. Afleveres sammen med intern og ekstern kontrakt til vejleder og kvaliteten diskuteres.

Foranalyse

Skal gennemlæses af alle projektdeltagere og af vejleder. På et møde diskuteres, om foranalysen er fyldestgørende nok til at danne grundlag for det videre analysearbejde. Eventuelle ændringer diskuteres.

6.3 Referencelinie 2: Analyse

6.3.1 Mellemprodukter og ønskede tilstande

Byg model

Modellen beskriver objektsystemets klasser, objekter, strukturer og dynamik.

Fastlæg krav

Resultatet heraf er en liste af overordnede krav til edb-systemet.

6.3.2 Kriterier for vurdering af mellemprodukter

Byg model

Denne skal indeholde: (1) en klasseliste, som viser hver af de udvalgte klasser med tilhørende hændelser, (2) et strukturdiagram, som viser objektsystemets klasser og strukturelle sammenhænge og (3) adfærdsmønster og attributter for hver klasse.

Fastlæg krav

Denne skal indeholde: (1) en komplet funktionsliste vurderet med hensyn til kompleksitet og (2) en grundlæggende repræsentations- og dialogform samt en komplet liste af elementer i brugergrænsefladen med skitse af typiske elementer.



6.3.3 Procedurer for vurdering af mellemprodukter

Byg model og Fastlæg krav

Begge skal gennemlæses af alle projektdeltagere samt afleveres til vejleder. På et vejledermøde skal kvaliteten af analysedokumentet samt dets eventuelle fejl og mangler diskuteres.

6.4 Referencelinie 3: Design

6.4.1 Mellemprodukter og ønskede tilstande

PDC

En Problem Domain Component.

HIC

En Human Interaction Component.

DMC

En Data Management Component.

6.4.2 Kriterier for vurdering af mellemprodukter

PDC, HIC og DMC

Alle tre komponenter skal overholde Coad & Yourdons standard for OOD, og disse skal godkendes af vejleder.

6.4.3 Procedure for vurdering af mellemprodukter

PDC, HIC og DMC

Skal gennemlæses af alle projektdeltagere samt afleveres til vejleder. På et vejledermøde diskuteres kvaliteten af designdokumentet, og det diskuteres om designdokumentet er et fyldestgørende grundlag for den videre implementering



6.5 Referencelinie 4: Implementering

6.5.1 Mellemprodukter og ønskede tilstande

Prototype 1: Interface

Det skal være muligt at kunne forbinde webserver og database med alle de funktioner, som er fundet i analysen (horisontal implementering).

Prototype 2: Interface + GUI

Her er der tilføjet en GUI (Graphical User Interface), og dette er den fulde implementering af prototype 1.

6.5.2 Kriterier for vurdering af mellemprodukter

Prototype 1: Interface

Skal indeholde de funktioner, som er fundet i analysen.

Prototype 2: Interface + GUI

Alle prototype 1's funktioner skal være fuldt implementerede heri.

6.5.3 Procedurer for vurdering af mellemprodukter

Prototype 1: Interface

Teknisk gennemgang med NFBs databaseadministrator.

Prototype 2: Interface + GUI

Teknisk gennemgang med NFBs databaseadministrator.

6.6 Referencelinie 5: Test

6.6.1 Mellemprodukter og ønskede tilstande

Fejlliste

Denne indeholder testen fundne bugs og mangler.

Forslag til designændringer

Disse forslag kan give anledning til mindre justeringer i designdokumentet samt i implementeringen.

6.6.2 Kriterier for vurdering af mellemprodukter

Fejlliste

Fejlene opdeles i prioritet. Dem med højst prioritet fikses først og så fremdeles.

Forslag til designændringer



Forslagene kan gennemgås ved hjælp af en teknisk gennemgang. Hvis en ændring bliver for omfattende i tid, stryges den – hvis ikke, udføres den.

6.6.3 Procedure for vurdering af mellemprodukter

Fejlliste

Fejllisten vurderes af projektgruppen efter fejlenes prioritet samt omfang i tid.

Forslag til designændringer

Forslagene vurderes i samråd mellem projektdeltagerne og projektvejlederen, og ændringernes relevans diskuteres.

6.7 Referencelinie 6: Aflevering

Aflevering

Projektet samles, trykkes og afleveres d. 2. November 1998 kl. 12⁰⁰.



Indholdsfortegnelse

7. PROBLEMOMRÅDE	3
7.1 BESKRIV SITUATIONEN	3
7.1.1 Ønsker til det nye system	3
7.1.2 Krav til det nye system	3
7.2 BATOFF-KRITERIET	3
7.2.1 BATOFF-kriterie 1	4
7.2.2 Systemdefinition 1	4
7.2.3 BATOFF-kriterie 2	4
7.2.4 Systemdefinition 2	5
7.3 ENDELIGT VALG AF SYSTEMDEFINITION	5
7.4 PROBLEMFORMULERING	5
7.5 PROBLEMAFGRÆNSNING	6
7.6 RISIKOSTYRING	6
7.7 KVALITETSMÅL	6
7.7.1 Brugbarhed	7
7.7.2 Integritet	7
7.7.3 Effektivitet	7
7.7.4 Korrekthed	7
7.7.5 Vedligeholdelsesvenlighed	8
7.7.6 Testbarhed	8
7.7.7 Fleksibilitet	8
7.7.8 Genbrugbarhed	8
7.7.9 Flytbarhed	8



Objektorienteret analyse er ikke den første aktivitet i et systemudviklingsforløb. Forud for analyseaktiviteten går en del arbejde, der skal resultere i et første valg af edb-løsning, og som i denne metode³ udtrykkes i en såkaldt systemdefinition. Foranalyseaktiviteten betegner perioden fra systemudviklingen påbegyndes og til opgaven er afgrænset og identificeret i form af en systemdefinition. Det vil sige, at resultatet af foranalyseaktiviteten er en systemdefinition. I foranalyseaktiviteten i OOA findes *systemvalget*. Systemvalget er baseret på tre delaktiviteter:

- Beskriv situationen
- Skab nye ideer
- Formulering og valg af systemdefinition

Den første delaktivitet fokuserer på udfordringerne, som de fremtræder i situationen. Her forsøger man at få overblik over situationen og over forskellige relevante fortolkninger. Projektgruppen har dog truffet det valg, ikke at anvende rige billeder til at beskrive situationen. I den anden delaktivitet er målet at skabe og vurdere nye ideer til design af edb-systemet. OOA tilbyder en række teknikker, som på forskellig vis kan introducere nytænkning og kreativitet. OOA tilbyder tre teknikker:

- Forbilleder
- Metaforer
- Eksperimenter

I den tredje delaktivitet foregår formuleringen og valg af systemdefinition. Her formuleres en eller flere systemdefinitioner, som vurderes i henhold til den eksisterende situation og de undersøgte ideer til nyskabelse. Men kort sagt er formålet med systemvalget altså dels, at indgå en overordnet aftale om den ønskede edb-løsning, og dels at etablere et relevanskriterium for den objektorienterede analyse. Resultatet af systemvalget er en systemdefinition, der opfylder BATOFF-kriteriet.

7. Problemområde

For at skabe sig et overblik over NFBs ønsker til det nye system, er det nødvendigt at komme frem til en definering af problemerne med det gamle system.

7.1 *Beskriv situationen*

Siden 1991 har NFB kørt med et fælles billetbestillingssystem via telefon samt en fælles database, der centralt bliver styret fra Imperial biografen i København. Det system, som bliver benyttet nu, er et gammelt DOS-baseret system, og mangler muligheden for kommunikation til Internettet. Til gengæld kan man om det gamle system sige, at det er gennemtestet og virker. For at NFB skal kunne benytte

³ Mathiassen, Lars et al.: Objektorienteret analyse



kundebestillinger over Internettet, bliver det nødvendigt at udvikle et interface til det nuværende system.

Ideen med billetbestilling over Internettet er rimelig ny i Danmark; ca. 3 år. Blandt andet har DSB kørt med betaling over Internettet i en testperiode med stor succes. Scanlines har ligeledes haft bestilling af færgebilletter på Internettet i et stykke tid. Så ud fra de oplysninger gruppen har fundet frem til, er det en temmelig ny bestillingsmetode, som NFB kan introducere i Danmark.

7.1.1 Ønsker til det nye system

For at give biografgængerne større service samt at følge med den teknologiske udvikling på Internettet, har NFB længe ønsket at give kunden mulighed for at bestille og betale biografbilletter via Internettet. Når en kunde har bestilt og betalt en billet, skal kunden kunne afhente billetten i nogle afhentningsautomater, der er opstillet rundt omkring. På denne måde reduceres de lange køer ved billetlugerne i biografene. Desuden kan kunden bestille sin billet samt indhente information om filmen, når det lige præcis passer kunden.

Der er også et ønske om, at EONs hjemmeside skal fremstå som et forbillede for gruppens færdigdesignede side.

7.1.2 Krav til det nye system

Kunden skal ikke have mulighed for at se, hvor mange sæder der allerede er solgt til den pågældende film. Dette krav er stillet af NFB, da man ikke ønsker at en konkurrerende biograf får muligheden for at se hvor mange billetter der er solgt til en given film. NFB har dog givet udtryk for, at dette krav kan fraviges, hvis det ikke tidsmæssigt er muligt at nå.

Ydermere er der et krav om, at der ikke må foretages mere end syv billetter pr. bestilling.

7.2 BATOFF-kriteriet

BATOFF-kriteriet er et vurderingskriterie, som kan anvendes til at vurdere, om en systemdefinition er velformuleret og klar. Kriteriet fortæller *ikke* om en systemdefinition er nyttig eller ej, men fortæller blot om systemdefinitionen indeholder de relevante elementer.

Hver af de seks bogstaver står for et element, BATOFF-kriteriet består af følgende 6 elementer:

Betingelser: De betingelser, som systemets videre udvikling og brug er underlagt.

Anvendelsesområde: En eller flere organisatoriske enheders administration, styring eller overvågning af et problemområde.

Teknologi: Den teknologi, som systemet udvikles til og ved hjælp af.

Objektsystem: En kommende brugers opfattelse af et problemområde.

Funktionalitet: De hovedfunktioner, som systemet tilbyder til understøttelse af arbejdsopgaver i anvendelsesområdet.

Filosofi: Den filosofi, der ligger bag edb-systemet.



7.2.1 BATOFF-kriterie 1

Betingelser: Brugere af systemet er individuelle biografgængere, som anvender hjemmesiden til bestilling af biografbillet over Internettet. Billetten skal ikke betales over nettet, men ved afhentning i afhentningsautomater.

Anvendelsesområde: Systemet skal håndtere billetbestilling til Nordisk Film Biografer via Internettet.

Teknologi: Hjemmesiden skal via en form for interface, arbejde som et supplement til den database, som Nordisk Film Biografer anvender. Dette interface kunne fx udvikles vha. Java.

Objektsystem: Kunde, biograf, rækker, sal, sæde, film, afhentningsautomat, billetreservation, forestilling, pristype, kategori, medvirkende.

Funktionalitet: Støtte til biografgængere til bestilling samt afhentning af billetter ved betaling i specielle afhentningsautomater.

Filosofi: Primært et kommunikationsværktøj mellem biografgængere og Nordisk Film Biografer.

7.2.2 Systemdefinition 1

Et edb-system, der anvendes i forbindelse med bestilling af biografbilletter over Internettet. Systemet skal fungere som et kommunikationsmedium mellem biografgængere og Nordisk Film Biografer. Det skal være muligt, efter endt bestilling, at betale og afhente sine billetter i specielt opstillede afhentningsautomater i eller omkring biografen. Dette gøres for at undgå bestillingskøen og personlig betjening. Bestilling af billetter foregår på Nordisk Film Biografers hjemmeside. Der skal ikke tages specielt hensyn til brugernes varierende edb-erfaring, da man må påregne visse edb-færdigheder ved brug af Internettet, men hjemmesiden og bestillingen skal alligevel være forholdsvis nem at gå til. Systemet skal ligge på en webserver, som skal være platformsuafhængig og skal kunne fungere uanset hvilken DBMS, man benytter.

7.2.3 BATOFF-kriterie 2

Betingelser: Brugere af systemet er individuelle biografgængere, som anvender hjemmesiden til bestilling af biografbillet over Internettet.

Anvendelsesområde: Systemet skal håndtere billetbestilling til Nordisk Film Biografer via Internettet.

Teknologi: Hjemmesiden skal via et interface arbejde som et supplement til den database, som Nordisk Film Biografer anvender. Systemet udvikles ved hjælp af fx Java.

Objektsystem: Kunder, biograf, sal, sæde, rækker, film, billetreservation, forestilling, pristype, kategori, medvirkende.

Funktionalitet: Støtte til biografgængere til bestilling af billetter.

Filosofi: Primært et kommunikationsværktøj mellem biografgængere og Nordisk Film Biografer.



7.2.4 Systemdefinition 2

Et edb-system, der anvendes i forbindelse med bestilling af biografbilletter over Internettet. Systemet skal fungere som et kommunikationsmedium mellem biografgængere og Nordisk Film Biografer. Det er muligt at bestille biografbilletter, men afhentning og evt. betaling af billetter skal ske ved personlig betjening i den givne biograf. Der skal ikke tages specielt hensyn til brugernes varierende edb-erfaring, da man må påregne visse edb-færdigheder ved brug af Internettet. Hjemmesiden og bestillingen skal alligevel være forholdsvis nem at gå til. Systemet skal ligge på en webserver, som skal være platformsuafhængig og skal kunne fungere uanset hvilken DBMS, man benytter.

7.3 Endeligt valg af systemdefinition

For at kunne komme videre med den egentlige analyse, skal der vælges en af de to systemdefinitioner. Vi har valgt systemdefinition 2, da vi ikke har ressourcer til at kunne leve op til systemdefinition 1. Ved valg af systemdefinition 1, skal der indgås samarbejde med et automatbygningsfirma, der er specialister inden for betalingsautomater. Et sådant samarbejde ligger langt uden for de rammer, som gruppen har sat. Gruppen mener dog, at ved at vælge systemdefinition 2 giver vi NFB det rette grundlag for bestilling via Internettet. Ydermere vil det egne sig til at blive udvidet med den fysiske afhentningsautomat, hvor kunderne skal betale og hente de bestilte billetter.

7.4 Problemformulering

For at benytte det nuværende system er det nødvendigt, at udvikle en "skal" til dette. Derved bliver det muligt at lave et bestillingssystem til Internettet, uden at lave om på det gamle velfungerende system. Denne overbygning på det nuværende system udarbejdes for, at undgå at skulle konstruere et helt nyt system, og de konsekvenser der følger heraf. Denne beslutning medfører nogle spørgsmål og overvejelser:

Er det muligt at udvikle et interface, der kan overbygge det nuværende system og database?

Er det muligt at simulere Internet-trafik i et lokalt testmiljø (under LAN)?

Er den valgte styringsmetode anvendelig til dette projekt?

Er den objektorienterede metode anvendelig på vores problemområde?

Er det muligt at implementere systemet udelukkende ved brug af Java?

7.5 Problemafgrænsning

Projektgruppen har valgt at afgrænse sig fra at lave et billetbestillingssystem, der giver muligheden for, at man kan betale sine billetter via Internettet eller ved afhentning i specielle automater. Systemet tillader kun bestilling af billetter. Projektgruppen afgrænser sig ligeledes fra at teste systemet på browsere, der ikke understøtter JDK 1.1. Grunden til dette er, at JDK 1.1 direkte understøtter JDBC og derfor ikke kræver eksterne moduler for at forbinde til en database.



7.6 Risikostyring

Projektgruppen har anerkendt, at der findes visse risici i dette projekt. For at få klarlagt disse risici har gruppen anvendt brainstorm, og har udarbejdet en prioriteret risiciliste, som også angiver hvad der bør gøres i en given situation.

Prioritet	Risiko	Håndteringsteknik
1	Fejlvurdering af projektets omfang	Afgrænsning
1	Bemanding	Afgrænsning
2	Manglende rutine i anvendelsen af udviklingsværktøj	Der skal bruges mere tid på at arbejde med værktøjet
2	Dokumentation	Rapporten skrives fortløbende
3	NFBs database kan ikke arbejde sammen med JDBC	Projektgruppen udvikler selv en database i Access

Tabel 7.1

Da man ofte i forbindelse med test konfronteres med resultatet af alle analyse- og designresultaterne, har projektgruppen valgt at anvende risikostyring, da det er bedre at forebygge risici end at komme i en brandslukningssituation senere i projektføreløbet.

For yderligere information om, hvad risikostyring er, se punkt 5.1.6.

7.7 Kvalitetsmål

Gruppen har opstillet nogle overordnede kriterier for kvalitetsmålene med det udviklede system. Gruppen har etableret disse mål ud fra to kilder⁴, hvori man opstiller forskellige overordnede krav til systemet både nu, men også fremover. Vi har valgt at tage de punkter med, vi mener, der skal gøres overvejelser omkring. I dette afsnit vil de opstillede kriterier så blive vurderet på deres relevans i vores system, både af gruppen, men også af de oplysninger vi fik fra NFB.

Kriterier	Meget Vigtigt	Vigtigt	Irrelevant
Brugervenlighed	X		
Integritet	X		
Effektivitet		X	
Korrekthed	X		
Vedligeholdelsesvenlighed		X	
Testbarhed	X		
Fleksibilitet			X
Genbrugbarhed		X	
Flytbarhed			X

Tabel 7.2

⁴ McCall et al. 1977 samt Lars Mathiassen et al.: Objekt orienteret analyse og design



Kvalitetsmålene vil nu blive beskrevet mere detaljeret for at få et indblik i hvilke krav NFB samt projektgruppen har til systemet.

7.7.1 Brugervenlighed

Et vigtigt krav fra NFB til systemet er, at det færdigudviklede system skal være nemt at gå til. Man skal som bruger ikke have de store tekniske overvejelser under bestillingen af billetter gennem systemet. Samtidigt skal det være logisk bygget op, så det ligner resten af EONs hjemmeside, så man ikke som bruger bliver forvirret ved brugen af bestillingssystemet.

7.7.2 Integritet

Da det system vi skal udvikle, skal håndtere billetbestillinger fra Internettet, er det uhyre vigtigt at der er en høj integritet. Det er vigtigt, at når en bruger vælger nogle sæder til bestilling, at de rent faktisk også bliver reserveret af systemet. Hvis dette kvalitetsmål ikke er opretholdt, vil det skabe et dårligt ry for NFB, og systemet vil simpelthen være ubrugeligt. Derfor er det nødvendigt, at der er 100% integritet i systemet.

7.7.3 Effektivitet

Der er ingen krav fra NFB, til effektiviteten af systemet. Systemet skal naturligvis fungere, men overvejelser omkring hastighed af hjemmesiderne er ikke et kvalitetsmål for NFB. Det er mere vigtigt for NFB, at systemet angiver korrekte informationer. Gruppen anser det dog som et kvalitetsmål at systemet har en forholdsvis høj hastighed, da vi anser dette som et godt kvalitetsmål for den videre brug af billetbestilling over Internettet.

7.7.4 Korrekthed

Kravene fra NFB er, at systemet skal fungere korrekt, både mht. visning af de rigtige film til de rigtige tider og sale samt udregning af den rigtige pris, da dette ville være fatalt for NFB hvis ikke kunden kunne stole på systemet. Samtidigt skal systemet virke troværdigt, så brugeren ikke finder systemet usikkert og derved ikke ville bruge det. Ved at opbygge en troværdighed på deres hjemmeside, vil det heller ikke i fremtiden blive noget problem i at indbygge en funktion, så brugeren kan benytte online betaling.

7.7.5 Vedligeholdelsesvenlighed

Idet dette system bliver en udbygning på et nuværende system og hjemmeside, er der ikke de store krav fra NFB om, hvor vidt vedligeholdelsen skal være nem. Systemet skal udvikles på en sådan måde at opdatering samt vedligeholdelse kræver minimal brug af tid. Gruppen vurderer ikke dette som et større problem, da databasens funktion netop vil ligge op til, at nye data bliver indtastet når det er nødvendigt. Desuden vil gruppen benytte hjemmesider, udviklet i HTML. Da dette ikke er synderligt svært at arbejde med, anser gruppen ikke dette som hindring for vedligeholdelsen.



7.7.6 Testbarhed

NFB ligger stor vægt på, at de nye systemer som de evt. skal benytte, er ekstremt gennemtestede. Det er et utroligt vigtigt mål for dem at benytte systemer der fungerer. NFB havde ikke som mål at benytte helt nyudviklet systemer, da disse tit viser sig at have små fejl som kan have store økonomiske konsekvenser for NFB. Derfor vil gruppen prøve at teste systemet så grundigt som muligt for fejl, og udvikle en detaljeret testrapport som går i dybden med systemet, for at påvise evt. fejl. Dette vil gøres under accepttesten, ved brug af en black-box test der viser systemets funktioner, en brugertest der viser brugervenligheden samt andre testemner såsom stresstest og volumetest til visning af systemets stabilitet.

7.7.7 Fleksibilitet

Gruppen mener ikke at fleksibilitet er vigtig i denne sammenhæng. Hvis NFB ønsker at udbygge systemet, bliver dette nemt. Men vi anser det ikke for sandsynligt at NFB skal ændre i det udviklet system som tilpasning til andre systemer, da dette er en udbygning af deres nuværende system.

7.7.8 Genbrugbarhed

Som udgangspunkt er genbrug altid godt. Derfor ønsker vi også at have for mål at systemet er udviklet med henblik på genbrugbarhed.

7.7.9 Flytbarhed

Da systemet er et Internet baseret system, anses flytbarhed ikke som et kvalitetsmål for systemet. Projektgruppen vurderer at systemet skal ligge placeret centralt på en server, og mulighed for flytning vil ske sammen med denne. Vi vil ikke tage højde for mulighederne forbundet med dette.



Indholdsfortegnelse

8. BYG MODEL	3
8.1 FORMÅL	3
8.2 BATOFF-KRITERIE OG SYSTEMDEFINITION	3
8.3 BESKRIVELSE AF BRAINSTORM	3
8.4 BESKRIVELSE AF DATA FLOW DIAGRAMMER [DFD]	4
8.5 FUNDNE KLASSE	4
8.6 KLASSEBESKRIVELSE	5
8.7 HÆNDELSES BESKRIVELSE	7
8.8 VURDERING AF STRUKTUR	8
8.8.1 Klyngestruktur	9
8.8.2 Generalisering/specialisering	9
8.8.3 Aggregering	10
8.8.4 Associering	10
8.8.5 Den dynamiske rollemodel	10
8.9 DYNAMIK	11
8.9.1 Beskrivelse af adfærdsdiagram for klassen 'Medvirkende'	11
8.9.2 Beskrivelse af adfærdsdiagram for klassen 'Pristype'	12
8.9.3 Beskrivelse af adfærdsdiagram for klassen 'Forestilling'	13
9. FASTLÆG KRAV	14
9.1 FUNKTIONER	14
9.1.1 Funktionsliste	14
9.1.2 Funktionsbeskrivelse	14
9.2 GRÆNSEFLADER	15
9.2.1 Brugsmiljøet	16
9.2.2 Brugssituationer	16
9.2.3 Dialogformer	16
9.2.4 Udformning af brugergrænsefladen	17
9.2.5 Produktets fremtræden	17
9.3 DEN TEKNISKE PLATFORM	19
9.4 EDB-SYSTEMETS NYTTE OG REALISERBARHED	20
9.5 STRATEGI FOR DET VIDERE UDVIKLINGSARBEJDE	20



Når man anvender Lars Mathiassens objektorienterede analysemetode, er der to overordnede aktiviteter, som skal gennemføres:

Byg model
Fastlæg krav

I disse to overordnede aktiviteter findes en masse delaktiviteter. I modelbygningsaktiviteten findes klasserne, objekterne og hændelserne, og når disse er vurderet skal der findes strukturer. Der skal findes klassestrukturer såsom generaliserings- og klyngestrukturer samt objektstrukturer såsom aggregeringer og associeringer. I den sidste aktivitet i modelbygningsaktiviteten skal objektsystemets dynamik fastlægges. Resultatet af modelbygningsaktiviteten skal være en sammenhængende model af et objektsystem.

I aktiviteten Fastlæg Krav skal man dels fastlægge edb-systemets indholdsmæssige brugsegenskaber og dels opstille kravene til edb-systemets grænseflader.

Resultatet af aktiviteten skal være en komplet liste af overordnede krav til edb-systemet.

Det samlede resultat af disse to overordnede aktiviteter er et analysedokument, som findes i denne del. Det skal bemærkes, at analyseaktiviteten ikke foregår sekventielt, som ovenstående måske kunne antyde. Analysen er en iterativ proces, hvor man ofte må gå tilbage og revurdere resultaterne af de enkelte delaktiviteter.

8. Byg model

8.1 Formål

Nordisk Film Biografer [NFB] er et søsterselskab til Nordisk Film, som ejes af Egmont-koncernen. Virksomheden har bestilt et edb-system, som skal håndtere bestilling af biografbilletter over Internettet.

Edb-systemet skal udvikles som en skræddersyet løsning til NFB. Der skal udvikles en hjemmeside, som via JDBC skal arbejde som supplement til den database, NFB anvender.

Edb-systemet skal koncentreres om billetbestilling til NFBs biografer. Det skal ikke anvendes i forbindelse med billetbetaling på Internettet.

En mere teoretisk gennemgang af OOAs begrebsdefinitioner findes i appendiks A.

8.2 BATOFF-kriterie og systemdefinition

Betingelser: Brugere af systemet er individuelle biografgængere, som anvender hjemmesiden til bestilling af biografbillet over Internettet.

Anvendelsesområde: Systemet skal håndtere billetbestilling til Nordisk Film Biografer via Internettet.

Teknologi: Hjemmesiden skal via JDBC arbejde som et supplement til den database, som Nordisk Film Biografer anvender. Systemet udvikles ved hjælp af Java.

Objektsystem: Kunder, biograf, sal, pladser, film, billet.



Funktionalitet: Støtte til biografgængere til bestilling af billetter.

Filosofi: Primært et kommunikationsværktøj mellem biografgængere og Nordisk Film Biografer.

Et edb-system, der anvendes i forbindelse med bestilling af biografbilletter over Internettet. Systemet skal fungere som et kommunikationsmedium mellem biografgængere og Nordisk Film Biografer. Det er muligt at bestille biografbilletter, men afhentning og betaling af billetter skal ske ved personlig betjening i den givne biograf. Der skal ikke tages specielt hensyn til brugernes varierende edb-erfaring, da man må påregne visse edb-færdigheder ved brug af Internettet, men hjemmesiden og bestillingen skal alligevel være forholdsvis nem at gå til. Systemet skal ligge på en webserver, som skal være platformsuafhængig og skal kunne fungere uanset hvilken DBMS, man benytter.

8.3 Beskrivelse af brainstorm

Gruppen foretog en brainstorm over hændelser der sker, når man bestiller en billet. Denne brainstorm blev brugt af gruppen, for at finde hændelser og klasser i systemet. Samtidig er dette en udmærket fremgangsmåde til at få hul på analysen, da det er en fortrinlig måde at, skabe sig et overblik over systemet på. Vi udførte en mere systematisk brainstorm som tog udgangspunkt i, at en kunde bestilte en biografbillet og de arbejdsgange der fulgte med Resultatet af brainstormen findes i appendiks B.

8.4 Beskrivelse af Data Flow Diagrammer [DFD]

For at skabe sig et yderligere overblik over systemet, inden gruppen går over til OOD, har vi valgt at vise systemet med et kontekstdiagram, et oversigtsdiagram samt et niveau-et-diagram for processen *Administrere Billetbestilling*. Dette sker for at fastslå, at gruppen har fundet alle hændelserne til systemet. Desuden mente gruppen, at DFD giver god indsigt i, hvordan systemet skal fungere, hvilke processer der er i systemet samt hvilket informationsflow, der er i systemet. Diagrammerne findes i appendiks B.

Diagrammerne blev brugt som arbejdsmodeller, og vil ikke blive beskrevet yderligere. Desuden tilhører disse modeller Struktureret Analyse, og da gruppen benytter Objektorienteret Analyse, vil gruppen ikke gå dybere ind i denne notationsform.

Ønskes der et dyberegående teoretisk kendskab, henvises til bogen "Struktureret Analyse"¹.

8.5 Fundne klasser

For at finde så mange klasser og hændelser som overhovedet muligt, foretog gruppen en brainstorming. Dette gøres for hurtigt at finde en stor samling klasser og hændelser, hvorfra det er muligt at udvælge dem, der er nødvendige for vores system.. Denne brainstorm kan ses i appendiks B.

¹ Delskov, Lise & Lange, Therese: Struktureret Analyse.



Her ses resultatet af brainstormen efter endt vurdering af de klasser der blev fundet.

Billettereservation

Hændelser: Opret, slet, rediger, check forestilling, check dato, check tider, indhent kundedata, returner information.

Attributter: Film, sal, række, sæder, tid, dato, pris.

Biograf

Hændelser: Opret, slet, rediger.

Attributter: Navn, adresse, postnr, by, faxnr, tlf.nr., sal antal.

Film

Hændelser: Opret, slet, rediger.

Attributter: Titel, årgang, spilletid, skuespillere, beskrivelse, filmkategori, instruktør.

Filmkategori

Hændelser: Opret, slet, rediger.

Attributter: Kategorinavn.

Forestilling

Hændelser: Opret, slet, rediger.

Attributter: Filmtitel, salnr., dato, tid.

Kunde

Hændelser: Reserver billet, afbestil billet, rediger billet, opret, redigér, slet.

Attributter: Navn, adresse, postnr., by, tlf.nr., email, password.

Medvirkende

Hændelser: Opret, slet, rediger.

Attributter: Navn, nationalitet, andre info.

Pristype

Hændelser: Opret, slet, rediger.

Attributter: Dyr, medium, billig.

Rækker

Hændelser: Check pristype.

Attributter: Nr, salnr, pristype.

Sal

Hændelser: Opret, slet, rediger.

Attributter: Nr, rækkenr, sædenr.

Sæde

Hændelser: Reserver, afbestil.

Attributter: Nr, rækkenr, salnr.



8.6 Klassebeskrivelse

I det følgende vil vi give en kort beskrivelse af de klasser, der er fundet.

Billettereservation

Denne klasse skal indeholde alle informationer der hører til en reservation. Dette være sig den pågældende films titel, hvilken sal den bliver fremvist i, række og sædenummer samt hvilken dato og tidspunkt, reservationen er til. Billetprisen skal også fremgå. Ved oprettelse af en reservation, bliver der oprettet en associering til en bestemt forestilling.

Biograf

Klassen indeholder kun oplysninger om én given biograf. Disse oplysninger består af stamdata, såsom adresse og telefonnummer.

Film

Her ligger forskellige informationer som fx titel, årgang, spilletid og beskrivelse. Når en film bliver oprettet, bliver der oprettet en associering til klassen

Forestilling.

Filmkategori

Denne klasse er simpel, idet den kun indeholder de forskellige filmkategorier, såsom action, komedie, drama, gyser og thriller.

Forestilling

Denne klasse binder **Film** og **Sal** sammen. Klassen er den centrale i en billetbestilling, da den indeholder filmens titel, hvilken sal den går i, hvilke datoer og tider den spilles på.

Kunde

Denne klasse skal bruges til at registrere kundens telefonnummer. Dette gøres for, at kunden blot skal oplyse sit telefonnummer ved afhentningen af de reserverede billetter. Der kunne dog i fremtiden let vælges at registrere kunden vha. fx email. Kundens telefonnummer er valgt, da det er lettere at huske frem for et maskinbestemt reservationsnummer.

Gruppen mener også, at det kunne være interessant at registrere kundens postnr., da man på den måde kunne lave en markedsanalyse over kunderne og hvor i landet de kommer fra.

Da vi har tænkt os at lave en loginprocedure, som skal give kunden den endelige adgang til at kunne reservere et antal sæder, behøver man der derfor også et password, som også skal gemmes.

Medvirkende

Denne klasse bestod i virkeligheden af tre klasser, nemlig *skuespiller*, *instruktør* og *forfatter*, men da disse havde fælles hændelser og fælles attributter, besluttede projektgruppen først at lægge disse klasser sammen under et. Dette har vi ikke gjort, da man derved ikke kunne modellere, at fx en instruktør kan



spille med i sin egen film. Derfor har man valgt en dynamisk rollemodel, men problemet kunne formentlig også være løst vha. multipel arv.

Pristype

I denne klasse ligger de tre priskategorier, der ud fra rækkens nummer bestemmer billetens pris. De tre pris kategorier er *Dyr*, *Medium* og *Billig*. Disse priser er bestemt ud fra rækkenummeret i en bestemt sal til en bestemt film.

Rækker

Da denne klasse bestemmer prisen på et sæde, ud fra klassen **Pristype**, er denne klasse rimelig essentiel for systemet. Klassen hænger sammen med klassen **Sal**, der derved beskriver hvor mange rækker der findes i den givne sal.

Sal

I en given biograf kan der være mange sale. I klassen **Sal** er der et id-nr, der bestemmer hvilken sal der er tale om. Dette bestemmer så igen hvilken pris, billetten skal have.

Sæde

Sammen med klassen **Rækker** er denne klasse vigtig i en billetbestilling. Som kunde bestemmer man ikke selv hvilke sæder man vil reservere, men man kan dog godt give et klart udtryk for, hvilken række man vil sidde på. Det er også vigtigt, at klassen skal kunne returnere hvilke sædenummer som er blevet valgt.

8.7 Hændelsesbeskrivelse

Vi vil i det følgende kort beskrive hvilke hændelser de forskellige klasser har.

Billetreservation:

Her er igen de klassiske *opret*, *slet*, *rediger*, men også de mere forespørgselsprægede *check forestilling*, *check dato*, *check tider*, *indhent kundedata*, *returner information*, som alle bliver kørt af systemet selv. Disse hændelser sker, når en kunde henvender sig for at bestille en billet. Operatøren skal i denne forbindelse sætte systemet i gang med at udføre disse hændelser, for at kunne servicere kunden på bedste vis.

Film

Opret, *slet*, *rediger*. Naturligvis er det nødvendigt at kunne oprette og slette en film, da der hele tiden kommer nye film til biografen og gamle film skal slettes fra systemet.

Filmkategori

Opret, *rediger*. Det er nødvendigt at oprette en filmkategori, da der sommetider kommer nye genrer på filmmarkedet.

Forestilling



Opret, slet, rediger. Naturligvis skal man kunne oprette en forestilling, bl.a. når en ny film udkommer, og derefter slette den senere, når filmen ikke længere spilles. Samtidig skal man kunne redigere i en forestilling, fx hvor og hvornår filmen spilles.

Kunde

Her er de tre klassiske hændelser *opret, slet og redigér* som udgør en livscyklus for et objekt af denne klasse, her dog med lidt andre navne. *Reserver billet* sker, når kunden henvender sig for at bestille en billet. *Afbestil billet* forekommer ikke så tit, men forekommer hvis en kunde melder afbud. Normalt giver kunderne ikke besked om en afbestilling, så derfor bliver systemet nødt til automatisk at slette forældede reservationer. Sidste hændelse er *redigér billet*, som forekommer, hvis en kunde ønsker at ændre sin bestilling. Denne hændelse forekommer for det meste i form af, at man ønsker at reservere et sæde mere eller mindre.

Medvirkende

Opret, rediger. Disse hændelser sker når en film skal indtastes i systemet. Herved oprettes også medvirkende i filmen som allerede ikke er i systemet, fx skuespillere, instruktører og forfattere.

Pristype

Opret, slet, rediger. Da pristyperne kan ændre sig pga. filmenes karakter (fx premierefilm) eller større driftsomkostninger, er det nødvendigt at kunne redigere pristyperne.

Række

Check pristype. Denne hændelse sker når kunderne har bestemt hvilken film og evt. række, de vil sidde på.

Sæde

Reserveret, afbestilt. Når man reserverer en billet, reserverer man et sæde i en række. Hvis man ikke kan komme alligevel, kan man afbestille det reserverede sæde.

8.8 Vurdering af struktur

Efter udarbejdelsen af strukturdiagrammet blev gruppen klar over, at der manglede nogle klasser og at én bestemt klasse (**Medvirkende**) kunne "deles ud". Disse rettelser er tilføjet og gruppen vil i det følgende omtale disse.

8.8.1 Klyngestruktur

Gruppen har valgt ikke at koncentrere os så meget om klyngestrukturer. Dels fordi projektet ikke er stort nok, og dels fordi gruppen mener, at strukturdiagrammet er overskueligt nok. Men naturligvis er der klynger, hvilket gruppen er helt klart over. Vi har i gruppen diskuteret meget om der i virkeligheden ikke var tre klynger og ikke kun to. De to forskellige forslag så sådan ud:



Forslag 1: En struktur med to klynger:

Forestilling består af: *Forestilling, Kunde, Billetreservation, Biograf, Sal, Række, Pristype* og *Sæde*.

Film består af: *Film, Kategori* og en dynamisk rollemodel *Medvirkende*.

Forslag 2: En struktur med tre klynger

Reservation: *Forestilling, Kunde, Billetreservation, Pristype*.

Biograf: *Biograf, Sal, Række* og *Sæde*.

Film: *Film, Kategori* og en dynamisk rollemodel *Medvirkende*.

Gruppen har valgt modellen med to klynger, da det virker mere logisk at opdele strukturdiagrammet på denne måde.

I forslag 2 mente vi, at det var meget naturligt at oprette klyngen **Biograf**, da den indeholdt de klasser, som en biograf består af. Klyngen **Reservation** var ligeledes også naturlig, da vi i første omgang mente at de fire indeholdende klasser hørte sammen i forbindelse med en billetreservation.

Da alle klasserne har en logisk sammenhæng til klassen **Film**, giver klyngen **Film** sig selv. Derfor fremgår den også i begge forslag.

Vi har alligevel valgt at benytte forslag 1, da klasserne **Rækker** og **Pristype** hænger meget sammen og derfor vil det ikke være logisk, at disse to klasser skulle findes i to forskellige klynger.

8.8.2 Generalisering/specialisering

Der findes kun en enkelt generalisering/specialisering i strukturdiagrammet.

Generaliseringsklassen er **Pristype** og denne har tre specialiseringer: *Dyr, Medium* og *Billig*. Det eneste som ligger i de specialiserede klasser er en pris, som afhænger af en række, som igen afhænger af en sal. Grunden til denne afhængighed er, at forskellige sale kan have et forskelligt antal rækker, og derfor kan have et varierende antal Dyr-, Medium- og Billigsæder.

Hvis denne afhængighed ikke var så stærk, fx hvis der kun var én sal i biografen, kunne man have lagt priserne op i generaliseringsklassen som attributter og derved have undgået en generaliserings/specialiserings-struktur.

8.8.3 Aggregering

På strukturdiagrammet findes tre aggregeringsstrukturer. Aggregeringen mellem **Sal** og **Rækker** er tydelig, da en sal består af rækker. Dog er det forskelligt fra sal til sal, hvor mange rækker der er.

Derudover er der en aggregering mellem **Rækker** og **Sæde**, da en række består af sæder. Disse aggregeringer giver tydeligt udtryk for, hvorfor de er der.

I starten var der en aggregering mellem **Forestilling** og **Sæde**, da et sæde kan reserveres til forskellige forestillinger. Men det gav ikke mening at sige, at en forestilling består af sæder. Derfor valgte gruppen i stedet at anvende en associering mellem **Billetreservation** og **Forestilling**, og slette aggregeringen mellem **Forestilling** og **Sæde**. Denne associeringsstruktur beskrives i det følgende om associeringer.



8.8.4 Associering

På strukturdiagrammet findes syv associeringer, men kun de fire af dem er interessante.

Gruppen har valgt at forklare to af associeringerne samtidig, nemlig associeringen mellem **Forestilling** og **Sal**, og mellem **Forestilling** og **Film**.

En forestilling består af oplysninger om en films fremvisningstidspunkter- og datoer samt hvilken sal, filmen fremvises i.

Vi har gjort dette, da en film altid er associeret med en forestilling, og en forestilling altid er associeret med en sal. Derfor har gruppen valgt ikke at associere en film direkte med en sal.

En **Pristype** er associeret med en **Forestilling**, da en pristype bestemmes ud fra en sals rækker samt ud fra hvilken film, der indgår i en forestilling (fx om det er en premierefilm).

En **Forestilling** er associeret med **Billetreservation**, da en billet altid reserveres til en bestemt film på et bestemt tidspunkt, og da disse informationer ligger i **Forestilling** er det naturligt, at knytte en forestilling og en billetreservation sammen på denne måde.

Gruppen kan nu konstatere, at strukturdiagrammet viser objektsystemets klasser og strukturelle sammenhænge på bedste vis. Diagrammet findes i appendiks B.

8.8.5 Den dynamiske rollemodel

Til at starte med var der ingen dynamisk rollemodel. Gruppen overvejede først om der burde laves en generalisering med hensyn til klassen **Medvirkende**. Denne klasse kunne specialiseres til henholdsvis **Skuespiller**, **Instruktør** og **Forfatter**. Men der ville kun være en enkelt attribut til forskel, nemlig den medvirkendes titel, så denne idé blev forkastet. Dels på grund af den ene attribut og dels på grund af, at fx en instruktør kan spille med i en af sine egne film. Dette problem kunne løses ved at anvende multipel arv, men også ved hjælp af en dynamisk rollemodel. Den dynamiske rollemodel blev derfor valgt, da bl.a. Java ikke direkte understøtter multipel arv.

8.9 Dynamik

I denne aktivitet skal der fokuseres på objektsystemets dynamik. Hensigten med dynamikaktiviteten er at karakterisere de enkelte objekter i objektsystemet gennem en analyse af deres adfærd og dataegenskaber.

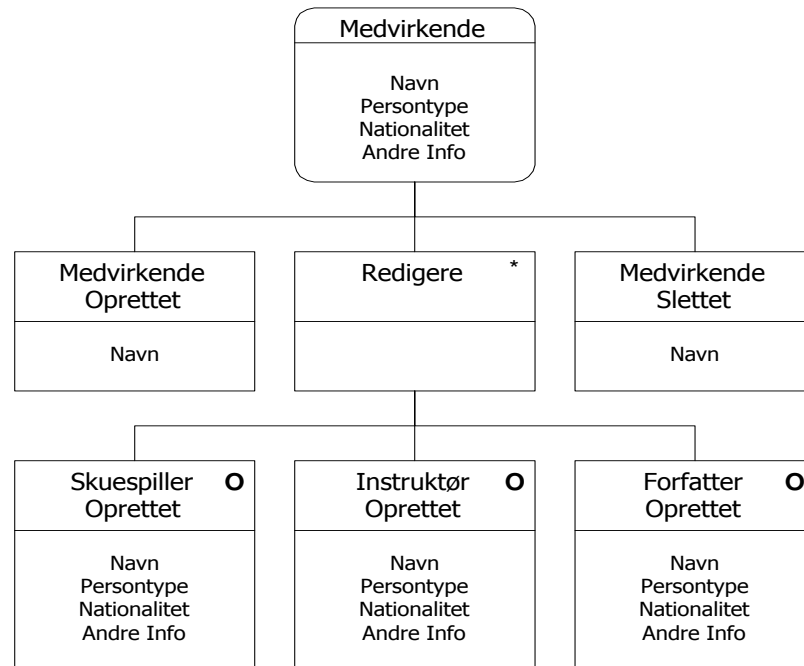
Gruppen har før arbejdet med dynamikaktiviteten, men har fra tidligere projekter erfaret, at disse ikke hjalp på nogen konstruktiv måde og faktisk slet ikke blev anvendt. Derfor har gruppen valgt ikke at koncentrere sig om dynamikaktiviteten, men for at vise at den behersker adfærdsdiagrammeringsteknikken, har gruppen udarbejdet tre adfærdsdiagrammer, som kan ses i det nedenstående.

Herunder beskrives tre adfærdsdiagrammer, som er betegnet som de mest interessante i vores strukturdiagram. Teorien omkring adfærdsmønstre findes i appendiks A.

8.9.1 Beskrivelse af adfærdsdiagram for klassen 'Medvirkende'



I klassen **Medvirkende** er det denne, der er superklassen. Denne klasse er speciel, da det er en dynamisk rollemodel, og dermed sagt, kan et objekt i denne klasse have forskellige roller samtidig. Her er det igen en sekvens, der sker i oprettelsen af en medvirkende. Man går fra venstre mod højre i objektforløbet. Når man skal oprette objekter i klassen sker der en iteration, en gentagelse, da man både kan være skuespiller, instruktør og forfatter på samme film. Selve tildelingen af objekterne i subklasserne er et valg, man som bruger tildeler de forskellige objekter.

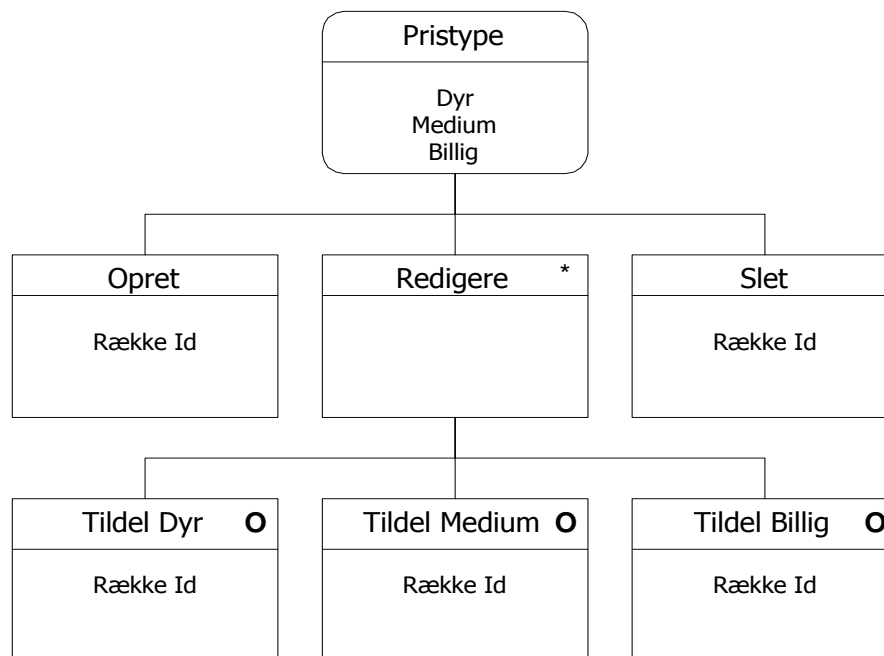


Figur 8.1



8.9.2 Beskrivelse af adfærdsdiagram for klassen 'Pristype'

Her er det klassen **Pristype** der er superklassen, og **Dyr**, **Medium** og **Billig** der er subklasserne. Da man først opretter, redigerer, og til sidst nedlægger en pristype, er det første niveau af adfærdsdiagrammet en sekvens, da man udfører hændelserne fra venstre mod højre. Det næste niveau, er at redigere rækkerne, så de får tildelt dyr, medium og billig. Da dette sker gentagne gange, er hændelsen *redigere* iterativ. Selve hændelsen tildel *dyr*, *medium* og *billig*, er en selektion, da man kun kan tildele én pristype til en række.

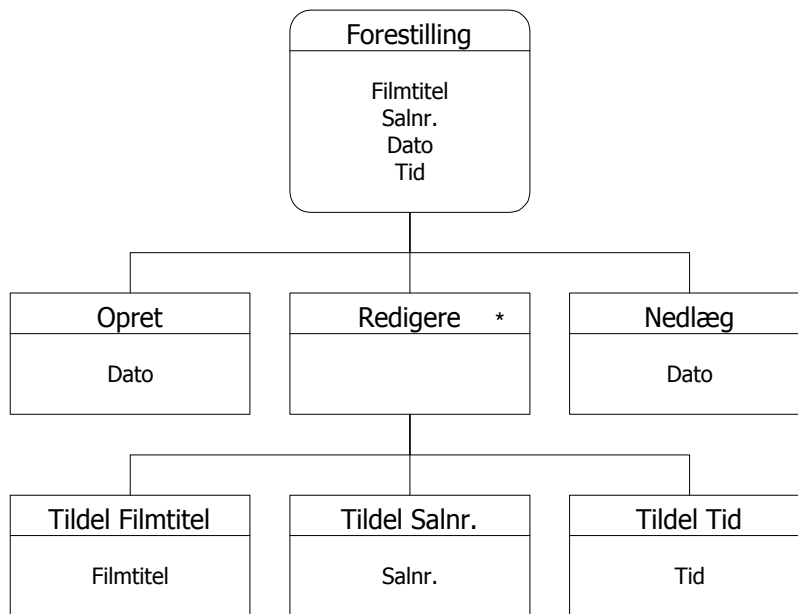


Figur 8.2



8.9.3 Beskrivelse af adfærdsdiagram for klassen 'Forestilling'

I den centrale klasse **Forestilling**, er det selve de interne hændelser, der er interessante. Denne klasse foretager tildeling af filmtitler, salnumre, datoer samt tidspunkter hvor en bestemt forestilling skal køre. Som nedenstående diagram viser, er det øverste niveau en sekvens der sker fra venstre mod højre. Man opretter, redigerer og nedlægger i denne rækkefølge. I hændelsen *redigere*, sker så tildelingen af de forskellige attributter i klassen. Selve redigeringen sker som en iteration, da den kan indtræffe flere gange. Tildelingen af attributterne i klassen sker som en sekvens, da disse skal tildeles alle sammen en gang, i den angivne rækkefølge fra venstre mod højre.



Figur 8.3

9. Fastlæg krav

9.1 Funktioner

For at finde nærmere ud af hvad det er, systemet skal lave, er det vigtigt af finde de funktioner som skal være tilstede for, at en bruger kan anvende vores billetbestillingssystem vha. Internettet. Da vores system ikke skal tage udgangspunkt i NFBs nuværende system, men derimod skal være en forlængelse af det, har vi koncentreret os om at finde netop de funktioner, der skal være tilgængelig for brugeren på Internettet og ikke hvilke funktioner, der skal kunne operere på vores tidligere fundne klasser af NFBs nuværende system. Derfor giver den nedenstående liste af funktioner også kun udtryk for dette. Her følger den liste af funktioner, som vi mener bør være tilgængelige for en bruger af systemet.



9.1.1 Funktionsliste

Funktioner	Kompleksitet
Afbestil reservation	Simpel
Bestil reservation	Kompleks
Find film	Middel
Find forestilling	Middel
Opret nyt medlem	Middel
Slet nyt medlem	Simpel

Tabel 9.1

9.1.2 Funktionsbeskrivelse

For at give et klarere indblik i hvad hver enkelt funktion betyder, følger her en funktionsbeskrivelse af systemets funktioner, der bør fremtræde når en kunde skal reservere billetter via Internettet. Vi har valgt at give denne beskrivelse nu, og ikke vente til designfasen, fordi vi under udarbejdelsen ubevidst gik i detaljer med hver enkelt funktion. Vi mener også, at det giver os et større overblik over det kommende systems egenskaber, ved at beskrive funktionerne allerede nu. Beskrivelsen af funktionerne består af en kort beskrivelse af funktionen selv samt hvilke klasser funktionen påvirker i problemområdet, og til sidst hvilken af de fire funktionstyper den tilhører (signalering, beregning, aflæsning og opdatering).

Afbestil reservation

Beskrivelse: Denne funktion sletter en reservering fra systemet.

Påvirker: Billetreservation, Sæde.

Type: Opdatering.

Bestil reservation

Beskrivelse: Denne funktion gemmer de informationer som kunden har valgt, dvs. film, tidspunkt og sædeønske. Desuden returneres den beregnede billetpris og afhentningstidspunkt.

Påvirker: Billetreservation, Sæde.

Type: Signalering, Beregning.

Find film

Beskrivelse: Denne funktion finder den film kunden har bedt om at se.

Påvirker: Ingen klasser.

Type: Aflæsning

Find forestilling

Beskrivelse: Udfra kundens valg af film og dato, sender denne funktion disse informationer via JDBC til NFBs database og returnerer alle valide forestillinger med disse betingelser.

Påvirker: Forestilling.



Type: Aflæsning.

Opret nyt medlem

Beskrivelse: Denne funktion sender email og password der er indtastet af kunden og opretter denne.

Påvirker: Kunde.

Type: Opdatering.

Slet nyt medlem

Beskrivelse: Denne funktion afbryder oprettelsen af medlemskab.

Påvirker: Ingen klasser.

Type: Signalerings.

9.2 Grænseflader

Formålet med grænsefladeaktiviteten er, at opstille kravene til edb-systemets grænseflader. Resultater skal være en grundlæggende repræsentations- og dialogform, en komplet liste af elementer i brugergrænsefladen med skitser af typiske elementer.

Gruppen har afgrænset sig fra at beskrive grænseflader til andre systemer og apparater, da disse ikke findes i gruppens problemområde.

9.2.1 Brugsmiljøet

Overvejelser om brugergrænseflader må tage udgangspunkt i en indlevelse i brugsmiljøet. Studiet af brugsmiljøet består i at analysere brugerprofilen og brugssituationerne.

Med hensyn til brugerprofilen er der tale om både tilfældige og faste brugere i alle aldre, som er biografgængere. De kan klassificeres, alt efter hvilke edb- og Internet-færdigheder den enkelte besidder. Tilfældige brugere ønsker en let tilgængelig brugergrænseflade, hvor faste brugere ønsker at de hurtigt kan udnytte systemet, fx ved hjælp af genveje.

På nuværende tidspunkt bliver brugergrænsefladen tilpasset tilfældige brugere. Man kunne forestille sig, at det på et senere tidspunkt vil være muligt at betale billetter ved bestillingen således, at faste brugere kan oprette en konto og derved slippe for at angive deres brugerinformationer gang på gang ved hver billetbestilling.

9.2.2 Brugssituationer

Gruppen har valgt at se bort fra egentlige brugssituationer, som Lars Mathiasen anbefaler, at man analyserer. Gruppen mener, at spørgsmål som fx:

Løser den enkelte bruger flere arbejdsopgaver parallelt?

Bevæger brugeren sig fysisk rundt under arbejdet?



Adskiller enkelte brugssituationer sig på afgørende måde fra de typiske brugssituationer?

henvender sig mere til en bruger, som skal udføre et stykke arbejde ved hjælp af edb-systemet. Billetbestillingssystemet kan betragtes mere som en service, der stilles til rådighed.

9.2.3 Dialogformer

Der er fem forskellige former for dialogformer:

Dialogform	Fordel	Ulempe
Menuvalg	Kort indlæring	Fare for mange menuer
Skemaudfyldelse	Kræver begrænset træning	Optager plads på skærmen
Kommandosprog	Hurtigt for faste brugere	Kræver omfattende træning
Naturligt sprog	Ingen systemer udviklet	-
Direkte manipulation	Let at lære	Vanskeligt at programmere

Tabel 9.2

Gruppen har valgt dialogformerne menuvalg og skemaudfyldelse, da disse virker mest naturlige, når man skal indtaste oplysninger på en hjemmeside. Det skal dog nævnes, at direkte manipulation implicit også er valgt, da vi benytte os af forskellige knapper på hjemmesiden, som skal aktiveres af brugeren vha. musen, fx knappen **Find forestilling**.

EON allerede gjort brug af disse dialogformer på deres hjemmeside. Disse former for dialog er de mest almindelige på nettet, og henvender sig til både faste og tilfældige brugere.

Dialogformen menuvalg går i vores system ud på, at man ved hjælp af combo-bokse kan vælge fx en film ud fra listen af film, som er tilgængelig for øjeblikket. I dette system går skemaudfyldelse ud på, at man indsamler information om brugeren såsom navn, adresse, postnr., email, password m.m.

Gruppen mener ikke, at kommandosprog egner sig til at fungere på en hjemmeside. Grunden til dette er, at en hjemmeside som regel er udarbejdet ved hjælp af HTML, som ikke er beregnet til kommandosprog, men snarere til indtastning og museklik.

9.2.4 Udformning af brugergrænsefladen

Udformning af brugergrænsefladen omfatter tre dele. Valg af dialogform, fastlæggelse af præsentation af model og funktioner samt koordinering af hele edb-systemets fremtræden.

Da det var et ønske fra NFB at hjemmesiden skulle ligne EONs, har gruppen valgt at læne sig opad denne. Derfor er det valgt, at der skal anvendes skemaudfyldelse og menuvalg som dialogform. Det grundlæggende valg står mellem at gøre dialogformen funktions- eller objektorienteret. Da der tages udgangspunkt i



funktioner, der kan virke rutineprægede, er dialogformen gjort funktionsorienteret.

Når funktionerne aktiveres, sker det ved menuer. Menuerne fremtræder som pop-up vinduer og knapper. Funktionernes konsekvenser visualiseres af browseren. I Netscape kan man fx oppe i højre hjørne se stjerneskud, mens den arbejder. Når man har fat i det ønskede, holder stjerneskudene op.

9.2.5 Produktets frem

Vi har lavet et udkast til
grov skitse, som skal v

Fx vælges forestillingen til kl.

sidens. Det er kun en

Trin 1: Find forestilling

Knappen *Find forestilling* skal aktivere en funktion som benytter brugerens valg af film og dato. Både *filmtitel* og *dato* er combo-bokse. Ved klik med musen på filmtitel får brugeren en liste af de film, der kan se i biografen. Ligeledes fungerer combo-boksen dato.

Trin 2: Vælg tidspunkt

Når der er valgt en bestemt film, kommer et nyt skærm billede frem med brugerens valg, og de aktuelle tidspunkter filmen går på, på den valgte dag. Tidspunkterne er såkaldte hyperlinks, som ved klik med musen springer videre til Trin 3.

Trin 3: Vælg sæder

Dette skærm billede er frembragt af det valg brugeren foretog i Trin 2. Her ses igen den valgte filmtitel og det tidspunkt filmen er valgt til at blive set på. Til højre ses et billede af den pågældende sals layout. Brugeren kan nu vælge de sæder der ønskes ved at markere sæderne på billedet. Derefter klikkes på knappen *Sæde ønsker*.

OK

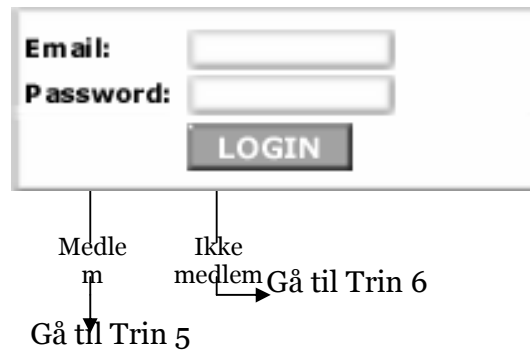
Ikke OK

Gå til Trin 7

Gå til Trin 4

**Trin 4: Login**

For at brugeren kan få lov at bestille sæderne, skal brugeren igennem en loginprocedure. Her skal brugeren blot indtaste sin email og et password og dernæst klikke på knappen login.

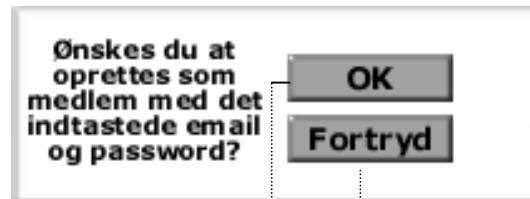


**Trin 5: Bestil/Fortryd**

Er brugeren medlem, kommer dette skærbillede frem, hvor brugeren skal vælge udfra de to knapper, om der ønskes at bestille- eller fortryde en reservation.

**Trin 6: Ikke medlemmer**

Er kunden ikke medlem fortsætter login proceduren med dette skærbillede.

**Trin 7: Best fitted seat**

Kan brugerens valg af sæder ikke imødekommes, vælger systemet selv de bedste sæder der kommer tættest på brugerens valg.



9.3 Den tekniske platform

Edb-systemet skal udvikles til en PC, der har adgang til Internettet. Systemet skal programmeres i det objektorienterede programmeringssprog Java. Da NFB ikke har tilladt os at arbejde direkte på deres egen database, af hensyn til sikkerheden, har vi valgt at benytte Access, som vha. en JDBC/ODBC-bridge kommunikere med brugeren, der sidder på Internettet.

Brugergrænsefladen skal være webbaseret og implementeres ved hjælp af HTML og brug af Servlets. Systemet skal betjenes ved hjælp af mus og tastatur. For nærmere information om Servlets, se appendiks E.



9.4 Edb-systemets nytte og realiserbarhed

Systemet har ikke været gennemgået i detaljer med NFB, men de ønskede at vi lagde os op ad EON Films design og funktionalitet. Derved er der opnået en indforstået accept fra begge parter om den ønskede funktionalitet.

Gruppen har ikke erfaring med udvikling af tilsvarende systemer med store dele af den valgte tekniske platform. Dette være sig JDBC/ODBC, Servlets og til dels Java. Det er på nuværende tidspunkt svært at fastslå, om systemet kan implementeres i den foreslåede udformning.

En eventuel fremtidig udvidelse af systemet, såsom betaling via Internettet, kan realiseres på den valgte tekniske platform, men med forholdsvis få modifikationer.

9.5 Strategi for det videre udviklingsarbejde

Da dette system er en del af et obligatorisk skoleprojekt, har der hele tiden hersket enighed blandt både gruppens medlemmer, vejleder og kontaktpersonen i NFB, om at dette system bør udvikles. Der hersker dog stadig en del tvivl om, om systemet vil kunne fungere med en høj belastning af samtidige brugere på Internettet.



Indholdsfortegnelse

10. PROBLEM DOMAIN COMPONENT	5
10.1 MODIFICERING AF ANALYSERESULTATER	5
10.1.1 Generalisering/specialisering	5
10.1.2 Associeringer og aggregeringer	5
10.1.3 Kardinaliteter	6
10.1.4 Rollemodellen	6
10.1.5 Tilføjelse af ekstra klasser	6
11. DATA MANAGEMENT COMPONENT	7
11.1 DESIGN AF DMCEN	7
11.2 VALG AF DMC	8
11.3 FRA OOA TIL E/R	8
11.3.1 Fra klasser til tabeller	8
11.3.2 Generalisering- og specialiseringsstruktur i den relationelle database	9
11.3.3 Associerings- og aggregeringsstruktur i den relationelle database	9
11.4 NORMALISERING	10
11.5 E/R	10
11.6 EER	10
11.6.1 Entitetstyper	11
11.6.2 Relationstyper	11
11.6.3 Attributter	11
11.6.4 Afhængigheder	11
11.6.5 Generalisering/specialisering	11
11.6.6 Total og delvis specialisering	12
11.6.7 Definerende prædikater	12
12. HUMAN INTERACTION COMPONENT	13
12.1 HVORDAN HIC?	13
12.1.1 Klassificering af personer	13
12.1.2 Spørgsmål til opbygning af en succesfuld website	14
12.1.3 Websitens form	16
12.1.4 Websitens indhold	16
12.2 HIC BESKRIVELSE	17
12.2.1 PDC - EIC - HIC interaktion ved login	17
12.2.2 PDC - EIC - HIC interaktion ved reservering	18
13. TABELDESIGN	19
13.1 KLOKKEREN VS. IKKE KLOKKEREN IMPLEMENTERING	19
13.1.1 Klokken implementering	19
13.1.2 Ikke klokken implementering	19
13.1.3 Overvejelser	20
13.1.4 Gruppens overvejelser	20
13.2 TABELBESKRIVELSER	21
13.3 INTEGRITET	26
13.4 DISKUSSIONEN OMKRING LÅSNING AF TABELLER	26
14. TASK MANAGEMENT COMPONENT	29
14.1 KORT SYSTEMBESKRIVELSE	29
14.2 TMCENS ROLLE I VORES SYSTEM	30
15. EXTERNAL INTERFACE COMPONENT	31
15.1 EXTERNAL INTERFACE DIAGRAM	31
15.1.1 Gruppens overvejelser	31



16. KOMPONENTERNES SAMSPIL	32
17. AFRUNDING AF OOD.....	33



Når man anvender Peter Coads og Edward Yourdons⁶ objektorienterede designmetode, er der fire forskellige komponenter, man kan designe ud fra, alt efter hvad man har brug for:

- Design af Problem Domain Component [PDC]
- Design af Data Management Component [DMC]
- Design af Human Interaction Component [HIC]
- Design af Task Management Component [TMC]

Udover dette vil gruppen også benytte John R. Ellis⁷ komponent, Eksternal Interface Component [EIC].

Som det fremgår af navnet, beskæftiger PDCen sig med problemområdet, som tager sit udgangspunkt i strukturdiagrammet, der blev udarbejdet under analysen. Det er også i arbejdet med PDCen, at man modificerer analyseresultaterne, hvis der forekommer ændringer.

HICen udgør designet af brugergrænsefladen og indeholder design af fx vinduer. Komponenten afspejler, hvordan en person vil interagere med systemet samt hvordan systemet vil præsentere information for brugeren.

DMCen er den komponent, som håndterer lagring og læsning af objekter fra data management systemet. Der er flere måder at tilgå disse på og ifølge Coad & Yourdon⁸ er de tre mest anvendte:

- Flade filer
- Relationelle databaser
- Objektorienterede databaser

Hvis et system skal håndtere flere processer samtidig, skal TMC komponenten benyttes. TMC bliver i sammenhæng med webbaseret systemer brugt til, at styre de tråde processer der opstår når flere klienter forspørger data'er fra serveren. Når man i nogle tilfælde ønsker, at et system skal kommunikere sammen med en ekstern komponent, benytter man en EIC. Denne komponent kan man så udskifte alt efter, hvilke informationer man ønsker. Man kan naturligvis også benytte flere eksterne komponenter. Et eksempel på brugen af en EIC, kan være når et system styre en ekstern måler. Komponenten der trigger måleren, vil så være EIC'en. I denne del dokumenteres designet af henholdsvis PDC, HIC, DMC, TMC og EIC. Gruppen har eksperimenteret med anvendelse af en udvidet version af E/R-modellen, nemlig EER-modellen. Da det er i DMCen man begynder at normalisere sine tabeller, har vi valgt at starte modelleringen af EER-diagram her. Vi har valgt at lægge beskrivelsen af EER-diagrammet i denne del samt selve diagrammet i appendiks C.

⁶ Coad, Peter & Yourdon, Edward: Object-oriented Design.

⁷ John R. Ellis: Objectifying Real Time Systems



10. Problem Domain Component

Som det fremgår af navnet, beskæftiger denne komponent sig med problemområdet, som tager sit udgangspunkt i strukturdiagrammet, der blev udarbejdet i OOA.

Udviklingen af PDCen har en række hovedformål, nemlig at man får:

- Større genbrug af design og klasser
- Grupperet problem domain specifikke klasser sammen
- Etableret en protokol, ved at tilføje en generaliseringsklasse
- Tilgodeset de understøttede niveauer af hierarki
- Forbedret performance
- Understøttet Data Management Component
- Tilføjet lavere niveauer komponenter
- Reviewet og udfordret tilføjelserne mod OOAs resultater

10.1 Modificering af analyseresultater

Under gruppens udarbejdelse af PDCen, er analyseresultaterne blevet revideret og der er foretaget visse ændringer i forhold til strukturdiagrammet. Se appendiks D.

10.1.1 Generalisering/specialisering

Generalisering/specialisering omkring *Dyr*, *Medium* og *Billig*, for *Pristype* er blevet fjernet. Gruppen indså, at denne nedrivning er irrelevant, da hver af specialiseringsklasserne kun indeholder én attribut hver, som desuden også lå i generaliseringsklassen. Da disse kun har én attribut til forskel, lægges specialiseringerne op i generaliseringsklassen *Pristype*. Desuden har gruppen tilføjet en attribut, *pristypeID*, i klassen *Pristype*, samt et *Id* i klassen *Billetreservering*. Begge er unikke nøgler for disse klasser.

10.1.2 Associeringer og aggregeringer

Associeringen mellem *Forestilling* og *Pristype* er desuden fjernet fra PDC-diagrammet, da det muligt at finde den eksakte pris igennem *Sal*, *Række* og *Pristype*.

Associeringen mellem *Film* og *Medvirkende* er blevet en aggregering. Det skal ses i lyset af, at en film består af medvirkende, og hvis der ingen medvirkende er, er der heller ingen film. Der vil altid, efter gruppens opfattelse, være tilknyttet en medvirkende til en film, om det så bare er en fotograf.

10.1.3 Kardinaliteter

Der er kardinaliteter på PDC diagrammet, som er ændrede i forhold til strukturdiagrammet. Mellem *Billetreservation* og *Kunde* kan en kunde oprettes uden at have en reservation. Derfor er kardinaliteten ændret til 0:M. Mellem *Billetreservation* og *Sæde*, er et sæde oprettet uden nødvendigvis at have en reservering. Associeringen mellem *Sal* og *Forestilling* er 0:1, da en *Sal* kan have op til én forestilling på et givent tidspunkt. Derfor kan der på et givet tidspunkt være én eller ingen forestilling i en sal.



10.1.4 Rollemodellen

Vi havde i starten modelleret den dynamiske rollemodel, så der kun fandtes tre former for roller, nemlig *Skuespiller*, *Forfatter* og *Instruktør*. Vi indså dog hurtigt, at det ville være en for stor begrænsning at gøre det på denne måde, da der findes mange flere former for roller end kun disse tre. Vi valgte derfor at omdøbe de forskellige roller til persontype1, 2, ... , *n*. Dette skal illustrere, at der findes et udefineret antal af roller, som en medvirkende kan besidde samtidigt.

Gruppen er klar over, at det er en noget utraditionel notationsform, vi har benyttet til at vise denne specielle struktur, men vi mener den viser ideen på bedste vis.

10.1.5 Tilføjelse af ekstra klasser

Desuden har gruppen tilføjet en ekstra klasse, *Postnrby*, til PDC diagrammet. Dette er en opslagstabel, der indeholder alle postnumre i Danmark.

11. Data Management Component

Data Management-komponenten [DMC] er den komponent, som tilbyder infrastrukturen til lagring og genfindning af objekter fra et data management system.

Da gruppen vil anvende en relationel database som lagringsmedie, har vi valgt at konvertere OOA-modellen (klasser, objekter, strukturer og dynamik) til et E/R-diagram, da dette vil give en hurtig implementering. Konverteringen vil betyde, at PDCen og DMCen kollapses og derved vil repræsentere begge komponenter. Når man designer en DMC, designer man normalt to ting: datalayout og services. Dette skal gruppen ikke, da den skal implementere databasen ved hjælp af et relationelt databaseværktøj, nemlig Access. Med design af datalayout menes definition og normalisering af databasens tabeller, og med services menes de services, som fx gemmer et objekt i databasen – eller måske ved databasen selv, hvordan den skal gemme et objekt. Når man arbejder med Access, er det ikke nødvendigt at designe services, da disse ligger i DBMSen. Disse services kan aktiveres på to måder: enten af brugerne, når de via Internettet henter fra, eller gemmer i, databasen, eller af DBAen, som direkte manipulerer databasens indhold.

En af grundene til, at vi anvender Access er, at vi kun bruger den som en kopi af den almindelige database, så vi har noget at teste på. Bortset fra det, er det nemt at oprette og redigere tabeller i Access, og det er samtidig nemt at udforme en ordentlig brugergrænseflade, hvis dette skulle blive nødvendigt. Disse er helt klare fordele, men én ulempe er i hvert fald, at Access har begrænsninger på, hvor mange tabeller og forekomster man kan lave. Dette bliver dog ikke et problem for os, da dette tal alligevel er så stort, at vi slet ikke når op i den størrelsesorden.

11.1 Design af DMCen

Som nævnt ovenfor designer man normalt services og datalayout. Denne DMC indeholder kun design af datalayout, da PDCen og DMCen kollapses til at udgøre E/R-diagrammet.

Med hensyn til services er der umiddelbart to måder at implementere DMCen:



Central DMC

Decentral DMC

En *central* DMC håndterer behandlingen af alle objekterne i PDCen. Hvis der tilføjes nye objekter vil disse blive behandlet af DMCen. En central DMC er mere kompliceret, da den ved hvert nyt objekt skal identificere hvilken klasse, som objektet er instans af, og derefter placere det i den rette sammenhæng. Ulempen er, at DMCen er svær at implementere, da den skal kunne genkende hvilken klassen, som et objekt er instans af. Dog er det en fordel, at styringen er placeret centralt.

En *decentral* DMC fortæller et objekt i PDCen, at det skal gemme sig selv, dvs. at, objektet selv skal kende tabellayoutet. Ulempen er, at hvis tabellayoutet ændres, skal alle objekter også ændres. Fordelen er dog, at DMCen bliver lettere at implementere pga. at den kun skal sende én besked til PDCen.

Proceduren for design af datalayout er, at man starter med at opstille alle klasser og de til hver klasse hørende attributter. Derefter normaliserer man listen til og med tredje normalform, hvilket derved resulterer i tredje normalforms tabeldefinitioner. Næstsidst definerer man en databasetabel for hver tredje normalformstabel, og til sidst ses der på performancekrav. Det kan fx være, at nogle tabeller skal lægges sammen for at minimere antallet af opslag m.m.

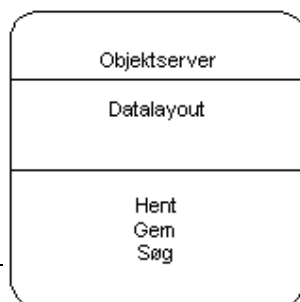
11.2 Valg af DMC

Da vi har tænkt os at implementere databasen ved hjælp af et relationelt databaseværktøj, har vi et mindre problem. Vi benytter os af en objektorienteret analysemetode, og en relationel database kan ikke implementeres udfra en objektorienteret tankegang. Næste skridt vil derfor være at konvertere de klasser og objekter, som vi har fundet i analysen. Denne konvertering kan foretages vha. Lars Mathiassens⁹ fremgangsmåde.

Ved en implementering af DMCen, ville vi vælge at implementere den som en central DMC, dvs. objekterne beder DMCen om at blive gemt. Selvom vores tabellayout ikke ændrer sig, har vi alligevel valgt at lade DMCen klare læsning og skrivning af objekter, da denne metode vil sørge for at systemet bliver mere konsistent samt øge genbrugeligheden.

Vi har som nævnt ikke behov for at implemente en DMC i vores system, men skulle den implementeres ville den kunne se ud som på følgende figur:

Figur 11.1



⁹ Mathiassen, Lars et al.: Objektorienteret analyse og design.



11.3 Fra OOA til E/R

Meningen med denne konvertering er, at vi ud fra vores objektmodel gerne skulle ende med, at få en færdig E/R-model, som kan bruges som udgangspunkt til den endelige implementering af den relationelle database.

11.3.1 Fra klasser til tabeller

Hver af de fundne klasser bliver til en entitet og hvert forbindelsesled bliver til en relation i E/R-modellen. Disse entiteter og relationer vil fremover blive kaldt tabeller. Den pågældende klassens navn, kan også bruges som navn til tabellen, og klassens attributter kan også overføres som attributter til tabellen. Attributterne udgør kolonner og hver objekt af en klasse svarer til en række.

Man skal være opmærksom på, at hver tabel skal have en unik identifikation. Findes der ikke en sådan, skal der tilføjes en, fx i form af et autonummer eller lignende.

11.3.2 Generalisering- og specialiseringsstruktur i den relationelle database

Denne form for struktur findes ikke direkte i en relationel database, men der er flere muligheder at implementere den på.

Vi kan som eksempel på dette bruge vores fundne genspec-struktur mellem klasserne *Medvirkende*, der er den generelle klasse, og klasserne *Persontype 1*, *Persontype 2*, ... , *Persontype n*, som er specialiseringsklasser.

Første mulighed: at implementere de fire klasser ved at oprette fire tabeller. Et objekt fra hver af klasserne *Persontype 1*, *Persontype 2*, ... , *Persontype n* skal nu repræsenteres i den generelle klasse *Medvirkende* ved hjælp af en unik nøgle til hver af klasserne. Det vil sige, at i tabellen *Medvirkende* laves en relation til tabellerne *Persontype 1*, *Persontype 2*, ... , *Persontype n* via en unik nøgle.

Anden mulighed: at man flytter alle attributter fra den generelle klasse ned i de tre specialiserede klasser. Denne løsning er bedst, når der er få attributter i den generelle klasse og, hvis man under søgning kender persontypen i forvejen.

Tredje mulighed: at oprette én tabel, nemlig den generelle klasse *Medvirkende*. Denne tabel skal så indeholde alle attributter fra samtlige tre specialiserede klasser. Denne løsning betragtes som bedst, når der er mange attributter i forvejen i den generelle klasse.

11.3.3 Associerings- og aggregeringsstruktur i den relationelle database.

Når man taler om aggregerings- og associeringsstrukturer i forbindelse med relationelle databaser, skelnes der implementeringsmæssigt ikke mellem disse to former for strukturer. De implementeres nemlig på samme måde – nemlig ved direkte brug af nøgler.

1. **Mange-til-mange associering/aggregering:** her oprettes én tabel for hver af de involverede klasser. For at skabe forbindelse mellem disse tabeller, oprettes en tredje tabel, der skal indeholde hver af de to førnævnte



tabellers unikke nøgler. På denne måde bliver den tredje tabel til et led, der forbinder de to hovedklasser sammen via nøglerne.

2. **En-til-mange associering/aggregering:** fremgangsmåden i dette tilfælde er også forholdsvis simpel. Ved hjælp af en fremmednøgle fra "en-klassen", kan man pege på objekterne i "mange-klassen". Er der her tale om aggregering, skal man dog tage følgende to spørgsmål i betragtning, når man skal henholdsvis oprette og slette objekter:

- Har helhedsobjekter altid nogle delobjekter, som skal oprettes sammen med det?
- Er delobjekters eksistens afhængig af helhedsobjektet, så delene skal nedlægges 7sammen med helheden?

Om helhedsobjektet i dette system altid har nogle delobjekter, må man svare ja til. Man kan sige, at man fx opretter en biograf uden at oprette en sal – men det giver ingen mening at gøre således.

Delobjekternes eksistens er afhængig af helhedsobjektet, fx er en rækkes eksistens afhængig af en sal, som den tilhører. Hvis man nedlægger en sal, må man også nedlægge de tilhørende rækker.

11.4 Normalisering

Når man nu har udarbejdet sit E/R-diagram over den kommende database, skal hver entitet/relation kvalitetstjekkes. Dette kvalitetstjek kaldes *normalisering*. Her ses der nærmere på felternes indbyrdes forhold og afhængighed. Normalisering af datamodellen resulterer i, at man undgår redundante data og derved sikrer, at databasen bliver konsistent.

Visse lærebøger anbefaler, at man som hovedregel bør opfylde alle seks normaliseringstrin, men gruppen anvender Coad & Yourdons¹⁰ designmetode, og disse forfattere anbefaler at man nøjes med, at normalisere til og med tredje normalform.

Der er naturligvis også ulemper ved normalisering. Der kommer flere tabeller, hvilket medfører flere læsninger og skrivinger ved opslag og opdateringer. Fordele ved normalisering, som opnås på længere sigt er ingen redundant data og en konsistent database.

I appendiks A ses definitionerne på første og til og med tredje normalform for, at man kan få en fornemmelse af, hvad der skal opfyldes på de forskellige trin.

11.5 E/R

Da gruppen primært har valgt at anvende EER-teknikken, vil den ikke gå ind i en dybere og mere detaljeret beskrivelse af E/R-diagrammet. E/R-modellen er kun blevet anvendt som springbræt til EER-modellen, som er beskrevet til mindste detalje i næste afsnit (appendiks C).

Der er i realiteten ikke så meget at forklare om E/R-diagrammet. Det består af entiteter, relationer og kardinaliteter. Der findes ingen itererende datagrupper,

¹⁰ Coad, Peter & Yourdon, Edward: Object-oriented Design.



men der er dog et subtypediagram. Entiteten **Medvirkende** er supertypen, og **Persontype 1**, **Persontype 2**, ... , **Persontype n** er subtyperne. Gruppen har valgt at anvende EER istedet for E/R, da man med EER kan modellere mange flere ting. Man kan modellere generaliserings/specialiseringsstrukturer, aggregeringernes afhængigheder samt forskellige slags attributter (almindelige, nøgleattributter, sammensatte, flerværdi). Ved første øjekast virker EER-diagrammet meget uoverskueligt, men hvis man bruger lidt tid til at sætte sig ind i teknikken vil man finde, at det er en særdeles raffineret måde at modellere E/R på.

11.6 EER

Gruppen har valgt at udarbejde et EER-diagram på grund af de semantiske fordele, som er nævnt i appendiks C. Nedenfor følger en gennemgang af gruppens EER-diagram.

11.6.1 Entitetstyper

Alle entitetstyper er almindelige entitetstyper. Der findes ingen svage entitetstyper, dvs. entitetstyper, som ikke har en komplet nøgleattribut. Man overvejede på et tidspunkt, om entitetstypen **Pristype** var en svag entitet. Den havde ikke fået en nøgleattribut, og derfor blev det diskuteret, om **Pristype** var ejet af **Række**. Gruppen blev dog enige om, at det ikke var en svag entitet, og den fik sin egen primærnøgle.

11.6.2 Relationstyper

Alle relationstyper er almindelige relationstyper. Der findes ingen identificerende relationstyper, da der ikke findes nogen svage entitetstyper. Man overvejede på et tidspunkt, om relationstypen **Pristype/Række** var den identificerende relation for den svage entitetstype **Pristype**, som skulle knyttes til sin ejer netop gennem denne identificerende relation. Gruppen blev enig om, at der ikke var tale om en identificerende relation, når **Pristype** ikke var en svag entitetstype.

11.6.3 Attributter

Alle nøgleattributter er understreget. Der findes ingen flerværdi-attributter, men derimod nogle enkelte tilfælde af sammensatte attributter.

11.6.4 Afhængigheder

Der findes tre afhængigheder, og disse afhængigheder stemmer overens med de tre aggregeringer, som ses på både strukturdiagrammet og PDC-diagrammet. Disse afhængigheder ses mellem:

Biograf og Sal
Sal og Række
Film og Medvirkende



Disse afhængigheder viser, at fx en sal ikke kan eksistere uden at en biograf eksisterer. Ligeså forholder det sig med de to andre afhængigheder.

11.6.5 Generalisering/specialisering

Som det ses er nogle af entitetstyperne relateret ved et generaliserings/specialiserings-forhold, som resulterer i et *er-et* forhold. Dette er genspec-strukturen omkring entitetstypen **Medvirkende**.

O'et i cirklen fra **Medvirkende** og ned til subclasserne angiver, at der findes et overlap således, at fx en Persontype 1 godt kan være en Persontype 2 (dvs. den samme entitet kan være medlem af mere end én subklasse).

11.6.6 Total og delvis specialisering

I genspec-strukturen er anvendt en dobbelt linie fra entitetstypen **Medvirkende** og ned til cirklen. Dette angiver, at der er tale om en delvis specialisering, hvilket vil sige, at et medlem af **Medvirkende** skal være medlem af en af subclasserne.

11.6.7 Definerende prædikater

På genspec-strukturen **Medvirkende** er anvendt definerende prædikater. Det første prædikat er *Persontype*, som er sat på linien fra **Medvirkende** og ned til cirklen. Dette prædikat er anvendt pga. attributten *Persontype*, efter hvilken subclasserne klassificeres.

12. Human Interaction Component

HICen udgør design af brugergrænsefladen og indeholder design af fx vinduer. Disse skal under normale omstændigheder designes i overensstemmelse med brugernes krav, men da denne HIC er en website, vil det blive svært at tilfredsstille alle brugeres ønsker. HICen designes dels med udgangspunkt i de krav, som gruppen fandt frem til i afsnittet om brugergrænseflader i analyseaktiviteten, og dels med udgangspunkt i EON Films website, som vi ligger os meget opad. HICen afspejler hvordan en person vil interagere med systemet og hvordan systemet vil præsentere information for brugeren. Normalt forholder det sig sådan, at man begynder med at studere brugsmiljøet *inden* man påbegynder sit HIC-design. Men brugsmiljøet er ikke anderledes end alle andre websites med hensyn til fremgangsmåden, så dette har gruppen valgt ikke at gøre så meget ud af.

12.1 Hvordan HIC?

Ifølge Coad & Yourdon¹¹ er strategien for at designe denne komponent som følgende:

- Klassificér personer
- Beskriv personerne og deres opgavescenarier
- Design kommandohierarkiet
- Design den detaljerede interaktion

¹¹ Coad, Peter & Yourdon, Edward: Object-oriented Design.



Fortsæt prototyping
 Design af HIC-klasserne
 Design ved at tage højde for GUI

Gruppen mener ikke, at gennemgang af alle ovenstående HIC-emner bliver så dybdegående, som hvis vi fx skulle udvikle et administrationssystem. Vi vil allerede nu afgrænse os fra at designe et kommandohierarki, og istedet besvare nogle spørgsmål, som vi finder mere relevante.

Der startes med klassificering af personer, og klassificering af disse omfatter også beskrivelse. Der beskrives ikke opgavescenarie, da dette er meget simpelt. Med hensyn til at fortsætte prototypingen, kan man sige at der prototypes hele tiden.

12.1.1 Klassificering af personer

Der klassificeres efter brugernes færdigheder, men vi vælger kun én klassificering – nemlig ”tilfældig bruger”. Man kunne forestille sig, at engang med tiden ville systemet blive udbygget således, at man som det første på hjemmesiden skulle vælge om man var novice, tilfældig eller fast bruger. Derved – alt efter hvad man vælger – ville man få fremvist en passende HIC.

Vi har valgt ”tilfældig bruger” set i lyset af, at brugerne på Internettet har meget varierende edb-erfaring.

Personbeskrivelse

Hvem Såvel unge som ældre

Formål At anvende systemet til bestilling af biografbilletter på

Internettet

Karakteristik *Uddannelse*

Kan være hvad som helst

Kritiske succesfaktorer

Systemet skal understøtte bestilling af biografbilletter, og bruger grænsefladen må ikke være for svær at anvende.

Gruppens karakteristika af de brugere der vil benytte bestillingssystemet, vil være en bruger der er vant til at surfe, og kender de gængse metoder indenfor Internettet.

Færdighedsniveau Alle brugere er klassificeret som tilfældige brugere

12.1.2 Spørgsmål til opbygning af en succesfuld website

Nedenstående spørgsmål er hentet fra <http://dir.dk/artikler-krav.htm> og omfatter en række spørgsmål, som en virksomhed bør overveje inden den får en website på Internettet. Gruppen har besvaret disse spørgsmål på bedste vis, så det er altså ikke virksomhedens meninger, men gruppens meninger, der gives udtryk for i nedenstående.

Er det overhovedet relevant for virksomheden at få en website, og bruger virksomhedens målgrupper Internet?



Gruppen mener, at det *er* relevant for NFB at komme på Internettet – det er relevant for de fleste servicevirksomheder, da Internettet får større og større indflydelse på vores hverdag. NFBs målgruppe er primært unge mennesker, og sekundært alle andre. De fleste unge mennesker i dag anvender Internet både hjemme og på arbejde eller i skolen. Flere ældre mennesker begynder også at interessere sig for Internettets muligheder, og ligeså forholder det sig også for dem, som endnu ikke har nået teenage-alderen.

Skal virksomheden gå på Internettet af strategiske grunde eller bare for at "være med". I givet fald, hvad er de strategiske overvejelser bag?

NFB bør overveje at gå på Internettet af strategiske grunde, men også bare for at være med. Alle er snart sagt på Internettet nu om dage, så hvorfor ikke NFB. De strategiske overvejelser handler nok mest om at skaffe sig kunder. Det er afgjort en fordel, at hvis man læser information om en film på en biograf's website, er det rart at kunne bestille en billet med det samme – og derved undgå personlig betjening, hvis det er det, man ønsker.

Hvilke målgrupper ønsker man at kunne servicere med websiten?

Man ønsker at kunne servicere alle målgrupper – personer i alle aldre, som bruger Internettet. Primært nok de unge mennesker, da de formentlig går mest ud, men også ældre såvel som meget unge.

Skal virksomheden tilbyde statiske eller dynamiske (databaseløsning) informationer?

NFB skal tilbyde dynamisk information. Det er nemlig en databaseløsning, som anvendes når brugerne af hjemmesiden bestiller en billet eller henter noget information.

Hvilke øvrige services skal målgrupperne have tilbudt?

Man kunne gøre som EON, hvor man kan melde sig til en ordning, hvor man får email hver gang der kommer en ny film, man kan bestille CD'er m.m. I fremtiden kunne man tilbyde billetbetaling såvel som billetbestilling på websiten.

Skal virksomheden forberede sin website til at kunne håndtere salg, når sikkerheden om kort tid er tilstrækkelig hertil?

Ja, det mener vi afgjort.

Skal virksomheden selv stå for opdatering af informationer? Hvem skal tage sig af administration af websiten – virksomhedens edb-afdeling, marketingsafdelingen eller 3. part?

NFB kan godt selv stå for opdatering af informationer om fx nye film, forudsat at det er en kvalificeret person, som gør det. NFBs tekniske afdeling kan godt stå for administration af websiten, da disse allerede står for administration af den database, som NFB anvender.



Skal virksomheden anvende Intranet som led i en eventuel Internet-strategi?

Gruppen mener ikke, at det er nødvendigt for NFB at anvende et internt internet (Intranet), da deres lokale netværk ikke er så stort. Men det kunne være smart, hvis man forestillede sig at man derved kunne benytte den allerede eksisterende brugergrænseflade fra Internettet til at registrere en ordre, der blev indtelefoneret.

Hvad gør virksomhedens konkurrenter vedrørende Internet? Bliver markedsføring via Internet en vigtig konkurrenceparameter indenfor den aktuelle branche?

Der findes ikke den store konkurrence mellem biografer i Danmark, da det stort set er Nordisk Film det hele. Gruppen mener dog, at markedsføring via Internettet kunne blive en vigtig konkurrenceparameter i fremtiden, da det kunne skaffe flere kunder og dermed mere indtjening.

Skal virksomheden anvende ekstern rådgivning eller benytte egne ressourcer til at udvikle sin website?

Hvis NFB anvender ekstern rådgivning, vil websiten sandsynligvis blive mere professionel og gennearbejdet. Medmindre NFB selv har nogle kapable medarbejdere med lidt teknisk snilde. Der er dog den risiko, at NFBs egne folk er ”blinde” overfor nytænkning og kreativitet eller er blevet fastlåst i nogle bestemte arbejdsgange. Derfor kunne det være fordelagtigt med ekstern rådgivning.

12.1.3 Websitens form

Formen, dvs. først og fremmest strukturen og anvendelse af grafik er vigtig for websitens funktionalitet og brugervenlighed. Det er væsentligt, at siden er overskuelig og logisk i sin opbygning.

På websitens hovedgrænseflade anvendes grafik for at gøre det nemmere og mere visuelt for brugeren. Noget visuelt virker altid mere tiltrækkende end en masse tekst, for som man siger: ”Et billede siger mere end tusind ord”.

Da brugervenlighed altid er et subjektivt begreb, er der svært at diskutere.

Gruppen selv mener, at websiten bliver brugervenlig. Vi mener, at den er nem at anvende, overskuelig og logisk i sin opbygning.

Gruppen har gået meget efter EONs webdesign, og deres website er sandsynligvis blevet analyseret, så gruppen mener at på denne måde kan gå helt galt i byen mht. websitens brugervenlighed og logiske komposition.

12.1.4 Websitens indhold

Der skal være en nøje balance mellem websitens indhold og form. Kvaliteten af de tilbudte services skal kunne fastholde brugernes interesse således, at brugeren vender tilbage for at anvende siden.

Ifølge artiklen om den succesfulde website skal, en god website altid have én eller flere kerneservices såsom adgang til:



Virksomhedens produktkataloger
Forhandlerliste
Supportfunktioner
Nyhedsinformationer

Og med de stadig øgede muligheder for at koble virksomhedens databaser til Internetmiljøet er der derved muligheder for at tilbyde relevant information til virksomhedens målgrupper på en effektiv måde.

Gennem NFBs website får brugerne mulighed for fra Internettet, at koble sig på NFBs database, hvorfra de kan hente information om film samt skrive til databasen ved at reservere billetter.

Udover kerneservices er det nærliggende at tilbyde forskellige services – det være sig både selvstændige services eller services, som supplerer kerneservices, fx kan NFB ønske at få informationer fra målgrupperne via websiten og disse informationer kan derefter indgå i NFBs videre markedsføringsarbejde. En måde at gøre det på kunne være en spørgeskemaundersøgelse direkte på Internettet vedrørende brugernes vurdering af websiten og NFBs webservices.

Det kunne også være forslag fra brugerne til, hvordan NFB kunne forbedre websiten og dens services.

På nuværende tidspunkt indeholder NFBs website kun de mest basale services, som skal anvendes for at kunne lave en billetbestilling.

Der citeres¹²:

En rigtig god website virker som en levende organisme, der forandrer og tilpasser sig brugernes behov. Menneskelig kreativitet og opfindsomhed er forudsætningen for, at en sådan tilpasning kan ske optimalt. Anvendelse af de tekniske muligheder er i den sammenhæng blot et værktøj.

12.2 HIC beskrivelse

Vores HIC består af en række HTML-dokumenter, som frembringes i forskellige situationer. Disse HTML-dokumenter består af en række knapper (buttons), Links, Combo (combo bokse) og Editfields (tekstfelter).

HTML-dokumentet Popup (se HIC-diagrammet i appendiks D), er i virkeligheden ikke noget dokument, men vi har lavet den for at vise at de tre underliggende HTML-dokumenter er popup dokumenter, dvs. at de ved login proceduren ligger sig oven på et andet HTML-dokument, fx Film-Dato-Tid.

12.2.1 PDC - EIC - HIC interaktion ved login

Her følger en beskrivelse af diagrammet i appendiks D. Diagrammet viser samspillet mellem PDC, EIC og HIC, når en bruger skal igennem login proceduren og evt. registrering.

For at en bruger kan foretage en reservering, skal brugeren være logget på. Er brugeren ikke registreret, skal vedkommende udfylde en online registreringsformular.

¹² <http://dir.dk/artikler-krav.htm>: Opbygning af den succesfulde website.



Når kunden logger sig på, sendes *kundeID* og *password* til PDC-klassen **Kunde** via EICen. Er brugeren medlem, er denne nu logget på HIC klassen **Reservering**, hvorefter vi henviser det videre forløb i PDC - EIC - HIC interaktion ved reservering i appendiks D.

Er kunden derimod ikke er medlem, returneres et "ikke medlem" til HICen via EICen, som frembringer klassen **Registrering**. Her skal brugeren indtaste de nødvendige informationer, for at kunne blive oprettet. Disse informationer bliver derpå sendt til PDC klassen **Kunde** gennem EICen, hvor det gemmes.

12.2.2 PDC - EIC - HIC interaktion ved reservering

Som det ses i diagrammet i appendiks D, viser diagrammet den interaktionen der sker når en bruger skal reservere sæder til en bestemt biografforestilling.

Brugeren har på dette tidspunkt valgt den film der ønskes at se samt dato og tidspunkt, dvs. *forestillingsID* og *kundeID* er allerede gemt i klassen

Billetreservation.

Først sendes *forestillingsID* til klassen **Forestilling**, for at finde hvilken sal forestilling er i. **Forestilling** sender *salnummer* videre til klassen **Sal**, for at kunne finde det layout den pågældende sal har. Sal-layoutet sendes til HICen gennem EICen, hvor sal-layoutet indgår i **Sædevalg** dokumentet.

Brugeren vælger nu de sæder der ønskes at reserveres. De valgte sæder og deres tilhørende rækker sendes nu til PDC klassen **Billetreservation** via EICen, hvor de valgte sæder/rækker gemmes.

Nu kan der ske to ting, enten er sæderne ledige eller også er de optagede.

Hvis sæderne er ledige, returneres et 'ok' gennem EICen til HIC-dokumentet **Normal reservering**, der fortæller at reserveringen er fuldført. Brugeren får nu muligheden for at fortryde sin reservering. Gør brugeren dette, sendes et fortryd til PDC-klassen **Billetreservering** igennem EICen, som sletter *ReservationsID*'et og dermed de reserverede sæder.

Hvis sæderne er optaget i forvejen, returneres et 'ikke ok' via EICen til HICens **Alternativ reservering**, som giver brugeren nogle alternative sæder, som kommer tættest på brugerens oprindelige ønske. Brugeren får også her muligheden for at fortryde de af systemet valgte sæder. Gør brugeren dette, slettes *reserveringsID* på samme vis som før. Dette viser det informations flow der er imellem PDCen, EICen og HICen. Gruppen medtog EICen i dette diagram, da systemet ikke kommunikere direkte imellem PDCen og HICen. Al kommunikation mellem disse, sker igennem EICen, da dette system er en smule specielt opbygget. Detaljer om denne ombygning og kommunikationsform, ligger i afsnittet omkring EIC-komponenten.

13. Tabeldesign

I det efterfølgende vil designet af systemets tabeller blive gennemgået.



13.1 Klokkeren vs. ikke klokkeren implementering

Når vi går fra E/R diagrammet og til at designe tabeller til et databasesystem, er det vigtigt at overveje om man vil benytte klokkeren eller ikke klokkeren implementering. De designovervejelser man bør gøre sig, skal gøres ud fra kravene til databasen, såsom søgetid og kompleksitet.

13.1.1 Klokkeren implementering

Hvis man ikke tager hensyn til EDB-maskiner og databasesystemer, vil en klokkeren implementering være ideel. Det skal ses i lyset af, at man ikke tager hensyn til virkelighedens verden, hvor der er begrænsninger, og hvor accesstiderne ikke er lynhurtige lige meget hvor komplekse tabellerne er. Her implementerer man alle entiteter, subtyper, datagrupper og relationer direkte ned i selvstændige tabeller, som man tilføjer et unikt ID, som entydigt skiller tabellerne fra hinanden. Disse unikke værdier bliver automatisk genereret af systemet, hver gang der oprettes en ny række i tabellen. Derpå skaber man en database, der afspejler virkeligheden, og har den optimale fremtidssikring. Det bliver nemt at overskue samt implementere nye tabeller og deres relationer i databasesystemet.

13.1.2 Ikke klokkeren implementering

I virkelighedens verden, vil et komplekst system – implementeret klokkerent – give en dårlig performance, grundet en dårligere måde at sortere på. Derved giver det utilsigtede lange accesstider, jo større systemet bliver. Dette kan undgås ved at benytte tre metoder:

Redundant lagring: her findes tre typer redundans; *kolonnekopiering*, *typefelter* samt *beregnete dataelementer*. Disse tre typer går hovedsagelig ud på at tilføje databasesystemet flere attributter eller join tabeller for at skabe direkte eller bedre accessveje mellem to eller flere tabeller. Dette skaber så en bedre performance, når systemet kan tilgå den ønskede information direkte, uden at skulle igennem flere tabeller. Der er så overvejelser, der skal gøres mht. opdateringer, lagerkapacitet samt overblik over databasen.

Tabelintegration: her ønsker man derimod at mindske antallet af tabeller ved, at integrere to eller flere tabeller sammen i en tabel, og derved nedsætte accesstiden samt overskueliggøre databasen, da der derved bliver mindre tabeller, systemet skal igennem.

Tabeludeladelse: man kan mere drastisk overveje om der er tabeller, der helt kan udelades af systemet.

13.1.3 Overvejelser

Som hovedregel skal man holde sig til en klokkeren implementering, da man derved skaber et godt databasedesign, som er baseret på den analyserede virkelighed og som giver den mest optimale fremtidssikring. Det kan også give problemer med redundans, da dette hurtigt kan skabe problemer omkring opdatering og sletning, altså om tabellerne indeholder de opdaterede værdier, eller



om rækken blev slettet sammen med alle dens relationsrækker, og om disse skulle slettes osv.

Hvis man ønsker at benytte en ikke klokkeren implementering, skal dette overvejes grundigt i de konkrete tilfælde, hvor man vægter for og imod for et godt databasedesign eller en bedre performance. Hvis man ønsker at indføre en ikke klokkeren implementering i sit databasedesign, bliver hele databasen betragtet som en ikke klokkeren implementering, da det er helhedsbilledet man kigger på.

13.1.4 Gruppens overvejelser

Gruppen har, så vidt det er muligt, implementeret databasen klokkeren, da det giver det bedste billede af den analyserede virkelighed, og et godt og overskueligt database design. Der er dog nogle tilfælde, hvor gruppen ikke har overført entiteterne og relationer direkte i databaseimplementeringen. Dette skete efter grundige overvejelser fra gruppens side, hvor vi gennemgik de problemer, der er beskrevet ovenfor i forbindelse med et ikke klokkeren design. De ændrede tabeller er beskrevet nedenfor, både mht. hvad der gør dem ikke klokkerene samt hvorfor gruppen ønskede at benytte dem.

1. Tabellen **Sal**

Denne tabel er ikke en klokkeren implementation, da primærnøglen salID ikke bliver genereret automatisk af systemet, selv ikke når der oprettes en ny række i tabellen. Denne overvejelse skete på baggrund af, at gruppen mente at det var DBAens opgave at oprette en ny sal, og ikke en automatisk handling fra systemet.

2. Tabellen **Biograf**

Ligesom den forgående er denne heller ikke en direkte implementering, da primærnøglen navn, ikke genereres automatisk af systemet. Det er sket, fordi denne tabel udelukkende indeholder stamdata over de biografer, der er tilsluttet systemet. Grunden til, at der ikke findes et unikt tal ID for denne tabel, er at der ikke er to biografer der har det samme navn, og hvis der skal oprettes en ny biograf i systemet, bliver denne oprettet af DBAen selv, og ikke af systemet.

3. Tabellen **Kategori**

Primærnøglen kategoriID bliver ikke automatisk genereret af systemet, da denne bare er en opslagstabel til entiteten Film. I denne tabel findes der en attribut, hvori man kan vælge kategorien udfra et opslag i tabellen Kategori.

4. Tabellen **FilmMedvirkende**

Denne tabel er lidt speciel. Den er lavet over en mange-til-mange relation mellem entiteterne Film og Medvirkende. Her opretter man en tabel der tager primærnøglen fra den ene tabel og primær nøglen fra den anden tabel, og ligger dem sammen i en samlet tabel. Denne er derfor ikke en klokkeren implementering, da nøglerne ikke bliver automatisk talt op, fordi de er primærnøgler i de to andre tabeller.




5. Tabellen **Postnrby**

Denne tabel er ligesom tabellen Kategori en opslagstabel. I den befinder der sig en liste over alle postnumre og bynavne i Danmark. Udfra det indtastede postnummer slår databasen selv op i denne tabel for at finde det tilhørende bynavn. Så derfor benyttes der ikke en automatisk optælling af nøglen, da denne er fastlagt i tabellen, og ikke må ændres.

Resultatet af gruppens overvejelser omkring disse tabeller, kan ses i næste afsnit. Resten af tabellerne er en direkte overføring af entiteterne fra E/R-diagrammet. Disse overholder alle kravene omkring automatisk generet primærnøgler, samt er der ingen integreret tabeller blandt disse. Gruppen så sig ikke nødsagede til, at lave nogen tabeludeladelse, da vi mente alle tabellerne skulle bruges i vores system.

13.2 Tabelbeskrivelser

Vi vil nu give en beskrivelse for hver af de tabeller vi har arbejdet os frem til. Vi vil ikke beskrive hver enkelt attribut, men kun dem vi finder mest interessante at nævne.

Symbolet  angiver, at det er den unikke nøgle i tabellen og en understregning af en et attributnavn betyder, at det er en fremmednøgle til en anden tabel. Alle primærnøgler er indekseret.

Tabellen **Kunde**

attributnavn	data type	format
<u>kundeID</u>	autonummer	langt heltal
alias	tekst	50
foedselsdag	tekst	50
foedselsmaaned	tekst	50
foedselsaar	tekst	50
<u>postnr</u>	tekst	4
tlf	tekst	12
koen	tekst	50
email	tekst	50
password	tekst	12

Tabel 13.1

En kunde får tildelt et *kundeID*, når kunden oprettes i databasen. *Postnr* er fremmednøgle til tabellen **Postnrby**. Ved ikke at have by som attribut, har vi fjernet den redundans det ville give, hvis man skulle indtast kundens by. *Email* og *Password* er vigtigt i denne sammenhæng, da det kræves at en kunde har begge disse, for at kunne lave en reservation fra Internettet. Email adressen kan ydermere bruges til at sende kunden en ordrebekræftelse.

Tabellen **Billetreservation**

attributnavn	data type	format
<u>PreservationsID</u>	autonummer	langt heltal
forestillingsID	tal	langt heltal
<u>raekkeID</u>	tal	langt heltal
<u>saedeID</u>	tal	langt heltal
kundeID	tal	langt heltal

Tabel 13.2

SaedeID er fremmednøgle til tabellen **Saede**, da man skal kunne finde ud af hvilke sæder der er reserveret til en pågældende reservation.

Tabellen **Forestilling**

attributnavn	data type	format
<u>forestillingID</u>	autonummer	langt heltal
<u>filmID</u>	tal	langt heltal
<u>salID</u>	tal	langt heltal
dato	tal	byte
tid	dato og klokkeslæt	langt klokkeslætsformat

Tabel 13.3

Denne centrale tabel danner relation mellem en reservation og en bestemt film på et bestemt tidspunkt i en bestemt sal. *FilmID* er fremmednøgle til tabellen **Film**. Fremmednøglen *salID* er vigtig, da den binder en bestemt forestilling til en bestemt sal vha. tabellen **Sal**. *Dato* skal angives ved dato-måned og årstal, fx 28-08-1998. *Tid* skal angives med størrelsen timer-minuter, fx 16:30.

Tabellen **Række_Sæde**

attributnavn	data type	format
<u>raekkeID</u>	autonummer	langt heltal
<u>saedeID</u>	Tal	langt heltal
<u>salID</u>	Tal	langt heltal
reserveret	ja/nej	ja/nej

Tabel 13.4

Denne tabel er opstået ved en sammenlægning af entiteterne **Række** og **Sæde**. Dette er gjort fordi klassen **Sæde** indeholdte informationer, der ligeså godt kunne hentes fra klassen **Række**, da det drejede sig om de samme informationer. *RækkeID*, *SædeID* og *SalID* udgør tilsammen den unikke identifikation til denne tabel.

Tabellen **Sal**

attributnavn	data type	format
<u>salID</u>	tal	langt heltal
map	tekst	50

Tabel 13.5

Map skal muligvis angive stien til det sted på web-serverens harddisk, hvor informationerne ligger til en bestemt sals layout.

Tabellen **Biograf**

attributnavn	data type	format
<u>navn</u>	tekst	25
adresse	tekst	50
postnr	tekst	4
tlfnr	tekst	12
faxnr	tekst	12
salantal	tal	3

Tabel 13.6

Denne tabel skal indeholde de pågældende stamdata for en biograf. Vi har valgt at bruge attributten *navn* som primærnøgle, da det er yderst sjældent at to biografer hedder det samme. Vores system indeholder kun én biograf, men man ville let kunne tilføje flere biografer. *Postnr* er fremmednøgle til tabellen **Postnrby**. *Salantal* angiver hvor mange sale der findes i biografen.

Tabellen **Pristype**

attributnavn	data type	format
<u>pristypeID</u>	auto nummer	langt heltal
<u>raekkeID</u>	tal	langt heltal
<u>salID</u>	tal	langt heltal
pris	tekst	8

Tabel 13.7

RaekkeID og *SalID* er begge fremmednøgler til tabellen **Raekke_Saede**. Det er nødvendigt for at kunne finde en bestemt rækkes pris i en bestemt sal.

Tabellen **Film**

attributnavn	data type	format
<u>filmID</u>	autonummer	langt heltal
filmtitel	tekst	50
aargang	tekst	4
spilletid	tal	langt heltal
<u>medvirkendeID</u>	tal	langt heltal
<u>kategoriID</u>	tekst	20
beskrivelse	tekst	1000

Tabel 13.8

I denne tabel skrives en films stamdata. *MedvirkendeID* er fremmednøgle til tabellen **Medvirkende**, da der som regel er flere medvirkende i en film. *KategoriID* er fremmednøgle til tabellen **Kategori**. Dette er desuden også for at undgå redundans.

Tabellen **Kategori**

attributnavn	data type	format
<u>kategoriID</u>	tekst	20

Tabel 13.9

I denne tabel ligger alle de forskellige filmkategorier, fx komedie, action, drama osv.

Tabellen **Medvirkende**

attributnavn	data type	format
<u>medvirkendeID</u>	autonummer	langt heltal



<u>navnID</u>	tal	langt heltal
<u>rolleID</u>	tal	langt heltal

Tabel 13.10

Denne tabel er skabt for at undgå redundans. *NavnID* er fremmednøgle til tabellen **Nanveliste**, som har til formål at skabe en relation til de rigtige navne, til en bestemt film. *RolleID* slår op i tabellen **Rolle**, for at skabe relationen til de roller en medvirkende har i en bestemt film

Tabellen **FilmMedvirkende**

attributnavn	data type	format
filmID	tal	langt heltal
medvirkendeID	tal	langt heltal

Tabel 13.11

Denne tabel skaber mange-til-mange relation mellem tabellerne **Film** og **Medvirkende**. Den indholder begge tabellers unikke ID.

Tabellen **Navnliste**

attributnavn	data type	format
navnID	autonummer	langt heltal
navn	tekst	75

Tabel 13.12

I denne tabel er listet alle navnene på alle implicerede i en film, da en impliceret kan medvirke i flere film. Dette vil mindske redundansen betydeligt ved at oprette denne specielle navneliste.

Tabellen **Rolle**

attributnavn	data type	format
rolleID	autonummer	langt heltal
rolle	tekst	30

Tabel 13.13

I denne tabel er listet alle de former for roller der findes, fx skuespillere, instruktører og forfattere.

Tabellen **Postnrby**

attributnavn	data type	format
postnr	tekst	4
by	tekst	50

Tabel 13.14

Her er listet alle byerne i Danmark samt deres tilhørende postnumre.

13.3 Integritet

Når man udvikler relationelle databaser er der tre krav, som altid skal være opfyldt:

Nøglekrav: et primærnøglefelt skal være entydig eller unik i alle poster i enhver tabel.

Entitetsintegritet: et primærnøglefelt må aldrig være tomt/blankt, dvs. der må ikke være en post i en tabel, hvis primærnøglefelt er tomt.

Referenceintegritet: et fremmednøglefelt i en barn-tabel skal findes i en modertabel som primærnøgle og indeholde samme oplysning. Dvs. fælles felter skal indeholde samme oplysninger.

Gruppen går naturligvis efter at opfylde både nøglekravet og entitetsintegritetsreglen, da man altid skal sørge for dette. Gruppen vil dog ikke konsekvent gennemtvunge referenceintegritet, men kun anvende denne hvor det findes nødvendigt.

13.4 Diskussionen omkring låsning af tabeller

Under design overvejelserne omkring adgang til databasen, stødte vi ind i en mindre diskussion omkring låsning af tabeller. Det er kritisk for databasens effektivitet og integritet, at man finder den optimale måde at låse tabeller på. Låsning er vigtigt idet, at databasen skal kunne håndtere, at flere brugere samtidigt kan hente og gemme data fra via hjemmesiden.

Det må aldrig være muligt for en bruger, at kunne låse en tabel i længere tid, da man derved låser for andre brugere.

Vi diskuterede først en løsning, hvor vi skulle benytte timeout på låsningen.

Metode 1, hvor sæderne er ledige:

1. Send ønske



2. Lås tabel
3. Check og finder ledige
4. Viser valg til bruger
5. Venter på brugervalg
6. Skriv valg
7. Lås op
8. Venter på brugervalg

Metode 1, hvor sæderne *ikke* er ledige:

1. Send ønske
2. Lås tabel
3. Check og finder med optaget
4. Laver ny reservation vha. best fit
5. Viser til brugeren
6. Venter til bruger foretager valg
7. Skriv valg
8. Lås op

I metode 1 er fremgangsmåden med låsning præcis den samme uanset, om sæderne er optaget eller ledige. Brugeren får altid vist sædevalget inden den endelige accept af pladserne skrives til tabellen (hvorefter den låses op). Foretager brugeren ikke noget valg, er systemet nød til at bruge timeout for at låse tabellen op igen. Timeout er derimod en meget dårlig løsning, da man ikke kan opretholde den ønskede effektivitet og integritet af systemet.

Metode 1 blev hurtigt forkastet til fordel for en langt bedre løsning, nemlig metode 2.

Metode 2, hvor sæderne *er* ledige:

1. Send ønske
2. Lås tabellen sæde
3. Checker og finder den ledige
4. Skriver reservation
5. Låser op

Metode 2, hvor sæderne *ikke* er ledige:

1. Send ønske
2. Lås tabellen
3. Checker og finder dem allerede reserveret



4. Lav ny vha. best fit
5. Skriver ny reservation
6. Låser op

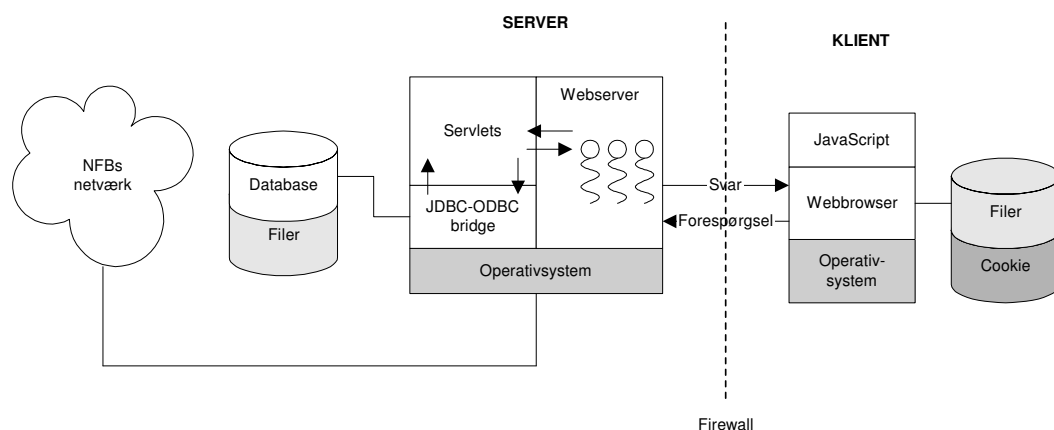
I Metode 2, er der stor forskel på, om sæderne er reserveret eller ej. Fremgangsmåden ved bedste tilfælde, dvs. hvor sæderne ikke er optaget, er langt mere optimal end bedste tilfælde i Metode 1. Grunden til dette er, at vi skriver bestillingen i tabellen med det samme – dvs. vi venter ikke på, at brugeren accepterer valget.

I værste tilfælde, finder systemet alternative sæder og skriver *disse* til tabellen. Derefter vises valget til brugeren, som så får muligheden for at acceptere eller fortryde systemets forslag. Fortrydes der, slettes bestillingen fra tabellen vha. denne algoritme:

1. Send afbestilling
2. Lås tabel
3. Slet sæder
4. Lås tabel op

14. Task Management Component

Denne komponent har til formål at styre de processer, der bliver sat i gang af en eller flere brugere. TMCen i vores system bliver især vigtig, da systemet skal kunne håndtere flere brugerforespørgsler samtidig. TMCens opgaver er at sørge for, at der bliver oprettet en proces (tråd) til hver af de indkommende forespørgsler. For at kunne forstå hvorfor netop vores system skal kunne håndtere flere forespørgsler samtidig, og hvordan webserveren reagerer på brugernes forespørgsler, har vi valgt at visualisere det med følgende figur:



Figur 14.1



14.1 Kort systembeskrivelse

Som det ses på figuren, har vi en klient og en server. Klienten skal her opfattes som Internetbrugeren og serveren er vores webserver. Klienten kan vha. sin browser sende kommandoen 'GET' eller kommandoen 'POST' til serveren, dette kalder vi med ét ord en forespørgsel. Alle forespørgsler skal igennem en firewall for at kunne nå frem til serveren. Serveren kan derefter returnere et svar til klienten. Selve serveren består af fire forskellige dele. Til at modtage forespørgslerne fra brugerne, bruges webserveren, som skal oprette en tråd for hver af de forespørgsler, der kommer fra brugerne af systemet. Disse tråde bliver så knyttet til hver af de servlets, som forespørgselen hører til.

De forskellige servlets henter de oplysninger de skal bruge, igennem JDBC/ODBC-bridgen, der sørger for at konvertere servletens forespørgsel til SQL-sætninger, som kan forstås af selve databasen. Derudover ligger der et operativsystem på serveren.

De fleste hjemmesider på Internettet er statisk opbygget, dvs. at det er på forhånd kodet HTML dokumenter der loades fra webserveren til klienten. En fordel ved dette er, at der ikke er de store vente tider på at HTML dokumentet bliver loadet ind i brugerens browser. En stor ulempe er dog, at det ikke er særligt fleksibelt og at det kræver meget vedligeholdelse af selve hjemmesiden.

Der er tale om en endnu større ulempe, hvis hjemmesidens informationer skal modificeres meget ofte. Dette er netop tilfældet i vores system, da HTML dokumenterne skal ændres alt efter brugerens valg af fx film og tidspunkt og at hjemmesiden skal opdateres hver dag med nye forestillinger m.m.

Derfor er det nødvendigt med et dynamisk modul, der er i stand til at hente de informationer der er relevante i forhold til netop brugerens valg. Denne dynamik kan kaldes Web Content Management [WCM]. Ved brug af WCM, vil hjemmesiden hele tiden være "up-to-date" med informationer, billeder osv., da alle informationer bliver genereret ud fra NFBs database.

Gruppen vil derfor benytte servlets til at opnå denne fleksibilitet i udformningen af hjemmesiden, og JDBC/ODBC-bridgen som "talerør" til kommunikation med databasen.

En mere detaljeret beskrivelse af principperne omkring servlets og JDBC/ODBC-bridge findes i appendiks E samt WCM i appendiks F.

14.2 TMCens rolle i vores system

Da vores system skal kunne bruges af mange samtidige brugere på Internettet, skal TMC sørge for, at hver forespørgsel fra hver bruger får sin egen separate tråd tildelt. Derved undgår man konflikter og kø på serveren. Det er derfor naturligt, at webserveren udgør vores TMC, da det er denne der modtager forespørgslerne fra brugerne.

TMCen har også en anden rolle, nemlig at udføre samtidighedskontrollen til databasen. Dette håndteres dog af DBMSen selv, så dette vil vi ikke beskæftige os nærmere med.

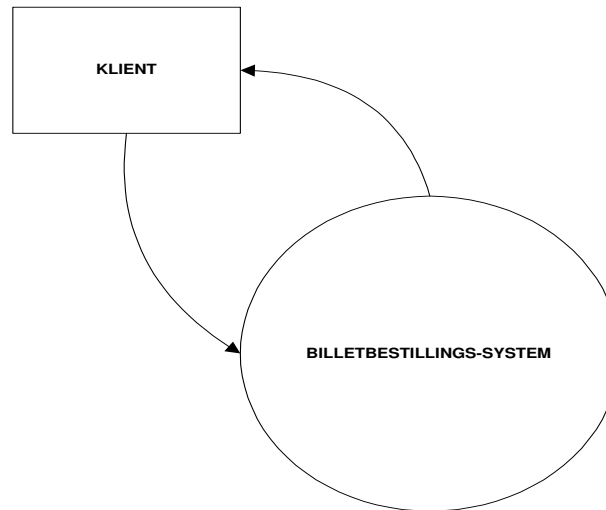
Det vigtige ved TMCen i vores system er, at den kan sætte flere funktioner i gang samtidigt og derved opnå en form for multi-tasking. Derved skaber man en bedre brugervenlighed, da brugeren ikke skal vente i "evigheder" på at få svar på en forespørgsel.



15. External Interface Component

15.1 External Interface Diagram

Figur 15.1



Det er selvsagt, at klienterne ligger udenfor systemet og derfor er klient et eksternt objekt.

Med hensyn til kommunikationen mellem det eksterne objekt og systemet, er der tale om *discrete dataflow*. Data er nemlig tilgængelig på forskellige tidspunkter, som ofte er forbundet med en hændelse, der adviserer systemet om dataenes tilstedeværelse.

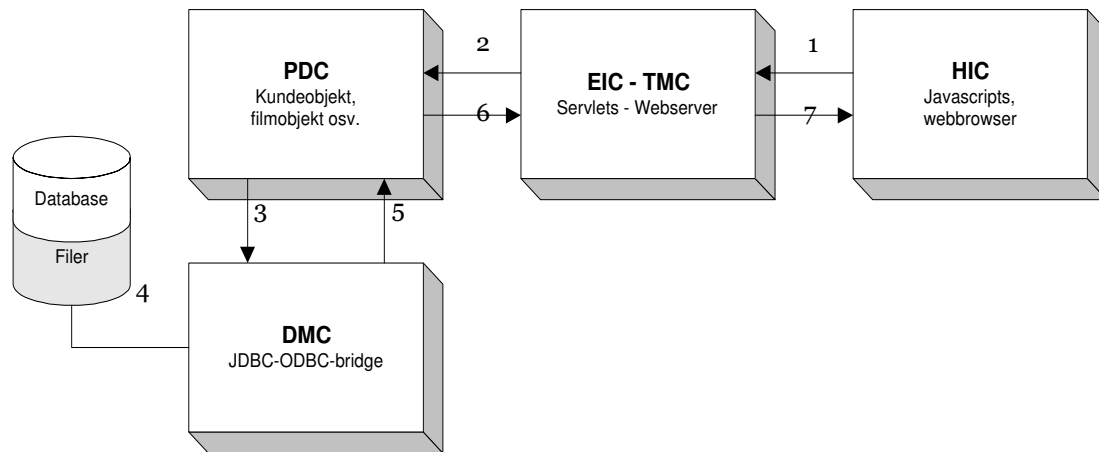
15.1.1 Gruppens overvejselser

Ifølge vores system, vil EICen være gruppens brug af servlets og webserver. De overordnede funktioner hos disse er, at generere HTML dokumenter ud fra brugerens valg af informationer fra bestillingsdokumentet og webserveren skal oprette en tråd for hver af de forespørgsler der modtages fra klienterne. Gruppen havde problemer i starten, med at definere EICens egentlige formål under vores system. Vi kunne se, at vi ikke kun kunne benytte et informationsflow mellem HICen og PDCen, da det ikke var en direkte manipulation der skete der. I vores tilfælde modtager EICen data fra klienterne, og ud fra dette, genereres der en HIC med de tilhørende/ønskede informationer fra PDCen. Dette giver samtidig EICen en rimelig central rolle i gruppens system. Normalt ser man ikke en så central rolle for EICen, men i vores tilfælde, pga. kompleksiteten og brugen af servlets, vil dette ikke være tilfældet. Hele systemet er baseret på den information der bliver genereret af servlets, dermed EICen, og systemet er faktisk generelt sagt, bygget op omkring EICen og ikke, som i andre tilfælde, over PDCen og HICen. I vores tilfælde får disse kun en "bi-rolle" i systemet, og EICen for den centrale "hovedrolle".



16. Komponenternes samspil

Vi har nu oprettet de komponenter vi mener skal til, for at vores system kan blive så fleksibelt som muligt. For at give et bedre overblik over hvad de forskellige komponenter skal indeholde og hvordan de skal samarbejde i vores system, er følgende figur udarbejdet:



Figur 16.1

Når en bruger ønsker at bestille biografbilletter skal vedkommende først ind på NFBs hjemmeside. Dette gøres ved, at brugeren taster NFBs URL ind vha. sin webbrowser. Herefter vil hver forespørgsel gå igennem 7 interaktionstrin mellem komponenterne, som det er vist på figur 16.1.

(1) Som det første modtager EICen en forespørgsel om at levere HICens indhold ud til brugeren. Dette gøres ved at TMC opretter en tråd, som skal sætte den rette servlet i gang med dette arbejde. (2) Da HICens indhold er dynamisk skal servleten kalde de nødvendige objekter i PDC komponenten der indholder de rette informationer.

(3) PDCen objekter er gemt i en relationel database, så det er derfor nødvendigt at konvertere alle forespørgslerne til SQL-sætninger. Dette klares af DMCen, som vha. en JDBC/ODBC-bridge omkonverterer funktionskaldene til SQL-sætninger for derpå at få adgang til (4) databasen.

(5,6,7) I databasen ligger alle de dynamiske informationer der skal bruges til HICen. Dvs. filminformationer, tidspunkter, billeder, sallayout osv. og informationerne går nu turen den anden vej, for at kunne returnere til servleten der skulle bruge informationerne for til sidst at kunne levere de forespurgte HIC indhold.

Systemets komponenter er i en lidt speciel rolle i vores system, da der er tale om dynamisk genereret indhold. Derfor var der i gruppen en del diskussion komponenternes placering i systemet, men vi mener dog at have fundet den mest optimale.



17. Afrunding af OOD

Fra analyse til design er der foretaget visse ændringer og justeringer. Gruppen mener, at disse er foretaget til det bedre og at design nu er en afsluttet aktivitet. De fire af de fem anvendte komponenter opfylder alle Coad & Yourdons standard for objektorienteret design, men vi har dog valgt at fylde et selvstændigt kapitel, som omhandler tabeldesign, ind i designdelen. Dette er gjort, da vi mener at Coad & Yourdons standard for beskrivelse af relationelle databasetabeller i DMCen ikke er fyldetsgørende nok, og man er også blevet nødt til at overveje klokken versus ikke klokken implementering.

En anden vigtig overvejelse, er de designmæssige krav til en client/server løsning. De teoretiske løsninger og de overvejelser gruppen har gjort sig omkring dette, kan læses i appendiks F.

Som det ses i designdelen er E/R anvendt som springbræt til anvendelse af EER, som er meget mere udtryksfuldt end E/R.

Umiddelbart mener vi selv, at der er grundlag nok for at gå videre over i implementeringsaktiviteten, hvor databasen og hjemmesiden skal udarbejdes, og der skal kodes servlets.



Indholdsfortegnelse

18. DESIGNOVERVEJELSER	3
18.1 UDELADELSE AF HEIGHT OG WIDTH I -TAGS	3
18.2 VISNING AF RESERVEREDE SÆDER	3
18.3 RESERVERING AF SÆDER	3
18.4 JDBC/SERVLET LØSNING	4
18.5 FORBEDRING AF SVARTIDER	4
19. SERVLETOVERSIGT	5
19.1 OVERSIGT	5
19.2 SERVLET-GENNEMGANG	5
<i>CancelSeatServlet.java</i>	5
<i>CheckSeatsServlet.java</i>	6
<i>GreetUserServlet.java</i>	6
<i>LoginServlet.java</i>	6
<i>LogoutServlet.java</i>	6
<i>MovieActorInfoServlet.java</i>	6
<i>MovieInfoServlet.java</i>	7
<i>MovieShowsServlet.java</i>	7
<i>OrderSeatsServlet.java</i>	7
<i>ReservationsServlet.java</i>	7
<i>ShowHallServlet.java</i>	7
<i>ShowOrdersServlet.java</i>	8
<i>TryLoginServlet.java</i>	8
20. KLASSEOVERSIGT	9
20.1 OVERSIGT	9
20.2 KLASSE-GENNEMGANG	9
<i>BilletPris.java</i>	9
<i>BilletReservation.java</i>	9
<i>CookieCheck.java</i>	10
<i>Film.java</i>	10
<i>Forestilling.java</i>	10
<i>JDBC.java</i>	11
<i>Kategori.java</i>	11
<i>Kunde.java</i>	11
<i>Sal.java</i>	12
<i>SalLayout.java</i>	12
21. BESKRIVELSE AF ESSENTIEL KODE	14
21.1 REDIRECT TIL ANDEN HTML-SIDE	14
21.2 BRUG AF COOKIES	14
21.3 FORBINDELSE TIL DATABASEN	15
21.4 SQL-SÆTNINGER	15
22. AFRUNDING AF IMPLEMENTERING	17



Denne del indeholder en del af dokumentationen for implementeringsaktiviteten. Gruppen har valgt, at den skal omhandle nedenstående:

- Fremhævelse af vanskelig læsbar kode
- Fremhævelse af central kode
- Moduloversigt
- Klassebeskrivelse

Selve koden findes i appendiks G, men vi mener at det er vigtigt at fremhæve kodedele, som umiddelbart er vanskelig at læse og forstå. Derudover er det også vigtigt at fremhæve kode, som er central og som fremhæver essensen af programmet.

For at gøre programdokumentationen overskuelig er der udarbejdet en moduloversigt samt en klassebeskrivelse. Klassebeskrivelserne er opbygget sådan, at de indeholder følgende:

- Klassens navn
- Klassens formål
- Klassens attributter
- Klassens metoder

En del af klassebeskrivelserne findes naturligvis også i analysedokumentet, men visse klasser kan være ændret eller nye klasser kan være fundet nødvendige at anvende efter analysedokumentets udarbejdelse. Derfor er det nødvendigt med en omtale af disse.

Gruppen har på nuværende tidspunkt, i samråd med NFB, besluttet at udvikle systemet sådan, at man rent faktisk skal kunne se de sæder i salen der allerede er optaget. På denne måde fik vi løst en masse problemer.

Dette er med til at yde brugeren en meget bedre service, fordi de på den måde, selv kan bestemme, præcis hvor i salen de skal sidde.

18. Designovervejelser

Både før og under selve implementeringen havde vi en række designovervejelser om måden at implementere visse ting på, og disse gennemgås i det følgende.

18.1 Udeladelse af *HEIGHT* og *WIDTH* i **-tags

Normalt, når man angiver et billede i et HTML-dokument, angives også højden og bredden på billedet vha. af de såkaldte *HEIGHT* og *WIDTH* attributter. Et billede på 100x100 pixels ville fx blive angives således:

```
<IMG SRC="/graphics/testpic.gif" WIDTH=100 HEIGHT=100>
```

** er selve den kode der henter et billede og *SRC* attributten angiver navnet og evt. stien på billedet.



Grunden til, at man angiver de forømtalte HEIGHT og WIDTH attributter er, at browseren på den måde kan indsætte billedets dimensioner og dermed formattere HTML-dokumentet uden at billedet nødvendigvis behøver at være fuldt hentet. Angiver man *ikke* HEIGHT og WIDTH, kan teksten først formatteres endeligt, når hele billedet er hentet.

Vi undlod at bruge disse attributter visse steder i vores servlets, da vi ellers skulle have oprettet et Image-objekt, der først skulle hente billedet, derefter finde dets dimensioner og tilsidst skrive disse ind i den genererede HTML. Vi mener, at dette ville tage længere tid, og desuden er det mere besværligt.

18.2 Visning af reserverede sæder

Vi valgte at vise alle reserverede sæder, selvom det var uønsket fra NFBs side. Grunden til, at vi viser hvilke sæder der er reserveret er, at vi på den måde meget nemmere kan oprette en reservation. Vi behøver ikke tjekke om sæderne er ledige, da de vil være det i 99% af tilfældene. Hvis sæderne dog alligevel ikke er ledige, bliver klienten nødt til at gå tilbage og prøve engang til. Dette vil kun ske, hvis en anden bruger har nået at reservere sædet inden billedet blev opdateret. Vi har hermed helt løst det store problem, der ville være i at programmere en 'best fit' metode.

18.3 Reservering af sæder

Vi valgte at ændre de primære nøgler i **billetreservation**-tabellen, da vi på den måde ikke behøvede at tjekke en reservering inden den skrives til databasen. Ved at ændre primærnøglerne til at være på **forestillingID**, **raekkeID** og **saedeID** opnår vi, at en SQLException bliver kastet, hvis vi prøver at reservere en allerede eksisterende reservation. Vi fanger (catch) denne Exception og fortæller brugeren, at det ikke er muligt at reservere de valgte sæder.

Vi har valgt at slette **raekke_saede** tabellen, da den er overflødig. I stedet bruger vi **billetreservation**-tabellen til at gemme sæderne i, da dette er mere logisk.

18.4 JDBC/Servlet løsning

Vi havde nogle problemer med at finde ud af, hvordan vi skulle forbinde Servlets med databasen på den bedst mulige måde. Vi ville gerne have en overordnet klasse der kunne oprette og nedlægge forbindelsen til databasen og alle Servlets skulle så bruge denne klasse når de skulle læse eller skrive fra/til databasen. Det var dog ikke helt nemt at finde ud af, hvordan det skulle gøres, da de fleste eksempler ikke brugte en JDBC/Servlet løsning. I vores endelige design, har vi dog valgt, at lade hver Servlet håndtere deres egen adgang til databasen, da vi mener det gør hver enkelt Servlet mere overskuelig.

18.5 Forbedring af svartider

Et andet problem var, at få de forskellige **klasser**, fundet i OOD, til at bruge databasen på den bedste og hurtigste måde. I begyndelsen lod vi hver klasse, fx **Film** eller **BilletReservation**, læse deres egne attributter fra databasen, dvs. at vi blev nødt til at oprette et Film-objekt, hvis vi skulle bruge navnet på filmen (attributten *filmtitel*). Det virkede også perfekt nok, men hastigheden var, som



forventet, ikke tilfredstillende, da hvert objekt sendte deres egne SQL-sætninger til databasen. I første omgang lavede vi en lille cache i vores objekter, dvs. vi tjekkede, om vi skulle bruge det objekt, der allerede var hentet fra databasen ved at gemme fx *FilmID* i Film-objektet og på den måde undgå at læse den samme fra databasen flere gange. Det forbedrede hastigheden væsentligt, men vi mente stadigvæk ikke, det var tilfredstillende. I vores endelige implementation af servlets, der bruger mange klasser, valgte vi i stedet at lave nogle avancerede SQL-sætninger, der bruger INNER JOIN (sammenlægning af flere tabeller) til at hente de nødvendige informationer på tværs af tabellerne. Dette er en mindre objektorienteret implementering, men den er meget hurtigere end de forrige.

19. Servletoversigt

Vores system kører på en gratis version af Apache serveren (<http://www.apache.org/>). Denne server har dog ikke muligheden for at bruge Java servlets direkte. Softwareproducenten Live Softwares JRun Servlet Engine 2.2a udbygger forskellige servere, deriblandt Apache, så det bliver muligt for serveren at kalde servlets. Nogle servere har direkte support for servlets, fx Suns Java Web Server, men da den downloadede version var meget ustabil og langsom (da en stor del Java implementeret), valgte vi ikke at bruge den. I det efterfølgende ses en oversigt over alle de, i programmet, anvendte servlets og senere i dette kapitel gennemgås hver servlet mere detaljeret.

19.1 Oversigt

CancelSeatServlet.java
CheckSeatsServlet.java
GreetUserServlet.java
LoginServlet.java
LogoutServlet.java
MovieActorInfoServlet.java
MovieInfoServlet.java
MovieShowsServlet.java
OrderSeatsServlet.java
ReservationsServlet.java
ShowHallServlet.java
ShowOrdersServlet.java
TryLoginServlet.java

19.2 Servlet-gennemgang

CancelSeatServlet.java
Formål



Sletter en reservation fra databasen og omdirigerer side til ReservationsServlet for at få alle tilbageværende reservationer.

Metoder

```
doGet (HttpServletRequest req, HttpServletResponse res)
```

CheckSeatsServlet.java**Formål**

Tjekker om sæder er ledige og skriver en reservation til databasen, hvis de er.

Metoder

```
init (ServletConfig config)
destroy ()
doGet (HttpServletRequest req, HttpServletResponse res)
```

GreetUserServlet.java**Formål**

Byder en bruger velkommen, når han eller hun logger ind på systemet.

Metoder

```
doGet (HttpServletRequest req, HttpServletResponse res)
```

LoginServlet.java**Formål**

Tilføjer en ny bruger til databasen, hvis 'NFB Navn' er ledigt. Ellers vil denne servlet returnere et 'sorry'-response.

Metoder

```
init (ServletConfig config)
doGet (HttpServletRequest req, HttpServletResponse res)
```

LogoutServlet.java**Formål**

Logger ud af systemet ved at reinitialisere cookie.

Metoder

```
doGet (HttpServletRequest req, HttpServletResponse res)
```

MovieActorInfoServlet.java**Formål**

Finder en specificeret medvirkende i alle film.

Metoder

```
init (ServletConfig config)
destroy ()
doGet (HttpServletRequest req, HttpServletResponse res)
insertActors (PrintWriter out)
insertParts (PrintWriter out)
```

MovieInfoServlet.java**Formål**

Finder informationer om en specificeret film.

**Metoder**

```
doGet (HttpServletRequest req, HttpServletResponse res)
public void doActorsTable (PrintWriter out)
public String doMovieSelector ()
```

MovieShowsServlet.java**Formål**

Henter information om movieshows fra en Access-database, som kører med en ODBC/JDBC-bridge

Metoder

```
init (ServletConfig config)
destroy ()
doSelector (java.io.PrintWriter out)
getMovieTitle (int pMovieID)
findMovieShows (java.io.PrintWriter out)
makeTimeOfShowLink (java.sql.Time t, int fID)
doGet (HttpServletRequest req, HttpServletResponse res)
```

OrderSeatsServlet.java**Formål**

Opdaterer databasen med en evt. brugerbetaling.

Metoder

```
init (ServletConfig config)
destroy ()
doPost (HttpServletRequest req, HttpServletResponse res)
```

ReservationsServlet.java**Formål**

Finder alle reservationer for en bruger, som er logget ind, og genererer HTML.

Metoder

```
doGet (HttpServletRequest req, HttpServletResponse res)
```

ShowHallServlet.java**Formål**

Opsætter et dokument med udvidet filminformation og anvender SalLayout-appletten til at vælge sæder.

Metoder

```
init (ServletConfig config)
doGet (HttpServletRequest req, HttpServletResponse res)
```

ShowOrdersServlet.java**Formål**

Viser alle reservationer for den bruger der er logget ind.

Metoder

```
init (ServletConfig config)
destroy ()
doGet (HttpServletRequest req, HttpServletResponse res)
formatTime (java.sql.Time atTime)
formatDate (int atDate)
```

**TryLoginServlet.java****Formål**

Søger gennem databasen ved at bruge 'NFB Navn' og 'Kodeord' som parametre. Hvis brugeren ikke findes, får han eller hun et 'do-you-want-to-register'-response. Ellers vil brugeren få adgang.

Metoder

doPost (HttpServletRequest req, HttpServletResponse res)

20. Klasseoversigt

I det efterfølgende ses en oversigt over alle de i programmet anvendte klasser, og senere i dette kapitel gennemgås hver klasse mere detaljeret.

20.1 Oversigt

BilletPris.java
BilletReservation.java
CookieCheck.java
Film.java
Forestilling.java
JDBC.java – parent klasse
Kategori.java
Kunde.java
Sal.java
SalLayout.java – applet

20.2 Klasse-gennemgang

BilletPris.java**Formål**

Håndterer adgangen til 'billetpris' tabellen.

Metoder



```
Billetpris(int fID)
getSalID()
getPrice(int fRow)
```

Attributter

```
int atSalID
int[] atPrices
```

BilletReservation.java**Formål**

Håndterer adgang til 'Billetreservation'.

Metoder

```
getForestillingID()
getKundeID()
getSuccess()
```

Attributter

```
int forestillingID = -1;
int kundeID = -1;
boolean dSuccess = false;
int atReservationsID;
int atForestillingID;
int atKundeID;
```

CookieCheck.java**Formål**

Læser cookies for at finde ud af, om en bruger har logget ind.

Metoder

```
CookieCheck(HttpServletRequest req)
isLoggedIn()
getKundeID()
```

Film.java**Formål**

Håndterer adgangen til 'film' tabellen.

Metoder

```
getID()
getTitle()
getYear()
getLength()
getDescription()
getPicture()
getCategoryID()
```

Attributter

```
int ID;
String atYear
String atTitle
int atLength
String atDescription
int atCategoryID
String atPicture
```

Forestilling.java**Formål**

Håndterer adgang til 'Forestilling'.

**Metoder**

```
Forestilling(int fID)
int getID()
getFilmID()
getTime()
getFormattedTime()
getDate()
getFormattedDate()
```

Attributter

```
int ID;
int atFilmID;
java.sql.Time atTime;
int atDate;
int atSalID;
```

JDBC.java**Formål**

Denne klasse indeholder forskellige ofte brugte metoder, fx forbindelse til databasen og hentning af reserverede sæder i en sal.

Metoder

```
JDBC()
finalize()
findKunde(String fAlias)
getReservedSeats(int fForestillingID)
printSQLException(SQLException ex)
```

Attributter

```
Connection con
```

Kategori.java**Formål**

Håndterer adgangen til 'kategori' tabellen.

Metoder

```
Kategori(int fID)
getID()
getCategory()
```

Attributter

```
atCategoryID
atCategory
```

Kunde.java**Formål**

Håndterer adgang til 'Kunde'-tabellen.

Metoder

```
Kunde()
Kunde(int fID)
Kunde(String fAlias, String fPassword)
isLoading()
getID()
getAlias()
setAlias(String fAlias)
getPassword()
setPassword(String fPassword)
getBirthDay()
setBirthDay(String fBirthDay)
getBirthMonth()
```




```

setBirthMonth(String fBirthMonth)
getBirthYear()
setBirthYear(String fBirthYear)
getPostalCode()
setPostalCode(String fPostalCode)
getPhone()
setPhone(String fPhone)
getGender()
setGender(String fGender)
getEmail()
setEmail(String fEmail)
setAll(String fAlias, String fPassword, String fBirthDay, String fBirthMonth, String fBirthYear,
String fPostalCode, String fPhone, String fGender, String fEmail, String fCardHolder, String
fCardNumber, fCardMonth, fCardYear)
addToDatabase()

```

Attributter

```

int atKundeID;
String atAlias;
String atPassword;
String atBirthDay;
String atBirthMonth;
String atBirthYear;
String atPostalCode;
String atPhone;
String atGender;
String atEmail;
String fCardHolder;
String fCardNumber;
String fCardMonth;
String fCardYear;

```

Sal.java**Formål**

Håndterer adgangen til 'sal' tabellen.

Metoder

```

Sal(int fID)
getID()
getMap()

```

Attributter

```

int ID
int atMap

```

SalLayout.java**Formål**

Dette er en applet til at vise og vælge sæder i en sal. Denne applet bruges af **to** forskellige servlets: ShowHallServlet og CheckSeatsServlet. Når den loades gennem ShowHallServlet, vil appletten være i stand til at vælge/fravælge sæder og derefter sende brugeren til en given adresse (URL). Når man bruger den anden servlet (CheckSeatsServlet), viser appletten kun de valgte sæder og gør det umuligt at ændre de valgte sæder eller sende information fra appletten.

Metoder

```

init()
destroy()
paint(Graphics g)
update(Graphics g)
drawTopGreenPanel(Graphics g)
drawTopGrayPanel(Graphics g)

```



```

ImagemapRectangle findSeat(MouseEvent e)
ImagemapRectangle findSeat(int Column, int Row)
findButton(MouseEvent e)
mouseMoved(MouseEvent e)
Map()
Map(int width, int height, int space)
getSpace()
getDimension()
setColor(Color color)
getColor()
setPrice(int price)
int getPrice()
getRects()
doMap(int x, int y, int columns, int rows, int startColumn, int startRow)
drawMap()
drawReservedSeats()
drawSeats()
sendClicked()
clearClicked()

```

21. Beskrivelse af essentiel kode

I det følgende vil vi beskrive noget af det mest centrale og vigtigste af vores kode.

21.1 Redirect til anden HTML-side

I visse tilfælde kan det være en fordel at guide brugeren over til en allerede eksisterende HTML-side, fx hvis der er opstået en fejl såsom, at brugeren prøver at bestille billetter uden at være logget på systemet.

Herunder ses den kode, der bruges til at omdirigeringen:

```

1 String location = "/no_user_found.html";
2
3 String scheme = req.getScheme();
4 String host = req.getServerName();
5 int port = req.getServerPort();
6
7 String redirectString = scheme + "://" + host;
8 if (port != 80) {
9     redirectString += ":" + port;
10 }
11
12 redirectString += location;
13 res.sendRedirect(redirectString);

```

Første laver vi en absolut URL ved at finde navn og port på serveren. Derefter bruger vi metoden **sendRedirect** fra brugerens Response-objekt (se linie 13) med den genererede URL. Response-objektet er en parameter fra en servlets servicemethode (fx **doGet** eller **doPost**).

21.2 Brug af Cookies

En Cookie bruges til at gemme små mængder af tilstandsinformationer, fx brugerkonfigurationer eller til sikkerhedskontrol på lavniveau.

Vi bruger Cookies, for at finde ud af, om en bruger er logget ind i vores system, fx er det ikke muligt at lave en billetreservation eller afbestille billetter, hvis man ikke



er logget på. Der findes forskellige former for cookies, fx persistente cookies og temporære cookies.

Persistente cookies bliver gemt på brugerens harddisk og kan derfor læses efter brugeren har haft lukket browseren og/eller slukket computeren, hvorimod den form for temporære cookies, som vi bruger, slettes hver gang man lukker browseren.

```
1 Kunde k = new Kunde("Peter", "uuijklk")
2 if (k.isLoaded()) {
3     Cookie c = new Cookie(CheckCookie.LOGIN, "" + k.getID())
4     c.setComment("NEFB Login Cookie")
5     res.addCookie(c)
6 }
```

I ovenstående kode tilføjer vi en Cookie til brugerens svar, hvis han/hun blev fundet i vores database.

I linie 1 søger vi gennem vores database ved at oprette et objekt med de informationer, brugeren gav os. I linie 2 tester vi om brugeren var fundet, da **isLoaded** vil returnere true, hvis han/hun var fundet. I linie 3 opretter vi en ny cookie med navnet "LogIn" (vores statiske konstant LOGIN fra CookieCheck klassen), og med kundeID'et som værdi (castet til en string). I linie 4 tilføjer vi en kommentar til cookien, så browseren kan vise brugeren *hvad* cookien indeholder. I linie 5 tilføjer vi endeligt cookien til brugerens response objekt (res).

Det er vigtigt, at man tilføjer cookien til response objektet *før* man begynder at generere HTML, da cookies sendes i headeren (dvs før evt. HTML-kode).

21.3 Forbindelse til databasen

Da vi bruger en ODBC/JDBC-bridge til at forbinde vores Access-database med vores Java implementering, skal vi første finde selve ODBC/JDBC-driveren, og derefter oprette forbindelsen til databasen. Ved hjælp af forskellige metoder fra java.sql pakken, oprettes forbindelsen på følgende måde:

```
1 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
2 Connection con = DriverManager.getConnection(databaseURL, DBuser, DBpassword);
```

DriverManageren opretholder en liste af registrerede Driver-klasser. I linie 1 loader vi en Driver-klasse explicit uden at afhænge af nogen form for ekstern setup. Linie 2 tester alle loadede drivere og bruger den driver, den givne URL virker på. Herpå oprettes et Connection-objekt der har adgang til databasen. Dermed kan fx SQL-sætninger sendes til den.

21.4 SQL-sætninger

Vi bruger mange forskellige SQL-forespørgelser i vores kode, fx når vi skal hente oplysninger om en film eller når vi skal indsætte en ny billetreservation i vores database. Vi har i det nedenstående givet eksempler på SELECT, INSERT og DELETE SQL-sætninger. Alle variabler starter med 'm_', og '?' symboliserer en udefineret variabel, der tildeles en værdi senere (før SQL-sætningen sendes til databasen).

SELECT-sætning, der henter information om en film med givet filmID (m_ID)



```
SELECT filmtitel, aargang, spilletid, beskrivelse, billed, kategoriID
FROM film
WHERE filmID = m_ID
```

SELECT-sætning, der henter alle forestillinger på en given dag ved brug af INNER JOIN.

```
SELECT forestilling.forestillingID, forestilling.tid, film.filmtitel, kategori.kategori
FROM kategori INNER JOIN (film INNER JOIN forestilling ON film.filmID = forestilling.filmID) ON
kategori.kategoriID = film.kategoriID
WHERE forestilling.dato = m_dageFraNu AND film.filmID = m_filmID
ORDER BY film.filmtitel, forestilling.tid ASC
```

DELETE-sætning, der sletter en billetreservation med givet forestillingID, kundeID, række og sæde.

```
DELETE FROM billetreservation
WHERE (forestillingID = m_forestillingID AND
      kundeID = m_kundeID AND
      række = m_række AND
      sæde = m_sæde)
```

INSERT-sætning, der indsætter en reservation med givne oplysninger om forestillingID, række, sæde og kundeID.

```
INSERT INTO billetreservation(forestillingID, række, sæde, kundeID)
VALUES (m_ForestillingID, ?, ?, m_KundeID)
```

22. Afrunding af implementering

Vi vil nu anse implementeringsaktiviteten som værende afsluttet. Vi mener at have afdækket de mere teoretiske områder af implementeringen, og med både de teoretiske implementeringsovervejelser samt den praktiske kodning på plads, kan vi nu bevæge os i retning af testaktiviteten.

Desværre er vi ikke helt tilfredse med vores implementering af appletten. Grunden til dette er, at vi vha. en pure JDBC-driver kunne have opnået en meget renere implementering, da vi med denne ville kunne forbinde direkte til databasen og på den måde ville kunne gå udenom en servlet. Det var desværre ikke muligt at få fat i en pure JDBC-driver, som kunne forbindes til en Access-database, som vi anvender.

Den endelige implementering anvender en servlet, der læser alle de reserverede sæder, og derefter sender dem til appletten vha. parametre (<PARAM>-tags). For yderligere information om servlets og JDBC, se appendiks E. Alt koden findes i appendiks G.



Indholdsfortegnelse

23. TESTDOKUMENT	3
24. TESTRAPPORT	6
24.1 FORMÅL	6
24.1.1 Referencer	6
24.1.2 Omfang og begrænsninger	6
24.2 ACCEPTTESTENS TESTEMNER	6
24.3 TESTDESIGN	7
25. TESTIMPLEMENTERING	9
25.1 STRESSTEST	9
25.2 VOLUMETEST	9
25.3 SIKKERHEDSTEST	10
25.4 KOMPATIBILITETSTEST	10
25.5 FEJLBEHANDLING	10
25.6 BLACKBOX-TEST	11
25.6.1 Navigationspanel	11
25.6.2 Login proceduren	12
25.6.3 Information indhentning	13
25.6.4 Billet bestillingsproceduren	14
25.6.5 Betalingsproceduren	15
25.6.6 Afbestillingsproceduren	16
25.6.7 Andet	16
25.6.8 Afrunding af Blackbox test	16
25.7 BRUGERTEST	16
25.7.2 Udførelse af brugertest	17
25.7.3 Forslag til program- og designændringer	17
25.7.4 Konklusion på brugertesten	18
25.8 UDFØRELSE AF TEST	19



Når man nu har analyseret, designet og implementeret sit system, skal man på en eller anden måde finde ud af, om systemet fungerer efter hensigten.

Der er flere grunde til, at det er vigtigt at teste. Nedenfor gives nogle grunde:

Mennesker laver fejl. Derfor skal tests udvikles med det formål at jage fejl. Jo flere fejl man finder, jo bedre.

Det tager tid at finde fejl. Derfor skal tests inkluderes i projektplanlægningen med realistiske tider. Erfaringer viser, at på større softwareprojekter har 50% af tiden været brugt til test. Derfor er *systematisk* testplanlægning med til at nedsætte dette tal.

Fejl skal findes så tidligt som muligt. Det er dårlig økonomi at udskyde alle testaktiviteter til de sidste faser.

Udviklere er blinde over for egne fejl. Derfor er det hensigtsmæssigt at bemande et projekt således, at ingen skal teste sit eget program alene.

Selv gode tests kan ikke redde et dårligt program. Derfor skal tests indbygges løbende.

Hvis man skal teste systematisk, skal man starte med at gøre sig klart, *hvad* det er man skal teste. Man skal samtidig gøre sig op med sig selv, hvorfor man tester, og hvor vigtigt det er, at det færdige produkt ikke fejler.

For at gøre testen så systematisk som muligt, kan man med fordel opdele testarbejdet i mindre og mere overskuelige aktiviteter. Man udvælger omhyggeligt testdata til hver enkelt test, og forudsiger derefter resultaterne. Dette kaldes med andre ord en *blackbox-test*, og er en udmærket teknik til at finde ud af, om systemet "gør det rigtige".

Det er også vigtigt, at man ikke er alene om at både planlægge og udføre en test. Man risikerer kun at få sine egne synsvinkler med, mens andre involverede kan bidrage med "andre øjne".

Man bør køre en passende mængde af tests, og tage resultaterne med ind til skrivebordet. Man kontrollerer dem ved at sammenligne resultatet med det forudsagte resultat.

Man kan også køre en *whitebox-test*, hvor man indbygger IF-sætninger for at se, om systemet kommer ud med det rigtige svar. Kort sagt er en *whitebox-test* en test, af den logiske struktur af implementationen, og er derfor rettet mod selve programstrukturen.

Gode tests kan også ses som en del af en kvalitetsstyringspolitik. Man tester løbende og planlægger nogle gode tests. Dvs. man indbygger kvalitet i programmet løbende. Dårlige erfaringer har vist, at det ikke kan betale sig at vente til sidst med at teste, da man så skal til at "lægge kvalitet på" tilsidst, og at dette altid ender med noget rod.

Så en god kvalitetspolitik er, at indbygge kvalitet i programmet løbende.

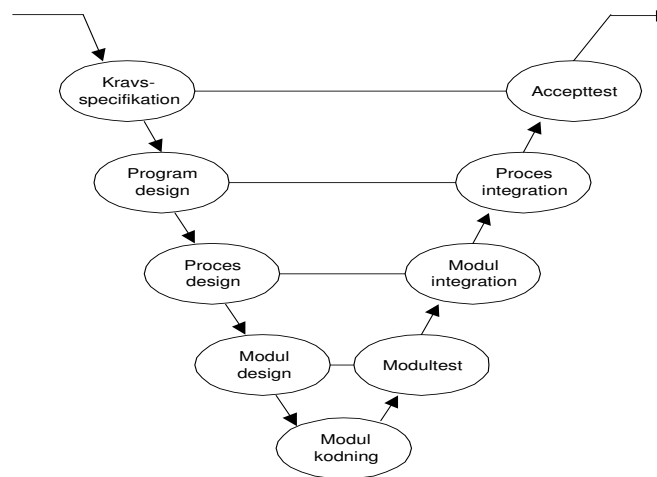
23. Testdokument

Formålet med dette testdokument er, at strukturere den testgennemgang, som gruppen vil udføre på det udarbejdede programmel. Dette gøres for systematisere



gruppens testgrundlag, så vi har en vejledning i testning, og dermed ved hvad der skal testes og hvordan, så gruppen undgår at teste ”med hovedet under armen”. Derfor vil vi planlægge testen nøje.

Til brug af testplanlægning vil gruppen benytte SPU’s (Struktureret Program Udvikling) ”V”-model¹³ som fremgangsmåde. ”V”-modellen ser ud som følgende:



Figur 23.1

Accepttest

Som man kan se udfra figuren hører kravspecifikationen sammen med acceptttesten. Det skal ses i lyset af, at acceptttesten er den sidste test man kører på systemet, hvor man holder kravspecifikationen sammen med det endelige system. Brugerens og gruppens krav til systemet kan ses i Del II, afsnit 7.2.

Procesintegration

Her samler man trinvis processerne, så man kan checke kommunikationen mellem disse. Gruppen vil ikke benytte sig af denne testprocedure, da vores processer udspringer fra serveren. Denne server er et allerede udarbejdet program, hvor gruppen kun har mulighed for at teste outputtet fra denne.

Modulintegration

Der er to muligheder for modulintegration, *bottom up* eller *top down*.

Bottom up: Her tester man modulerne fra bunden af og op. Man starter med moduler, der ikke kalder andre moduler og går opad i systemet. Ulempen ved denne metode er, at man først kommer til at teste hovedmodulet meget sent i testperioden.

Top down: I top down er det det modsatte der sker. Her tester man fra oven og nedefter. Fordelen er, at man fra starten kan teste hovedmodulet, dog med simulerede stubbe. Ulempen er, at man først kan teste et modul fuldt ud, når det

¹³ Udleveret notat: ”Vejledning i Softwaretest”



underliggende modul er lavet.

Gruppen vil benytte top down, da vi tester ud fra serveren i systemet, og derefter kobler modulerne på efterhånden som de er lavet. Under integrationstest findes der igen to muligheder, nemlig trinvis integration og samlet integration. Ved trinvis integration kobler man modulerne på et efter et. Ved samlet integration kobler man alle modulerne på samtidig. Denne metode kaldes også "Big bang" og kan ikke anbefales. Gruppen benytter da også trinvis integration.

Modultest

Her tester man hvert enkelt modul separeret fra hinanden. Dette er dog en langvarig proces, men det kan betale sig, da moduler der er gennemtestet separat, som regel ikke volder store problemer ved en evt. samling. Gruppen benytter sig da også i nogen grad af denne testform, for at sikre, at modulerne fungerer efter hensigten før de kobles på systemet.

Der findes også fire forskellige testteknikker man kan benytte sig af, hvilke er listet som følger:

Interaktiv testkørsel

Her indtaster testkøreren manuelt de testinput, som man ønsker testet på outputtet. Dette sammenlignes så med det ønskede output.

Automatisk testkørsel

Denne metode kan benyttes, hvis man ønsker at udvikle et bestemt testprogram til systemet. Dette kan gøres helt eller delvis automatisk ved at køre en simpel kommando, der sætter testen i gang. Hvis der er tale om en fuldautomatisk testkørsel vil det udarbejdede testprogram selv lave sammenligningerne mellem det output programmet leverer, og det output man ønsker.

Testinstrumentering

Hvis man ønsker en dybere indsigt i hvad programmet foretager sig, kan det være nødvendigt at indbygge "if-sætninger" i programmet, så man kan se om systemet internt reagerer som det skal. Det er ikke altid kun output-informationen, man er interesseret i. Man indbygger testinstrumenteringen i det færdige system, så man derved kan få udskrevet vigtige parametre på centrale punkter i systemet, eller man kan holde statistik på, hvor mange gange systemet har kaldt en monitor.

Debuggere

Debuggere er kendt i et hvert udviklingsværktøj. Formålet er, at gennemkøre hele eller dele af programmet for at finde evt. programmeringsfejl.

Til testning af gruppens system vil vi benytte en interaktiv testkørsel. Dette sker på baggrund af systemets kompleksitet, samt at det umiddelbart er det nemmeste og det meste resultatgivende i vores tilfælde. Da vores system er meget skærmorienteret, er det nemt at opdage eventuelle output fejl på de udarbejdede brugergrænseflader.

For at teste et system er det vigtigt at fastslå, hvilke input man ønsker testet og det pågældende output, som systemet skal generere. Man kan vælge testcases ud fra to synsvikler:



Funktionelt (Blackbox-test)

Her fastslår man om systemet udefra reagere efter hensigten. Man sammenligner i bund og grund det genererede output med det ønskede output.

Programmæssigt (Whitebox-test)

Her kigger man ind i systemet, og ser på den logiske opbygning og strukturen af instruktioner, forgreninger, tilstande osv.

24. Testrapport

Gruppen har på nuværende tidspunkt opnået det ønskede resultat mht. implementeringsfasen, og vil nu teste systemet. Til det vil gruppen benytte nedenstående fremgangsmåde, de opstillede testemner samt det udarbejdede brugertestdokument. Afsnittet vil slutte med en konklusion på testen samt de evt. fundne rettelser.

24.1 Formål

Gruppen tester for at påpege evt. fejl i systemet, og for at vise at systemet er udviklet på baggrund af de krav, der var i starten af projektperioden til systemet. Da gruppen løbende har udviklet og testet de moduler, som skal bruges i systemet, vil dette testdokument kun omhandle selve accepttesten, og hvilke testmetoder vi ønsker at udsætte vores system for.

24.1.1 Referencer

Gruppen refererer til kravsspecifikationen del II, afsnit 7.2 samt hele del IV som omhandler OOD og de moduler, gruppen har udviklet. Desuden kan henvises til brugervejledningen, i appendiks H.

24.1.2 Omfang og begrænsninger

Gruppen ønsker at teste det samlede system for at se, hvordan det reagerer i de situationer, som gruppen vil sætte programmet i. Gruppen vil, så vidt det er muligt, teste systemet i de omgivelser og af de brugere, som kommer til at benytte systemet. Gruppen vil teste systemet i kombination af lokalnetværk samt Internet. Grunden til denne kombination vil blive beskrevet mere detaljeret i afsnit 24.3 omkring testdesign

24.2 Accepttestens testemner

Følgende testemner vil blive gennemgået i denne accepttest, som gruppen vil udføre som en afslutning på hele projektforløbet. Accepttesten udføres for at sikre, at det afleverede produkt lever op til de stillede krav til systemet.

Stresstest

Her kunne det fx være interessant at spørge: Virker systemet, hvis 25 brugere opretter det samme navn samtidig?

Volumetest



I denne test handler det fx om: Hvordan opfører systemet sig ved stor tilgang af data gennem lang tid?

Brugertest

Gruppen ønsker at teste systemets brugervenlighed på mange forskellige brugere, men alle har det tilfælles, at de er brugere på Internettet. Ydermere ønskes det, at brugerne kommer med forslag til, hvad der kan laves om, og hvad de finder forvirrende. Der er udarbejdet brugergrænsefladerne på baggrund af den allerede eksisterende EON hjemmeside, hvilket var et ønske som gruppen havde til den færdige hjemmeside, så det er begrænset hvad gruppen vil ændre i det designmæssige. Hvis der er forslag i den retning, vil gruppen dog overveje disse ændringer af designet.

Sikkerhedstest

Et af de mere spændende spørgsmål vil være: Kan man snyde loginproceduren på hjemmesiden?

Kompatibilitetstest

Projektgruppen vil her bl.a teste om systemet kan bruges med ældre versioner af browsere?

Fejlbehandling

I fejlbehandling drejer det sig fx om, hvordan reagerer systemet på fejl?

Blackbox

Her vil gruppen gennemgå alle funktioner, for at fastslå om de virker.

Disse testemner ønskes gennemgået, og der vil til hvert punkt blive beskrevet hvordan systemet, brugere m.v. reagerer på disse fejl.

24.3 Testdesign

Da gruppen mener, at brugerne af dette system er folk, der surfer på Internettet fra deres hjemme-PC, vil en del af testen foregå i "hjemlige" omgivelser. Brugere bliver folk der surfer på Internettet, hvor gruppen har fundet to typer, den øvede og den let øvede. Disse to brugertests vil foregå udfra brugertestdokumentet, som en "tænke-højt-prøve", og vil blive optaget på video. Disse to tests vil dog blive udført på en standalone PC, hvor gruppen benytter en PC i private omgivelser, for at skabe det rette miljø for brugerne. Gruppen havde overvejet at teste systemet på Internettet, men da ingen af gruppens medlemmer havde fast forbindelse til Internettet, er gruppen tvunget til at teste systemet lokalt.

Gruppen vil dog i visse tilfælde teste systemet i et lokalnetværk for at opnå flerbruger-adgang til systemet, så vi kan teste hvordan systemet reagerer på testemner omkring stresstest, volumetest, sikkerhedstest, kompatibilitetstest samt



fejlbehandling. Disse testemner vil blive testet i et lokalnetværk, da gruppen mener at disse emner godt kan testes lokalt med et fordelagtigt resultat, i mangel af fast Internet adgang. Mht. stresstesten, sikkerhedstesten og fejlbehandlingstesten vil gruppen benytte flere brugere, da gruppen ikke er i stand til at udføre disse tests selv. Det skal ses i lyset af bemanningen i gruppen, men også fordi det altid er en fordel at få andre end gruppen selv til at teste systemet, da de er mere objektive og derfor ofte vil være istand til at finde fejl eller misinformation, som gruppen ikke er opmærksom på.

Alle funktioner i systemet vil blive gennemgået i blackbox testen, hvor gruppen vil beskrive de testede funktioner, det forventede resultat samt om hvorvidt systemet reagerede som forventet. Desuden vil der ved nogle af funktionerne, blive tilknyttet en kommentar, hvor gruppen ser det nødvendigt.

25. Testimplementering

Gruppen har naturligvis al mulig tiltro til det udviklede system. Det er også gruppens overbevisning, at systemet ”opfører sig som det skal”, når det bliver udsat for de beskrevne testemner. Med andre ord, systemet fungerer normalt under stresstesten, volumetesten samt fejlbehandlingstesten. Systemet kan ikke snydes under sikkerhedstesten, systemet er kompatibelt med alle gængse browsere samt alle fejlbeskeder er forståelige, hvis de skulle opstå. For at bevise disse påstande, har gruppen valgt at udføre en detaljeret blackbox-test, hvori gruppen gennemfører alle procedurer som systemet tilbyder. Dette sker for at påpege, at systemet generede det output, som gruppen havde forventet.

25.1 Stresstest

Til denne test har vi valgt at stille følgende spørgsmål:

Virker systemet, hvis 25 brugere opretter det samme navn samtidig?

Ja, da det er utrolig svært at alle bruger trykker på retur nøjagtigt samtidigt, har systemet ikke besvær med denne opgave.

Virker systemet, hvis 25 brugere logger sig på samtidig?

Denne opgave har samme karakter som den første opgave, så der er heller ikke her problemer med udførslen.

Virker systemet, hvis 25 brugere ønsker det samme sæde?

Systemet virker, men kun én af de 25 brugere får reserveret sædet.

Kan systemet håndtere 25 brugeres ”surfen rundt” på hjemmesiden?

Der er heller ikke her nogle problemer med dette antal af brugere på hjemmesiden.

25.2 Volumetest



Hvordan opfører systemet sig ved stor tilgang af data gennem lang tid?

Af forskellige årsager, har det ikke været gruppen muligt, at kunne teste systemet gennem længere tid. Derfor kan der pt. ikke svares på dette spørgsmål. Dog viser en dags konstant brug, ingen tegn på problemer.

Hvordan reagerer databasen på meget datatilgang?

Ved megen efterspørgsel på data nedsættes serverens hastighed, men dog ikke til stor målbar gene.

25.3 Sikkerhedstest

Kan man snyde loginproceduren på hjemmesiden?

Ved gentagende forsøg, har vi ikke fundet det muligt at snyde loginproceduren.

Gruppen har indbygget et maksimum på 7 sædebestillinger pr. reservation. Kan der snydes med dette?

Det er ikke muligt at snyde direkte med max af sæderbestillinger pr. reservation, men det er dog muligt at bestille flere end 7 sæder ved at lave endnu en reservation, hvor man dog igen har max på 7 sæder.

25.4 Kompatibilitetstest

Kan systemet bruges med ældre versioner af browsere?

Nej, ikke med fuld funktionalitet, da de ældre browsere ikke understøtter den java version vi benytter til vores applet.

Kan systemet bruges af forskellige browsere, såsom både Netscape Navigator og Microsoft Explorer?

Begge begge browsere virker, såfremt det er en version fra 4.00 og opefter.

25.5 Fejlbehandling

Hvordan reagerer systemet på fejl?

Det kommer med fejlmeddelelser alt efter vilke type af fejl det drejer sig om.

Kommer systemet korrekt ud efter en fejl?

I de tilfælde vi har provokeret fejl frem, har systemet kommet korrekt ud bagefter.

Er fejlbeskederne forståelige?

Ja, det mener vi bestemt at erfaringerne viser.

Hvordan reagerer systemet på eksterne fejlsituationer, såsom strømsvigt, harddiskfejl osv.?



Der tages ikke højde for disse situationer i systemet, så derfor reagere det naturligvis dårligt.

25.6 Blackbox-test

Gruppens udførelse af blackbox-testen vil ske på baggrund af alle funktioner i systemet. Gruppen vil skrive det forventede output, det output som systemet genererer samt om systemet kan udføre de valgte procedurer.

25.6.1 Navigationspanel

De nedstående links, findes på NFB sitens forside.

Hvad testes	Forventede resultat	Resultat	Evt. kommentar
Log in linket	Login pop-up fremkommer	✓	
Log ud linket	Man sendes tilbage til forsiden og man kan se i nederste frame, at der ikke er logget nogen bruger på.	✓	Inden klik på log ud, er man i dette tilfælde logget på systemet.
Bestillingslinket	Skærmbilledet til bestilling af billetter fremkommer	✓	Inden klik på linket, er man i dette tilfælde logget på.
Afbestillingslinket	Skærmbillede til afbestilling af billetter fremkommer	✓	
Betalingslinket	Skærmbillede til betaling af billetter fremkommer	✓	
Film Info linket	Skærmbillede til film info fremkommer	✓	
Skuespiller Info linket	Skærmbillede til skuespiller info fremkommer	✓	

Tabel 25.1



25.6.2 Login proceduren

Denne procedure omhandler "log ind", "Opret ny bruger", systemmeddelelser og "Log ind ikke gennemført" samt "Log ind gennemført", som alle er pop-up meddelelser.

Hvad testes	Forventede resultat	Resultat	Evt. kommentar
Kan man taste sit bruger navn ind	Man kan se det indtastede navn i tekstboken	✓	
Kan man ses passwordet ved indtastning	Man bør kun kunne se en stjerne for hvert tegn der indtastes	✓	
Minimum og maksimums grænsen for antal tegn i brugernavn og password	At systemet kræver minimum tre tegn og maksimum 20 tegn i brugernavn. Der er et minimum på 1 tegn og et maksimum på tolv tegn i passwordet . Systemet accepterer ikke æ ø å.	✓	
Tages der højde for store og små bogstaver i brugernavn og password	Der tages højde for store og små bogstaver	%	Store og små bogstaver skelnes ikke.
Login med ikke eksisterende bruger info	Pop-up systemmeddelelse om at bruger ikke findes	✓	
Oprettelse af ny bruger med allerede eksisterende brugerinformationer	Pop-up systemmeddelelse om at bruger allerede eksisterer	✓	
Linket "Registrér dig her"	Nyt pop-up skærbillede til indtastning af ny brugerinformation fremkommer	✓	
Login med forkerte antal tegn i postnr., tlf. nr. samt fødselsdato	Pop-up systemmeddelelse om at et ugyldigt antal tegn i et af de givne felter er indtastet forkert	✓	Her kommer en hjælp til brugeren om hvad der er tastet forkert i forhold til hvad systemet forventer
"Videre" knappen	Meddelelse om at login er gennemført fremkommer	✓	I pop-up meddelelsen findes en "Luk" knap som sender brugeren til startside.
"Afbryd" knappen	Afbryder login proceduren og sender brugeren til startbilledet uden at vedkommende er logget på.	✓	
Kan man se at brugeren er logget på	I nederste frame, venstrehjørne kan ses brugernavn samt email adresse som tilkendegiver, at brugeren er logget på.	✓	
Kan det ses om brugeren kan betale	I nederste højre hjørne, ses status for onlinebetaling	✓	



online			
--------	--	--	--

Tabel 25.2

25.6.3 Information indhentning

Indhentning af information tager sit udgangspunkt i de to skærbilleder "Film info" samt "Skuespiller info".

Hvad testes	Forventede resultat	Resultat	Evt. kommentar
Combo boksen "Skuespiller" i skærbilledet "Skuespiller info"	En liste med skuespiller navne vises i combo boksen	✓	
Combo boksen "Rolle" i skærbilledet "Skuespiller info"	En liste med alle rolletyper fremkommer i combo boksen	✓	Fx Hovedrolle, Bi-rolle, Instruktør, Forfatter
"Søg" knappen i skærbilledet "Skuespiller info"	En liste med filmtitler fremkommer med de valgte parametre fra combo boksen	✓	
Kan alle navne og alle roller vælges i skærbilledet "Skuespiller info"	En komplet liste med alle film, navne og roller fremkommer	✓	
Links på filmtitlerne i skærbilledet "Skuespiller info"	Her fremkommer informationerne om den valgte film	✓	Her bringes man over i et andet skærbillede, omhandlende filminformationer.
Combo boksen med filmtitler i skærbilledet "Film info"	Ved klik vises en liste med alle film fra databasen	✓	
"Søg" knappen	Den valgte films informationer fremkommer	✓	Der vises både et resumé af filmen, spilletid, produktionsår, billede fra filmen samt alle medvirkende
Skal man være logget ind for at kunne søge disse informationer	Nej, man vil altid kunne læse film informationer samt skuespiller informationer uden at være logget ind	✓	
"Bestil billetter" knappen	Nyt skærbillede "Bestil billetter" fremkommer med den valgte film	✓	Herefter skal der vælges dato og tidspunkt til pågældende film

Tabel 25.3



25.6.4 Billet bestillingsproceduren

Denne test tager sit udgangspunkt i skærbilledet "Bestil Billetter".

Hvad testes	Forventede resultat	Resultat	Evt. kommentar
Combo boksen "Film"	Ved klik fremkommer en liste med alle aktuelle filmtitler	✓	
Combo boksen "Dato"	Ved klik fremkommer en liste med datoerne på den næste syv dage	✓	
"Søg" knappen	Alle forestillinger vises med den valgte film på den valgte dato	✓	
Linket på filmtitlen	Skærbilledet "Film info" fremkommer	✓	
Linket på spilletidspunkt	Skærbillede med filmens titel, valgte dato, valgte tidspunkt samt layout på den sal filmen bliver fremvist i, fremkommer.	✓	I billedet over den aktuelle sal vises hvilke sæder der allerede er optaget. Disse er markeret med rødt.
Oplyses knapperne "Reserver sæder" samt "Nulstil valg", når musen bevæges over knappen	Knapperne lyser op	✓	
Knappen "Reserver sæder"	Skærbillede der viser de valgte sæder i salen samt deres samlede pris vises. Samtidig vises det seneste afhentningstidspunkt for billetterne.	✓	
Den samlede pris på de valgte billetter	Den viste pris er korrekt	✓	
Afhentningstidspunktet af billetterne	Det viste afhentningstidspunkt er korrekt	✓	Billetterne skal afhentes et kvarter før.
Knappen "Nulstil valg"	Ved klik på knappen, er de valgte sæder ikke længere markeret	✓	
Valg af flere end syv sæder	Et valg af et ottende sæde er ikke muligt	✓	Brugeren er ved tekst allerede oplyst om at dette ikke kan lade sig gøre
Valg af optaget sæde	Dette er ikke muligt	✓	Sædet kan ikke markeres og der fremkommer en "brumme" lyd.
Det korrekte sæde- og rækkenummer på de valgte sæder	De korrekte numre vises ovenover billedet af salen	✓	
Fravælgelse af valgt sæde inden tryk på knappen "Reserver sæder"	Det er muligt at fravælge et netop valgt sæde af den pågældende bruger	✓	



Valgt af tilfældige sæde rundt omkring i salen	Det er muligt at vælge tilfældige ledige sæder rundt omkring i salen	✓	
Aktivering af knappen "Reservér sæder" uden at have valgt nogen sæder	Der sker ingenting	✓	Ved klik på knappen, checkes automatisk om der er valgt nogen sæder.
Bestille uden at være logget på	Skærm billede med meddelelsen om at man skal logge sig på først, fremkommer	✓	
Knappen "Betal billetter"	Skærm billedet "Betal billetter" fremkommer	✓	

Tabel 25.4

25.6.5 Betalingsproceduren

Her involveres skærm billedet "Betal billetter"

Hvad testes	Forventede resultat	Resultat	Evt. kommentar
Skrives de rigtige informationer i forhold til det valgte	At informationerne er korrekte	✓	Informationerne er filmtitel, dato, tid, antal pladser og samlet pris
Opdateres prisen ved fravalg af hel reservation	Prisen opdateres korrekt	✓	Fravalget findes i kolumnen "Valgt" til højre, og vises ved et "flueben"
Om prisen er korrekt, hvis der er flere reservationer	Prisen udregnes korrekt i forhold til de enkelte reservationers samlede pris	✓	
Knappen "Betal"	Systemmeddelelse om betaling er gennemført	✓	
Ingen valgte reservationer, ved tryk på "Betal" knappen	Systemmeddelelse om at ingen reservationer er valgt	✓	

Tabel 25.5



25.6.6 Afbestillingsproceduren

Under denne procedure involveres skærbilledet "Afbestil billetter".

Hvad testes	Forventede resultat	Resultat	Evt. kommentar
Om de valgte sædereservationer listes med den rigtige dato, filmtitel samt tidspunkt	At de korrekte reservationer vises	✓	
Linket "Afbestil billetter"	Ved klik fjernes det valgte sæde og vises dermed ikke på listen	✓	
Opdateres reservationslisten	At den fjerner de fravalgte sædereservationer	✓	

Tabel 25.6

25.6.7 Andet

Hvad testes	Forventede resultat	Resultat	Evt. kommentar
Browserens "Refresh" knap	Siden bliver opdateret med evt nye informationer	✓+ %	De nyeste informationer bliver opdateret, men det "ser ud" som brugeren bliver logget af

Tabel 25.7

25.6.8 Afrunding af Blackbox test

Efter at have gennemgået sitens funktionalitet, mener vi at kunne fastslå, at alt virker efter hensigten.

25.7 Brugertest

Ved realiseringen af brugertesten har gruppen udarbejdet et dokument, som består af opgaver i systemet, som brugeren skal udføre. Denne seance bliver videofilmet, og skal foregå ved at brugeren tænker højt under gennemgang af opgaverne i systemet. Gruppen har sat et forventet tidsforbrug på 10 minutter til gennemgang af opgaven, både for den let øvede og den øvede bruger. Dette tidsforbrug skal ses som en parameter, for at checke brugervenligheden ved "surfing" gennem systemet. Brugertestdokumentet ser ud som følgende:

Følgende opgaver ønskes gennemgået i billetbestillingssystemet til NFB:

Opgave

Reservér 5 sæder til *X-Files* filmen klokken 19.30, dags dato.

Opgave

Hvad blev den samlede pris?

Opgave

Afbestil 2 sæder fra den foregående reservation.



Opgave

Bestil 10 sæder til *Godzilla* filmen klokken 20.00, imorgen.

Opgave

Log af systemet.

Opgave

Giv en kommentar til systemet, både funktionalitet samt brugervenlighed. Syntes du, at systemet fungerede optimalt, og var skærmbillederne forståelige? Var der noget som burde være med, eller gjort anderledes?

25.7.2 Udførelse af brugertest

Brugertesten blev udført i de beskrevne omgivelser samt med de beskrevet spørgsmål. Brugertesten blev videofilmet, og selve testen kan ses på den vedlagte cd. Gruppen benyttede to brugere, som gennemgik de opgaver som gruppen havde stillet til systemet. Brugerne havde generelt et godt indtryk af systemet, og der var da også kun et par få forslag til ændringer af dette.

25.7.3 Forslag til program- og designændringer

Brugerne havde følgende forslag til program- og designændringer af det udviklede system:

Feedback fra let øvet bruger

Han fandt placeringen af knappen *Søg* ret ulogisk, da han associerede den med datoen og ikke både med datoen og filmen. Han mente umiddelbart, at han ville få en liste med alle filmene frem når han så trykkede på *Søg*. Her mente han, at han kunne vælge hvilken film han ville se, udfra dato. Dette ser gruppen som en misforståelse af systemet, da gruppen mente han overså film-combo boksen. Han fandt dog hurtig selv ud af logikken i systemet.

Under udførelse af opgave 4, hvor brugeren skulle bestille ti billetter til *Godzilla*, kunne han ikke fuldføre opgaven, men kunne kun bestille syv billetter.

Gruppen observerede, at den let øvede bruger brugte omkring 15 minutter til gennemgang af opgaven, hvilke efter gruppens opfattelse var i overkant.

Gruppen havde regnet med, at hele bestillingsforløbet kun tog højst 10 minutter, selv for den let øvede.

På spørgsmålet om farvevalget samt udformningen var der positiv feedback fra brugeren.

Gruppens vurdering af feedbacken fra den let øvede bruger

En ændring af placering af knappen *Søg*, mener gruppen er unødvendig. Men for at tydeliggøre knappens overordnede funktion, kan man ændre baggrundsfarven, så indtrykket af knappen bliver af en mere overordnet karakter.



Gruppen indrømmer, at bestillingen af mere end syv billetter kan være indviklet. Denne feature med bestilling af syv og højest syv billetter pr. reservation, var et krav fra NFB grundet sikkerhedsmæssige årsager. Gruppen har dog lavet en metode, så bestilling af mere end syv billetter kan lade sig gøre. Dette gøres ved at man igen vælger den samme forestilling og vælger de ekstra sæder man ønsker. Man kan dog stadig kun bestille syv sæder ad gangen. Gruppen betragter det store tidsforbrug som en ”begynderfejl”. Gruppen mener at næste gang den let øvede bruger logger sig på systemet, vil der gå betydelig kortere tid, både fordi brugeren nu kender systemet, men også fordi brugeren er oprettet i systemet.

Feedback fra øvet bruger

Den øvede bruger fandt måden, som opgave 4 skulle udføres på, lidt vanskelig. Brugeren kunne dog udføre opgaven ved at vælge den samme forestilling og bestille de ekstra sæder, han manglede. Brugeren fandt også det valgte design og farvevalg godt, da brugeren mente det var logisk og overskueligt opbygget. Den øvede bruger brugte kun otte minutter til gennemførelse af de stillede opgaver, hvilket var mindre end det, som gruppen havde forventet.

Gruppens vurdering af feedbacken fra den let øvede bruger

Gruppen er opmærksom på, at bestilling af flere billetter kan være vanskelig, men da det er et stillet krav fra NFB, kan gruppen ikke ændre denne sikkerhedsmekanisme. Gruppen var dog tilfreds med, at brugeren kunne finde ud af at bestille mere end syv billetter til samme forestilling. Dette skete på den forømtalte måde.

Gruppen var overordentlig tilfreds med den øvede brugers tidsforbrug på de stillede opgaver. Det skal ses i lyset af at brugeren brugte tid på at oprette sig selv, og så derefter kan komme til at bestille billetter. Man skal dog også se dette, som at gruppen anser denne bestillingsform som impulsiv, da man også ville bruge tid på at læse om filmen osv. Vi benyttede tidsforbruget som en parameter til at måle systemets brugervenlighed.

25.7.4 Konklusion på brugertesten

Gruppen har et generelt godt indtryk af testen og mener, at det udviklede system lever op til de forventede resultater. De brugere, der testede systemet var også positivt indstillede overfor det udviklede system. Brugerne mente, at både farvevalg og funktionalitet var udarbejdet på en overskuelig måde, og helhedsindtrykket var positivt fra brugernes side. Gruppens konklusion på testen må være, at vi på nuværende tidspunkt har udviklet et system, som lever op til NFBs krav om brugervenlighed og funktionalitet.

Vi tester nu videre på systemet for at se, hvordan det reagerer på de forskellige testemner, der har til formål at prøve at ”ligge” systemet ned. Vi designer ikke videre på systemet nu, da designet i systemet fik positiv kritik.



25.8 Udførelse af test

Efter at have udarbejdet en testrapport og udført de beskrevne testemner samt de opgaver, der er beskrevet i brugertesten, kan gruppen lave en konklusion på de arrangerede tests. Denne konklusion skal ses som en beskrivelse af de problemer og designovervejelser, som brugerne havde til systemet. Så vidt der er muligt vil gruppen rette de fundne fejl samt korrigere designet, så det svarer til brugernes forslag. Fejl, der er fundet, men som ikke bliver rettet pga. tidsnød, vil kun blive beskrevet.



Indholdsfortegnelse

25. EVALUERING	2
25.1 PROCES-EVALUERING.....	2
25.1.1 Evaluering af gruppens samarbejde.....	2
25.1.2 Evaluering af samarbejde med Nordisk Film Biografer	2
25.1.3 Evaluering af samarbejde med vejleder	3
25.1.4 Konklusion på procesevalueringen	3
25.2 PRODUKTEVALUERING.....	3
25.2.1 Systemet.....	3
25.2.2 Rapporten.....	4
26. KONKLUSION.....	7
27. LITTERATURLISTE	9
27.1 BØGER	9
27.2 ARTIKLER	10
27.3 WEBOGRAFI.....	11

25. Evaluering

25.1 Procesevaluering

Procesevaluering går ud på, at evaluere projektets forløb med hensyn til beslutning, socialisering og kommunikation internt i gruppen.

25.1.1 Evaluering af gruppens samarbejde

På trods af, at et af gruppemedlemmerne aldrig har arbejdet sammen med de tre andre i gruppen før, synes vi at samarbejdet er gået godt. Samarbejdet har på intet tidspunkt føltes akavet, men har derimod gennem hele forløbet været præget af enighed, gensidig respekt og især humor. Sandie blev i starten valgt som projektleder, men projektlederrollen har egentlig været temmelig flydende, da vi alle stort set har deltes meget godt om den. Vi har alle sammen gennem hele forløbet haft det store, forkromede overblik.

Der har været mindre uoverensstemmelser, men det har udelukkende været i forbindelse med rapportens udseende (fonte, opstilling m.m.), og aldrig med hensyn til indhold og opbygning.

Vores individuelle arbejdsform – eller den måde, som vi har været vant til at arbejde på i tidligere projekter – har ikke divergeret synderligt.

Dette projekt er det længste projekt, som vi har arbejdet på under datamatikeruddannelsen. Men egentlig er det også det projekt, som er gået mest smertefrit i og med, at vores erfaring fra tidligere projekter har skinnet kraftigt igennem. Af de erfaringer, vi har gjort under tidligere projekter, har vi både brugt de bedste og de dårligste – de bedste for at overkomme og undgå konflikter, og de dårligste for at undgå at havne i for komplicerede situationer. Erfaringerne er naturligvis anvendt for at opnå det bedst mulige resultat.

En af de positive erfaringer, som vi har gjort så mange gange før, har været anvendelsen af referencelinier til del-styring af projektforsløbet (vi har anvendt referencelinier til at styre vandfaldsmodellen). Det er altid en fordel at have nogle



faste holdepunkter og deadlines for, hvornår et bestemt delprodukt skal være færdigt. På denne måde har vi også fået indført noget kvalitetsstyring i projektet.

25.1.2 Evaluering af samarbejde med Nordisk Film Biografer

Samarbejdet med Nordisk Film Biografer gik i starten fint, men har derefter været lidt af en bølgegang. I begyndelsen af projektperioden holdt vi et par møder med vores kontaktperson i NFB. Møderne gik fint og der var altid god stemning. Vi kunne have ønsket os, at NFB havde været en smule mere engageret, men de har dog svaret på de spørgsmål, som vi har haft. Vi har måske ikke selv været gode nok til at informere NFB om, hvad det egentlig var vi forventede af dem og vi mener ikke, at NFB påtog sig den egentlige kunderolle, som vi forventede af dem. Dette mener vi selvfølgelig var en fejltagelse, da bedre information fra os muligvis ville have givet et bedre samarbejde. Derfor har vi haft meget frie hænder – til tider har det været herligt, men andre gange har det også været frustrerende, da vi manglede noget konkret at holde os til.

25.1.3 Evaluering af samarbejde med vejleder

Samarbejdet med vejlederen er utrolig vigtigt, da det er denne som skal have det overordnede ”objektive” overblik. Han/hun er den, som bedømmer hvad der er godt nok, og hvad som ikke er. Samarbejdet med vores vejleder kan dog kun roses. Gruppen mener, at det har været yderst tilfredsstillende at afholde vejledermøde en gang ugentligt. Vejlederen har til alle tider været til rådighed og har samtidig udvist stort engagement i gruppens projektarbejde. Vi i gruppen er enige om, at hvis vi skulle afvikle et lignende projektforsøg til, ville vi helt klart vælge den samme vejleder igen.

25.1.4 Konklusion på procesevalueringen

Sluttelig kan vi, af procesevalueringen, konkludere, at samarbejdet har fungeret optimalt og alle i gruppen har fået dækket deres individuelle interesser (med hensyn til vores specialefag på fjerde semester) under projektforsøget.

25.2 Produktevaluering

Produktevaluering går ud på, at evaluere slutproduktets kvalitet, og påpege eventuelle svagheder og mulige forbedringer. Vores slutprodukt består faktisk af to delprodukter: systemet og rapporten.

25.2.1 Systemet

Da vi ikke på forhånd kendte teknikken bag servlets, JDBC og webserveren, er en del af tiden gået med at prøve sig frem samt at indhente informationer om disse teknikker og værktøjer. Meget information om servlets er hentet på denne fortrinlige site: <http://www.javasoft.com>, som bl.a. indeholder adskillige lærerige tutorials.

Vi synes bestemt ikke det har været nemt at designe HTML siderne ved hjælp af servlets. Da vi skulle lave vores design, blev vi nødt til at designe HTML siderne igennem en almindelig HTML-editor og derefter overføre denne kode til den servlet, der genererer siden. Det bliver derfor meget svært at ændre i designet, når man først har fået programmeret sin servlet. Hvis vi havde brugt fx ASP ville det



umiddelbart være noget nemmere, da man skriver ASP-kode direkte i HTML-dokumentet - dvs det bliver muligt at teste sit HTML dokument sideløbende med, at man skriver sin kode, uden recompile sit program hver gang.

Dog er vi fuldt ud tilfredse med vor design - det krævede bare et større overblik over det generelle design.

Funktionaliteten er begrænset til kun at tilbyde bestilling af billetter samt indhentning af diverse informationer vedr. film. Systemet mangler også en udbygelse, hvori man kan ændre allerede indtastet brugeroplysninger, såsom navn og email adresse. Der er dog foretaget en udbygning af systemet, som gør det muligt at foretag direkte online betaling. Dette skete da de fleste oplysninger allerede er registreret, når en bruger har bestilt billetter. Det eneste der mangler er, at der foretages en direkte transaktion fra brugerens købekort, og dermed en betaling af de bestilte billetter over Internettet. Databasen blev udbyggede, så denne nu kan registrere en brugers købekort nummer, og dermed er der ikke så langt til en direkte betaling. Vi har valgt at implementere dette, for at vise systemets umiddelbare fremtidssikring. Gruppen har dog ikke foretaget nogen videre overvejelser, omkring problematikken vedr. de lovmæssige forhold der er tilknyttet dette område, samt overvejelser omkring afbestilling ved betalte billetter. Der er heller ikke, fra gruppens side, udført analyse- og designmæssige drøftelser angående implementationen af dette modul i systemet. Denne udbygelse skal udelukkende ses som et eksempel på systemets fremtidige udvidelses muligheder med forskellige moduler.

Problematikken omkring bestilling af flere end syv billetter pr. forestilling, har gruppen med vilje valgt, at implementere i systemet. Dette gøres for at give brugeren mulighed for at bestille flere end syv billetter til en forestilling, uden at skulle ringe til NFBs billetbestilling. Begrænsningen i, kun at kunne bestille syv billetter ad gangen, gøres for at en bruger ikke kan bestille en hel sal, i en enkelt reserveringen.

Brugervenligheden af disse funktionaliteter, har igennem vores brugertest vist sig, at være fortrinlig. Selv en uøvet bruger vil allerede anden gang denne er i berøring med systemet, ikke have svært ved at navigere rundt på siden. Dette mener gruppen er ganske tilfredstillende, men for at sikre at en førstegangsbrowser kan navigere rundt på samme uhindret måde, kunne der evt. tilføjes forklarende tekst til nogle af de mest avancerede funktioner.

25.2.2 Rapporten

Vi har valgt at opdele rapportevalueringen på samme måde, som selve rapporten er opdelt på.

Projektetablering

Projektetableringen har under hele projektforsløbet fungeret som et godt holdepunkt, både med hensyn til tidsplanlægning og referencelinier. Det har været et absolut plus at arbejde med referencelinier og en tidsplan, da disse to ting sammen har hjulpet til at man når som helst har kunnet vurdere projektets status. Dog erfarede vi i implementeringsaktiviteten, at referencelinie 5 ikke holdt stik. Derfor udarbejdede vi et tillæg, som viser hvordan denne referencelinie skulle have set ud fra starten. Dette tillæg findes i appendiks I, figur I.1.



Foranalyse

Foranalysen har givet gruppen en god ballast i problemdefineringen. Ved at opstille BATOFF-kriterier og udarbejde systemdefinitioner, og derefter vælge en af disse, får man et udmærket overblik over, hvad det egentlig er man skal udvikle. Gruppen anser foranalysen som et godt udgangspunkt for det videre analyseforløb, da de fleste problemer allerede i foranalyseaktiviteten klarlægges.

Kvalitetsmål

Overvejelserne omkring de opstillede kvalitetsmål i foranalysen, har indvirket at gruppen fra starten havde nogle mål at arbejde ud efter, mht. udviklingen af systemet. Gruppen anser det for en god ting, at man allerede i begyndelsen havde nogle forholdsvis konkrete overvejelser omkring opbygningen af systemet. Generelt mener vi at have opfyldt alle de opstillede kvalitetsmål til systemet, og vil i denne evaluering kun gå i dybden med dem som gruppen vurderede som meget vigtige. Disse mål var:

Brugervenlighed

Efter udførelsen af brugertesten, har gruppen bevist at det udviklede system har en høj brugervenlighed, set udfra et brugervenligt synspunkt. Målet for gruppen var at udvikle et system, som gjorde det nemt og hurtigt at bestille billetter over Internettet. Brugertesten viste, at efter brugeren havde været i kontakt med systemet én gang, var der et stort kendskab til fremgangsmåden ved bestilling. Derfor må vi konkludere, at dette kvalitetsmål er opfyldt i henhold til det forventede.

Integritet og korrekthed

Vi har igennem vores tests fået fastslået at den meget vigtige opretholdelse af integriteten og korrektheden, er opfyldt i forhold til de krav vi satte fra starten. Disse ting blev bl.a fastslået gennem blackbox-testen, hvor vi påviste at bl.a priserne og reservationerne blev registreret og var rigtige.

Effektivitet

Igennem de gennemførte streestests, påviste gruppen at de forespørgelser som systemet udførte på databasen blev udført hurtigt. Overordnet har systemet en temmelig høj hastighed, som også gør det mere attraktivt at bruge systemet, set fra en brugers synsvinkel. Alt i alt er gruppen overordenlig tilfreds med hastigheden af systemet, dog må man påregne en vis ventetid på hentning af applet'en. Dette kunne måske være optimeret en smule, hvis vi havde valgt at bruge en ældre version af JDKen eller hvis vi havde undgået at bruge billeder og lydfiler.

Flytbarhed

Det er umiddelbart ret let at flytte systemet over på en anden server, da alle filnavne er relative samt koden selv tager højde for at finde



serverens navn og evt. port. Derfor har kvalitetsmålet for flytbarhed oversteget det forventede og givet en god dynamik, som kun kan være en fordel.

Analyse

Gruppen mener, at vi kunne have anvendt analysen til at analysere de behov, der måtte være til det kommende system, i stedet for at analysere det nuværende. Ikke dermed sagt, at analysen er benyttet forkert, da denne udmundede i en dummy-database, som vores system skulle anvende. Vi har dog i analyseaktiviteten fundet flere nyttige klasser, strukturer og funktioner, som er benyttet gennem såvel design som implementering.

Design

Vi startede med at ville designe tre komponenter, men efterhånden som designaktiviteten skred frem, fandt vi ud af, at vi faktisk havde brug for to mere. Til at starte med var der lidt problemer med EIC-komponenten, da vi ikke kunne se at vi skulle anvende denne. Da vi så havde fundet ud af, at vi skulle benytte en EIC blev gruppen opmærksom på, at der også optrådte en TMC-komponent i systemet. Vi havde lidt problemer med at designe begge disse komponenter, da vi aldrig har gjort det før, og vi fandt det en smule besværligt. Gruppen blev også introduceret for en ny datamodellerings teknik: EER. Gruppen havde stort set minimalt kendskab til denne teknik ved projektets start, men kan idag se de fordele ved brug af denne modelleringsmetode. Vi er i hvert fald selv ret tilfredse med vores designdokument.

Implementering

Gruppen er utilfreds med, at refresh af cachen (Ctrl+F5) fjerner brugerloginstatus selvom brugeren **ikke** bliver logget af. Det kan skabe sikkerhedsproblemer, da en bruger kan tro han/hun er logget ud selvom dette ikke er tilfældet. Vores applet har også et lille problem med opdatering af en 'mouseover' event. Hvis brugeren har cursoren placeret over appleten mens denne initialiseres kan der forekomme en lille grafikfejl der kan se ud som om, et sæde er reserveret.

Test

Den gennemgaaede testperiode mener gruppen var overordnet god. Gruppen havde dog visse problemer med at bl.a. stressteste systemet, da den planlagte testperiode lå i efterårsferien. Dette gjorde, at gruppen ikke havde mulighed for at udføre visse af de opstillede testemner, og derfor måtte udsætte disse til senere. Det resulterede i, at testperioden blev rimelig presset, men gruppen er overordenlig tilfreds med de testresultater, vi opnåede. Samtidig mener gruppen, at det testede system fungerer bedre end forventet.



26. Konklusion

Konklusionens formål er, at afrunde og besvare de, i problemformuleringen, stillede spørgsmål. Projektet er nu afsluttet og vi har konkluderet følgende:

Er det muligt at udvikle et interface, der kan overbygge det nuværende system og database?

Det er *ikke* muligt at udvikle et interface, som kan overbygge det nuværende system og database. Den primære grund er, at NFB af sikkerhedsmæssige årsager ikke lod os se databasen. Der er flere sekundære grunde:

NFB kendte ikke så meget til deres eget system, da dette er udviklet af et engelsk firma

Det var på baggrund af ovenstående ikke muligt at fremskaffe de nødvendige drivere, der skulle til for at kunne overbygge det nuværende system og den nuværende database

Er det muligt at simulere Internet-trafik i et lokalt testmiljø?

Der er to grunde til, at det ikke har været muligt at simulere Internet-trafik i et lokalt testmiljø. For det første har det ikke været muligt at simulere de store trafikmængder, som forekommer på Internettet. For det andet har det ikke været muligt at simulere de mange forskellige platforme, som i realiteten udgør Internettet. Dette kunne dog have været interessant at undersøge, da det har stor betydning for designets udformning.

Er den valgte styringsmetode anvendelig til dette projekt?

Den valgte styringsmetode har været det rigtige valg. Den eneste afvigelse fra en ren vandfaldsmodel har været, at implementeringsaktiviteten har været nødsaget til at køre parallelt med alle andre aktiviteter i og med, at vi fra starten var nødt til at afsætte ressourcer til at undersøge teknikken bag de valgte metoder og teknikker til implementeringen.

Som sagt før, valgte vi at styre vandfaldsmodellen med referencelinier, og dette har vist sig yderst anvendeligt, da vi mener at vandfaldsmodellen, med tilhørende referencelinier, passer godt sammen. Så snart en aktivitet i vandfaldsmodellen er afsluttet, kan der meget passende komme en referencelinie.

Er den objektorienterede metode anvendelig på vores problemområde?

Den objektorienterede systemudviklingsmetode (OOA og OOD) har været særdeles anvendelig på vores problemområde. Den er anvendelig i den forstand, at det er en iterativ metode, hvor man i gennem både analyse og design anvender de tidligere resultater og går mere i dybden. Man finder ud af nogle ting, bygger videre på det og finder flere detaljer.

Da vores opfattelse af SA og SD er, at man skal færdiggøre hver fase inden man påbegynder den næste mener vi, at det har været en klar fordel at anvende OOA og OOD. Vores erfaring med OOA og OOD er, at det er en friere metode, hvor man



stort set selv kan vælge de ting, som man har lyst til at gå i dybden med, for at skabe et bedre overblik for én selv.

På et par vigtige punkter i vores system finder vi ikke den objektorienterede metode tilstrækkelig i og med, at vi implementerer i en relationel database. Selvom det med Coad & Yourdons designmetode går forholdsvis smertefrit, går meningen med det objektorienterede alligevel lidt tabt, da man – når man laver klasser om til tabeller – mister ideen med objektorientering.

Derfor udviklede vi en EER-model, som netop bibeholder den semantik, som man mister ved udarbejdelse af E/R-modellen, når man anvender OOA og OOD. Ved at have udarbejdet EER-modellen er det nemmere at udskifte databasetypen med en anden. Dog mener vi, at man – i forbindelse med Coad & Yourdons designmetode – skal overveje DMCens udformning, hvis man udskifter databasetypen.

Er det muligt at implementere systemet udelukkende ved brug af Java?

Det er muligt at implementere systemet udelukkende ved brug af Java, da adgangen til databasen klares ved hjælp af JDBC, og dynamisk generering af HTML-dokumenter klares af servlets; begge Java-APIer. Dog er de fleste statiske oplysninger prægenereret HTML-sider.



Alt i alt kan man sige, at de fleste ting er lykkedes og slutresultatet – efter gruppens mening – er blevet bedre end de forventninger, som vi havde da projektet begyndte. Dette gælder både rapportens indhold og opbygning samt selve webløsningen.

27. Litteraturliste

27.1 Bøger

Andersen, Niels Erik et al.
Professionel Systemudvikling
Teknisk Forlag A/S 1986

Biering-Sørensen, Stephen et al.
Struktureret Program Udvikling
Teknisk Forlag A/S 1991

Coad, Peter & Edward, Yourdon
Object-oriented Design
Prentice Hall, Inc. 1991

Delskov, Lise & Lange, Therese
Struktureret Analyse
Teknisk Forlag A/S 1986



Elmasri, Ramez & Navathe, Shamkant B.
Fundamentals of Database Systems
The Benjamin/Cummings Publishing Company, Inc. 1994

Flanagan, David
Java In a Nutshell
O'Reilly & Associates, Inc. 1996

Flanagan, David
Java Examples in a Nutshell
O'Reilly & Associates, Inc. 1996

Huber, Peter et al.
Informationsanalyse
Teknisk Forlag A/S 1996

Mathiassen, Lars et al.
Objektorienteret Analyse
Forlaget Marko ApS 1993

Mathiassen, Lars et al.
Objektorienteret Analyse og Design
Forlaget Marko ApS 1997

27.2 Artikler

Boehm, Barry W.
A Spiral Model of Software Development and Enhancement
IEEE Computer vol.21 1988

Ellis, John R.
Objectifying Real Time Systems
SIGS Books 1994

Lee, Byung S.
OODB design with EER
Joop 1996

Low, G.C. et al.
Incorporation of distributed computing concerns into OO-methodologies
Journal of Object Oriented Programming 1996

Sippl, Roger
ODBC and Unix: Port no more?
Unix Review 1995



William H. Evans
Ideal foundations for web content management

Object magazine, June 1997

27.3 Webografi

<http://dir.dk/artikler-krav.htm>

Denne webadresse indeholder en række spørgsmål, som man kan stille sig selv, når man prøver at klarlægge, om en virksomhed kan drage fordel af at have en website og i så fald, hvad der gør den brugervenlig.

<http://java.sun.com/docs/books/tutorial>

Denne omfattende site indeholder mange tutorials om fx servlets, JDBC, applets, etc.

<http://www.apache.org>

Dette er adressen til Apache serveren. Indeholder informationer om opsætning, bugs, faqs etc.

<http://www.film.eon.dk>

Denne site er forbilledet for gruppens webdesign og indeholder informationer om alle aktuelle film i de fleste af landets biografer.

<http://www.livesoftware.com>

Firmaet bag JRun. Indeholder nyeste download af deres Servlet Engine og informationer om opsætning, ofte stillede spørgsmål, etc.

<http://www.microsoft.com/msdn>

En omfattende site omhandlede megen Windows informationer, fx brug af ASP.

<http://www.nordisk-film.dk>

Nordisk Films egen hjemmeside med informationer om virksomheden.

<http://www.sigs.com/publications/docs/joop/9602/joop9602.f.lee.html>

På denne website findes en artikel, der omhandler EER-modellen, som vi har arbejdet med. Artiklen handler mest om design af objektorienterede databaser med EER, men den kan være nyttig lige meget hvilken databasetype, man anvender.



Indholdsfortegnelse

OOA DEFINITIONER.....	2
OBJEKTER	2
KLASSER.....	2
HÆNDELSER	3
STRUKTUR	3
Klyngestruktur.....	3
Generalisering/Specialisering.....	4
Aggregeringsstruktur	4
Associeringsstruktur.....	5
ADFÆRD	5
Adfærdsmønstre.....	5
FUNKTIONER.....	6
GRÆNSEFLADER	6
DEFINITIONER PÅ NORMALFORMER.....	7
FØRSTE NORMALFORM.....	7
ANDEN NORMALFORM	7
TREDJE NORMALFORM.....	7

OOA definitioner

Objekter

Ved et objekt forstås det ifølge Lars Mathiassens¹ definition, at det er:

■ *En helhed med identitet, tilstand og adfærd.*

Objekter arbejder altså med tre egenskaber, der har til formål at klassificere hvert objekt. Et objekts identitet skiller et objektet ud fra alle andre objekter i objektsystemet. Dette sker typisk med en form for primær nøgle, som kender det fra databaser, der består af et nummer der udskiller et objekt fra en bestemt klasse. En tilstand kan både have statiske egenskaber, som fx navne eller tidsangivelser, og dynamiske angivelser som fx svingende mængder, flyvehøjde og temperaturer. Adfærd er den sekvens af hændelser, som objektet kan udføre eller påføres i sit livsforløb. Et objekt kan være involveret i en given hændelse som enten aktiv eller passiv part. Hvis en kunde indsætter et beløb på en konto, er kunden aktiv og kontoen passiv, og hændelsen er fælles for begge parter.

Klasser

Ved en klasse forstås det ifølge Lars Mathiassens definition, at der er:

■ *En mængde objekter med samme egenskaber og adfærdsmønstre*

Når klasserne skal udvælges bør alle kandidater skrives ned, uden dog at blive vurderet. Først fokuseres på navneord, derefter generelle typer af klasser. Man kan

¹ Mathiassen, Lars et al.: Objektorienteret Analyse



desuden vurdere ud fra andre kendte systemer. For at udvælge navne til klasserne, er der fire ”idealer” der kan benyttes:

- Anvend simple og læsbare betegnelser
- Brug betegnelser fra objektsystemet
- Benyt enkelte navneord
- Brug betegnelser for en enkelt forekomst

For at finde ud af om klassen skal med i objektsystemet, skal man kunne svare ja på følgende spørgsmål:

- Omfatter klassen flere objekter?
- Kan et objekt i en klasse identificeres entydigt?
- Har klassen en overskuelig mængde hændelser?
- Indeholder klassen unik information?

Hændelser

Ved en hændelse forstås det ifølge Lars Mathiassens definition, at der er

■ *En øjeblikkelig begivenhed, som involverer et eller flere objekter*

Når man finder hændelser, skal man fokusere på udsagnsord samt generelle typer og tilsvarende edb-systemer. En hændelse kunne være den før omtalte kunde, der sætter et beløb ind på sin konto. Hændelsen er ”at indsætte” og hændelsen vil komme til at hedde ”indsæt beløb”.

For at vurdere de hændelser man har fundet frem til, skal man kunne svare ja på følgende spørgsmål:

- Er hændelsen øjeblikkelig?
- Er hændelsen atomar?
- Kan hændelsen identificeres, når den sker?

Struktur

Forskellen på en klassestruktur og en objektstruktur er, at klassestrukturen er statisk hvor objektstrukturen er dynamisk. For at man samlet kan beskrive den begrebsmæssige sammenhæng mellem to eller flere klasser i en klassestruktur, er man derfor nødt til at benytte to begreber, klynge og generalisering. For at kunne beskrive sammenhæng mellem objekter bruges aggregering og associering.

Klyngestruktur

Formålet med en klynge er, at give overblik ved at samle flere klasser under ét begreb. Klynger afspejler en overordnet forståelse af problemområdet, idet det opdeler det i nogle få delområder. Definitionen man bruger om klynger er, at disse er:



■ *En samling af klasser, som er indbyrdes forbundne.*

En klynge betegnes ofte ved navnet på en central klasse. Når man opstiller en klyngestruktur skal man gennemgå følgende overvejelser:

Hvilke delområder består problemområdet af? Her tvinges man endnu en gang til at tage stilling til, om klassen skal med i modellen eller ej.

Tag udgangspunkt i de øvrige strukturer: generaliserings- og aggregeringsstrukturer samler klasserne i en klynge, associeringsstrukturer adskiller klynger.

Hvis en klasse kan være i to klynger, bør man oprette en særskilt klynge for denne klasse.

Generalisering/Specialisering

Formålet med generalisering er, at samle fælles egenskaber og adfærdsmønstre fra forskellige klasser i en mere generel klasse, superklassen. Superklassen beskriver de egenskaber og det adfærdsmønster, som er fælles for et antal specielle klasser, subklasserne. Superklassens egenskaber og adfærdsmønstre nedarves til subklasserne, via de egenskaber og adfærdsmønstre, som er beskrevet i den generelle klasse. Disse nedarves til alle de specialiserede klasser.

Man kan godt lave en generalisering over et begreb uden objekter, fx hvis der i et objektsystem kun findes ansatte og kunder, kan man lave en generalisering over begrebet 'person', selv om der ikke findes noget objekt af klassen 'person'.

Personklassen anvendes derimod kun til at beskrive fælles forhold for ansatte og kunder, og de fælles forhold nedarves så til disse to specielle klasser.

Specialisering er subklasserne, altså en specialisering af en superklasse, eksempelvis er frisør og lærling en subklasse/specialisering af superklassen ansat. Der sker en nedarvning af en "over"-klasse til specielle klasser. Ved en sproglig formuleringen af specialiseringer, bruger man udtrykket "er en", eksempelvis "en frisør er en ansat".

Aggregeringsstruktur

Når der er tale om aggregeringsstruktur

■ *Aggregering er en struktur, hvor et overordnet objekt, helheden, består af et antal underordnede objekter, delene.*

Man kan også sige, at en aggregeringsstruktur beskriver en relation mellem et objekt og de andre objekter, som udgør dets bestanddele. Eksempelvis kan man sige at objektet 'en sal' kan bestå af, eller er en aggregering af, forskellige objekter som fx rækker, lærred eller fremviser. Rækker kan så igen bestå af sæder. Sprogligt kan vi udtrykke en aggregering ved, at et objekt kan 'dekomponeres' i mindre dele. Man kan også sige, at en sal har rækker, sæder, lærred og fremviser.

For at finde aggregeringsstrukturer skal man overveje følgende:

For hvert klassepar skal man overveje, om den ene classes objekter er en dekomponering af den andens. Dvs. er et eller flere af den ene classes objekter indeholdt i et eller flere af den anden classes objekter?



For hvert klassepar skal man overveje, om det vil være relevant at aggregere dem i en ny klasse.

For hver enkelt klasse skal man overveje, om den kan dekomponeres i nogle relevante klasser, som ikke allerede findes i ens model.

For at kunne skelne aggregeringer fra associeringer, kan man stille følgende spørgsmål:

Kan objekterne eksistere uafhængigt af hinanden?

Er objekterne sideordnede og ligestillede?

Kan forbindelsen skifte til andre objekter af samme klasse?

Et ja til et af disse spørgsmål peger på associering, et nej peger på aggregering.

Associeringsstruktur

■ *Associering er en struktur, hvor et antal sideordnede eller ligestillede objekter knyttes til hinanden.*

Fx kan en kunde have mange reservationer og en reservation kan være foretaget af mange kunder. I dette tilfælde sideordner man objekterne 'kunde' og 'reservation', fordi det ikke giver mening af sige, at det ene objekt indeholdes i det andet.

Hverken kunden eller reservationen er en defineret egenskab ved det andet objekt. Sprogligt kan vi udtrykke associeringen med at kunden 'er forbundet med' reservationen eller omvendt.

For at finde associeringsstrukturer er man nødt til at inddrage alle øvrige klassepar, og overveje følgende:

For hvert klassepar skal man overveje, om de overhovedet har noget med hinanden at gøre. Skal de overhovedet med i modellen?

Kan man give forbindelsen et navn, som fx 'ejerskab'?

Overvej om der mangler klasser til at beskrive associeringer, fx en 'lejekontrakt.'

Adfærd

Adfærdsmønstre

Lars Mathiassen definerer et adfærdsmønster som følgende:

■ *Et abstrakt mønster af hændelser, der fastlægger de mulige eller ønskelige hændelsesforløb for alle objekter i en klasse.*

Adfærdsmønsteret kan bruges til at beskrive faktisk adfærd, men også til at foreskrive hvordan objekter *skal* opføre sig. De hændelser, der i klasseaktiviteten blev knyttet til hver enkelt klasse indgår i adfærdsmønsteret for den pågældende klasse.

Et adfærdsdiagram ordner de enkelte hændelser ved hjælp af konstruktionerne sekvens, selektion og iteration.



Sekvens: et hændelsesforløb foregår ved, at de indgående hændelsesforløb indtræffer én for én i den angivne rækkefølge.

Selektion: et hændelsesforløb foregår ved, at netop ét af de indgående hændelsesforløb indtræffer. En selektion bliver angivet med en 'O'.

Iteration: et hændelsesforløb indtræffer nul eller flere gange. Iteration bliver angivet med en '*'.

Så adfærdsmønsteret fastlægger altså på klasseniveau den mulige eller ønskelige adfærd for objekterne i en klasse. De konkrete forekomster af hændelser, som udgør adfærden for et bestemt objekt, sammenfattes i begrebet hændelsesforløb. Et adfærdsmønster beskriver derfor et fælles mønster, som gælder for alle objekter i en klasse.

Funktioner

Definitionen på en funktion er ifølge Lars Mathiassen således:

En ressource, som stilles til rådighed for brugeren og nyttiggør modelkomponenten i udførelsen af arbejdsopgaver.

Grænseflader

Lars Mathiassen definerer en grænseflade som:

Edb-systemets brugergrænseflade og direkte forbindelser til andre systemer og apparater.

Definitioner på normalformer

Første normalform

Hvis en tabel har en gentagende gruppe af felter indenfor samme næsten primærnøgle, så skal denne gruppe af felter fjernes fra tabellen og lægges over i en ny tabel sammen med en kopi af primærnøglen.

Hvis der ikke findes nogen gentagende gruppe af felter i en tabel, opfylder tabellen allerede første normalform.

Anden normalform

Hvis en tabel har en sammensat primærnøgle (bestående af to eller flere felter), og der findes et felt i tabellen, som ikke indgår i primærnøglen og er afhængigt af den del af den sammensatte primærnøgle (et eller flere felter i primærnøglen kaldet del-primærnøgle), så skal dette afhængige felt fjernes fra tabellen og lægges over i en ny tabel sammen med en kopi af del-primærnøglen, som bliver primærnøgle i den nye tabel.

Hvis primærnøglen ikke er sammensat i en tabel, dvs. består af et felt, opfylder tabellen allerede anden normalform.

Hvis primærnøglen er sammensat af alle felterne i tabellen, dvs. ingen felter som ikke indgår i



primærnøglen, opfylder tabellen allerede anden normalform.

Hvis hvert enkelt felt i tabellen uden for primærnøglen kun er afhængigt af hele primærnøglen, opfylder tabellen allerede anden normalform.

Tredje normalform

Hvis en tabel har felter, der ikke indgår i primærnøglen, og der findes et felt i tabellen, som ikke indgår i primærnøglen og er afhængigt af et andet felt (evt. flere felter tilsammen), der heller ikke indgår i primærnøglen, så skal dette afhængige felt fjernes fra tabellen, og lægges over i en ny tabel sammen med en kopi af det/de felter, som feltet afhang af, som bliver primærnøgle i den nye tabel.

Hvis der kun er et felt i en tabel, som ikke indgår i primærnøglen, opfylder tabellen allerede tredje normalform.

Hvis primærnøglen er sammensat af alle felterne i tabellen, dvs. ingen felter som ikke indgår i primærnøglen, opfylder tabellen allerede tredje normalform.

Hvis hvert enkelt felt i tabellen udenfor primærnøglen kun er afhængig af hele primærnøglen, opfylder tabellen allerede tredje normalform.

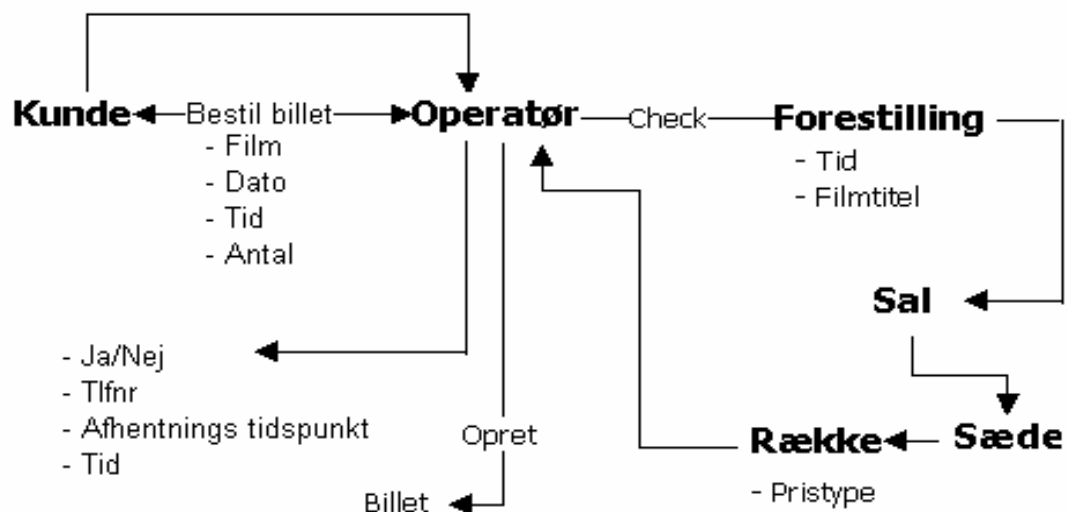


Indholdsfortegnelse

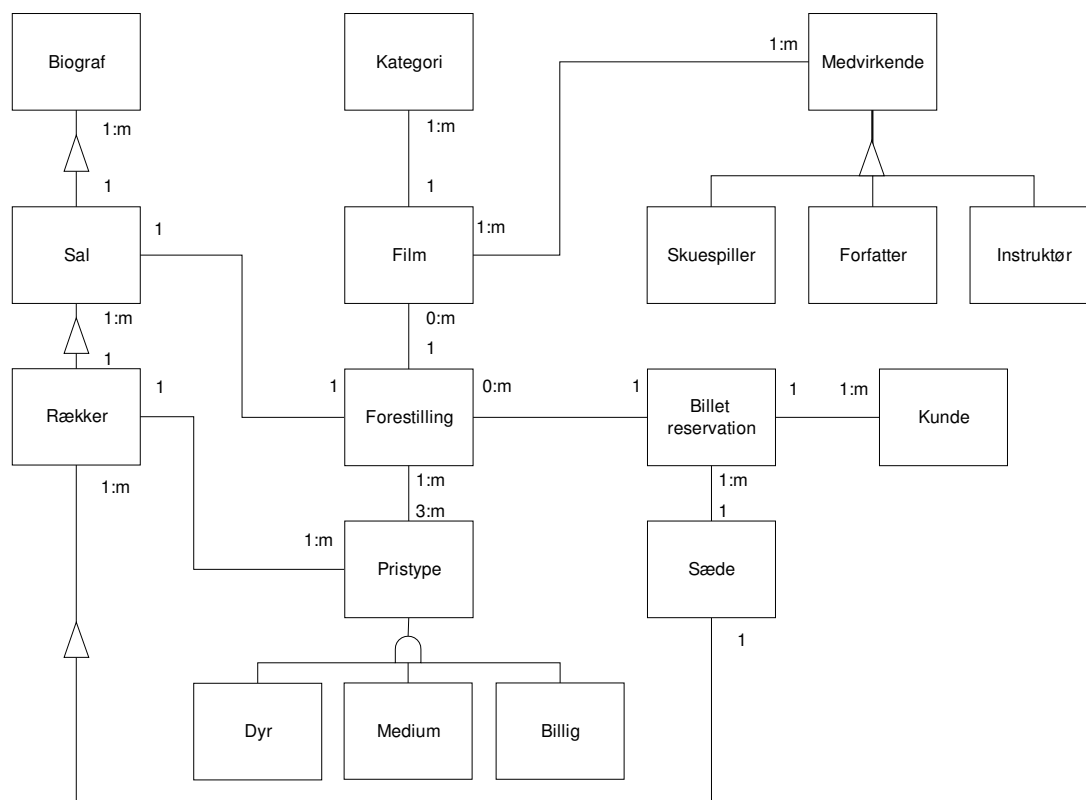
DIAGRAMMER TIL OOA	2
BRAINSTORM FOR SYSTEMETS ARBEJDSGANGE	2
STRUKTURDIAGRAM	3
KONTEKSTDIAGRAM	4
ESSENTIEL MODEL NIVEAU 1 – ADMINISTRÉR BILLETRESERVATION	5
OVERSIGTSDIAGRAM 1	6
OVERSIGTSDIAGRAM 2	7

Diagrammer til OOA

Brainstorm for systemets arbejdsgange



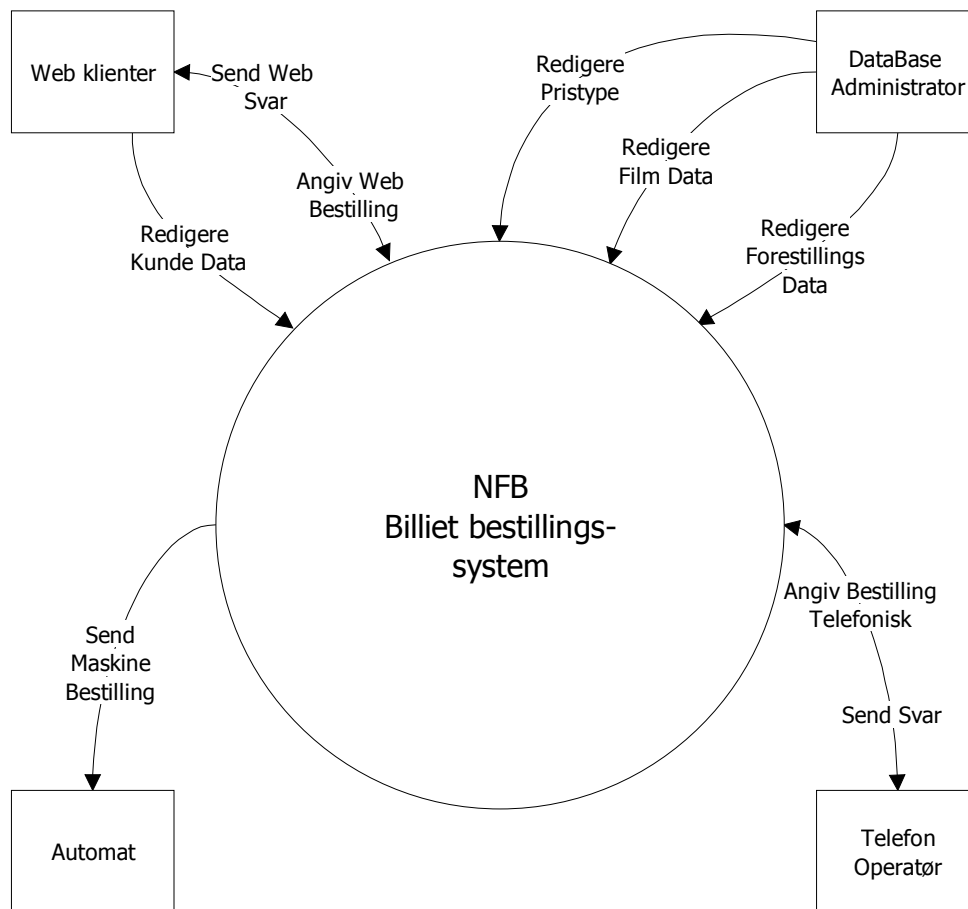
- Kunde: Bestil billet, afbestil billet, rediger bestilling
- Operator: Check forestilling, check tider, check dato, spørg om data, slet reservering, rediger reservering, opret reservering, returner information
- Forestilling: Check sæder, check tid
- Sal: Check ledige plader



Strukturdiagram

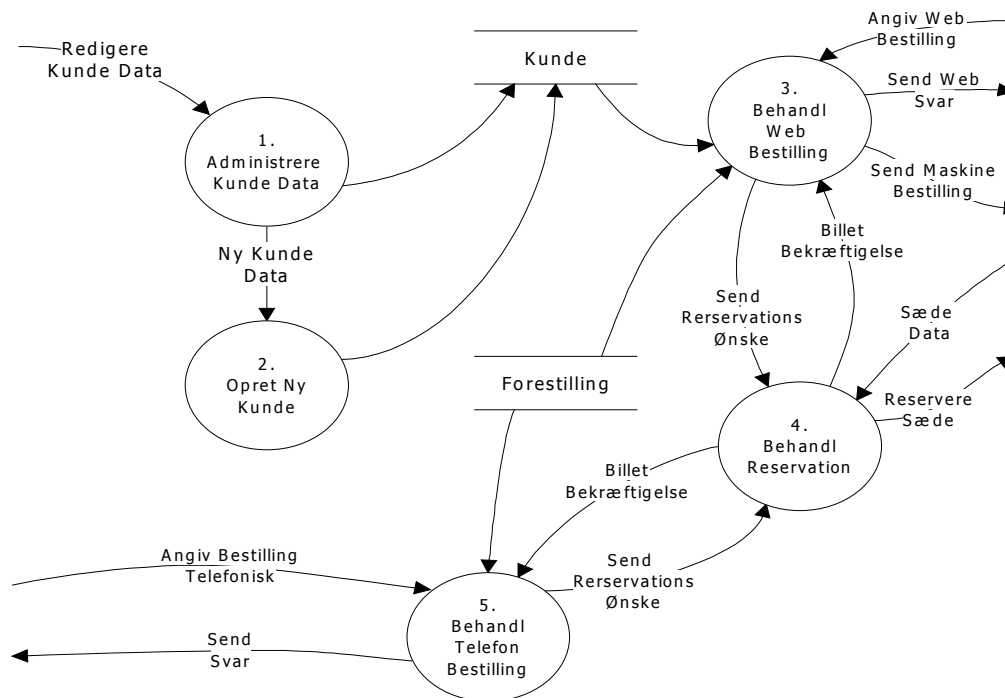


Kontekstdiagram



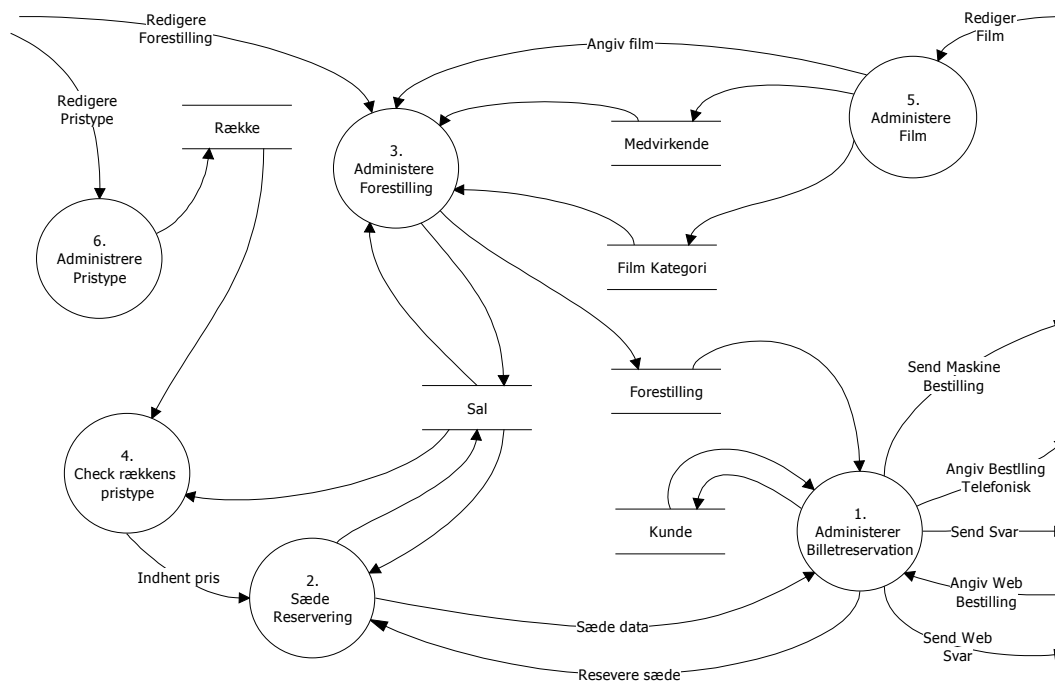


Essentiel model niveau 1 – Administrér billetreservation



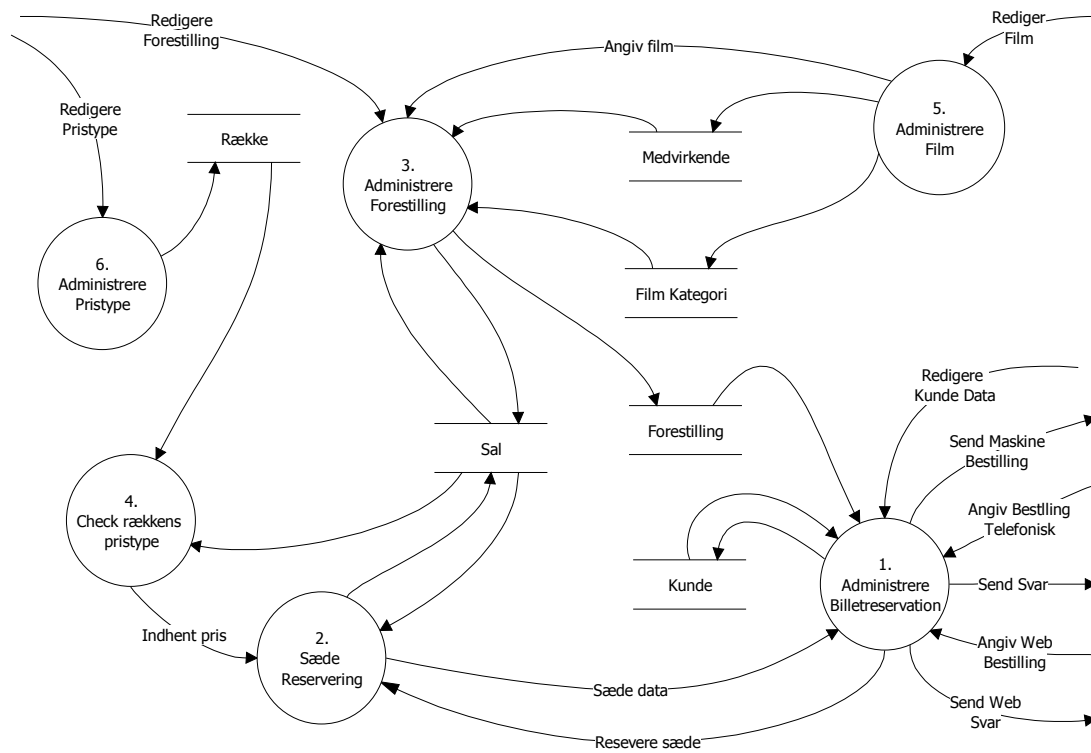


Oversigtsdiagram 1





Oversigtsdiagram 2





Indholdsfortegnelse

EER-MODELLEN.....	2
Hvad er EER?.....	2
Modelleringskoncepter.....	2
Fordele ved EER.....	3
Ulemper ved EER.....	4
BESKRIVELSE AF GRUPPENS EER-MODEL.....	4
Entitetstyper.....	5
Relationstyper.....	5
Attributter.....	5
Afhængigheder.....	5
Generalisering/specialisering.....	6
Total og delvis specialisering.....	6
Definerende prædikater.....	6

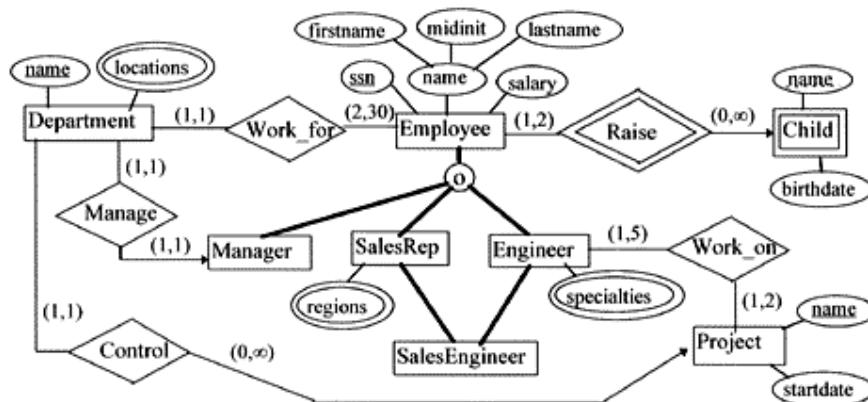
EER-modellen

Hvad er EER?

EER-modellen er en forbedret/udvidet version af E/R-modellen. EER-modellen indeholder alle de modelleringskoncepter, som man kender fra almindelig E/R, men indeholder også koncepterne om specialisering, generalisering, arv og kategorier. Denne udvidelse har ledt frem til EER-modellen, og EER står for *Enhanced ER*. I dette appendiks vil emnerne arv og kategorier dog ikke blive behandlet.

Modelleringskoncepter

Koncepterne i EER-modellen vil blive forklaret ud fra følgende figur¹:



Figur C.1: Eksempel på en EER-model

¹ Hentet fra Lee, Byung S.: OODB Design with EER. Se litteraturliste for webadresse.



Entitetstyper vises i bokse, som fx EMPLOYEE. En dobbeltrammet boks som fx CHILD viser, at entitetstypen er en *svag entitetstype*, dvs. en entitetstype, som ikke har en komplet nøgleattribut.

En *relationstype* vises som en rhombe som fx WORK_FOR. Hvis relationen er dobbeltrammet som fx RAISE viser det, at det er en *identificerende relationstype*. Dvs. en identificerende relation er en relation, som relaterer en svag entitetstype til dens "ejer".

Attributterne vises i ovaler. Hvis attributten i ovalen er understreget er der tale om en nøgleattribut. Hvis ovalen er dobbeltrammet viser det, at det er en *multivalued attribut* (fx for to-farvede biler skal en attribut 'Farve' være en multivalued attribut).

En attribut kan også være sammensat. Dette ses fx for entitetstypen EMPLOYEE, hvor attributten 'Name' er sammensat af fornavn, midtinitialer og efternavn.

Kardinaliteterne angiver minimums- og maksimumsværdier. Fx kan CHILD være tilknyttet mindst én, men højst to forekomster af EMPLOYEE, hvorimod EMPLOYEE kan have opdraget nul eller uendeligt mange børn.

En afhængighedsbetingelse på en relation er vist ved en pil. Fx kan PROJECT kun eksistere, hvis der findes et DEPARTMENT.

Entitetstyper kan være relateret ved generaliserings- og specialiseringsforhold, som resulterer i *er et/er-en* forhold. Entitetstyperne, som er forbundet med de fremhævede linier, viser *er-et/er-en* forholdet. Det er værd at lægge mærke til, at en genspec-struktur kan være et *specialiseringshierarki* eller et *specialiseringsmatrix*. Et specialiseringshierarki har den betingelse, at hver subklasse *skal* deltage i et klasse/subklasse-forhold. For et specialiseringsmatrix kan en subklasse være subklasse i mere end et klasse/subklasse-forhold.

På figuren findes også en *shared subclass*, SALESENGINEER. Shared subklasser leder frem til begrebet *multipl arv*. Man skal måske bemærke, at shared subklasser leder frem til matrixer. Hvis der ikke er nogen shared subklasser, ville man have et hierarki rettere end et matrix.

O'et i cirklen midt på figuren angiver, at der findes et *overlap*. Der vises en specialisering og med hensyn til dette er der to betingelser: der kan være tale om enten *disjointness* eller *overlap*.

Den første betingelse er *disjointness*, som specificerer at subklasserne i specialiseringen skal være adskilte. Dette betyder, at en entitet kan være medlem af højst én af subklasserne i specialiseringen. Hvis subklasserne ikke er disjoint, kan deres sæt af entiteter overlappe hinanden. Dvs. den samme entitet kan være medlem af mere end én subklasse. På denne figur må der være et overlap pga. den *shared subclass* SALESENGINEER. Hvis en MANAGER også kunne være en SALESREP ville man formentlig også her have anvendt en *shared subclass*.

Man har brugt en enkelt linie fra EMPLOYEE og ned til cirklen for at vise en *partial specialization*. Dette betyder, at en entitet ikke behøves at tilhøre en af subklasserne. Hvis man havde brugt en dobbeltlinie havde man vist en *total specialization*, hvori der specificeres at hvert medlem af superklassen *skal* være medlem af én af subklasserne.

Relationer er direkte repræsenteret i EER, hvorimod de i OO kun er indirekte repræsenteret gennem objektreferencer (associeringer og aggregeringer). EER-relationerne giver mulighed for at udtrykke semantikken, kardinaliteter og afhængighederne mellem entitetstyperne. Foruden det kan man repræsentere



mere sofistikerede betingelser på EER-diagrammet i form af tekst (fx kan man sætte *definerende prædikater* på en specialisering). Man kunne i ovenstående figur sætte definerende prædikater på linierne fra EMPLOYEE og ned til hhv. MANAGER, SALESREP og ENGINEER.

Fordele ved EER

Man kan ved hjælp af et EER-diagram repræsentere betydningen af generaliserings/specialiserings-strukturer samt afhængighederne i aggregeringer. Man kan også udtrykke, om en entitet kan indgå i kun en enkelt eller måske i flere subklasser ved at anvende de to specialiseringsbetingelser, som EER tilbyder: *disjoint* og *overlap*, og man kan anvende definerende prædikater på linierne til hver subklasse, for at indikere en entitets medlemskab af en given subklasse. Man kan på EER-diagrammet vise, om en entitet i superklassen *skal* være medlem af en subklasse eller ej. Hvis en entitet i superklassen *skal* være medlem af en subklasse kan dette vises ved at lave en dobbeltlinie fra superklassen og ned til cirklen. Dette kaldes også en *total specialisering*. Hvis en entitet i superklassen kan nøjes med at være medlem af superklassen og af ingen af subklasserne, kunne vi have vist dette med en enkelt linie. Dette kaldes *delvis specialisering*. Alt i alt mener gruppen, at EER-modellen giver bedre overblik end E/R-modellen, da EER udtrykker mange flere ting og man mister ikke betydningen af hverken generaliserings/specialiserings-strukturer eller aggregeringsstrukturer her, som man gør ved almindelig E/R-modellering.

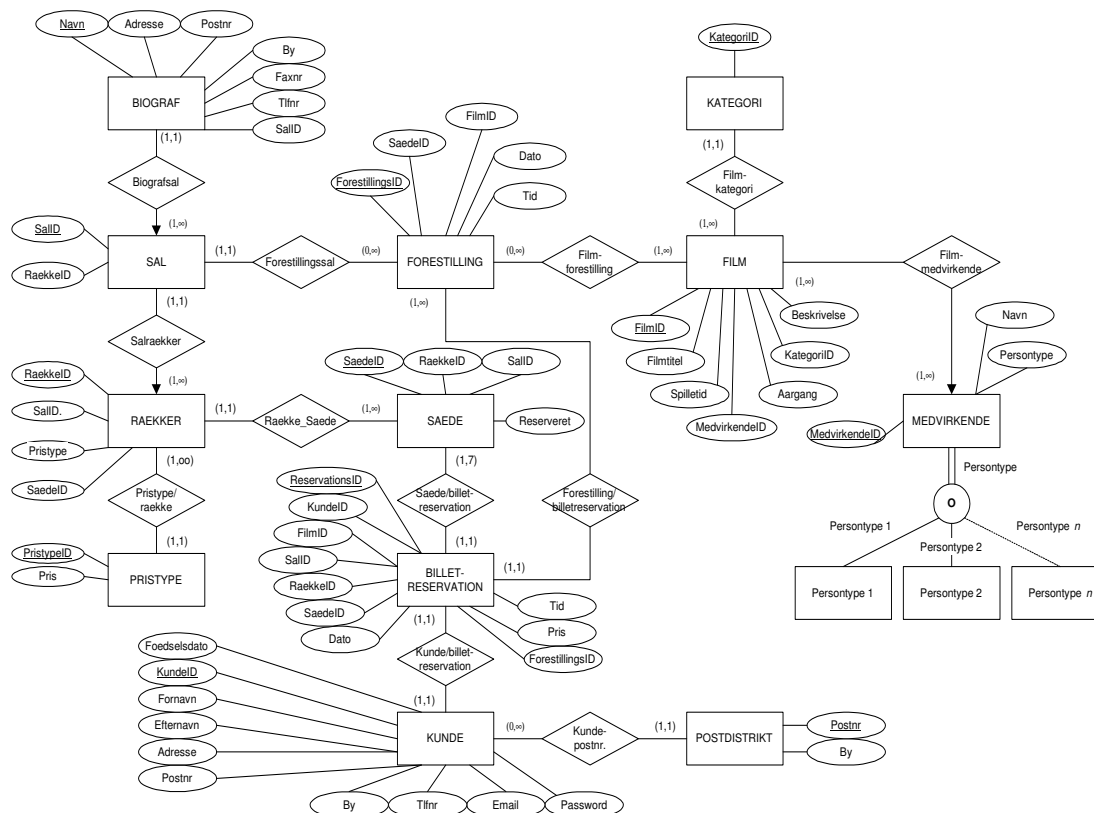
Ulemper ved EER

Den eneste ulempe, som gruppen mener, der er omkring EER, er den måde hvorpå attributterne angives. Men dette er udelukkende fordi diagrammet hurtigt bliver meget stort pga. attributrepræsentationen. Hvis der er jævnt mange attributter til hver entitetstype, skal diagrammet fylde mere end et almindeligt stykke papir for at være overskueligt.

Beskrivelse af gruppens EER-model



Gruppen har valgt at udarbejde et EER-diagram på grund af de semantiske fordele, som er nævnt ovenfor. Nedenfor følger en gennemgang af gruppens EER-diagram.



Figur C.2: Vores udarbejdede EER-model

Entitetstyper

Alle entitetstyper er almindelige entitetstyper. Der findes ingen svage entitetstyper, dvs. entitetstyper, som ikke har en komplet nøgleattribut. Man overvejede på et tidspunkt, om entitetstypen 'Pristype' var en svag entitet. Den havde ikke fået en nøgleattribut, og derfor blev det diskuteret, om 'Pristype' var ejet af 'Række'. Men gruppen blev enig om, at det ikke var en svag entitet, da den fik sin egen primærnøgle.



Relationstyper

Alle relationstyper er almindelige relationstyper. Der findes ingen identificerende relationstyper, da der ikke findes nogen svage entitetstyper. Man overvejede på et tidspunkt, om relationstypen 'Pristype/Række' var den identificerende relation for den svage entitetstype 'Pristype', som skulle knyttes til sin ejer netop gennem denne identificerende relation. Men gruppen blev enig om, at der ikke var tale om en identificerende relation, når 'Pristype' ikke var en svag entitetstype.

Attributter

Alle nøgleattributter er understreget. Der findes ingen flerværdi-attributter eller sammensatte attributter.

Afhængigheder

Der findes tre afhængigheder, og disse afhængigheder stemmer overens med de tre aggregeringer, som ses på både strukturdiagrammet og PDC-diagrammet. Disse afhængigheder ses mellem:

Biograf og Sal

Sal og Række

Film og Medvirkende

Disse afhængigheder viser, at fx en sal ikke kan eksistere uden at en biograf eksisterer. Ligeså forholder det sig med de to andre afhængigheder.

Generalisering/specialisering

Som det ses er nogle af entitetstyperne relateret ved et generaliserings/specialiserings-forhold, som resulterer i et *er-et* forhold. Dette er genspec-strukturen omkring entitetstypen **Medvirkende**.

O'et i cirklen fra **Medvirkende** og ned til subclasserne angiver, at der findes et overlap således, at fx en instruktør godt kan være skuespiller (dvs. den samme entitet kan være medlem af mere end én subklasse).



Total og delvis specialisering

I genspec-strukturen er anvendt en dobbelt linie fra entitetstypen **Medvirkende** og ned til cirklen. Dette angiver, at der er tale om en delvis specialisering, hvilket vil sige, at et medlem af **Medvirkende** skal være medlem af en af subklasserne.

Definerende prædikater

På genspec-strukturen **Medvirkende** er anvendt definerende prædikater. Det første prædikat er *Persontype*, som er sat på linien fra **Medvirkende** og ned til cirklen. Dette prædikat er anvendt pga. attributten *Persontype*, efter hvilken subklasserne klassificeres.



Indholdsfortegnelse

DIAGRAMMER TIL OOD 2

 PDC-DIAGRAM 2

 E/R-DIAGRAM..... 3

 HIC-DIAGRAM OVER LOGINPROCEDURE 4

 HIC-DIAGRAM OVER RESERVATIONSPROCEDURE 5

 HIC-DIAGRAM OVER SAMLET SYSTEM..... 6

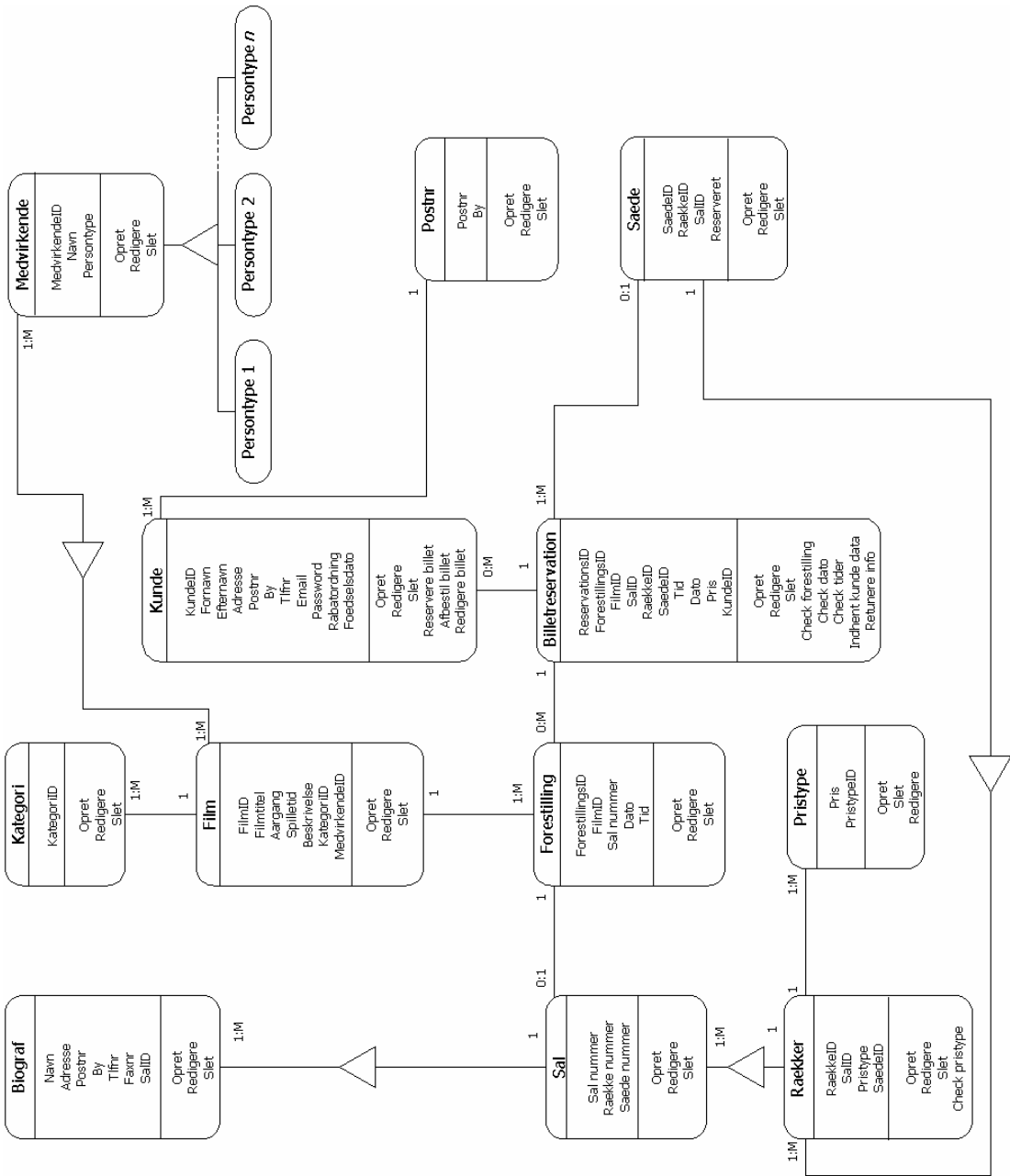
 INTERAKTIONSDIAGRAM OVER LOGINPROCEDURE..... 7

 INTERAKTIONSDIAGRAM OVER BESTILLING 8



Diagrammer til OOD

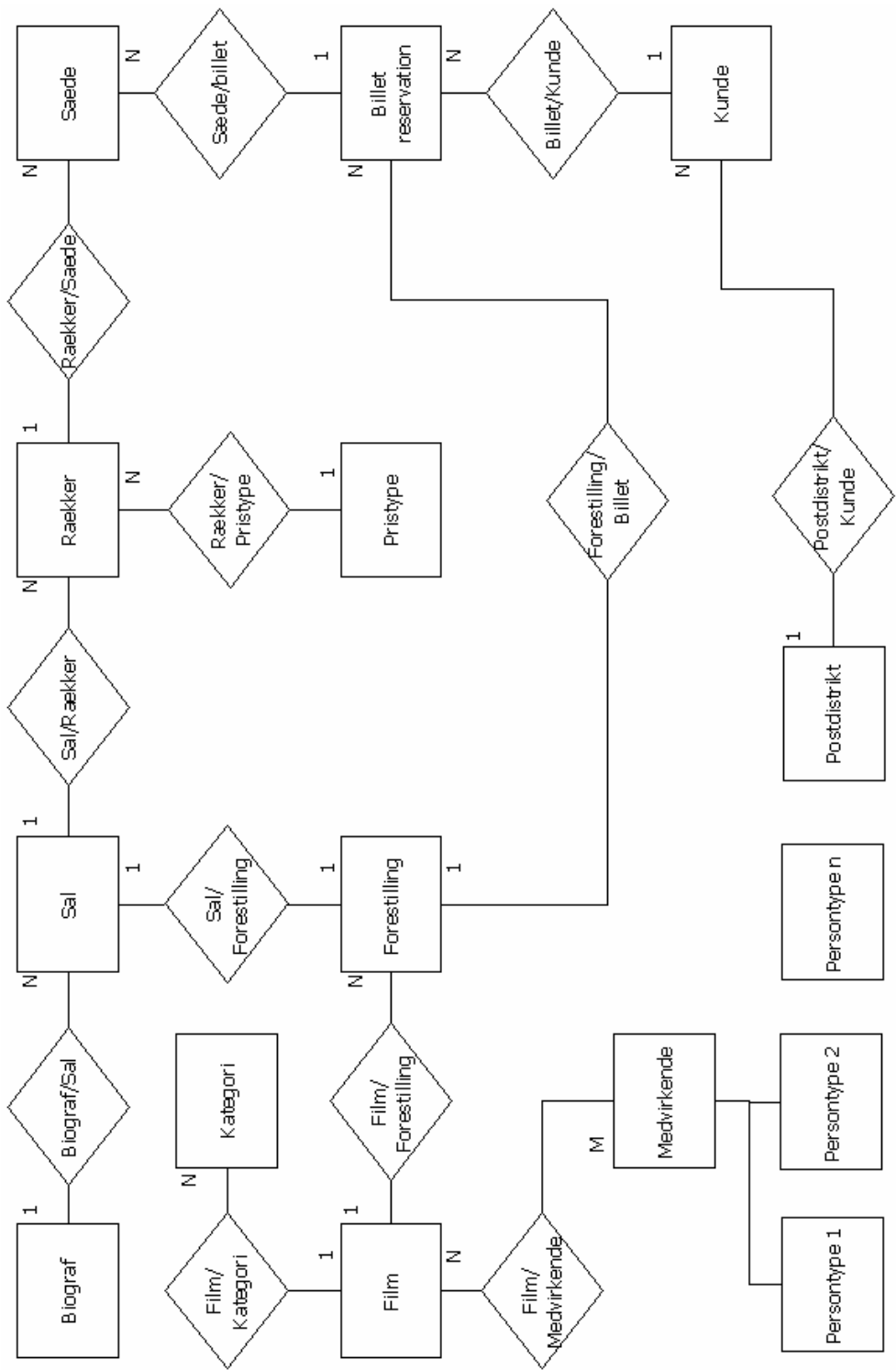
PDC-diagram



Figur D.1: Endelige version af PDC-diagram



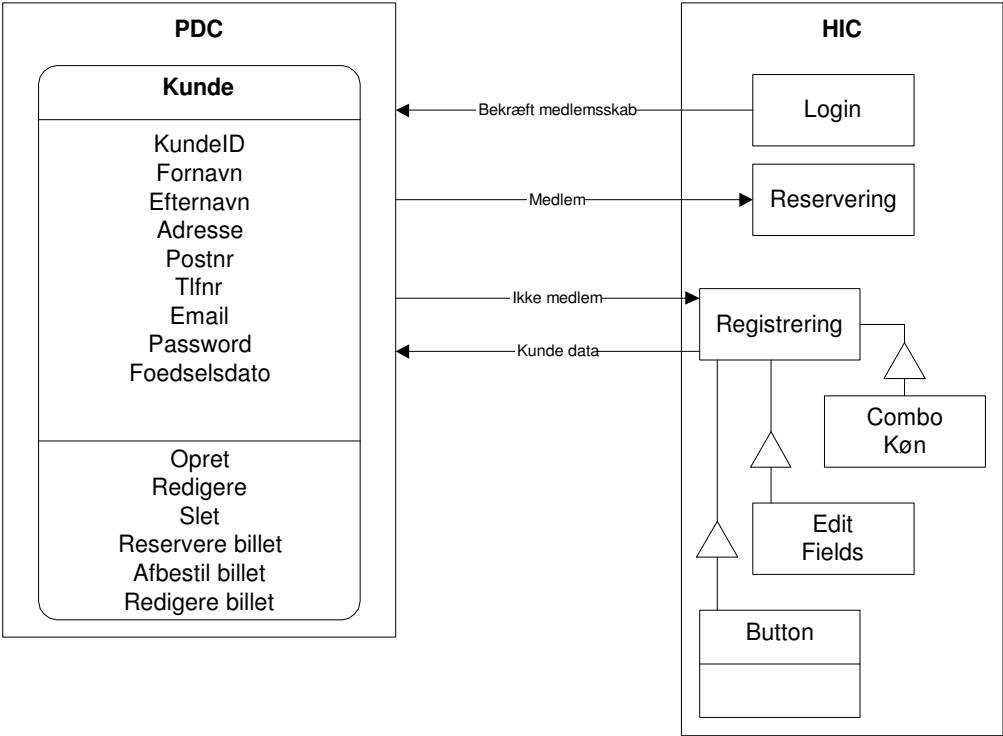
E/R-diagram



Figur D.2: Endelige version af E/R-diagram



HIC-diagram over loginprocedure



Figur D.3: HIC-digram over login-proceduren

```
graph LR
    subgraph PDC
        direction TB
        subgraph Billetreservation
            B1[ReservationsID  
ForestillingsID  
FilmID  
SalID  
RaekkeID  
SaedeID  
Tid  
Dato  
Pris  
KundeID]
        end
        subgraph Forestilling
            F1[ForestillingsID  
FilmID  
Sal nummer  
Dato  
Tid]
        end
        subgraph Sal
            S1[Sal nummer  
Raekke nummer  
Saede nummer]
        end
    end

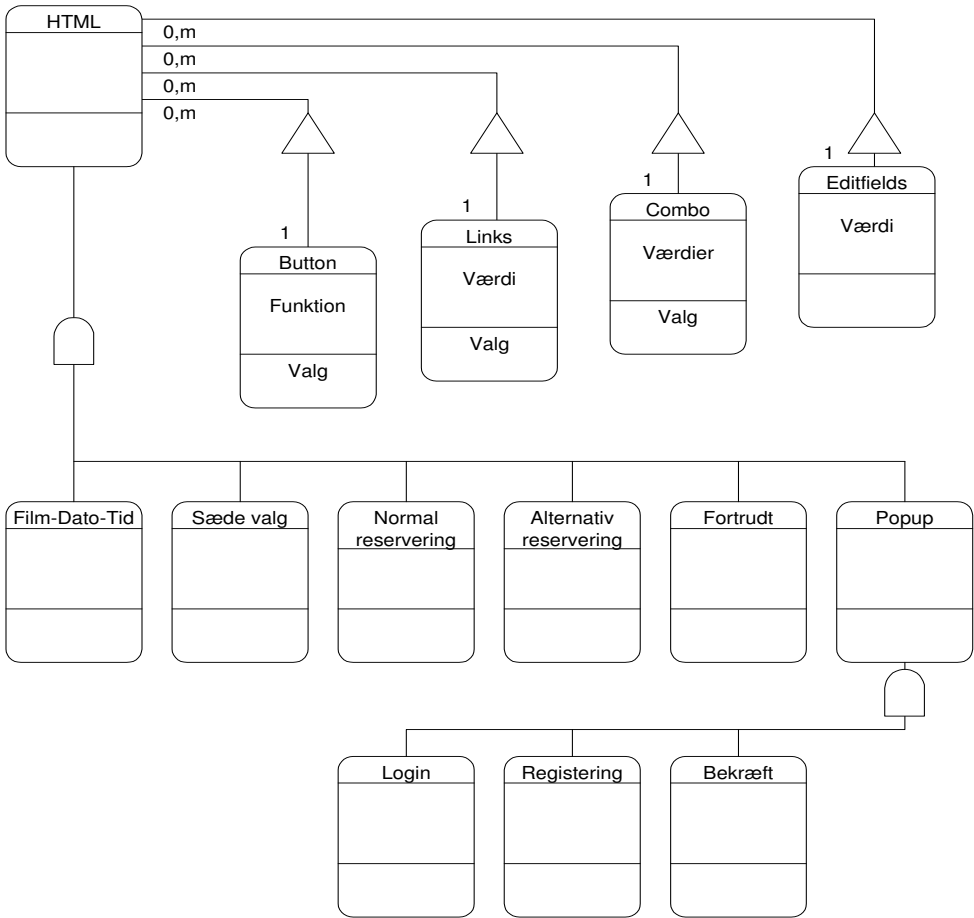
    subgraph HIC
        direction TB
        S1[Saedevalg]
        S2[Sallayout]
        S3[Normal Reservering]
        S4[Buttons]
        S5[Alternativ Reservering]
        S1 --|> S2
        S3 --|> S4
    end

    PDC -- "FilmID  
Dato  
Tid" --> S1
    S1 -- "Sæde & Række valg" --> PDC
    S1 -- "Rækker + Sæder" --> S2
    PDC -- "Ok" --> S3
    S3 -- "Fortryd" --> PDC
    PDC -- "Ikke Ok" --> S5
    S5 -- "Fortryd" --> PDC
    S5 --> S1
```

appendiks D – side 141 af 1



HIC-diagram over samlet system

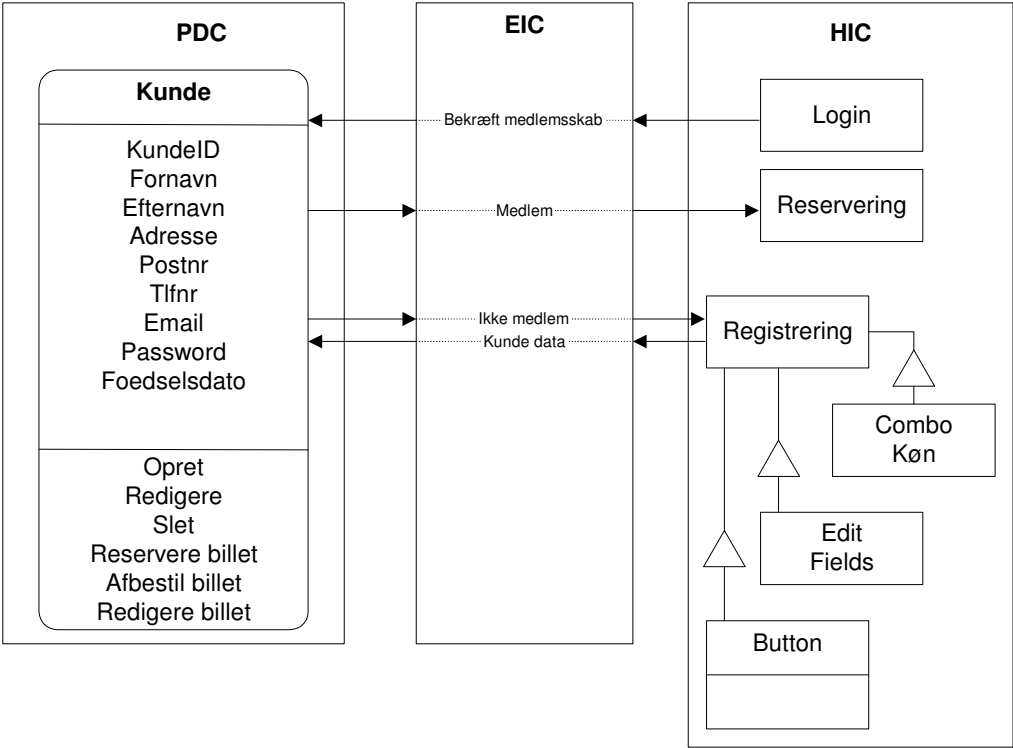


Figur D.5: HIC-diagram over det samlede system



Interaktionsdiagram over loginprocedure

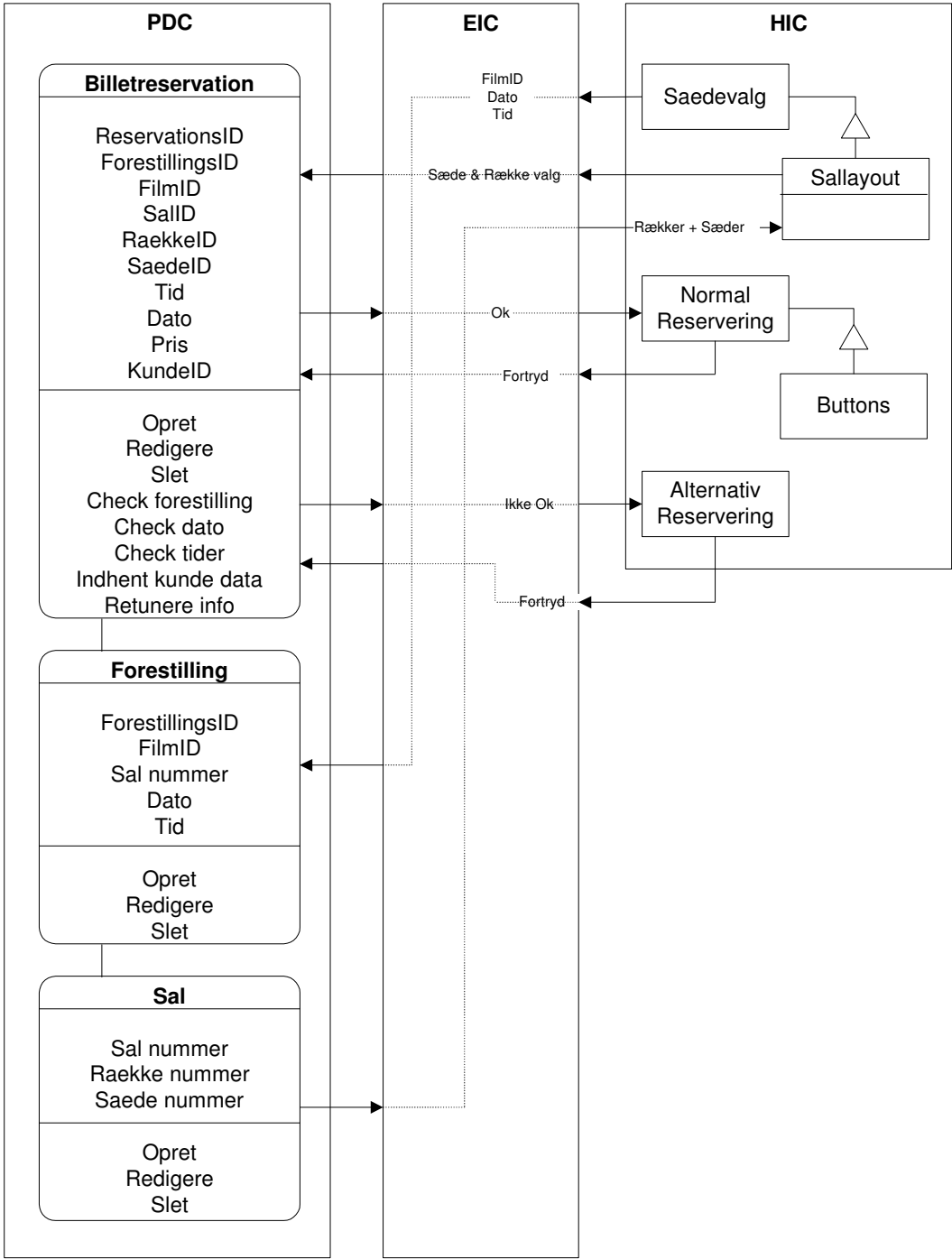
Figur D.6: Interaktionsdiagram over loginprocedure





Interaktionsdiagram over bestilling

Figur D.7: Interaktionsdiagram over bestilling





Indholdsfortegnelse

ODBC/JDBC	2
HVAD ER ODBC?	2
<i>Hvordan virker ODBC?</i>	2
HVAD ER JDBC?	3
<i>Hvad gør JDBC?</i>	4
<i>Hvordan virker JDBC med en JDBC/ODBC-bridge?</i>	4
JDBC vs. ODBC	4
SERVLETS	6
SERVLETS ISTEDET FOR CGI SCRIPTS	6
BRUG AF SERVLETS	6
SERVLET-PAKKENS ARKITEKTUR	7
<i>Servlet interface</i>	7
<i>Klientinteraktion</i>	7
<i>Interfacet fra Servlet Request</i>	7
<i>Interfacet for ServletResponse</i>	8
EN SIMPEL SERVLET	8
<i>Opsummering</i>	9
EN SERVLETS LIVSCYKLUS	10
<i>Initialisering af en servlet</i>	10
<i>Interaktion med klienter</i>	10
<i>Nedlæggelse af en servlet</i>	11

ODBC/JDBC

I dag, hvor udbredelsen af informationsbehov er stort – både for virksomheder og privatpersoner – er det nødvendigt at have et modul, som kan tale sammen med flere databaser og operativsystemer samtidig.

Hvis man alene ser på udvalget af databasesoftware i dag, kan en forholdsvis stor virksomhed få problemer med opbevaring af informationer, da det er sjældent at alle kører på den samme form for software. For, som princip, må virksomheden enten køre med en homogen database, dvs. det samme software ligger på alle maskiner, eller også må virksomheden indføre enten ODBC (Open DataBase Connectivity), JDBC eller anden global database styring, til styring af adgangen til deres databaser.

Den dyberegående gennemgang af JDBC og Servlets er baseret på information hhv. om JDBC fra <http://www.javasoft.com/products/jdbc/index.html> og Servlets fra <http://java.sun.com>.

Hvad er ODBC?

ODBC er et uafhængigt API (Application Programming Interface), hvilket vil sige at ligemeget hvilken relationel database man ønsker data fra, sørger API'en for denne tilgang vha. SQL-sætninger. Til hver af disse forskellige databaser findes der tillige en ODBC-driver, som konverterer forespørgslen fra den pågældende applikation til SQL-sætninger vha. ODBC, som så forespørger drivermanageren til en hvilken som helst DBMS.



Hvordan virker ODBC?

Hver applikation anvender ODBC APIen til at interagere med en eller flere datakilder gennem DBMS-specifikke drivere. ODBC-arkitekturen definerer fire komponenter:

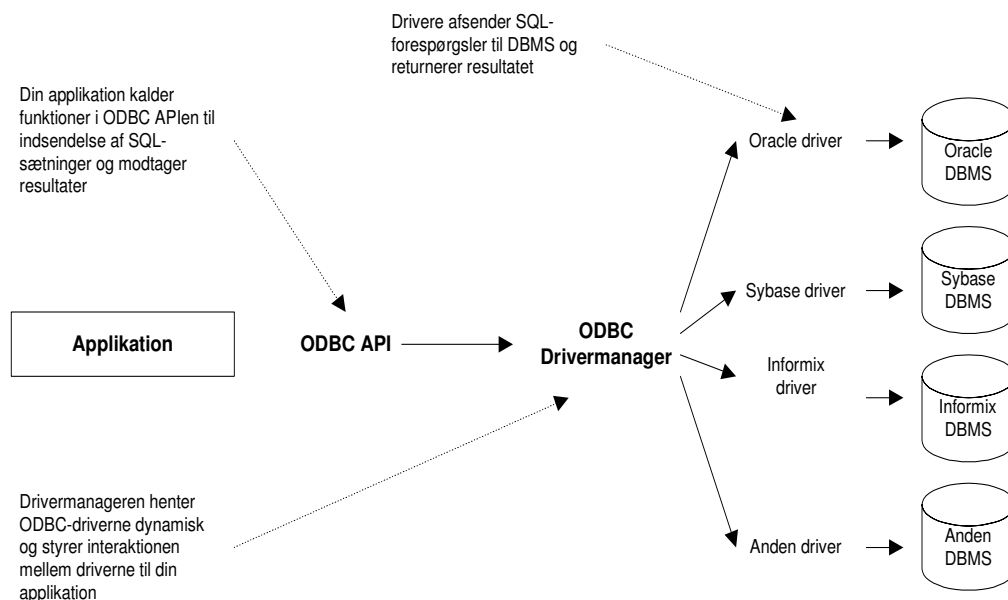
- Databaseapplikationen
- Drivermanageren
- Driveren
- Datakilden

Figuren¹⁴ på næste side illustrerer ODBC-konceptet:

Databaseapplikationen kalder ODBC-funktioner for hhv. at etablere forbindelse til, sende til, modtage fra samt afbryde forbindelsen til en datakilde. Databaseapplikationen behandler og viser data, som er modtaget via ODBC-driveren.

Drivermanageren leverer information til en applikation, loader drivere dynamisk efterhånden, som de bliver nødvendige.

Driveren udfører ODBC-funktionens kald og styrer alle dataudskiftninger mellem applikationen og den aktuelle DBMS. Driveren kan oversætte standard-SQL til andre datatilgangssprog, som er nødvendige for de forskellige datakilder.



Figur E.1: ODBC-konceptet

¹⁴ Hentet fra artiklen af Sippl, Roger: ODBC and Unix: Port no more?



Der er ikke så meget at sige om selve datakilderne andet end at disse kan være forskellige relationsdatabaser.

Hvad er JDBC?

JDBC er et Java API til at eksekvere SQL-sætninger. JDBC består af et sæt klasser og interfaces skrevet i Java. JDBC tilbyder en standard-API til værktøjs- og databaseudviklere, og gør det muligt at skrive databaseapplikationer ved at bruge en "ren" Java API.

Ved at bruge JDBC er det nemt at sende SQL-sætninger til hvilken som helst relationel database. Med andre ord er det med JDBC APIen ikke nødvendigt at skrive ét program for at tilgå en Sybase-database, ét andet for at tilgå en Oracle-database osv. Man kan derimod nøjes med at skrive et enkelt program ved at bruge JDBC APIen, og dette program kan så sende SQL-sætninger til den rigtige database. Tilmed behøver man ikke bekymre sig om at skrive forskellige applikationer, som skal køre på forskellige platforme. Kombinationen af Java og JDBC sætter programmøren i stand til at skrive programmet én gang og køre det hvor som helst.

Så når Java-applikationer skal tale med et bredt spektrum af forskellige databaser, er JDBC det rigtige valg.



Hvad gør JDBC?

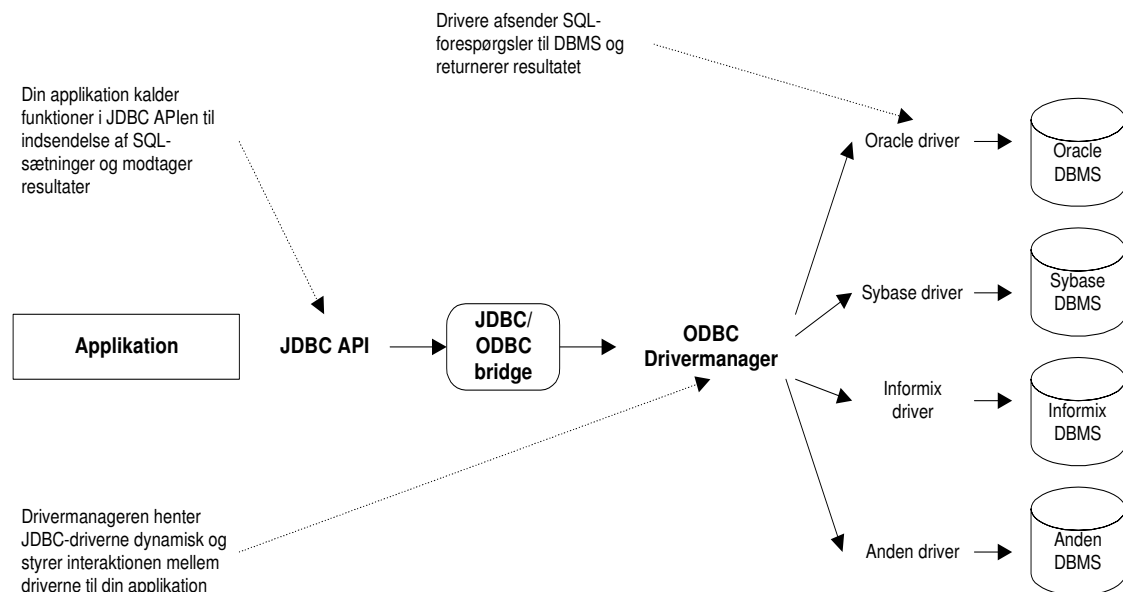
Sagt på en enkel måde muliggør JDBC tre ting:

- Etablerer en forbindelse til en database
- Sender SQL-sætninger
- Processerer resultatet

Hvordan virker JDBC med en JDBC/ODBC-bridge?

JDBC virker meget som ODBC, og fremgår af nedenstående figur¹⁵:

Databaseapplikationen kalder funktioner i JDBC APIen på samme måde, som den kalder funktioner i ODBC APIen. JDBC/ODBC-bridgen konverterer JDBC til ODBC og sender det til ODBC-drivermanageren og den anden vej rundt. Denne leverer så information til en applikation og loader driverne til de respektive



datakilder.

Figur E.2: JDBC/ODBC konceptet

JDBC vs. ODBC

Microsofts ODBC API er formentlig det mest anvendte programmeringsinterface til tilgang af relationelle databaser. ODBC tilbyder muligheden for, at man kan forbinde sig til næsten alle databaser på næsten alle platforme. Så hvorfor ikke bare bruge ODBC fra Java?

¹⁵ Udarbejdet ud fra forrige figur, hvor gruppen har tilføjet et JDBC API samt en JDBC/ODBC-bridge



Man kan faktisk bruge ODBC fra Java, men dette gøres bedst med lidt hjælp fra JDBC i form af en JDBC/ODBC-bridge. Men hvorfor behøver man JDBC? Nedenfor gives nogle grunde:

ODBC er ikke passende til direkte brug fra Java, da det anvender et C-interface. Kald fra Java til C-kode har nogle ulemper med hensyn til sikkerhed, implementation, robusthed og automatisk portabilitet af applikationer.

En bogstavelig oversættelse af ODBC C-APIen til en Java API er ikke at ønske. Fx bruger Java ikke pointere, men det gør ODBC. Man kan tænke på JDBC som ODBC oversat til et objektorienteret interface.

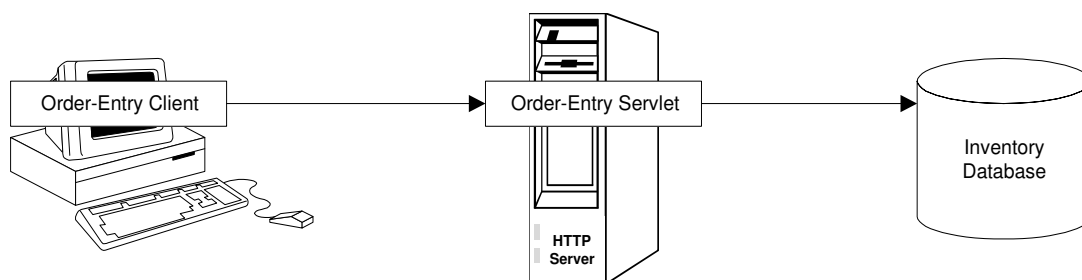
ODBC er svært at lære. Det mikser simple og avancerede features sammen og har komplekse valgmuligheder – selv for simple forespørgsler. JDBC, derimod, blev designet til at holde simple ting simple, mens det samtidig tillader mere avancerede muligheder, når disse behøves.

En Java API som JDBC er nødvendig for at muliggøre en "ren Java" løsning. Når ODBC bruges skal ODBC-drivermanageren og driverne installeres manuelt på hver klientmaskine. Når JDBC-driveren er skrevet kun i Java, er JDBC-koden automatisk installerbar, portabel og sikker på alle Java-platforme lige fra netværkscomputere til mainframes.

Kort sagt er JDBC APIen et naturligt Java-interface til de grundlæggende SQL-abstraktioner og begreber. Den bygger på ODBC i stedet for at starte på tegnebrættet, så derfor vil programmører, som er bekendt med ODBC finde det nemt at lære JDBC.

Servlets

Servlets er moduler, som udvider request/response-orienterede servere såsom servere, der kan køre Java. En servlet være fx være ansvarlig for at tage data i en HTML order-entry form og tilføre den forretningslogik, som bruges til at opdatere en virksomheds ordredatabase.



Figur E.3: Brugen af servlets

Servlets er for servere, hvad applets er for browsere. Men forskellen mellem servlets og applets er, at servlets ikke har en grafisk brugergrænseflade. Servlets kan implementeres på mange forskellige måder, da servlet APIen, som man bruger til at skrive servlets, antager intet om serverens omgivelser eller



protokol. Servlets bliver mest anvendt indenfor HTTP-servere og mange webservere understøtter Servlet API'en.

Servlets istedet for CGI scripts

Servlets er en effektiv erstatning for CGI scripts. De tilbyder en måde, hvorpå man kan generere dynamiske dokumenter, som er både nemmere at skrive og hurtigere at køre. Servlets tager også højde for problemet ved kørsel på forskellige platforme. Da de udvikles med Java Servlet API'en opnås platformsuafhængighed.

Brug af servlets

Her er to af de mange anvendelsesmuligheder for servlets:

Tillade kollaboration mellem mennesker. En servlet kan håndtere flere requests samtidig, og kan synkronisere requests. Dette tillader en servlet at understøtte systemer såsom online-konferencer.

Forwarding requests. Servlets kan forwarde requests til andre servere og servlets. Derfor kan servlets bruges til at balancere load mellem flere servere og til at partitionere en enkelt logisk service over flere servere.

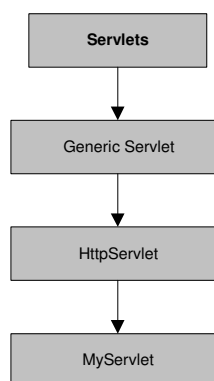
Servlet-pakkens arkitektur

Java.`javax.servlet` pakken tilbyder interfaces og klasser til at skrive servlets. Pakkens arkitektur beskrives nedenfor.

Servlet interface

Den centrale abstraktion i Servlet API'en er `Servlet` ^{api} interfacet. Alle servlets implementerer dette interface, enten direkte eller mere almindeligt ved at udvide en klasse som implementerer det som `HttpServlet` ^{api}.

Figur E.4: Servlets klassehierarki



Servlet-interfacet erklærer, men implementerer ikke metoder, som håndterer servlet'en og dens kommunikation med klienter.



Klientinteraktion

Når en servlet accepterer et kald fra en klient, modtager den to objekter:

- En `ServletRequest`^{api}, som indkapsler kommunikationen fra klienten til serveren
- En `ServletResponse`^{api}, som indkapsler kommunikationen fra servlet'en og tilbage til klienten

`ServletRequest` og `ServletResponse` er interfaces, som defineres af `javax.servlet` pakken.

Interfacet fra Servlet Request

`ServletRequest` interfaces giver servlet'en adgang til:

- Information såsom de parameternavne, som klienten kommer med, protokollen som bruges af klienten samt navnene på henholdsvis den fjerne host, som lavede forespørgslen, og serveren, som modtog forespørgslen.
- Inputstrømmen `ServletInputStream`^{api}. Servlets bruger inputstrømmen til at skaffe data fra klienter, som bruger applikationsprotokoller såsom HTTP POST og PUT.

Interfaces, som udvider `ServletRequest` interfacet, tillader servlet'en at få mere protokolspecifik data. `HttpServletRequest`^{api} interfacet indeholder fx metoder til at tilgå HTTP-specifikke header-information.

Interfacet for ServletResponse

`ServletResponse` interfacet giver servlet'en metoder til at svare klienten. Den:

- Tillader servlet'en at sætte længden for indholdet og MIME¹⁶ typen for svaret.
- Tilbyder en outputstrøm, `ServletOutputStream`^{api}, og en `Writer` gennem hvilken servlet'en kan sende svardata.

Interfaces, som udvider `ServletResponse` interfacet giver servlet'en flere protokolspecifikke muligheder. `HttpServletResponse`^{api} interfacet indeholder fx metoder, som tillader servlet'en at manipulere HTTP-specifik headerinformation.

En simpel Servlet

Den følgende klasse definerer en servlet:

```
public class SimpleServlet extends HttpServlet
{
    /**
     * Håndter HTTP GET metoden ved at bygge en simpel website
     */

    public void doGet (HttpServletRequest request, HttpServletResponse response)
```

¹⁶ MIME: Multipurpose Internet Mail Extensions. Fx en standard for, hvordan man skriver mails.



```
throws ServletException, IOException
{
    PrintWriter    Out;
    String  title = "Simple Servlet Output";

    //sæt indholdets type og andre response headerfelter først

    response.setContentType("text/html");

    //skriv så data af response

    out = response.getWriter();

    out.println("<HTML><HEAD><TITLE>");
    out.println(title);
    out.println("</TITLE></HEAD><BODY>");
    out.println("<H1>" + title + "</H1>");
    out.println("<P>Dette er output fra SimpleServlet.");
    out.println("</BODY></HTML>");
    out.close();
}
}
```

Opsummering

Dette er en kort gennemgang af ovenstående programeksempel:

SimpleServlet udvider HttpServlet klassen, som implementerer Servlet interfacet.

SimpleServlet overdefinerer doGet metoden i HttpServlet klassen. doGet metoden kaldes, når en klient laver en GET request, og resulterer i, at den simple HTML-side returneres til klienten.

Indeni doGet metoden:

- Brugers request repræsenteres ved et HttpServletRequest objekt
- Svaret til brugeren repræsenteres ved et HttpServletResponse objekt
- Da tekstdata returneres til klienten, sendes svaret ved at bruge Writer objektet, som kommer fra HttpServletResponse objektet.



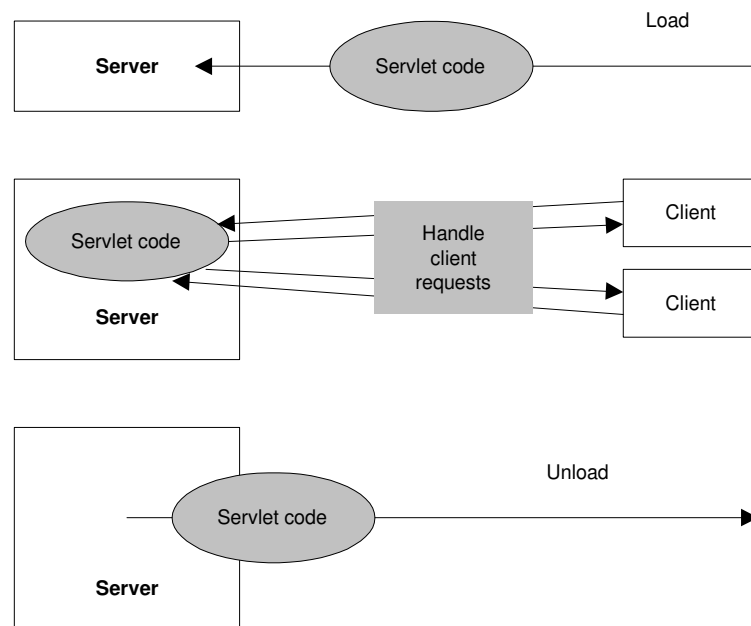
En servlets livscyklus

Hver servlet har samme livscyklus:

En server loader og initialiserer servlet'en

Servlet'en håndterer nul eller flere klientrequests

Serveren fjerner servlet'en (nogle servere gør kun dette, når de lukkes ned)



Figur E.5: En servlets livscyklus

Initialisering af en servlet

Når en server loader en servlet, kører serveren servlet'ens `init` metode.

Initialiseringen færdiggøres før klientrequests håndteres.

Selvom de fleste servlets kører i multitrådede servere, har servlets ingen samtidighedstilfælde under initialiseringen. Serveren kalder `init` metoden en gang, når serveren loader servlet'en, og kalder ikke `init` metoden igen, medmindre serveren reloader servlet'en. Serveren kan ikke reloadere servlet'en før serveren har nedlagt servlet'en ved at kalde `destroy` metoden.

Interaktion med klienter

Efter initialisering kan servlet'en håndtere klientrequests.

Nedlæggelse af en servlet

Servlets kører indtil serveren nedlægger dem, fx på systemadministratorens kommando. Når en server nedlægger en servlet, vil serveren ikke køre servlet'en igen før serveren reloader og reinitialiserer servlet'en.





Indholdsfortegnelse

WEB CONTENT MANAGEMENT [WCM].....	2
KRAV OM WEB CONTENT MANAGEMENT.....	2
Arkitekturen bag WCM.....	2
Optimal ydeevne.....	2
URL syntese.....	3
OPTIMERING AF YDEEVNE.....	3
Båndbredde ydeevne.....	3
Disk ydeevne.....	4
CPU ydeevne.....	4
ODB ER IDEELLE FOR LINK MANAGEMENT.....	4
RELATIONELLE DATABASER VS. OBJEKTORIENTERET DATABASER.....	5
CLIENT/SERVER BESKRIVELSE.....	6
Beskrivelse af serveren.....	6

Web Content Management [WCM]

Da vores hjemmeside sandsynligvis vil blive besøgt af rigtig mange brugere, vil dette betyde at der vil være en række krav til webservernes evne til at håndtere disse mange brugere og den information der skal sendes ud til brugeren.

Krav om Web Content Management

Webservere skal altså kunne håndtere mange brugere døgnet rundt. Det betyder derfor også at der skal overføres mere data, flere datatyper og dette vil ofte kræve, at der er mere end én person der skal stå for vedligeholdelse af hjemmesiden og dens indhold.

Derfor er det vigtigt, at man sørger for at man opnår en høj pålidelighed af de links og data som hjemmesiden indeholder samt at de hele tiden er up-to-date. Dette kan WCM-konceptet hjælpe med at opnå.

Arkitekturen bag WCM

WCM er faktisk bare et andet ord for en database, der består af webindhold, dvs. HTML, Javascripts, billeder, VRML [[Virtual Reality Modeling Language](#)] etc., og web-metadata, dvs. links, versions historie, accesshistorie etc., skal ligge i en database. De førnævnte krav om pålidelighed kan opnås, hvis man først genererer hjemmesidens indhold efter behovet fra brugeren.

Man skal altså bruge en database og en webserver, der kan modtage forespørgslerne fra brugerne på Internettet, og en eller anden form for Application Programming Interface [API], som kan foretage selve genereringen. Det kunne fx være ASP, NSAPI [Netscape Server Application Programming Interface], ISAPI [Internet Server API], CGI [Common Gateway Interface] eller Servlets. Sidstnævnte er den API vi har valgt at benytte os af.

Databasens kommunikationsform forgår vha. SQL, så for at kunne udnytte databasens indhold mangler vi en bro mellem APIen og WCMen (databasen). Her kan man benytte en såkaldt ODBC/JDBC-bridge, som sørger for, at omkonvertere APIens funktionskald til SQL forespørgsler.



Databasen kunne, som i vores tilfælde, være en relationel database, men en objektorienteret database ville være optimal, da en sådan har nogle flere fordele end en relationel database. En sammenligning mellem den relationelle database og den objektorienterede database kan ses i tabel F.1.

Optimal ydeevne

For at både webserver og WCM kan opnå den optimale forespørgselsydeevne, må de være istand til at kunne håndtere flere samtidige brugere. Webservere opretter generelt en ny tråd (proces) for hver forespørgsel den modtager fra Internettet. Mange databaser har også denne mulighed, men langt fra alle. Derfor er det vigtigt at undersøge hvad en database har at tilbyde netop på dette felt, hvis man vil bruge den som WCM.

Hele ideen med WCM er, at systemet bliver mere fleksibel, og derfor kan man faktisk godt slippe heldigt fra at have valgt en database der ikke kan klarer efterspørgslen af sine data, da man forholdsvis let kan skifte databasen ud med en anden. Man skal blot sørge for at man samtidigt også skifter sin ODBC/JDBC-bridge driver.

Dette må siges at være en stor fordel, både økonomisk og forretningsmæssigt. Udskiftningen vil nemlig tage langt mindre tid, end hvis man skulle skifte alt ud, derfor får man mindre utilfredse brugere og dermed mindre prestigetab.

URL syntese

Der findes også andre måder at skabe bedre fleksibilitet og pålidelighed på. Man kan fx gemme HTML indhold uden URL'er, men fastholde alle links som logiske associationer. Lige inden siden skal leveres til webserveren bliver disse erstattet med fysiske links (URLer). Denne egenskab vil ligge i API'en. Denne form for URL syntese nedsætter ydeevnen med 20%, men URL'en vil altid være korrekt. Bruges en flad-fil løsning har man URL'en liggende direkte i koden, og skal derfor opdateret hver gang der sker en ændring med target. Eksempel på fysisk og logisk link

```
<a href="http://www.nfb.dk/site/images/nfblogo.jpg">Nordisk Film Biografer</a>  
<a href="images/nfblogo.jpg">Nordisk Film Biografer</a>
```

Dette vil også medføre, at man let kan replikere indholdet over på en anden server, blot man bare beholder den samme filstruktur. Dette kaldes for, at man opretter en Mirror Site, da man laver en form for spejl af den oprindelige server.

Optimering af ydeevne

For at optimere både webserverens og databasens ydeevne kan man foretage forskellige tiltag, for at forbedre dette.

Båndbredde ydeevne

Båndbredde er den mængde af trafik et net maksimalt kan håndtere. En kombination mellem distribution og replikation, er muligvis den bedste løsning til at hjælpe på dette problem.



Distribution er, når man kopierer dele af indholdet til en anden lokation, server eller database.

Replikation er, når man tager en nøjagtig kopi af en mængde data og placerer det på en anden server eller database. Man kan derfor kalde replika for syntetisk båndbredde.

En af måderne til at bestemme hvad der behøves at blive replikeret er, efter hvilke sider der er blevet cachet for nyligt. Dette tilbydes også af OODB.

Disk ydeevne

Harddiskens I/O evne kan også være en flaskehals for ydeevnen, men dette kan løses med caching, klustering og replikering til spindler.

- *Caching* - For det meste får en hjemmeside de fleste hits (antal besøgende) på ca. 10% af hele hjemmesidens samlede indhold. Dette kaldes, det "hotte". Dette kunne fx være et firmas startside eller nyhedssektion. Når indholds volumen er lav og adgangen til disse data er høj, kan ekstra servercache hjælpe på problemet.
- *Klustering* - Man kan klustre webrelateret indhold på fortløbende sektorer på harddisken. Klikkes der fx på "produkter" vil alt tekst, billeder m.m., der er på produktsiden, blive hentet ved kun at læse en gang fra harddisken.

Denne fremgangsmåde kan forbedre ydeevne drastisk samt give mulighed for at hente andre relaterede sider ind i cachen på forhånd.

- *Replikering til spindler* - Man replikerer det "hotte" til flere servere, på den måde kan flere få adgang til det samtidigt. En router kan fordele de indkomne forespørgsler til de 3 server (Round Robin, SNMP [Simpel Network Management Protocol] baseret). Det "kolde", altså de resterende 90%, kan distribueres på normal vis.

En ulempe ved replikering er, at ydeevnen ved en skrivning nedsættes pga. at alle replika skal opdateres. En stor fordel er dog, at man forbedrer læseydeevnen til de applikationer der bliver læst mest.

Replikering er ikke speciel dyr i diskomkostninger, da administratoren selv kan vælge hvilke HTML-dokumenter der skal replikeres.

CPU ydeevne

CPUen kan også blive sat på en prøve, især når det gælder om at generere HTML kode o.lign. efter behov/forespørgelse. Dette kan betyde endog meget store krav om CPU kraft.

Et eksempel på dette kunne være, hvis man ønskede at generere en statistik ud fra et givent tidspunkt. Dette vil kræve, at CPUen skulle udregne alle tallene til statistikken og måske endda lave et billede med en kurve på.

For at forbedre dette, kunne man undgå at generere billeder, men i stedet vise det i form af en tabel.

ODB er ideelle for link management

En moderne OODBs har som regel faciliteter for fysisk klustering af data på disk, management af transaktioner og data replikation, som alle kan forbedre ydeevnen



langt mere end ved en flad-filsløsning som fx er tilfældet i Access. Derfor vil det være en fordel at vælge en OODB, når man skal arbejde med WCM. Fordelene ved OODB kan ses i tabel F.1.



Relationelle databaser vs. objektorienteret databaser

Da vi benytter Access som en slags dum database, mener vi ikke kravene til databasen er særlig vigtige, men det kunne alligevel være interessant at sammenligne mulighederne i en objektdatabase og i en relationel database som fx Access.

Objekt databaser	Relationelle databaser
Benytter netværksmodellen	Benytter den relationelle datamodel
Har egenskaber for Metadata management	Har egenskaber for Metadata management
Har egenskaber for samtidighedskontrol	Har egenskaber for samtidighedskontrol
Kan gemme alle former for rige datatyper	Kan ikke gemme rige datatyper
Har egenskaber for sofistikeret replikation [✕]	Har egenskaber for begrænset replikation
Understøtter ACID kriterierne	Understøtter ACID kriterierne
Understøtter lange transaktioner	Understøtter lange transaktioner*
Kan arbejdes på uden på, uden adgangen fra Internettet afbrudt.	Låser en tabel for Internet adgang, når den åbnes af DBAen.

Tabel F.1: Sammenligning af ODB og relationelle databaser

[✕] Man kan lave replikering efter hvilke sider der er blevet cache for nyligt

* Access fare - Brug af CommitTrans eller Rollback påvirker alle operationer på alle forbindelser og databasen inden for dette arbejdsrum

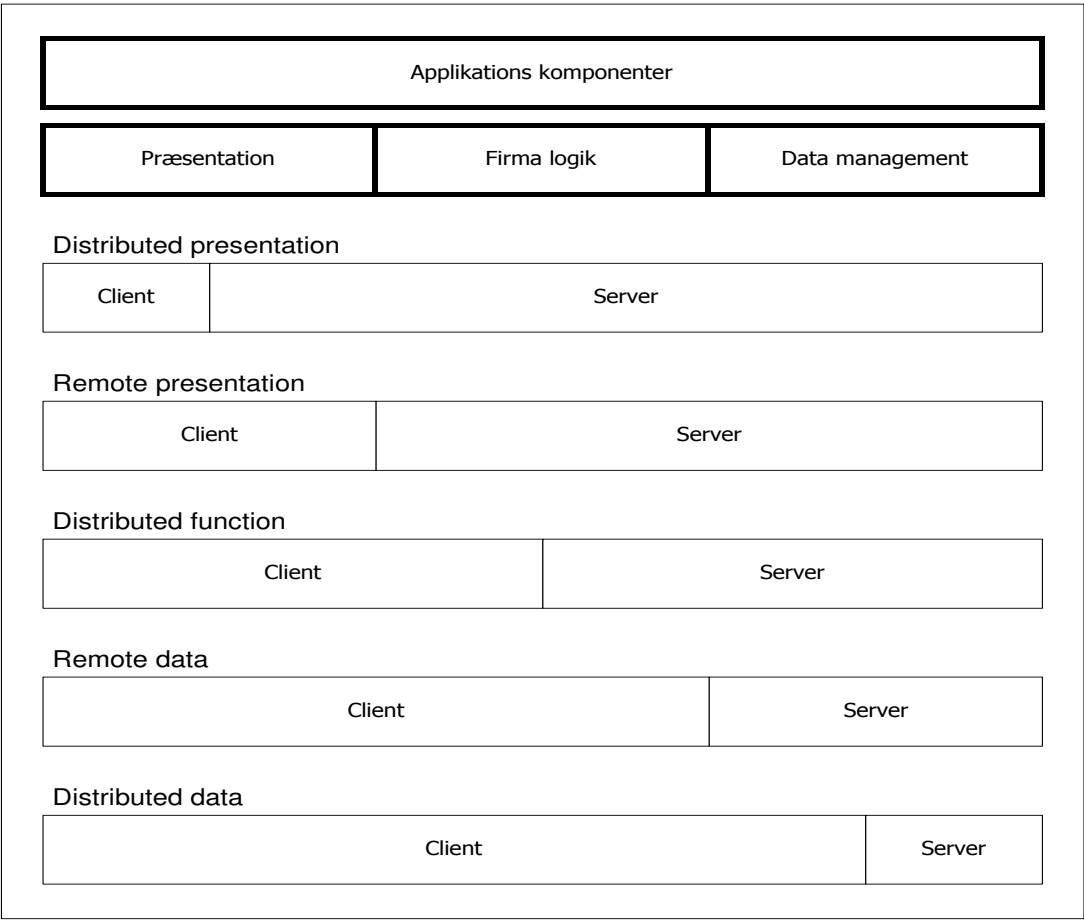
Client/server beskrivelse

Serverens rolle i det system som gruppen udarbejder til NFB, kan vises vha. de opstillede topologier for designovervejelser som G. C. Low¹⁷ beskriver i sin artikel vedr. Objektorienterede metrologier i et distribueret system. Grunden til, at gruppen vil benytte denne topologi er for at beskrive serverens rolle i systemet.

Beskrivelse af serveren

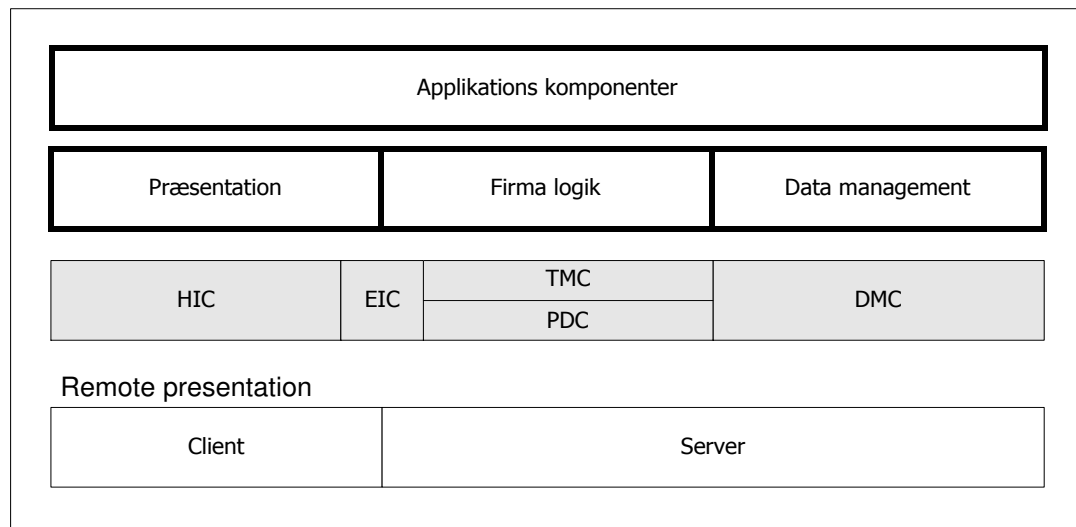
Gruppen benytter et allerede udarbejdet server program, der som tidligere beskrevet, sætter en proces i gang for hver klient, der er online på NFBs hjemmeside. Hvis man ser på serverens rolle i systemet, kan man se, at denne er rimelig central. Dette kan vises med nedenstående figur, som omhandler de topologier man benytter, når man bestemmer hvilke moduler der skal placeres hos serveren, og hvilke der bliver sendt til klienterne.

¹⁷ G. C. Low et al.: "Incorporation of distributed computing concerns into object-oriented methodologies"



Figur F.1: Gartner-modellen

Hvis man betragter Præsentation som HIC-komponenten, firma logik som PDC- og TMC-komponenterne og Data management som DMC-komponenten kan man ud fra denne figur se at gruppen har valgt at benytte applikationstopologien "remote presentation" i designet af serverens rolle. Gruppen har modificeret den topologi, så den passer til vores tilfælde, og for at give et bedre overblik over modulernes rolle i systemet. Det skal fastslås, at EICens placering er imellem klienten og serveren som kommunikationsled. Ud af disse overvejelser, fremstår følgende figur som en beskrivelse af serverens og klientens rolle i vores system.



Figur F.2: Client/Server modulopbygning

Hvis man sammenholder denne figur med afsnittet omkring komponenternes samspil (Del IV – kapitel 16) kan man helt klart danne sig et overblik over serverens rolle i systemet. Man kan ud fra figuren se, at hele HICen er placeret hos klienten, som taler sammen med serveren via EICen. Hos serveren findes så PDCen, som styrer tilgangen til DMCen og dermed databasen. TMCens rolle er, at styre de forskellige forespørgsler der er på databasen.

Brugen af disse design overvejelser sker fordi gruppen for det første ville fastslå serverens rolle i systemet samt modulernes placering og for det andet deres kommunikation med hinanden.



Indholdsfortegnelse

SERVLETS	2
CANCELSEATSERVLET.JAVA	2
CHECKSEATSERVLET.JAVA	5
GREETUSERSERVLET.JAVA	10
LOGINSERVLET.JAVA	12
LOGOUTSERVLET.JAVA	15
MOVIEACTORINFOSERVLET.JAVA	16
MOVIEINFOSERVLET.JAVA	22
MOVIESHOWSERVLET.JAVA	27
ORDERSEATSERVLET.JAVA	33
RESERVATIONSERVLET.JAVA	35
SHOWHALLSERVLET.JAVA	40
SHOWORDERSERVLET.JAVA	44
TRYLOGINSERVLET.JAVA	49
KLASSERNE	51
BILLETPRIS.JAVA	51
BILLETRESERVATION.JAVA	53
COOKIECHECK.JAVA	55
FILM.JAVA	56
FORESTILLING.JAVA	58
JDBC.JAVA – SUPERKLASSE	60
KATEGORI.JAVA	62
KUNDE.JAVA	63
SAL.JAVA	69
SALLAYOUT.JAVA – APPLLET	70
PREGENEREREDE HTML SIDER	80
ABOUT.HTML	80
ERROR_RESERVATION.HTML	81
INDEX.HTML	82
LOGIN.HTML	83
MAIN.HTML	88
MENU.HTML	89
NO_CARD_INFO.HTML	90
NO_LOGIN.HTML	91
NO_USER_BOTTOM.HTML	92
NO_USER_FOUND.HTML	93
TOP.HTML	94
TRYLOGIN.HTML	95

Servlets

Følgende afsnit indeholder al servlet-relateret Javakode sorteret i alfabetisk orden.

CancelSeatServlet.java

```
/**
 * @(#)CancelSeatServlet.java
 * @version 1.3
 *
 * Copyright (c) August–November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theil (peter@conquerware.dk).
 * All Rights Reserved.
 */
```



```

* Requirements:
* + Java Servlet Developers Kit 2.0 (SDK2.0)
*
* Notes:
* <none>
*
*/

import java.io.*;
import java.sql.*;

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Deletes a reservation from database and redirects page to
 * ReservationsServlet to get all remaining reservations. We're
 * implementing SingleThreadModel since we're going to update the
 * database and don't want anybody to disturb it while doing so.
 */
public class CancelSeatServlet extends HttpServlet
implements SingleThreadModel {

    /**
     * Private variables for in-parameters
     */
    private int m_forestillingID;
    private int m_kundeID;
    private int m_raekke;
    private int m_saebe;

    /**
     * Private connection to database
     */
    private Connection con;

    /**
     * Initialize servlet by first calling parent configuration, then
     * finding JDBC/ODBC Bridge driver and last make the connection to
     * ODBC database (MS Access).
     */
    public void init(ServletConfig config) throws ServletException {

        // Perform standard configuration for Servlet
        super.init(config);

        try {

            // Load the jdbc-odbc bridge driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            // Attempt to connect to a driver. Each one
            // of the registered drivers will be loaded until
            // one is found that can process this URL
            con = DriverManager.getConnection("jdbc:odbc:NEB",
            "svend", "jensen");

        }
        catch(Exception e) {
            throw new UnavailableException(this,
            "Unable to open a connection to the Access DB");
        }

    } // -> public void init(ServletConfig config) throws ServletException

    public void destroy() {

        try {

            // Close the connection to the database
            con.close();

        }
        catch(SQLException sqle) {

            System.out.println("Unable to close Connection");

            while (sqle != null) {
                System.out.println();
                System.out.println("SQLState: " + sqle.getSQLState());
                System.out.println("Message: " + sqle.getMessage());
                System.out.println("Vendor: " + sqle.getErrorCode());
                sqle = sqle.getNextException();
            }

        } // -> catch (SQLException sqle)

    } // -> public void destroy()

    /**
     * Handles a GET request from Client by reading parameters
     * and delete specified seat.
     */
    public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

```



```

/**
 * Get all parameters from query. Four parameters are needed to
 * delete a seat from database: forestillingID, kundeID, rækkeID
 * and sædeID.
 */
m_forestillingID = Integer.parseInt(req.getParameter("forestillingID"));
m_kundeID = Integer.parseInt(req.getParameter("kundeID"));
m_række = Integer.parseInt(req.getParameter("rækkeID"));
m_sæde = Integer.parseInt(req.getParameter("sædeID"));

/**
 * Make SQL query using retrieved four parameters as input.
 */

final String mSQL = "DELETE FROM billetreservation "
    + "WHERE (forestillingID = " + m_forestillingID
    + " AND kundeID = " + m_kundeID
    + " AND række = " + m_række
    + " AND sæde = " + m_sæde + ")";

try {

    Statement stmt = con.createStatement();
    stmt.executeUpdate(mSQL);

    stmt.close();

}
catch (SQLException ex) {

    /**
     * Error occurred while trying to delete seat from database.
     */

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<HTML><HEAD><TITLE>Fejl ved afbestilling</TITLE></HEAD>");
    out.println("<BODY BGCOLOR=#333333 TEXT=#FFFFFF " +
        "LINK=#CC9933 VLINK=#CC9933 ALINK=#CC9933>");

    out.println("Fejl i afbestilling af sædeligde!<BR>");
    out.println("<TD>");
    while (ex != null) {
        out.println("<BR>SQLState: " + ex.getSQLState ());
        out.println("<BR>Message: " + ex.getMessage ());
        out.println("<BR>Vendor: " + ex.getErrorCode ());
        ex = ex.getNextException();
    }
    out.println("</TD>");

    out.println("</BODY></HTML>");

}

/**
 * Redirect browser to ReservationServlet which now will find
 * all reservations minus the one we just removed from database.
 */

String location = "/servlet/reservations";

String scheme = req.getScheme();
String host = req.getServerName();
int port = req.getServerPort();

String redirectString = scheme + "://" + host;
if (port != 80) {
    redirectString += ":" + port;
}

redirectString += location;

res.sendRedirect(redirectString);

} // -> public void doGet(HttpServletRequest req, HttpServletResponse res)
}

```



CheckSeatsServlet.java

```

/**
 * @(#)CheckSeatsServlet.java
 * @version 1.8
 *
 * Copyright (c) August–November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (SDK2.0)
 *
 * Notes:
 * <none>
 *
 */

import java.io.*;
import java.util.*;
import java.text.*;
import java.lang.Integer;
import java.awt.*; // Point

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

import Billetpris;
import BilletReservation;
import Film;
import Forestilling;
import Sal;

import CookieCheck;

/**
 * Checks if seats are available and write a reservation to
 * Database if they are. We're implementing SingleThreadModel
 * since we're going to insert a new reservation into database.
 */
public class CheckSeatsServlet extends HttpServlet
    implements SingleThreadModel {

    private int forestillingID = -1;
    private int kundeID = -1;
    private Vector vSeats;

    Film m_Film;
    Forestilling m_Forestilling;
    Sal m_Sal;
    Billetpris m_billetpris;
    BilletReservation m_BilletReservation;

    /**
     * Handles a 'GET' request from a Client by reading parameters
     * and see if all specified seats are available for reservations.
     */
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        /**
         * Read 'forestillingID' from parameter query.
         */
        forestillingID = Integer.parseInt(req.getParameter("forestillingID"));

        /**
         * Find out if user has logged in by checking Cookie.
         */
        CookieCheck cookie = new CookieCheck(req);

        if (!cookie.isLoggedIn()) {

            /**
             * No user was logged into our system so we will have to
             * inform him/her of this by redirecting user to a predefined
             * 'no user logged in' HTML page.
             */

            String location = "/no_login.html";

            String scheme = req.getScheme();
            String host = req.getServerName();
            int port = req.getServerPort();

            String redirectString = scheme + "://" + host;
            if (port != 80) {
                redirectString += ":" + port;
            }

            redirectString += location;
            res.sendRedirect(redirectString);

        } else {

```



```

/**
 * User was logged into system so we will have to write
 * selected seats to database and return status of the
 * operation.
 */

vSeats = new Vector();

String s;
int _i = 0;
int _x = 0;
int _y = 0;

// Get seats 'seat0', ..., 'seat6' from parameters
try { s = req.getParameter("seat"+_i); }
catch (Exception e) { s = null; };

while(s != null) {

    try {
        _x = Integer.parseInt(s.substring(0, s.indexOf("-")));
        _y = Integer.parseInt(s.substring(s.indexOf("-")+1));
    }
    catch (Exception e) {
        System.out.println("Error reading: " + s);
    }

    Point p = new Point(_x, _y);
    vSeats.addElement(p);

    _i++;

    try { s = req.getParameter("seat"+_i); }
    catch (Exception e) { s = null; };

}

// ** All seats are now placed in a Vector Object called vSeats

// Find out if the seats are available
mBilletReservation = new BilletReservation(forestillingID,
cookie.getKundeID(), vSeats);

mForestilling = new Forestilling(forestillingID);

// Get 'filmID' by reading Forestilling table using provided ID
mFilm = new Film(mForestilling.getFilmID());
dSal = new Sal(mForestilling.getSalID());

System.out.println("T Billetpris");
mBilletpris = new Billetpris(mForestilling.getSalID());
System.out.println("D Billetpris");

/**
 * Calculate price
 */
int price = 0;

for (int i = 0; i < vSeats.size(); i++) {

    Point p = (Point)vSeats.elementAt(i);

    if (p != null)
        price += mBilletpris.getPrice(p.x);

}

if (!mBilletReservation.getSuccess()) {

    /**
     * It was not possible to reserv selected seats. This
     * happens if another user just has reserved them and
     * but the Applet still shows them as available.
     */

    String location = "/error_reservation.html";

    String scheme = req.getScheme();
    String host = req.getServerName();
    int port = req.getServerPort();

    String redirectString = scheme + "://" + host;
    if (port != 80) {
        redirectString += ":" + port;
    }

    redirectString += location;
    res.sendRedirect(redirectString);

} else {

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<H1>");
    out.println("<HEAD>");
    out.println("<TITLE>Plads reservation</TITLE>");

```



```

out.println("</HEAD>");
out.println("<BODY BGCOLOR=#333333 TEXT=#FFFFFF LINK=#CC9933 "
    + "VLINK=#CC9933 ALINK=#CC9933 TOPMARGIN=0 LEFTMARGIN=0 "
    + "MARGINWIDTH=0 MARGINHEIGHT=0>");

out.print("<IMG SRC=\"/graphics/bestil_billetter.gif\" "
    + "WIDTH=474 HEIGHT=59>");

out.println("<CENTER>");
out.println("<TABLE WIDTH=98% BORDER=0 CELLSPACING=0 CELLPADDING=0>");
out.println("<TR>");
out.println("    <TD BGCOLOR=#999966 VALIGN=TOP>"
    + "<IMG SRC=\"/graphics/02left.gif\" WIDTH=15 HEIGHT=15 "
    + "BORDER=0 ALT=\"Left Curve\"></TD>");

// Insert Title
out.println("    <TD BGCOLOR=#999966 WIDTH=100% ALIGN=CENTER NOWRAP>"
    + "<FONT FACE=\"Verdana, Geneva, Arial\" SIZE=5 "
    + "COLOR=#000000><B> + m_Film.getTitle() + "</B></FONT>"
    + "</TD>");

// Insert applet
out.println("    <TD ROWSPAN=3 VALIGN=TOP>");

out.println("<APPLET CODEBASE=\"/\" CODE=\"SalLayout.class\" "
    + "WIDTH=420 HEIGHT=315>");

/**
 * Transfer forestillingID to applet (this makes it
 * possible to write selected seats to database).
 */
out.println("<PARAM NAME=\"forestillingID\" "
    + "VALUE=\"" + forestillingID + "\">");

/**
 * Now, we'll have to parse testfile with all seats in
 * specified hall. The testfile is parsed by using the
 * StreamTokenizer Object which will allow us to get every
 * number (integer) separately, because every number are
 * padded with a space.
 */

FileReader r = new FileReader(req.getRealPath("/") + dSal.getMap());
StreamTokenizer st = new StreamTokenizer(r);

try {

    int numSect;
    int x, y, columns, rows, sCol, sRow, Rcolor, Gcolor, Bcolor;
    numSect = 0;

    x = 90; y = 100; columns = 110; rows = 120; sCol = 130; sRow = 140;
    Rcolor = 0; Gcolor = 0; Bcolor = 0;

    int nt;
    while ( (nt = st.nextToken()) != StreamTokenizer.TT_EOF ) {

        x = (int)st.nval;

        st.nextToken();
        y = (int)st.nval;

        st.nextToken();
        columns = (int)st.nval;

        st.nextToken();
        rows = (int)st.nval;

        st.nextToken();
        sCol = (int)st.nval;

        st.nextToken();
        sRow = (int)st.nval;

        st.nextToken();
        Rcolor = (int)st.nval;

        st.nextToken();
        Gcolor = (int)st.nval;

        st.nextToken();
        Bcolor = (int)st.nval;

        /**
         * Add all read informations to <PARAM> tag.
         */
        out.println("<PARAM NAME=\"sect\" + numSect + \"\" VALUE=\""
            + x + ", " + y + ", " + columns + ", " + rows + ", " + sCol + ", "
            + sRow + ", " + Rcolor + ", " + Gcolor + ", " + Bcolor + "\">");

        numSect++;
    }

} catch (NoSuchElementException e) {
    System.out.println(e.getClass().getName());
}

```



```

catch(IOException ioe) { System.out.println("Error reading mapping..."); }

// Load all reserved seats
Vector vTemp = m_BilletReservation.getReservedSeats(forestillingID);

for (int i = 0; i < vTemp.size(); i++) {

    Point p = (Point)vTemp.elementAt(i);
    out.println("<PARAM NAME=\"reserv\" + i + \"\" VALUE=\"\" + p.x
        + \",\" + p.y + \"\">");

}

// Load used-selected seats
for (int i = 0; i < vSeats.size(); i++) {

    Point p = (Point)vSeats.elementAt(i);
    out.println("<PARAM NAME=\"pick\" + i + \"\" VALUE=\"\" + p.x
        + \",\" + p.y + \"\">");

}

out.println("</APPLET>");
out.println("</TD>");

out.print("<D bgcolor=#999966 align=right valign=top height=33>");
out.println("<IMG SRC=\"\"/graphics/02right.gif\" width=16 height=15 \"
    + \"border=0 alt=\"\"Right Curve\"\"></D>");
out.println("</TR>");

out.println("<TR>");
out.print("<D bgcolor=#CC9933 colspan=2 align=center>");
out.println("<IMG SRC=\"\"/graphics/pixel.gif\" width=1 height=2 \"
    + \"border=0 alt=\"\"></D>");
out.println("<D bgcolor=#CC9933 align=center\"
    + \"<IMG SRC=\"\"/graphics/pixel.gif\" width=1 height=2 \"
    + \"border=0 alt=\"\"></D>");
out.println("</TR>");

out.println("<TR valign=\"\"top\">");
out.println("<TD><IMG SRC=\"\"/graphics/pixel.gif\" width=15 \"
    + \"height=315 border=0 alt=\"\"></TD>");
out.println("<TD>");
out.println("<D align=center>");
out.println("<TABLE width=100% cellpadding=0 cellspacing=0>");
out.println("<TR bgcolor=#666666>");

// Insert date
out.println("<D height=18 align=left nowrap>\"
    + \"<FONT FACE=\"\"Verdana, Geneva, Arial\"\" size=1>\"
    + \"<br>\" + m_Forestilling.getFormattedDate()
    + \"</FONT>></TD>");

// Insert time
out.println("<D align=right nowrap>\"
    + \"<FONT FACE=\"\"Verdana, Geneva, Arial\"\" size=1>\"
    + \"Klokken \" + m_Forestilling.getFormattedTime()
    + \"<br></FONT>></TD>");
out.println("</TR>");

out.println("<TR>");
out.println("<D colspan=2 align=justify>");
out.println("<FONT FACE=\"\"Verdana, Geneva, Arial\"\" size=4>\"
    + \"<b>Sådan reserveret</b></FONT><b>");
out.println("<b>");

/**
 * Okay, this is a bit cumbersome. I want to show user an
 * absolute time of when he has to pick up his reserved
 * tickets. Therefore I will have to subtract 15 minutes
 * from time of show. I create a Calendar object (it makes
 * it possible to subtract minutes) and initializes it
 * with time read from Forestilling object. Then I
 * subtract 15 minutes and retrieves the new time (hours
 * and minutes) from it; padding a zero (0) if minutes are
 * below 10 (only one figure).
 */

Calendar calGetTickets = Calendar.getInstance();
calGetTickets.setTime((Date)m_Forestilling.getTime());
calGetTickets.add(Calendar.MINUTE, -15);

String TicketsPickup = calGetTickets.get(Calendar.HOUR_OF_DAY) + ":";
TicketsPickup += (calGetTickets.get(Calendar.MINUTE)<10 ? "0" : "")
    + calGetTickets.get(Calendar.MINUTE);

out.println("<FONT FACE=\"\"Verdana, Geneva, Arial\"\">\"
    + \"<p align=justify>Alle valgte sæligerder er blevet \"
    + \"reserveret! Husk, at De skal hente billetterne senest \"
    + \"kl. \" + TicketsPickup + \"<b><br>salen til højre, \"
    + \"er dit valg angivet med lysegrøn og andre \"
    + \"reserverede sæligerder er angivet med rødsalshvid.\"
    + \"</TD>");
out.println("</TR>");

out.println("<TR><TD><br></TD></TR>");

```




```

// Insert price
out.println(" <tr>");
out.println(" <td align=center colspan=2>");
out.println(" <table width=80%>");
out.println(" <tr>");
out.println(" <td><NR><TD>" + vSeats.size()
+ " s&aelig;de" + (vSeats.size() == 1 ? "" : "r")
+ " til i alt</TD></NR></TD>");
out.println(" <td align=right><NR><TD><B>kr. " + price
+ ",</B></TD></NR></TD>");
out.println(" </tr>");
out.println(" </table>");
out.println(" </td>");
out.println(" </tr>");

// Insert 'pay for tickets' button
out.println(" <tr>");
out.println(" <td align=center colspan=2><B><B>"
+ "&A href=/servlet/showorders">"
+ "<img src=/graphics/betal_billetten_button.gif" "
+ "height=17 width=126 border=0/></td>");
out.println(" </tr>");

out.println(" ");
out.println(" </table>");
out.println(" ");
out.println(" </td>");
out.println(" <td><img src=/graphics/pixel.gif" width=15 "
+ "height=315 border=0 alt=\\\"></td>");
out.println("</tr>");

out.println("</table>");
out.println("</center>");

out.println("</body>");
out.println("</html>");

out.close();

} // --> if (mBilletReservation.getSuccess()) else

} // --> if (!cookie.isLoggedin()) else

} // --> public void doGet(HttpServletRequest req, HttpServletResponse res)

}

```



GreetUserServlet.java

```

/**
 * @(#)GreetUserServlet.java
 * @version 1.2
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (SDK2.0)
 *
 * Notes:
 * <none>
 *
 */

import java.io.*;

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Greet user when he/she logs into our system by changing bitmap
 * in bottom frame and print customers alias and email.
 */
public class GreetUserServlet extends HttpServlet {

    private int m_kundeID;

    /**
     * Handles a GET request from a Client.
     */
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
        ServletException, IOException {

        try {
            m_kundeID = Integer.parseInt(req.getParameter("kundeID"));
        }
        catch (NumberFormatException nfe) {
            m_kundeID = -1;
        }

        if (m_kundeID != -1) {

            /**
             * Get user informations by creating a new 'Kunde' object.
             */
            Kunde k = new Kunde(m_kundeID);

            res.setContentType("text/html");
            PrintWriter out = res.getWriter();

            out.println("<HTML>");
            out.println("<HEAD>");
            out.println(" <TITLE>Greet User " + k.getAlias() + "</TITLE>");
            out.println("</HEAD>");

            out.println("<BODY BGCOLOR=#000000 TEXT=#FFFFFF LINK=#CC9933 "
                + "VLINK=#CC9933 ALINK=#CC9933 TOPMARGIN=0 LEFTMARGIN=0 "
                + "MARGINWIDTH=0 MARGINHEIGHT=0>");

            out.println("<TABLE BORDER=0 CELLSPACING=2 CELLPADDING=2>");
            out.println("<TR>");
            out.println("<TD VALIGN=MIDDLE>"
                + "<IMG SRC=\"/graphics/"
                + (Integer.parseInt(k.getGender()) == 0 ? "boy.gif" : "girl.gif")
                + ".gif\" WIDTH=20 HEIGHT=38></TD>");
            out.println("<TD WIDTH=100% VALIGN=MIDDLE>");
            out.println("<FONT FACE=\"Tahoma, Verdana, Geneva, Arial\" "
                + "SIZE=1 COLOR=#6699CC>");
            out.println("<B>" + k.getAlias() + "</B><BR>");
            out.println(k.getBrail());
            out.println("</FONT>");
            out.println("</TD>");

            // Is online payment available? i.e. is there any card info?
            out.println("<TD ALIGN=RIGHT>"
                + "<FONT FACE=\"Tahoma, Verdana, Geneva, Arial\" SIZE=2 "
                + "COLOR=#CC99CC>Online&nbsp;&nbsp;&nbsp;betaling&nbsp;&nbsp;&nbsp;<BR>&nbsp;&nbsp;&nbsp;"
                + (k.isCardLoaded() ? "" : "<B>Ikke</B>")
                + "&nbsp;&nbsp;&nbsp;tillgængelig&nbsp;&nbsp;&nbsp;</FONT></TD>");

            out.println("</TR>");
            out.println("</TABLE>");

            // End HTML document
            out.println("</BODY>");
            out.println("</HTML>");
            out.close();

        } else { // -> if (m_kundeID != -1)

```



```
/**
 * No valid user has logged on so we will have to give him
 * a 'no user logged in' HTML in bottom frame.
 */

String location = "/no_user_bottom.html";

String scheme = req.getScheme();
String host = req.getServerName();
int port = req.getServerPort();

String redirectString = scheme + "://" + host;
if (port != 80) {
    redirectString += ":" + port;
}

redirectString += location;
res.sendRedirect(redirectString);

} // -> if (n_kundeID != -1) else

} // -> public void doGet(HttpServletRequest req, HttpServletResponse res)

}
```



LoginServlet.java

```
/**
 * @(#)LoginServlet.java
 * @version 1.8
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (SDK2.0)
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 */

import java.io.*;
import java.util.*;
import java.text.*;
import java.lang.Integer;

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

import Runde;

/**
 * Adds new user to database if 'NEB Navn' is available. Otherwise this
 * Servlet will return a 'sorry' response.
 */
public class LoginServlet extends HttpServlet {

    /**
     * Kunde-object used when trying to find user in database and
     * when adding new user to it.
     */
    private Kunde m_kunde;

    private String fAlias, fPassword, fBirthday, fBirthmonth, fBirthyear;
    private String fPostalcode, fPhone, fGender, fEmail, fCardHolder;
    private String fCardNumber, fCardMonth, fCardYear;

    /**
     * Handles a <code>GET</code> request from client
     *
     * @param <code>req</code> Request from client
     * @param <code>res</code> Response to client
     */
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
        ServletException, IOException {

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Qnet Nyt Login</TITLE>");
        out.println("<STYLE TYPE='text/css'>\" TITLE='\"Input style='>");
        out.println("<INPUT { font-size : 10pt;");
        out.println("          font-weight: bold;");
        out.println("          font-family : Verdana, Tahoma, Geneva, Arial;}");
        out.println("</STYLE>");
        out.println("</HEAD>");

        out.println("<BODY BGCOLOR=#000000 TEXT=#FFFFFF LINK=#FFFFFF "
            + "VLINK=#FFFFFF ALINK=#FFFFFF>");

        try {

            // Read all user informations from parameters

            fAlias = req.getParameter("Alias");
            fPassword = req.getParameter("password");
            fBirthday = req.getParameter("birthday");
            fBirthmonth = req.getParameter("birthmonth");
            fBirthyear = req.getParameter("birthyear");
            fPostalcode = req.getParameter("postalcode");
            fPhone = req.getParameter("phone");
            fGender = req.getParameter("gender");
            fEmail = req.getParameter("email");
            fCardHolder = req.getParameter("card_name");
            fCardNumber = req.getParameter("card_number");
            fCardMonth = req.getParameter("card_month");
            fCardYear = req.getParameter("card_year");

        }
        catch (Exception e) {
            System.out.println("Error reading some of the user info!");
        }
    }
}
```



```

/**
 * Create a Kunde object so we'll be able to check if user
 * already exist in database.
 */
m_kunde = new Kunde();

if (m_kunde.find(fAlias)) {

    /**
     * Alias (NEB Navn) was found in database so we will have
     * to tell client to use another alias.
     */

    out.println("<CENTER>");
    out.println("<FORM>");
    out.println("<TABLE BORDER=0 CELLSPACING=2 WIDTH=277>");
    out.println("<TR>");
    out.println("    <TD COLSPAN=3 HEIGHT=27>");
    out.println("        <FONT COLOR=#6699CC SIZE=4 "
            + "FACE=\"Verdana, Geneva, Helvetica, Arial\">");
    out.println("        <B>NEB NAVN (TAGET</B>");
    out.println("        </FONT>");
    out.println("    </TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("    <TD ALIGN=TOP ALIGN=JUSTIFY COLSPAN=2>"
            + "<P ALIGN=JUSTIFY>");
    out.println("        <FONT SIZE=2 FACE=\"Verdana, Tahoma, Geneva, Arial\">");
    out.println("        Dit valgte NEB Navn, \"\" + fAlias + "\", er allerede "
            + "optaget af en anden bruger. Gearing; venligst tilbage og "
            + "%saelig;lg et andet NEB Navn.");
    out.println("    </FONT>");
    out.println("    </TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("    <TD COLSPAN=3><CENTER>");
    out.println("        <HR SIZE=1 NOSHADE WIDTH=85% COLOR=#6699CC></CENTER>");
    out.println("    </TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("    <TD>");
    out.println("        <CENTER>");
    out.println("        <TABLE>");
    out.println("            <TR>");
    out.println("                <TD>");
    out.println("                    <INPUT LANGUAGE=\"JavaScript\" TYPE=\"button\" "
                            + "\"VALUE=\"&#171; Tilbage\" OnClick=\"history.back()\" \" "
                            + "\"NAME=\"btnBack\">");
    out.println("                </TD>");
    out.println("            </TR>");
    out.println("            <TR>");
    out.println("                <INPUT LANGUAGE=\"JavaScript\" TYPE=\"button\" "
                            + "\"VALUE=\" Afbryd \" OnClick=\"parent.close()\">");
    out.println("            </TR>");
    out.println("        </TABLE>");
    out.println("        <CENTER>");
    out.println("    </TD>");
    out.println("</TR>");
    out.println("    <TD COLSPAN=3><CENTER>");
    out.println("        <HR SIZE=1 NOSHADE WIDTH=85% COLOR=#6699CC></CENTER>");
    out.println("    </TD>");
    out.println("</TR>");
    out.println("</TABLE>");
    out.println("</FORM>");
    out.println("</CENTER>");

} else { // --> If (m_kunde.find(fAlias))

    /**
     * Add new user to database. All informations have already
     * been read so you just have to send these informations
     * to a 'Kunde' method which will write them to database.
     */

    m_kunde.setAll(fAlias, fPassword, fBirthday, fBirthmonth, fBirthyear,
        fPostalcode, fPhone, fGender, fEmail, fCardHolder, fCardNumber,
        fCardMonth, fCardYear);

    /**
     * Write info to database and check result.
     */
    boolean ok = m_kunde.addToDatabase();

    // Generate HTML response
    out.println("<CENTER>");
    out.println("<TABLE BORDER=0 CELLSPACING=0 WIDTH=263>");
    out.println("<TR>");
    out.println("    <TD COLSPAN=3 HEIGHT=27>");
    out.println("        <FONT COLOR=#6699CC SIZE=4 "
            + "FACE=\"Verdana, Geneva, Arial\">");
    out.println("        <B>TAK...</B></FONT>");
    out.println("    </TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("    <TD COLSPAN=3 ALIGN=TOP>");
    out.println("        <FONT SIZE=1 FACE=\"Verdana, Geneva, Arial\">");

```

appendiks G – side 174 af 95



LogoutServlet.java

```

/**
 * @(#)LogoutServlet.java
 * @version 1.3
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (SDK2.0)
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 *
 */

import java.io.*;

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Logs out of system by reinitialize cookie to its default value.
 */
public class LogoutServlet extends HttpServlet {

    /**
     * Handles a GET request from a client by resetting cookie and
     * generating a HTML document with 'you have logged out' info.
     */
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
        ServletException, IOException {

        /**
         * Initialize cookie with default value and add it to response.
         * Since we can't delete a cookie we have to delete its contents.
         */
        Cookie c = new Cookie(CookieCheck.LOGIN, CookieCheck.DEFAULT_VALUE);
        res.addCookie(c);

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<HTML>");
        out.println("<HEAD>");
        out.println(" <TITLE>Bogser er logget ud</TITLE>");
        out.println(" <META HTTP-EQUIV='refresh' CONTENT='0'; URL='\" +
            "http://\" + req.getHeader("Host") + "/main.html;\">");
        out.println("</HEAD>");

        out.println("<BODY BGCOLOR=#333333 TEXT=#FFFFFF LINK=#CC9933 \"
            + \"VLINK=#CC9933 ALINK=#CC9933 TOPMARGIN=0 LEFTMARGIN=0 \"
            + \"MARGINWIDTH=0 MARGINHEIGHT=0>");

        out.println("<IMG SRC='\" + graphics/leftcurve.gif\"' WIDTH=32 HEIGHT=25 \"
            + \"BORDER=0 ALT='\" + Left Curve\"'>");

        /**
         * 'Update' bottom frame by sending -1 as kundeID to our
         * GreetUserServlet.
         */
        out.println(" <SCRIPT LANGUAGE='\" + JavaScript\"'>");
        out.println(" <!-->");
        out.println(" parent.BottomFrame.location = \"\"
            + "/servlet/greetuser?kundeID=-1\"");
        out.println(" // -->");
        out.println("</SCRIPT>");

        // End HTML document
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();

    } // -> public void doGet(HttpServletRequest req, HttpServletResponse res)
}

```



MovieActorInfoServlet.java

```

/**
 * @(#)MovieActorInfoServlet.java
 * @version 1.2
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (SDK2.0)
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 */

import java.io.*;
import java.sql.*;

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * The <code>MovieActorInfoServlet</code> Servlet formats informations
 * about a specified actor and/or character he/she plays.
 */
public class MovieActorInfoServlet extends HttpServlet {

    /**
     * Private variables for in-parameters.
     */
    private int m_actorID;
    private int m_partID;

    /**
     * Private connection to database
     */
    private Connection con;

    /**
     * Initialize servlet by first calling parent configuration, then
     * finding JDBC/ODBC Bridge driver and last make the connection to
     * ODBC database (MS Access).
     */
    public void init(ServletConfig config) throws ServletException {

        // Perform standard configuration for Servlet
        super.init(config);

        try {

            // Load the jdbc-odbc bridge driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            // Attempt to connect to a driver. Each one
            // of the registered drivers will be loaded until
            // one is found that can process this URL
            con = DriverManager.getConnection("jdbc:odbc:NFB",
                "svend", "jensen");

        }
        catch (Exception e) {
            throw new UnavailableException(this,
                "Unable to open a connection to the Access DB");
        }

    } // --> public void init(ServletConfig config) throws ServletException

    public void destroy() {

        try {

            // Close the connection to the database
            con.close();

        }
        catch (SQLException sqle) {

            System.out.println("Unable to close Connection");

            while (sqle != null) {
                System.out.println();
                System.out.println("SQLState: " + sqle.getSQLState());
                System.out.println("Message: " + sqle.getMessage());
                System.out.println("Vendor: " + sqle.getErrorCode());
                sqle = sqle.getNextException();
            }

        } // --> catch (SQLException sqle)
    }
}

```




```

} // -> public void destroy()

/**
 * Handles a GET request from client by reading actorID and
 * partID parameters from query string and then find all
 * associated info i.e. movies from database.
 */
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

    /**
     * Get parameter from query. ID of actor and part are needed.
     */
    try {
        m_actorID = Integer.parseInt(req.getParameter("actorID"));
        m_partID = Integer.parseInt(req.getParameter("partID"));
    }
    catch (NumberFormatException nfe) {

        /**
         * We were unable to convert parameters to integers. This
         * is fatal, so we will have to make sure we don't try to
         * read these values.
         */

        m_actorID = -1;
        m_partID = -1;
    }

    /**
     * Make sure you have all needed information (actorID and
     * partID) before continuing.
     */
    if (m_actorID == -1) {

        /**
         * Inform user of error!
         */
        System.out.println("Desværre, der mangler nogle oplysninger!");
    }
    else { // -> if (m_actorID == -1)

        /**
         * Sounds good - all informations were retrieved successfully
         * so we will have to query database with the IDs.
         */

        /**
         * Make SQL query using retrieved parameters. We will have to
         * generated four different SQL queries:
         * + actorID and partID
         * + actorID and no partID
         * + no actorID and partID
         * + no actorID and no partID
         *
         * a ID of '0' is used when no ID was specified.
         */
        String mSQL = "";

        if ((m_actorID != 0) && (m_partID != 0))
            mSQL = "SELECT film.filmID, film.filmtitel, navneliste.navn, rolle.rolle "
                + "FROM ((film INNER JOIN medvirkende ON film.filmID = medvirkende.filmID) INNER JOIN rolle ON medvirkende.rolleID = rolle.rolleID) INNER JOIN navneliste ON "
                + "medvirkende.navnID = navneliste.navnID "
                + "WHERE navneliste.navnID = " + m_actorID + " AND rolle.rolleID = " + m_partID + " "
                + "ORDER BY film.filmtitel ASC";
        else if ((m_actorID != 0) && (m_partID == 0))
            mSQL = "SELECT film.filmID, film.filmtitel, navneliste.navn, rolle.rolle "
                + "FROM ((film INNER JOIN medvirkende ON film.filmID = medvirkende.filmID) INNER JOIN rolle ON medvirkende.rolleID = rolle.rolleID) INNER JOIN navneliste ON "
                + "medvirkende.navnID = navneliste.navnID "
                + "WHERE navneliste.navnID = " + m_actorID + " "
                + "ORDER BY film.filmtitel, rolle.rolle ASC";
        else if ((m_actorID == 0) && (m_partID != 0))
            mSQL = "SELECT film.filmID, film.filmtitel, navneliste.navn, rolle.rolle "
                + "FROM ((film INNER JOIN medvirkende ON film.filmID = medvirkende.filmID) INNER JOIN rolle ON medvirkende.rolleID = rolle.rolleID) INNER JOIN navneliste ON "
                + "medvirkende.navnID = navneliste.navnID "
                + "WHERE rolle.rolleID = " + m_partID + " "
                + "ORDER BY film.filmtitel ASC";
        else if ((m_actorID == 0) && (m_partID == 0))
            mSQL = "SELECT film.filmID, film.filmtitel, navneliste.navn, rolle.rolle "
                + "FROM ((film INNER JOIN medvirkende ON film.filmID = medvirkende.filmID) INNER JOIN rolle ON medvirkende.rolleID = rolle.rolleID) INNER JOIN navneliste ON "
                + "medvirkende.navnID = navneliste.navnID "
                + "ORDER BY film.filmtitel, rolle.rolle ASC";

        /**
         * Tell browser we're going to return some HTML code.
         */
        res.setContentType("text/html");

        /**
         * Get writer so we will be able to give client some
         * information back.
         */
        PrintWriter out = res.getWriter();

        out.println("<HTML>");
    }
}

```



```

out.println("<HEAD>");
out.println("<TITLE>Spilletidspunkter</TITLE>");
out.println("<STYLE TYPE='text/css' TITLE='Input style'>");
out.println("  INPUT { font-size : 9pt;");
out.println("        font-weight: bold;");
out.println("        font-family : Tahoma, Geneva, Arial;});");
out.println("  SELECT { font-size : 9pt;");
out.println("        font-family : Tahoma, Geneva, Arial;});");
out.println("</STYLE>");
out.println("</HEAD>");

out.println("<BODY BGCOLOR=#333333 TEXT=#FFFFFF LINK=#CC9933 "
+ "VLINK=#CC9933 ALINK=#CC9933 TOPMARGIN=0 LEFTMARGIN=0 "
+ "MARGINWIDTH=0 MARGINHEIGHT=0>");

out.print("<IMG SRC='\"/graphics/skuespiller_info.gif\" "
+ "WIDTH=474 HEIGHT=59>");

// Initialize HTML code (starts creating two TABLEs)
out.println("<!-- Full Table begin -->");
out.println("<TABLE BORDER=0 CELLPADDING=2 CELLSPACING=0 WIDTH=100%>");
out.println("<TR>");
out.println("<TD WIDTH=80% COLSPAN=2 ALIGN=CENTER VALIGN=TOP>");

/**
 * Generate two combos with actors and parts
 */
out.println("<!-- Combos Table begin -->");
out.println("<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0 WIDTH=98%>");
out.println("<TR VALIGN=MIDDLE>");
out.println("<TD BGCOLOR=#999966 VALIGN=TOP>"
+ "<IMG SRC='\"/graphics/02left.gif\" WIDTH=15 HEIGHT=15>"
+ "</TD>");
out.println("<TD BGCOLOR=#999966 ALIGN=CENTER>");

out.println("<!-- ActorCombo, PartCombo and Find begin -->");
out.println("<TABLE BORDER=0 CELLPADDING=2 CELLSPACING=0>");
out.println("<TR>");
out.println("<TD><FORM ACTION='\"/servlet/movieactorinfo\" "
+ "METHOD='GET'></TD>");

/**
 * Get actors
 */
out.println("<TD><FONT FACE=Arial COLOR=#FFFFFF>"
+ "Skuespiller</FONT>&nbsp;&nbsp;&nbsp;&");

insertActors(out);

out.println("</TD>");

/**
 * Get parts
 */
out.println("<TD>&nbsp;&nbsp;&nbsp;& <FONT FACE='Arial' COLOR=#FFFFFF>"
+ "Rolle</FONT>&nbsp;&nbsp;&nbsp;&");

insertParts(out);

out.println("</TD>");

/**
 * Setup 'find' button
 */
out.println("<TD><INPUT SRC='\"/graphics/soeg.gif\" TYPE='IMAGE' "
+ "BORDER=0 NAME='submit' WIDTH=43 HEIGHT=17 "
+ "ALT='\"Skoslasty efter film\" "
+ "ALIGN=TEXTTOP></TD>");

// End HTML coding
out.println("<TD></FORM></TD>");
out.println("</TR>");
out.println("</TABLE>");
out.println("<!-- ActorCombo, PartCombo and Find end -->");

out.println("</TD>");

out.println("<TD BGCOLOR=#999966 ALIGN=RIGHT VALIGN=TOP HEIGHT=33>"
+ "<IMG SRC='\"/graphics/02right.gif\" WIDTH=15 HEIGHT=15>"
+ "</TD>");
out.println("</TR>");

out.println("<TR>");
out.println("<TD BGCOLOR=#CC9933 COLSPAN=3 ALIGN=CENTER>"
+ "<IMG SRC='\"/graphics/pixel.gif\" WIDTH=1 HEIGHT=2></TD>");
out.println("</TR>");
out.println("</TABLE>");
out.println("<!-- Combos Table end -->");

/**
 * Create list of movies matching query
 */
out.println("<!-- Movies begin -->");
out.println("<TABLE BORDER=0 CELLPADDING=2 WIDTH=98%>");
out.println("<TR>");

```



```

out.println(" <ID bgcolor=#666666 align=center>"
+ "<font face='Verdana, Geneva, Arial' size=2>"
+ "<b>Film Titel</b></font></td>");

out.println(" <ID bgcolor=#666666 align=center>"
+ "<font face='Verdana, Geneva, Arial' size=2>"
+ "<b>Skuespiller</b></font></td>");

out.println(" <ID bgcolor=#666666 align=center>"
+ "<font face='Verdana, Geneva, Arial' size=2>"
+ "<b>Rolle</b></font></td>");

out.println("</tr>");

try {

    /**
     * Create a Statement object so we can submit SQL
     * statements to the driver.
     */
    Statement stmt = con.createStatement();

    int row = 0;

    /**
     * Submit a query, creating a ResultSet object
     */
    ResultSet rs = stmt.executeQuery(mSQL);

    /**
     * Loop through all found movies, creating a <tr> (table
     * row) for each occurrence.
     */
    while (rs.next()) {

        int filmID = rs.getInt(1);
        String filmtitel = rs.getString(2);
        String navn = rs.getString(3);
        String rolle = rs.getString(4);

        /**
         * Paint every second row with a brighter color.
         */
        out.println("<tr" + (row % 2 == 0 ? ">" : " bgcolor=#383838>"));

        out.print("<td valign=top>"
+ "<font face='Verdana, Geneva, Arial' size=2>");
        out.print("<a href='\"/servlet/movieinfo?filmID=" + filmID + "\">"
+ filmtitel + "</a>");
        out.println("</font></td>");

        out.print("<td valign=top align=center>"
+ "<font face='Verdana, Geneva, Arial' size=2>");
        out.print(navn);
        out.println("</font></td>");

        out.print("<td valign=top align=center>"
+ "<font face='Verdana, Geneva, Arial' size=2>");
        out.print(rolle);
        out.println("</font></td>");

        out.println("</tr>");

        row++;

    } // -> while (rs.next())

    /**
     * Close the result set and statement
     */
    rs.close();
    stmt.close();

    /**
     * You have to make special action if no movies were found
     * with specified criteria.
     */
    if (row == 0) {
        out.println("<tr>");
        out.print("<td colspan=3 align=justify>>"
+ "<font face='Verdana, Geneva, Arial' size=2>");
        out.println("Denne skuespiller har ikke medvirket i en film der "
+ "passer p acring; den valgte rolle.</font>");
        out.println("</td>");
        out.println("</tr>");
    } // -> if (row == 0)

    // Insert table end tag
    out.println("</table>");
    out.println("<!-- Movie Shows end -->");

}

catch (SQLException ex) {

    // A SQLException was generated. Catch it and
    // display the error information. Note that there
    // could be multiple error objects chained

```



```
// together

System.out.println ("\n*** SQLException caught ***\n");

while (ex != null) {
    System.out.println("SQLState: " + ex.getSQLState ());
    System.out.println("Message: " + ex.getMessage ());
    System.out.println("Vendor: " + ex.getErrorCode ());
    ex = ex.getNextException ();
    System.out.println ("");
}
} // -> catch (SQLException ex)

out.println("</ID>");
out.println("</TABLE>");
out.println("<!-- Full Table end -->");

out.println("</BODY>");
out.println("</HTML>");

} // -> if (m_actorID == -1) else

} // -> public void doGet (HttpServletRequest req, HttpServletResponse res)

/**
 * Inserts all actors into a 'SELECT' form item.
 */
public void insertActors(PrintWriter out) {

    final String mSQL = "SELECT navnID, navn "
        + "FROM navneliste "
        + "ORDER BY navn ASC";

    out.println(" <SELECT NAME=\"actorID\" SIZE=1>");
    out.println(" <OPTION VALUE=\"0\">Alle skuespillere</OPTION>");

    try {

        Statement stmt = con.createStatement ();
        ResultSet rs = stmt.executeQuery (mSQL);

        int f_navnID;
        String f_navn;

        /**
         * Loop through all found actornames
         */
        while (rs.next ()) {

            f_navnID = rs.getInt ("navnID");
            f_navn = rs.getString ("navn");

            out.println(" <OPTION" + (f_navnID == m_actorID ? " SELECTED " : " ")
                + "VALUE=\"" + f_navnID + "\">" + f_navn + "</OPTION>");

        } // -> while (rs.next ())

        rs.close ();
        stmt.close ();

    }
    catch (SQLException sqle) {

        /**
         * Unable to get actors from database
         */

        System.out.println ("Error getting actors from database");

    } // -> catch (SQLException sqle)

    out.println(" </SELECT>");

} // -> public void insertActors(PrintWriter out)

/**
 * Inserts all parts into a 'SELECT' form item.
 */
public void insertParts(PrintWriter out) {

    final String mSQL = "SELECT rolleID, rolle "
        + "FROM rolle "
        + "ORDER BY rolle ASC";

    out.println(" <SELECT NAME=\"partID\" SIZE=1>");
    out.println(" <OPTION VALUE=0>Alle roller</OPTION>");

    try {

        Statement stmt = con.createStatement ();
        ResultSet rs = stmt.executeQuery (mSQL);

        int f_rolleID;
        String f_rolle;
```



```
/**
 * Loop through all found parts
 */
while (rs.next()) {

    f_rolleID = rs.getInt("rolleID");
    f_rolle = rs.getString("rolle");

    out.println(" <OPTION" + (f_rolleID == m_partID ? " SELECTED " : " ")
        + "VALUE=\"" + f_rolleID + "\">" + f_rolle + "</OPTION>");

} // --> while (rs.next())

rs.close();
stmt.close();

}
catch (SQLException sqle) {

    /**
     * Unable to get parts from database
     */

    System.out.println("Error getting parts from database");

} // --> catch (SQLException sqle)

out.println(" </SELECT>");

} // --> public void insertParts(PrintWriter out)

}
```



MovieInfoServlet.java

```
/**
 * @(#)MovieInfoServlet.java
 * @version 1.2
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (SDK2.0)
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 */

import java.io.*;
import java.sql.*;

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * The <code>MovieInfoServlet</code> Servlet formats extended
 * information about a movie, e.g. description, playlength, category,
 * productionyear and full list of actors.
 */
public class MovieInfoServlet extends HttpServlet {

    /**
     * Private variable for in-parameter
     */
    private int m_filmID;

    /**
     * Private connection to database
     */
    private Connection con;

    /**
     * Handles a GET request from client by reading filmID parameter
     * and then reading all movie informations from database,
     * including all actors associated with it.
     */
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        /**
         * Get parameter from query. Only ID of movie is needed.
         */
        m_filmID = Integer.parseInt(req.getParameter("filmID"));

        /**
         * Make SQL query using retrieved parameter.
         */

        final String mSQL = "SELECT film.filmtitel, film.aargang, film.spilletid, film.beskrivelse, film.billedd, kategori.kategori "
            + "FROM film INNER JOIN kategori ON film.kategoriID = kategori.kategoriID "
            + "WHERE ((film.filmID)= " + m_filmID + ")";

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Spilletidspunkter</TITLE>");
        out.println("<STYLE TYPE='text/css' TITLE='Input style'>");
        out.println("  INPUT { font-size : 9pt;");
        out.println("         font-weight: bold;");
        out.println("         font-family : Tahoma, Geneva, Arial;}");
        out.println("  SELECT { font-size : 9pt;");
        out.println("         font-family : Tahoma, Geneva, Arial;}");
        out.println("</STYLE>");
        out.println("</HEAD>");

        out.println("<BODY BGCOLOR=#333333 TEXT=#FFFFFF LINK=#C99933 "
            + "VLINK=#C99933 ALINK=#C99933 TOPMARGIN=0 LEFTMARGIN=0 "
            + "MARGINWIDTH=0 MARGINHEIGHT=0>");

        out.print("<IMG SRC='\"/graphics/film_info.gif\" "
            + "WIDTH=474 HEIGHT=58>");

        try {
            con = DriverManager.getConnection("jdbc:odbc:NFB",
                "svend", "jensen");

            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(mSQL);

            // Only one row will be found!
```



```

if (rs.next()) {

    /**
     * First of all make sure to get all values from resultset.
     * It's important you only read every column once.
     */
    String m_filmtitel = rs.getString(1);
    String m_aargang = rs.getString(2);
    int m_spilletid = rs.getInt(3);
    String m_beskrivelse = rs.getString(4);
    String m_billede = rs.getString(5);
    String m_kategori = rs.getString(6);

    out.print("<CENTER>");
    out.println("<TABLE WIDTH=98% BORDER=0 CELLSPACING=0 CELLPADDING=0>");
    out.println("<FORM ACTION=\"/servlet/movieinfo/>");
    out.println("<TR>");
    out.println(" <TD BGCOLOR=#999966 VALIGN=TOP>"
        + "<IMG SRC=\"/graphics/02left.gif\" WIDTH=15 HEIGHT=15 "
        + "BORDER=0 ALT=\"Left Curve\"></TD>");
    out.println("");

    // Insert movie title
    out.println(" <TD BGCOLOR=#999966 WIDTH=100% NOWRAP>"
        + "<FONT FACE=\"Verdana, Geneva, Arial, Sans-Serif\" "
        + "SIZE=5 COLOR=#000000><B> " + m_filmtitel + "</B></FONT>"
        + "</TD>");

    // Insert movie selector
    out.println(" <TD BGCOLOR=#999966 ALIGN=RIGHT VALIGN=MIDDLE>"
        + "<INPUT SRC=\"/graphics/soeg.gif\" TYPE=\"IMAGE\" "
        + "BORDER=0 NAME=\"submit\" WIDTH=43 HEIGHT=17 "
        + "ALT=\"Søslashty efter valgte film\" "
        + "ALIGN=TEXTTOP><br> " + dbMovieSelector() + "</TD>");

    out.println(" <TD BGCOLOR=#999966 ALIGN=RIGHT VALIGN=TOP HEIGHT=33>"
        + "<IMG SRC=\"/graphics/02right.gif\" WIDTH=16 HEIGHT=15 "
        + "BORDER=0 ALT=\"Right Curve\"></TD>");
    out.println("</TR>");
    out.println("");
    out.println("<TR>");
    out.println(" <!-- Orange line -->");
    out.println(" <TD BGCOLOR=#CC9933 COLSPAN=4 ALIGN=CENTER>"
        + "<IMG SRC=\"/graphics/pixel.gif\" WIDTH=1 HEIGHT=2 "
        + "BORDER=0 ALT=\"\"></TD>");
    out.println("</TR>");
    out.println("");
    out.println("<TR VALIGN=TOP>");
    out.println(" <TD WIDTH=15><IMG SRC=\"/graphics/pixel.gif\" WIDTH=15 "
        + "HEIGHT=100 BORDER=0 ALT=\"\"></TD>");
    out.println("");
    out.println("<TR VALIGN=TOP>");
    out.println("");
    out.println(" <!-- 'Category', 'Picture' & 'Description' table -->");
    out.println(" <TABLE WIDTH=100% CELLSPACING=0 CELLPADDING=0 "
        + "BORDER=0>");

    // Insert category
    out.println(" <TR BGCOLOR=#666666>");
    out.println(" <TD HEIGHT=18 ALIGN=LEFT>"
        + "<FONT FACE=\"Verdana, Geneva, Arial, Sans-Serif\" "
        + "SIZE=1><br><B> " + m_kategori + "</B></FONT></TD>");
    out.println("</TR>");

    // Insert picture and description
    out.println(" <TR>");
    out.println(" <TD ALIGN=JUSTIFY>"
        + "<IMG SRC=\"/graphics/\" + m_billede + "\" ALIGN=RIGHT "
        + "BORDER=0 ALT=\"Screenshot of \" + m_filmtitel + "\" "
        + "HSPACE=4 VSPACE=4>"
        + "<FONT FACE=\"Verdana, Geneva, Arial, Sans-Serif\">"
        + "<P ALIGN=JUSTIFY> " + m_beskrivelse + "</TD>");
    out.println("");
    out.println("</TR>");
    out.println("</TABLE>");
    out.println(" ");
    out.println("</TD>");
    out.println("");
    out.println("<TD VALIGN=TOP>");
    out.println("");
    out.println(" <!-- 'Medvirkende' and 'ProductionYear' & "
        + "'PlayLength' tables -->");
    out.println(" <TABLE WIDTH=100% CELLSPACING=0 CELLPADDING=0 "
        + "BORDER=0>");
    out.println(" <TR BGCOLOR=#666666>");
    out.println(" <TD HEIGHT=18 ALIGN=RIGHT>"
        + "<FONT FACE=\"Verdana, Geneva, Arial\" SIZE=1>"
        + "<B>Medvirkende</B><br></FONT></TD>");
    out.println(" </TR>");
    out.println(" ");
    out.println(" <TR VALIGN=TOP>");
    out.println(" <TD>");

    // Insert actors
    dbActorsTable(out);

    out.println(" </TD>");

```

appendiks G – side 184 af 95



```

+ "WHERE film.filmID = " + m_filmID + " "
+ "ORDER BY navneliste.navn ASC";

out.println("<!-- 'Actors' table -->");
out.println("<TABLE WIDTH=100% CELSPACING=2 CELLPADDING=0 BORDER=0>");

int row = 0;

try {
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(fSQL);

    /**
     * Loop through resultset, inserting a row for each loop
     */
    while (rs.next()) {

        String f_rolle = rs.getString(1);
        String f_navn = rs.getString(2);

        out.println("<TR" + (row % 2 == 0 ? ">" : " BGCOLOR=#383838>"));
        out.println("<TD>"
            + "<FONT FACE='Verdana, Geneva, Arial'>"
            + f_navn + "<br></TD>");
        out.println("<TD><br>"
            + "<FONT FACE='Verdana, Geneva, Arial'>"
            + f_rolle + "</TD>");
        out.println("</TR>");

        row++;
    } // --> while (rs.next())

    rs.close();
    stmt.close();

}
catch(SQLException sqle) {
    /**
     * Problems! We were unable to get actors
     */
    out.println("<TR><TD COLSPAN=2>"
        + "Problemer med at finde medvirkende"
        + "</TD></TR>");

}

// Insert another row to get some space
out.println("<TR VALIGN=TOP>");
out.println("<TD COLSPAN=2><br></TD>");
out.println("</TR>");
out.println("</TABLE>");

} // --> public void dbActorsTable(PrintWriter out)

/**
 * Reads all movies from database and inserts them into
 * a <SELECT> form.
 */
public String dbMovieSelector() {

    final String fSQLQuery = "SELECT filmID, filmtitel " +
        "FROM film " +
        "ORDER BY filmtitel";

    String temp = "<SELECT NAME='\"filmID\"' SIZE=10>";

    int f_ID;
    String f_filmtitel;

    try {
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(fSQLQuery);

        // Display data, fetching until end of the result set
        while (rs.next()) {

            f_ID = rs.getInt("filmID");
            f_filmtitel = rs.getString("filmtitel");

            temp += " <OPTION" + (f_ID == m_filmID ? " SELECTED " : " ")
                + "VALUE='\"" + f_ID + "\"'>"
                + f_filmtitel + "</OPTION>";

        } // --> while (rs.next())

        rs.close();
        stmt.close();
    }
    catch(SQLException sqle) {
        /**
         * Problems! We were unable to get movies
         */
        System.out.println("Unable to movietitles.");
    }
}

```



```
}  
  
temp += "</SELECT>";  
  
return temp;  
} // -> public String doMovieSelector()  
}
```



MovieShowsServlet.java

```

/**
 * @(#)MovieShowsServlet.java
 * @version 1.5
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (JSDK2.0)
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Servlet tutorial:
 * + http://java.sun.com/docs/books/tutorial/servlets/TOC.html
 *
 * Notes:
 * <none>
 */

import java.io.*;
import java.util.*;
import java.text.*;
import java.lang.Integer;

// Import JDBC classes (JDK 1.1.6 and above)
import java.net.URL;
import java.sql.*;

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

//import Kategori;
//import Film;

/**
 * Retrieves informations about movieshows from an Access
 * Database running an ODBC/JDBC Bridge.
 */
public class MovieShowsServlet extends HttpServlet {

    /**
     * URL of the two servlets we call from this one:
     * movieshows -> this servlet
     * showhall -> servlet getting forestilling, setting up hall-
     * applet, etc.
     * movieinfo -> servlet showing extended information about
     * movie.
     */
    final String fMovieShowsServlet = "/servlet/movieshows";
    final String fShowHallServlet = "/servlet/showhall";
    final String fMovieInfoServlet = "/servlet/movieinfo";

    /**
     * This is the two parameters we need get in query string.
     */
    private int m_filmID, iDaysFromNowID;
    String query = "SELECT * FROM film";

    // Our connection to the Access 97 Database
    private Connection con;

    /**
     * Information used when connecting to database. These informations
     * comes from the 'servlet.properties' initialization file provided
     * when starting the servletrunner (or a Java-based Server)
     */
    String databaseURL, DBuser, DBpassword;

    /**
     * Initialize servlet by first calling parent configuration, then
     * finding JDBC/ODBC Bridge driver and last make the connection to
     * ODBC database (MS Access).
     */
    public void init(ServletConfig config) throws ServletException {

        // Perform standard configuration for Servlet
        super.init(config);

        try {

            // Load the jdbc-odbc bridge driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            // Show output on default out ($DEBUG)
            DriverManager.setLogStream(System.out);

            // Retrieve URL of database
            databaseURL = getInitParameter("databaseURL");

            // Retrieve user name

```



```

DBuser = getInitParameter("user");

// Retrieve password
DBpassword = getInitParameter("password");

// Attempt to connect to a driver. Each one
// of the registered drivers will be loaded until
// one is found that can process this URL
con = DriverManager.getConnection(databaseURL, DBuser, DBpassword);

}
catch(Exception e) {
    throw new UnavailableException(this,
        "Unable to open a connection to the Access DB");
}

} // --> public void init(ServletConfig config)

/**
 * Release connection to database before destroying servlet.
 */
public void destroy() {

    try {

        // Close the connection to the database
        con.close();

    }
    catch(SQLException sqle) {

        System.out.println("Unable to close Connection");

        while (sqle != null) {
            System.out.println();
            System.out.println("SQLState: " + sqle.getSQLState());
            System.out.println("Message: " + sqle.getMessage());
            System.out.println("Vendor: " + sqle.getErrorCode());
            sqle = sqle.getNextException();
        }

    }

} // --> public void destroy()

/**
 * Handles a 'GET' request by showing all shows on using
 * specified parameters.
 * @param <D>req/D> Request from client
 * @param <D>res/D> Response to client
 */
public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {

    /**
     * Retrieve servlet parameters from querystring.
     */
    m_filmID = Integer.parseInt(req.getParameter("filmID"));
    iDaysFromNowID = Integer.parseInt(req.getParameter("DaysFromNow"));

    /**
     * Tell browser we're going to return some HTML code.
     */
    res.setContentType("text/html");

    /**
     * Get writer so we will be able to give client some
     * information back.
     */
    PrintWriter out = res.getWriter();

    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Spilletidspunkter</TITLE>");
    out.println("<STYLE TYPE='text/css' TITLE='Input style'>");
    out.println("    INPUT { font-size : 9pt;");
    out.println("        font-weight: bold;");
    out.println("        font-family : Tahoma, Geneva, Arial;}");
    out.println("    SELECT { font-size : 9pt;");
    out.println("        font-family : Tahoma, Geneva, Arial;}");
    out.println("</STYLE>");
    out.println("</HEAD>");

    out.println("<BODY BGCOLOR=#333333 TEXT=#FFFFFF LINK=#CC9933 "
        + "<VLINK=#CC9933 ALINK=#CC9933 TOPMARGIN=0 LEFTMARGIN=0 "
        + "<MARGINWIDTH=0 MARGINHEIGHT=0>");

    out.print("<IMG SRC='\"/graphics/bestil_billeter.gif\" "
        + "<WIDTH=474 HEIGHT=59>");

    try {

        // Initialize HTML code (starts creating two TABLEs)
        out.println("<!-- Full Table begin -->");
        out.println("<TABLE BORDER=0 CELLPADDING=2 CELLSPACING=0 WIDTH=100%>");
        out.println("<TR>");
        out.println("<TD WIDTH=80% COLSPAN=2 ALIGN=CENTER VALIGN=TOP>");
    }
}

```



```

out.println("<FONT FACE=\"Verdana, Tahoma, Geneva, Arial\" SIZE=2>");

out.println("<!-- Selector Table begin -->");
out.println("<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0 WIDTH=98%>");
out.println("<TR VALIGN=MIDDLE>");
out.println("<TD BGCOLOR=#999966 VALIGN=TOP>"
    + "<IMG SRC=\"\"/graphics/02left.gif\" WIDTH=15 HEIGHT=15>"
    + "</TD>");

out.println(" <TD BGCOLOR=#999966 ALIGN=CENTER>");

/**
 * Make comboboxes (creates a <TABLE>) with movietitles and
 * next weeks dates.
 */
dbSelector(out);

out.println("</TD>");

out.println("<TD BGCOLOR=#999966 ALIGN=RIGHT VALIGN=TOP HEIGHT=33>"
    + "<IMG SRC=\"\"/graphics/02right.gif\" "
    + "<IMG SRC=\"\"/graphics/02right.gif\" WIDTH=15 HEIGHT=15 ALT=\"\"Right Curve\"></TD>");
out.println("</TR>");

out.println("<!-- Orange line row -->");
out.println("<TR>");
out.println(" <TD BGCOLOR=#CC9933 COLSPAN=3 ALIGN=CENTER>"
    + "<IMG SRC=\"\"/graphics/pixel.gif\" WIDTH=1 HEIGHT=1></TD>");
out.println("</TR>");
out.println("</TABLE>");
out.println("<!-- Selector Table end -->");

/**
 * By now you're able to get all shows (forestilling) for
 * selected movie.
 */
findMovieShows(out);

out.println("</TD>");
out.println("</TABLE>");
out.println("<!-- Full Table end -->");

}
catch (SQLException ex) {

    /**
     * A SQLException was generated. Catch it and display the
     * error information. Note that there could be multiple
     * error objects chained together.
     */
    System.out.println ("\\n*** SQLException caught ***\\n");

    while (ex != null) {
        System.out.println("SQLState: " + ex.getSQLState ());
        System.out.println("Message: " + ex.getMessage ());
        System.out.println("Vendor: " + ex.getErrorCode ());
        ex = ex.getNextException ();
        System.out.println ("");
    }
}
catch (java.lang.Exception ex) {

    // Got some other type of exception. Dump it.
    ex.printStackTrace ();

}

// End HTML document
out.println("</BODY>");
out.println("</HTML>");
out.close();

}

/**
 * Creates two comboboxes with <B>Movie Titles</B> and
 * <B>next 7 dates</B>.
 *
 * @param <D>out</D> A PrintWriter ready to receive a table
 * (i.e. writer isn't closed)
 * @return Complete <TABLE> (with new <FORM>) holding two
 * comboboxes
 */
public void dbSelector(PrintWriter out) throws SQLException {

    // Initialize HTML (no data read)
    out.println("<!-- Find, FilmCombo and DateCombo begin -->");
    out.println("<TABLE BORDER=0 CELLPADDING=2 CELLSPACING=0>");
    out.println("<TR>");
    out.println("<TD <FORM ACTION=\"\" + @MovieShowsServlet + \"\" "
        + "<METHOD=\"GET\"></TD>");

    // *** Film Titler ***
    out.println("<TD <FONT FACE=Arial COLOR=#FFFFFF>Film</FONT><br>");
    out.println("<SELECT NAME=\"filmID\" SIZE=1>");
    out.println("<OPTION VALUE=\"0\">Alle film</OPTION>");

```

appendiks G – side 190 af 95



```

* m_filmID will be zero (0) if no movie was selected (i.e. we
* have to get all movies).
*/
if (m_filmID == 0) {

    /**
    * SQL Query for finding shows of all movies.
    */
    SQLQuery = "SELECT forestillingID, forestilling.tid, film.filmtitel, kategori.kategori "
        + "FROM kategori INNER JOIN (film INNER JOIN forestilling ON film.filmID = forestilling.filmID) ON kategori.kategoriID = film.kategoriID "
        + "WHERE forestilling.dato = " + iDaysFromNowID + " "
        + "ORDER BY film.filmtitel, forestilling.tid ASC";

} else {

    /**
    * SQL Query for finding shows of a single movie.
    */
    SQLQuery = "SELECT forestillingID, forestilling.tid, film.filmtitel, kategori.kategori "
        + "FROM kategori INNER JOIN (film INNER JOIN forestilling ON film.filmID = forestilling.filmID) ON kategori.kategoriID = film.kategoriID "
        + "WHERE forestilling.dato = " + iDaysFromNowID + " AND film.filmID = " + m_filmID + " "
        + "ORDER BY film.filmtitel, forestilling.tid ASC";

}

// Create table header
out.println("<!-- Movie Shows begin -->");
out.println("<TABLE BORDER=0 CELLPADDING=2 WIDTH=98%>");
out.println("<TR>");

out.println(" <TD BGCOLOR=#666666 ALIGN=CENTER>"
    + "<FONT FACE='\"Verdana, Geneva, Arial'\" SIZE=2 COLOR=#FFFFFF>"
    + "<B>Film Titel</B></FONT></TD>");

out.println(" <TD BGCOLOR=#666666 ALIGN=CENTER>"
    + "<FONT FACE='\"Verdana, Geneva, Arial'\" SIZE=2 COLOR=#FFFFFF>"
    + "<B>Spilletidspunkt</B></FONT></TD>");

out.println(" <TD BGCOLOR=#666666 ALIGN=CENTER>"
    + "<FONT FACE='\"Verdana, Geneva, Arial'\" SIZE=2 COLOR=#FFFFFF>"
    + "<B>Film Kategori</B></FONT></TD>");

out.println("</TR>");

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(SQLQuery);

int row = 0;

boolean more = rs.next();
while (more) {

    /**
    * Get needed info from row.
    */
    int forestillingID = rs.getInt(1);
    Time tid = rs.getTime(2);
    String filmtitel = rs.getString(3);
    String kategori = rs.getString(4);

    /**
    * Shift between two different row colors.
    */
    out.println("<TR> " + (row % 2 == 0 ? ">" : ">" : " BGCOLOR=#383838>"));

    /**
    * Insert movietitel, time and category into tabledata.
    */
    out.println(" <TD VALIGN=TOP>"
        + "<FONT FACE='\"Verdana, Geneva, Arial'\" SIZE=3 COLOR=#FFFFFF>"
        + "<A HREF='\"\" + iMovieInfoServlet + "?filmID=" + m_filmID"
        + ">\">\" + filmtitel + "</A></FONT></TD>");

    out.println(" <TD VALIGN=TOP ALIGN=CENTER>"
        + "<FONT FACE='\"Verdana, Geneva, Arial'\" SIZE=3 COLOR=#FFFFFF>"
        + makeTimeOfShowLink(tid, forestillingID) + "</FONT></TD>");

    out.println(" <TD VALIGN=TOP ALIGN=CENTER>"
        + "<FONT FACE='\"Verdana, Geneva, Arial'\" SIZE=3 COLOR=#FFFFFF>"
        + kategori + "</FONT></TD>");

    out.println("</TR>");

    // Fetch the next result set row
    more = rs.next();

    row++;

}

/**
* You have to make special action if no shows were found
* with specified criteria.
*/
if (row == 0) {
    out.println("<TR>");

```



```

        out.println("<ID COLSPAN=3 ALIGN=JUSTIFY><P ALIGN=JUSTIFY>");
        out.println("Ingen spilletider passer på din søgning.</P>"
            + "Det kan enten være fordi ingen film passer til "
            + "søgningen faktisk spiller på den angivne dag, "
            + "eller fordi spilletidene på den pågældende dag "
            + "ikke kendes endnu. Hvis du har søgt på en dato "
            + "lidt ude i fremtiden kan du prøve igen senere.");
        out.println("</ID>");
        out.println("</TR>");
    }

    // Close access to resultset.
    rs.close();
    stmt.close();

    // Insert table end tag
    out.println("</TABLE>");
    out.println("<!-- Movie Shows end -->");

} // --> public void findMovieShows(PrintWriter out)

/**
 * Formats a <A HREF>-tag with informations about movieID and
 * showID.
 *
 * @param   <Dt</D>   A sql.Time object with time of current
 *                   show.
 * @param   <D>FID</D> ID of show.
 * @return  a href link which links to another servlet
 *         formatting output and loading correct applet.
 */
public String makeTimeOfShowLink(Time t, int fID) throws SQLException {

    String URL = "<A HREF=\"\" + fShowHallServlet + \"?\"";

    // Append extra '0' if minutes between 0..9 (single character)
    String minutes = "" + t.getMinutes();
    minutes = (minutes.length() == 1 ? "0" : "") + minutes;

    URL += "forestallingID=" + fID + "\">" + t.getHours() + ":" + minutes +
        "</A>";

    return URL;

} // --> public String makeTimeOfShowLink(Time t, int fID)

}

```




OrderSeatsServlet.java

```

/**
 * @(#)OrderSeatsServlet.java
 * @version 1.2
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (SDK2.0)
 *
 * Notes:
 * <none>
 *
 */

import java.io.*;
import java.sql.*;

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * The <code>OrderSeatsServlet</code> updates 'billetreservation'
 * in database with ordered tickets. Implements SingleThreadModel
 * since we're updating database.
 */
public class OrderSeatsServlet extends HttpServlet
    implements SingleThreadModel {

    /**
     * Private variables for in-parameters
     */
    private int m_kundeID;

    /**
     * Private connection to database
     */
    private Connection con;

    /**
     * Initialize servlet by first calling parent configuration, then
     * finding JDBC/ODBC Bridge driver and last make the connection to
     * ODBC database (MS Access).
     */
    public void init(ServletConfig config) throws ServletException {

        // Perform standard configuration for Servlet
        super.init(config);

        try {

            // Load the jdbc-odbc bridge driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            // Attempt to connect to a driver. Each one
            // of the registered drivers will be loaded until
            // one is found that can process this URL
            con = DriverManager.getConnection("jdbc:odbc:NFB",
                "svend", "jensen");

        }
        catch(Exception e) {
            throw new UnavailableException(this,
                "Unable to open a connection to the Access DB");
        }

    } // --> public void init(ServletConfig config) throws ServletException

    /**
     * Release connection to database before destroying servlet.
     */
    public void destroy() {

        try {

            // Close the connection to the database
            con.close();

        }
        catch(SQLException sqle) {

            System.out.println("Unable to close Connection");

            while (sqle != null) {
                System.out.println();
                System.out.println("SQLState: " + sqle.getSQLState());
                System.out.println("Message: " + sqle.getMessage());
                System.out.println("Vendor: " + sqle.getErrorCode());
                sqle = sqle.getNextException();
            }

        }

    }

}

```



```

    } // -> catch(SQLException sqle)

} // -> public void destroy()

/**
 * Handles a POST request by reading ID of customer and all
 * reservations he/she wants to pay.
 */
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

    /**
     * Get all parameters from query.
     */
    m_kundeID = Integer.parseInt(req.getParameter("kundeID"));

    /**
     * Multiple 'payfor' parameters are allowed, so we will have
     * to use an array of Strings.
     */
    String paying[] = req.getParameterValues("payfor");

    String mSQL = "UPDATE billetreservation SET billetreservation.betalst = True "
        + "WHERE (billetreservation.kundeID = " + m_kundeID + ") ";

    for(int i = 0; i < paying.length; i++) {
        mSQL += (i == 0 ? " AND " : " OR ");
        mSQL += "(billetreservation.forestillingID = " + paying[i] + ") ";
    }

    mSQL += ";";

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<HTML><HEAD>"
        + "<TITLE>Betalning af reservationer</TITLE>"
        + "</HEAD>");

    out.println("<BODY BGCOLOR=#333333 TEXT=#FFFFFF LINK=#CC9933 "
        + "VLINK=#CC9933 ALINK=#CC9933 TOPMARGIN=0 LEFTMARGIN=0 "
        + "MARGINWIDTH=0 MARGINHEIGHT=0>");

    out.println("<IMG SRC=\"/graphics/betal_billetter.gif\" "
        + "WIDTH=474 HEIGHT=59>");

    try {

        Statement stmt = con.createStatement();
        int rowsUpdated = stmt.executeUpdate(mSQL);

        out.println("<BR><BR>");

        if (rowsUpdated != 0) {

            out.println("&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&Tak for din betaling!");

        } else {

            out.println("&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&Det var desv&aelig;lign&euml; ikke muligt at "
                + "registr&euml;re din betaling.");

        }

        stmt.close();

    }

    catch (SQLException ex) {

        /**
         * Error occurred! Might have changed database or needs some
         * informations there wasn't provided.
         */

        System.err.println("SQL Exception - OrderSeatsServlet");
        out.println("&nbsp;&nbsp;&nbsp;&nbsp;&Der opstod et problem ved registrering af "
            + "Deres betaling. Venligst check 'betal billetter' for at "
            + "se ubetalte reservationer.");

    }

    out.close();

} // -> public void doPost(HttpServletRequest req, HttpServletResponse res)

}

```



ReservationsServlet.java

```

/**
 * @(#)ReservationsServlet.java
 * @version 1.6
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (SDK2.0)
 *
 * Notes:
 * <none>
 *
 */

import java.io.*;
import java.util.*;
import java.text.*;
import java.lang.Integer;
import java.awt.*;
import java.sql.*;

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

import BilletReservation;
import Film;
import Forestilling;
import Sal;

import CookieCheck;

/**
 * Finds all reservations of logged in user and generates HTML.
 */
public class ReservationsServlet extends HttpServlet {

    private Connection con;

    /**
     * Initialize servlet by first calling parent configuration, then
     * finding JDBC/ODBC Bridge driver and last make the connection to
     * ODBC database (MS Access).
     */
    public void init(ServletConfig config) throws ServletException {

        // Perform standard configuration for Servlet
        super.init(config);

        try {

            // Load the jdbc-odbc bridge driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            /**
             * Attempt to connect to a driver. Each one of the registered
             * drivers will be loaded until one is found that can process
             * this URL.
             */
            con = DriverManager.getConnection("jdbc:odbc:NEB", "sven", "jensen");

        }
        catch(Exception e) {
            throw new UnavailableException(this,
                "Unable to open a connection to the Access DB");
        }

    } // -> public void init(ServletConfig config)

    /**
     * Release connection to database before destroying servlet.
     */
    public void destroy() {

        try {

            // Close the connection to the database
            con.close();

        }
        catch(SQLException sqle) {

            System.out.println("Unable to close Connection");

            while (sqle != null) {
                System.out.println();
                System.out.println("SQLState: " + sqle.getSQLState());
                System.out.println("Message: " + sqle.getMessage());
                System.out.println("Vendor: " + sqle.getErrorCode());
            }

        }

    }

}

```



```

        sqle = sqle.getNextException();
    }

}

} // → public void destroy()

/**
 * Handles a GET request from Client by reading kundeID and
 * finding all his/hers reserved seats.
 */
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {

    // Get a cookie so we can get kundeID
    CookieCheck cookie = new CookieCheck(req);

    // Can't show reservations info if user hasn't logged in
    if (!cookie.isLoggedIn()) {

        /**
         * No user was logged into our system so we will have to
         * inform him/her of this by redirecting user to a predefined
         * 'no user logged in' HTML page.
         */

        String location = "/no_login.html";

        String scheme = req.getScheme();
        String host = req.getServerName();
        int port = req.getServerPort();

        String redirectString = scheme + "://" + host;
        if (port != 80) {
            redirectString += ":" + port;
        }

        redirectString += location;
        res.sendRedirect(redirectString);

    } else {

        /**
         * Customer was logged in, so we will be able to show all
         * his reservations by sending 'kundeID' to database.
         */

        // Get all customer information (e.g. Alias)
        Kunde kunde = new Kunde(cookie.getKundeID());

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<HTML><HEAD>"
            + "<TITLE>"
            + "Sælligdereserveringer for " + kunde.getAlias()
            + "</TITLE></HEAD>");

        out.println("<BODY BGCOLOR=#333333 TEXT=#FFFFFF LINK=#CC9933 "
            + "VLINK=#CC9933 ALINK=#CC9933 TOPMARGIN=0 LEFTMARGIN=0 "
            + "MARGINWIDTH=0 MARGINHEIGHT=0>");

        out.print("<IMG SRC=\"/graphics/afbestil_billetter.gif\" "
            + "WIDTH=474 HEIGHT=59>");

        out.println("<TABLE BORDER=0 CELLPADDING=2 CELLSPACING=0 WIDTH=100%>");
        out.println("<TR>");
        out.println(" <TD WIDTH=80% COLSPAN=2 ALIGN=CENTER VALIGN=TOP>");
        out.println(" <FONT FACE=\"Tahoma, Verdana, Geneva, Arial\" SIZE=2>");
        out.println(" <TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0 WIDTH=98%>");
        out.println(" <!-- Green area with username begin -->");
        out.println(" <TR VALIGN=MIDDLE>");
        out.println(" <TD BGCOLOR=#999966 VALIGN=TOP>");
        out.print("<IMG SRC=\"/graphics/02left.gif\" WIDTH=15 HEIGHT=15>");
        out.println("</TD>");
        out.println(" <TD ALIGN=CENTER BGCOLOR=#999966>");
        out.println(" <!-- Username area begin -->");
        out.println(" <TABLE BORDER=0 CELLPADDING=2 CELLSPACING=0>");
        out.println(" <TR><TD ALIGN=CENTER><B><FONT COLOR=#FFFFFF SIZE=3 "
            + "FACE=\"Verdana, Tahoma, Geneva, Arial\">"
            + "Reserverede sælligder for " + kunde.getAlias()
            + "</FONT></B></TD></TR>");
        out.println(" </TABLE>");
        out.println(" <!-- Username area end -->");
        out.println(" </TD>");
        out.println(" <TD BGCOLOR=#999966 ALIGN=RIGHT VALIGN=TOP HEIGHT=33>");
        out.print("<IMG SRC=\"/graphics/02right.gif\" WIDTH=15 HEIGHT=15>");
        out.println("</TD>");
        out.println(" </TR>");
        out.println(" <!-- Green area with username end -->");
        out.println(" <!-- Orange line begin -->");
        out.println(" <TR>");
        out.println(" <TD COLSPAN=3 ALIGN=CENTER BGCOLOR=#CC9933>");
        out.print("<IMG SRC=\"/graphics/pixel.gif\" WIDTH=1 HEIGHT=2>");
        out.println("</TD>");
        out.println(" </TR>");
        out.println(" <!-- Orange line end -->");
        out.println(" </TABLE>");
    }
}

```



```

out.println("<!-- Gray information area begin -->");
out.println("<TABLE BORDER=0 CELLPADDING=2 WIDTH=98%>");
out.println("<TR>");

// Title
out.println(" <TD ALIGN=CENTER BGCOLOR=#666666>");
out.println(" <FONT COLOR=#FFFFFF SIZE=2 " +
    "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println(" <B>Film Titel</B>");
out.println(" </FONT>");
out.println(" </TD>");

// Date Caption
out.println(" <TD ALIGN=CENTER BGCOLOR=#666666>");
out.println(" <FONT COLOR=#FFFFFF SIZE=2 " +
    "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println(" <B>Date</B>");
out.println(" </FONT>");
out.println(" </TD>");

// Time (hour:minutes) Caption
out.println(" <TD ALIGN=CENTER BGCOLOR=#666666>");
out.println(" <FONT COLOR=#FFFFFF SIZE=2 " +
    "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println(" <B>Tick</B>");
out.println(" </FONT>");
out.println(" </TD>");

// Row Caption
out.println(" <TD ALIGN=CENTER BGCOLOR=#666666>");
out.println(" <FONT COLOR=#FFFFFF SIZE=2 " +
    "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println(" <B>R&aelig;ke</B>");
out.println(" </FONT>");
out.println(" </TD>");

// Seat Caption
out.println(" <TD ALIGN=CENTER BGCOLOR=#666666>");
out.println(" <FONT COLOR=#FFFFFF SIZE=2 " +
    "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println(" <B>S&aelig;de</B>");
out.println(" </FONT>");
out.println(" </TD>");

// Paid Caption
out.println(" <TD ALIGN=CENTER BGCOLOR=#666666>");
out.println(" <FONT COLOR=#FFFFFF SIZE=2 " +
    "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println(" <B>Betalt</B>");
out.println(" </FONT>");
out.println(" </TD>");

// Cancel reservation Caption
out.println(" <TD ALIGN=CENTER BGCOLOR=#666666>");
out.println(" <FONT COLOR=#FFFFFF SIZE=2 " +
    "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println(" <B>Afbestilling</B>");
out.println(" </FONT>");
out.println(" </TD>");
out.println("</TR>");

out.println("<!-- Reservations begin -->");

/**
 * SQL query retrieving information we want to show user and
 * sorts by title, day, time, row and seat.
 */
final String mSQL = "SELECT billetreservation.raekke, billetreservation.saeds, billetreservation.kundeID, film.filmtitel, forestilling.forestillingID, forestilling.tid,
forestilling.dato, billetreservation.betalt "
    + "FROM film INNER JOIN (billetreservation INNER JOIN forestilling ON billetreservation.forestillingID = forestilling.forestillingID) ON film.filmID =
forestilling.filmID "
    + "WHERE (((billetreservation.kundeID)= " + cookie.getKundeID() + ")) "
    + "ORDER BY film.filmtitel, forestilling.dato, forestilling.tid, billetreservation.raekke, billetreservation.saeds ASC";

try {

    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(mSQL);

    /**
     * Tells us which row we're on when generating HTML. This
     * makes it possible to change every second rows color.
     */
    int row = 0;

    /**
     * Attributes we're going to fetch from resultset.
     */
    int m_raekkeID, m_saedsID, m_kundeID;
    String m_filmtitel;
    int m_forestillingID;
    java.sql.Time m_tick;
    int m_dato;
    boolean m_betalt;

    // Loop through all user reservations

```



```

boolean more = rs.next();
while (more) {

    /**
     * Get all values from current row.
     */
    m_raekkeID = rs.getInt(1);
    m_saedeID = rs.getInt(2);
    m_kundeID = rs.getInt(3);
    m_filmtitel = rs.getString(4);
    m_forestillingsID = rs.getInt(5);
    m_tid = rs.getTime(6);
    m_dato = rs.getTime(7);
    m_betalt = rs.getBoolean(8);

    out.println("<TR" + (row % 2 == 0 ? ">" : " bgcolor=#383838>"));

    // Print Movie Title
    out.println(" <TD VALIGN=TOP>");
    out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">");
    out.println(" " + m_filmtitel);
    out.println(" </FONT>");
    out.println(" </TD>");

    // Print Date
    out.println(" <TD ALIGN=CENTER VALIGN=TOP>");
    out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">");
    out.println(" " + formatDate(m_dato));
    out.println(" </FONT>");
    out.println(" </TD>");

    // Print Time
    out.println(" <TD ALIGN=CENTER VALIGN=TOP>");
    out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">");
    out.println(" " + formatTime(m_tid));
    out.println(" </FONT>");
    out.println(" </TD>");

    // Print row (raekke)
    out.println(" <TD ALIGN=CENTER VALIGN=TOP>");
    out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">");
    out.println(" " + m_raekkeID);
    out.println(" </FONT>");
    out.println(" </TD>");

    // Print column (saede)
    out.println(" <TD ALIGN=CENTER VALIGN=TOP>");
    out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">");
    out.println(" " + m_saedeID);
    out.println(" </FONT>");
    out.println(" </TD>");

    // Paid for seat (betalt)
    out.println(" <TD ALIGN=CENTER VALIGN=TOP>");
    out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">");
    out.println(" (m_betalt ? "Ja" : "Nej")");
    out.println(" </FONT>");
    out.println(" </TD>");

    // Cancel link (afbestil link)
    out.println(" <TD ALIGN=CENTER VALIGN=TOP>");
    out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">");
    out.println(" <A HREF=\"/servlet/cancelseat?"
        + "forestillingID=" + m_forestillingsID
        + "&kundeID=" + m_kundeID
        + "&raekkeID=" + m_raekkeID
        + "&saedeID=" + m_saedeID + "></FONT>");
    out.println(" afbestil billet</A></FONT>");
    out.println(" </TD>");

    out.println("</TR>");

    more = rs.next();

    row++;

} // -> while (more)

/**
 * Close ResultSet, Statement and Connection to database.
 */
rs.close();
stmt.close();

}
catch (SQLException ex) {

    /**
     * Error occurred! Might have changed database or needs some
     * informations there wasn't provided.
     */

    System.err.println("SQL Exception - ReservationsServlet");

}

out.println("<!-- Reservations end -->");

```



```

        out.println("<ID><ID></ID></ID>");
        out.println("</TABLE>");
        out.println("<!-- Gray information area end -->");
        out.println("</ID>");
        out.println("</TABLE>");
        out.println("<!-- final end -->");

        out.println("</BODY>");
        out.println("</HTML>");

        out.close();

    } // --> if (not logged in) else

} // --> public void doGet (HttpServletRequest req, HttpServletResponse res)

/**
 * Formats sql.Time object to a string using hours:minutes
 * format.
 *
 * @param    atTime    Time object with time of show
 * @return    String with Time in hours:minutes format
 */
public String formatTime (java.sql.Time atTime) {

    int t = atTime.getMinutes();
    return (atTime.getHours() + ":" + (t < 10 ? "0" : "") + t);

} // --> public String formatTime (java.sql.Time atTime)

/**
 * Formats relative DaysFromNow variable from database into a
 * absolute date by reading current date and appending
 * 'DaysFromNow' number.
 *
 * @param    atDate    Relative DaysFromNow variable ranging from
 *                    0 (today) to 7 (next week)
 * @return    Absolute date in format specified by system settings.
 */
public String formatDate (int atDate) {

    // Returns formatted date => currentDate + DaysFromNow (atDate)
    DateFormat fulldate = DateFormat.getDateInstance (DateFormat.FULL);
    Calendar cal = Calendar.getInstance();
    cal.add (Calendar.DATE, atDate);

    return fulldate.format (cal.getTime());

} // --> public String formatDate (int atDate)

}

```



ShowHallServlet.java

```

/**
 * @(#)ShowHallServlet.java
 * @version 1.6
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (SDK2.0)
 *
 * Notes:
 * For discussion of servlets, consider joining the JSRV-INTEREST mailing
 * list. For information on subscribing to this list, check our webpage at
 * http://jsrv.javasoft.com/
 *
 * Compiling:
 * javac -classpath \jdk2.0\lib\jck.jar -sourcepath \site\ <file.java>
 */

import java.io.*;
import java.util.*;
import java.text.*;
import java.lang.Integer;
import java.awt.*;

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

import BilletReservation; // getReservedSeats()
import Film;
import Forestilling;
import Sal;
import CookieCheck;

/**
 * The <code>ShowHallServlet</code> Servlet sets up a document with
 * extended information about specified movie and reads all reserved
 * seats in hall for use with our applet 'SalLayout'.
 */
public class ShowHallServlet extends HttpServlet {

    /**
     * ID of currently selected show. This ID is going to be
     * initialized with a parameter from servlets query string.
     */
    private int forestillingID = -1;

    private Film dFilm;
    private Forestilling dForestilling;
    private Sal dSal;

    /**
     * Handles a GET request from a Client by setting up a page with
     * extended movieinfo and an applet which allows the client to
     * reserve seats for specified show (forestillingID).
     */
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        /**
         * Check cookie to find out, if customer has logged into our
         * system. If that's the case, we will have to set up an
         * advanced HTML; otherwise we will redirect client to a
         * pregenerated 'you-have-to-be-logged-on' HTML page.
         */
        CookieCheck cookie = new CookieCheck(req);

        if (!cookie.isLoggedIn()) {

            /**
             * No user was logged into our system so we will have to
             * inform him/her of this by redirecting user to a predefined
             * 'no user logged in' HTML page.
             */

            String location = "/nq_login.html";

            String scheme = req.getScheme();
            String host = req.getServerName();
            int port = req.getServerPort();

            String redirectString = scheme + "://" + host;
            if (port != 80) {
                redirectString += ":" + port;
            }

            redirectString += location;
            res.sendRedirect(redirectString);
        }
    }
}

```




```

} else { // → if (!cookie.isLoggedIn())

/**
 * Get information about selected show (forestillingID) and
 * generate HTML code for user. This includes getting info
 * about hall, reserved seats, moviedescription, movielength,
 * productionyear, etc.
 */

/**
 * Retrieve ID of show from parameter string.
 */
forestillingID = Integer.parseInt(req.getParameter("forestillingID"));

/**
 * Retrieve information from database about show with the
 * forestillingID we just have read.
 */
dForestilling = new Forestilling(forestillingID);

/**
 * Get 'filmID' by reading Forestilling table using provided ID
 */
dFilm = new Film(dForestilling.getFilmID());
dSal = new Sal(dForestilling.getSalID());

res.setContentType("text/html");
PrintWriter out = res.getWriter();

out.println("<H1>");
out.println("<HEAD>");
out.println("<TITLE>S&aelig;dereservering til " + dFilm.getTitle()
    + "</TITLE>");
out.println("</HEAD>");

out.println("<BODY bgcolor=#333333 text=#ffffff link=#009933 "
    + "vlink=#009933 alink=#009933 topmargin=0 leftmargin=0 "
    + "marginwidth=0 marginheight=0>");

out.print("<img src=\"/graphics/bestil_billetter.gif\" "
    + "width=474 height=59>");

out.println("<center>");
out.println("<table width=98% border=0 cellpadding=0>");
out.println("<tr>");
out.print("<td bgcolor=#999966 valign=top>");
out.println("<img src=\"/graphics/02left.gif\" width=15 height=15 "
    + "border=0 alt=\"Left Curve\"></td>");

// Insert Title of movie
out.print("<td bgcolor=#999966 width=100% align=center nowrap>"
    + "<font face=\"Verdana, Tahoma, Geneva, Arial, Helvetica, "
    + "Sans-Serif\" size=5 color=\"#000000\">");
out.println("<b>" + dFilm.getTitle() + "</b></font></td>");

// Insert applet
out.println("<td rowspan=3 valign=top>");

out.println("<applet codebase=\"/\" code=\"SalLayout.class\" width=420 "
    + "height=315>");

/**
 * Transfer forestillingID to applet, which makes it possible
 * to write selected seats to database.
 */
out.println("<param name=\"forestillingID\" "
    + "value=\"" + forestillingID + "\">");

FileReader r = new FileReader(req.getRealPath("/") + dSal.getMap());
StreamTokenizer st = new StreamTokenizer(r);

try {

    int numSect;
    int x, y, columns, rows, sCol, sRow, Rcolor, Gcolor, Bcolor;
    numSect = 0;

    x = 90; y = 100; columns = 110; rows = 120; sCol = 130; sRow = 140;
    Rcolor = 0; Gcolor = 0; Bcolor = 0;

    int nt;
    while ( (nt = st.nextToken()) != StreamTokenizer.TT_EOF ) {

        x = (int)st.nval;

        st.nextToken();
        y = (int)st.nval;

        st.nextToken();
        columns = (int)st.nval;

        st.nextToken();
        rows = (int)st.nval;

        st.nextToken();
        sCol = (int)st.nval;

```



```

        st.nextToken();
        sRow = (int)st.nval;

        st.nextToken();
        Rcolor = (int)st.nval;

        st.nextToken();
        Gcolor = (int)st.nval;

        st.nextToken();
        Bcolor = (int)st.nval;

        out.println("<PARAM NAME=\"sect\" + numSect + \"\" VALUE=\""
            + x + "\",\" + y + "\",\" + columns + "\",\" + rows + "\",\" + sCbl + "\",\"
            + sRow + "\",\" + Rcolor + "\",\" + Gcolor + "\",\" + Bcolor + "\">");

        numSect++;
    }

}

catch (NoSuchElementException e) {
    System.out.println(e.getClass().getName());
}

catch (IOException ioe) {
    System.out.println("Fatal error when reading hall-mapping...");
}

}

/**
 * Create new BilletReservation so we're able to get all
 * reserved seats in hall.
 */
BilletReservation m_BilletReservation = new BilletReservation();

// Load all reserved seats
Vector vTemp = m_BilletReservation.getReservedSeats(forestillingID);

for (int i = 0; i < vTemp.size(); i++) {

    Point p = (Point)vTemp.elementAt(i);
    out.println("<PARAM NAME=\"reserv\" + i + \"\" VALUE=\"" + p.x
        + "\",\" + p.y + "\">");

}

out.println("</APPLED>");
out.println("</ID>");

out.println("<ID bgcolor=#999966 align=right valign=top height=33>"
    + "<IMG SRC=\"/graphics/02right.gif\" width=16 height=15 \"
    + \"ALT=\"Right Curve\"></ID>");
out.println("</TR>");

/**
 * Draw orange line.
 */
out.println("<TR>");
out.println("<ID bgcolor=#CC9933 colspan=3 align=center>"
    + "<IMG SRC=\"/graphics/pixel.gif\" width=1 height=1></ID>");
out.println("</TR>");

out.println("<TR valign=top>");
out.println("<ID><IMG SRC=\"/graphics/pixel.gif\" border=0 \"
    + \"width=15 height=315 alt=\"|\"></ID>");
out.println("<ID align=center>");
out.println("<TABLE width=100% cellpadding=0 border=0>");
out.println("<TR bgcolor=#666666>");

/**
 * Insert date of show by reading 'Forestilling' object.
 */
out.println("<ID height=18 align=left nowrap>"
    + "<FONT FACE=\"Verdana, Geneva, Arial\" size=1>&nbsp;"
    + dForestilling.getFormattedDate() + "</FONT></ID>");

/**
 * Insert time of show by reading 'Forestilling' object.
 */
out.println("<ID align=right nowrap>"
    + "<FONT FACE=\"Verdana, Geneva, Arial\" size=1>"
    + "Klokken " + dForestilling.getFormattedTime()
    + "&nbsp;</FONT></ID>");
out.println("</TR>");

out.println("<TR>");
out.println("<ID colspan=2 align=justify>");
out.println("<FONT FACE=\"Verdana, Geneva, Arial\" size=2>"
    + "<B>Reservering af sæder</B></FONT><BR><B>"
    + "<FONT FACE=\"Verdana, Geneva, Arial\">"
    + "<P align=justify>Vælg sæder ved at trykke på salen til "
    + "højre. Tryk endnu en gang for at fravalge sæder. Du "
    + "kan maksimalt vælge 7 sæder. Når du vil reservere, "
    + "tryk på 'reservér sæder' over salen.</FONT></ID>");
out.println("</TR>");

out.println("</TABLE>");
out.println("</ID>");
out.println("<ID><IMG SRC=\"/graphics/pixel.gif\" border=0 "
```



```
+ "WIDTH=15 HEIGHT=315 ALT=\\\"\\\"></ID>");
out.println("</TR>");

out.println("</TABLE>");
out.println("</CENTER>");

out.println("</BODY>");
out.println("</HTML>");

out.close();

} // if (!cookie.isLoggedIn()) else

} // -> public void doGet(HttpServletRequest req, HttpServletResponse res)

}
```



ShowOrdersServlet.java

```

/**
 * @(#)ShowOrdersServlet.java
 * @version 1.4
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (JSDK2.0)
 *
 * Notes:
 * <none>
 *
 */

import java.io.*;
import java.util.*;
import java.sql.*;
import java.awt.*;
import java.text.*;

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

import CookieCheck;
import Kunde;

/**
 * The <code>ShowOrdersServlet</code> finds all unpaid
 * reservations of currently logged in user and makes it possible
 * for user to select/deselect which reservations to paid.
 */
public class ShowOrdersServlet extends HttpServlet {

    /**
     * Private variables
     */
    private CookieCheck cookie;

    /**
     * Private connection to database
     */
    private Connection con;

    /**
     * Initialize servlet by first calling parent configuration, then
     * finding JDBC/ODBC Bridge driver and last make the connection to
     * ODBC database (MS Access).
     */
    public void init(ServletConfig config) throws ServletException {

        // Perform standard configuration for Servlet
        super.init(config);

        try {

            // Load the jdbc-odbc bridge driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            // Attempt to connect to a driver. Each one
            // of the registered drivers will be loaded until
            // one is found that can process this URL
            con = DriverManager.getConnection("jdbc:odbc:NEB",
                "svend", "jensen");

        }
        catch(Exception e) {
            throw new UnavailableException(this,
                "Unable to open a connection to the Access DB");
        }

    } // -> public void init(ServletConfig config) throws ServletException

    /**
     * Release connection to database before destroying servlet.
     */
    public void destroy() {

        try {

            // Close the connection to the database
            con.close();

        }
        catch(SQLException sqle) {

            System.out.println("Unable to close Connection");

            while (sqle != null) {

```



```

        System.out.println();
        System.out.println("SQLState: " + sqle.getSQLState());
        System.out.println("Message: " + sqle.getMessage());
        System.out.println("Vendor: " + sqle.getErrorCode());
        sqle = sqle.getNextException();
    }

    } // --> catch (SQLException sqle)

    } // --> public void destroy()

/**
 * Handles a GET request by showing all reservations of
 * currently logged in user.
 */
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

    // Get a cookie so we can get kundeID
    cookie = new CookieCheck(req);

    // Can't show reservations info if user hasn't logged in
    if (!cookie.isLoggedIn()) {

        /**
         * No user was logged into our system so we will have to
         * inform him/her of this by redirecting user to a predefined
         * 'no user logged in' HTML page.
         */

        String location = "/no_login.html";

        String scheme = req.getScheme();
        String host = req.getServerName();
        int port = req.getServerPort();

        String redirectString = scheme + "://" + host;
        if (port != 80) {
            redirectString += ":" + port;
        }

        redirectString += location;
        res.sendRedirect(redirectString);

    } else {

        Kunde kunde = new Kunde(cookie.getKundeID());

        /**
         * Customer has to have entered information about his/her
         * card (holder, number, expmonth and expyear).
         */

        if (!Kunde.isCardLoaded()) {

            /**
             * No card info was found so we will have to inform him/her
             * of this by redirecting user to a predefined 'no card
             * info' HTML page.
             */

            String location = "/no_card_info.html";

            String scheme = req.getScheme();
            String host = req.getServerName();
            int port = req.getServerPort();

            String redirectString = scheme + "://" + host;
            if (port != 80) {
                redirectString += ":" + port;
            }

            redirectString += location;
            res.sendRedirect(redirectString);

        } else {

            /**
             * User is logged in and has valid card info.
             */

            res.setContentType("text/html");
            PrintWriter out = res.getWriter();

            out.println("<HTML>");
            out.println("<HEAD>");
            out.println("<TITLE>Utbetalte reserveringer</TITLE>");

            out.println("<STYLE TYPE='text/css' TITLE='Input style'>");
            out.println("    INPUT { font-size : 8pt;");
            out.println("           font-weight: bold;");
            out.println("           font-family : Tahoma, Geneva, Arial;});");
            out.println("</STYLE>");

            out.println("<SCRIPT LANGUAGE='JavaScript'>");
            out.println("<!--");

```



```

out.println("function calcPrice(das_objekt, das_string) {");
out.println();
out.println("    var i;");
out.println();
out.println("    i = parseInt(document.Pay_Reservation.total.value);");
out.println();
out.println("    if (das_objekt.checked) {");
out.println("        i += das_string;");
out.println("    } else {");
out.println("        i -= das_string;");
out.println("    }");
out.println();
out.println("    document.Pay_Reservation.total.value = i;");
out.println("    return (true);");
out.println();
out.println("}");
out.println();
out.println("function checkOrders() {");
out.println("    if (parseInt(document.Pay_Reservation.total.value)==0)");
out.println("    {");
out.println("        window.alert(\"Ingen reserveringer valgt!\");");
out.println("        return (false);");
out.println("    } else");
out.println("    {");
out.println("        return (true);");
out.println("    }");
out.println("// -->");
out.println("</SCRIPT>");

out.println("</HEAD>");

out.println("<BODY BGCOLOR=#333333 TEXT=#FFFFFF LINK=#CC9933 "
+ "VLINK=#CC9933 ALINK=#CC9933 "
+ "TOPMARGIN=0 LEFTMARGIN=0 MARGINWIDTH=0 MARGINHEIGHT=0>");

out.println("<IMG SRC=\"/graphics/betal_billletter.gif\" WIDTH=474 "
+ "HEIGHT=59>"
+ "<TABLE BORDER=0 CELLSPACING=2 CELLSPACING=0 WIDTH=100%>");

out.println("<TR>");
out.println("    <FORM ACTION=\"/servlet/orderseats\" METHOD=\"POST\" "
+ "NAME=\"Pay_Reservation\" "
+ "OnSubmit=\"return checkOrders()\">");
out.println("    <INPUT TYPE=\"hidden\" NAME=\"kundeID\" VALUE=\""+
+ kunde.getID() + "\">");
out.println("    <ID WIDTH=80% COLSPAN=2 ALIGN=CENTER VALIGN=TOP>");
out.println("    <FONT FACE=\"Tahoma, Verdana, Geneva, Arial\" SIZE=2>");
out.println("    <TABLE BORDER=0 CELLSPACING=0 CELLSPACING=0 WIDTH=98%>");
out.println("    <!-- Green area with username begin -->");
out.println("    <TR VALIGN=MIDDLE>");
out.println("    <ID BGCOLOR=#999966 VALIGN=TOP>"
+ "<IMG SRC=\"/graphics/02left.gif\" WIDTH=15 HEIGHT=15>"
+ "</ID>");
out.println("    <ID ALIGN=CENTER BGCOLOR=#999966>");
out.println("    <!-- Username area begin -->");
out.println("    <TABLE BORDER=0 CELLSPACING=2 CELLSPACING=0>");
out.println("    <TR><ID ALIGN=CENTER><B><FONT COLOR=#FFFFFF SIZE=3 "
+ "FACE=\"Verdana, Tahoma, Geneva, Arial\">"
+ "Ubetalt reserveringer for " + kunde.getAlias()
+ "</FONT></B></ID></TR>");
out.println("    </TABLE>");
out.println("    <!-- Username area end -->");
out.println("    </ID>");
out.println("    <ID BGCOLOR=#999966 ALIGN=RIGHT VALIGN=TOP HEIGHT=33>"
+ "<IMG SRC=\"/graphics/02right.gif\" WIDTH=15 HEIGHT=15>"
+ "</ID>");
out.println("    </TR>");
out.println("    <!-- Green area with username end -->");
out.println("    <!-- Orange line begin -->");
out.println("    <TR>");
out.println("    <ID COLSPAN=3 ALIGN=CENTER BGCOLOR=#CC9933>"
+ "<IMG SRC=\"/graphics/pixel.gif\" WIDTH=1 HEIGHT=2></ID>");
out.println("    </TR>");
out.println("    <!-- Orange line end -->");
out.println("    </TABLE>");
out.println("    <!-- Gray information area begin -->");
out.println("    <TABLE BORDER=0 CELLSPACING=2 WIDTH=98%>");
out.println("    <TR>");
out.println("    <ID ALIGN=CENTER BGCOLOR=#666666>");
out.println("    <FONT COLOR=#FFFFFF SIZE=2 "
+ "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println("    <B>Film Titel</B>");
out.println("    </FONT>");
out.println("    </ID>");
out.println("    <ID ALIGN=CENTER BGCOLOR=#666666>");
out.println("    <FONT COLOR=#FFFFFF SIZE=2 "
+ "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println("    <B>Dato</B>");
out.println("    </FONT>");
out.println("    </ID>");
out.println("    <ID ALIGN=CENTER BGCOLOR=#666666>");
out.println("    <FONT COLOR=#FFFFFF SIZE=2 "
+ "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println("    <B>Tick</B>");
out.println("    </FONT>");
out.println("    </ID>");
out.println("    <ID ALIGN=CENTER BGCOLOR=#666666>");

```



```

out.println(" <FONT COLOR=#FFFFFF SIZE=2 "
+ "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println(" <B>Placer</B>");
out.println(" </FONT>");
out.println(" </TD>");
out.println(" <ID ALIGN=CENTER BGCOLOR=#666666>");
out.println(" <FONT COLOR=#FFFFFF SIZE=2 "
+ "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println(" <B>Pris</B>");
out.println(" </FONT>");
out.println(" </TD>");
out.println(" <ID ALIGN=CENTER BGCOLOR=#666666>");
out.println(" <FONT COLOR=#FFFFFF SIZE=2 "
+ "FACE=\"Verdana, Tahoma, Geneva, Arial\">");
out.println(" <B>Alt</B>");
out.println(" </FONT>");
out.println(" </TD>");
out.println(" </TR>");

out.println("<!-- Reservations begin -->");

String mSQL = "SELECT film.filmittel, forestilling.dato, forestilling.tid, forestilling.forestillingID, Sum(billetpris.pris) AS SumOfpris,
Count(forestillingID) AS CountOfforestillingID "
+ "FROM ((film INNER JOIN (billetreservation INNER JOIN forestilling ON billetreservation.forestillingID = forestilling.forestillingID) ON film.filmID =
forestilling.filmID) INNER JOIN kunde ON billetreservation.kundeID = kunde.kundeID) INNER JOIN billetpris ON (forestilling.salID = billetpris.salID) AND
(billetreservation.raekke = billetpris.raekkeID) "
+ "GROUP BY film.filmittel, forestilling.tid, forestilling.dato, forestilling.forestillingID, kunde.kundeID, billetreservation.betal"
+ "HAVING (((kunde.kundeID=? ) AND ((billetreservation.betal)=false)) "
+ "ORDER BY film.filmittel, forestilling.forestillingID;";

try {

PreparedStatement stmt = con.prepareStatement(mSQL);
stmt.setInt(1, kunde.getID());
ResultSet rs = stmt.executeQuery();

int totalprice = 0;
int row = 0;

boolean moreResults = rs.next();
while (moreResults) {

/**
* Read all values to make sure they're only read once.
* Reading the same column twice will crash the result.
*/
String m_filmittel = rs.getString(1);
int m_dato = rs.getInt(2);
java.sql.Time m_tid = rs.getTime(3);
int m_forestillingID = rs.getInt(4);
int m_pris = rs.getInt(5);
int m_placer = rs.getInt(6);

totalprice += m_pris;

out.println("<TR" + (row % 2 == 0 ? ">" : " BGOLOR=#383838>"));

out.println(" <ID VALIGN=MIDDLE>");
out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">"
+ m_filmittel + "</FONT></TD>");

out.println(" <ID ALIGN=CENTER VALIGN=MIDDLE>");
out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">"
+ formatDate(m_dato) + "</FONT></TD>");

out.println(" <ID ALIGN=CENTER VALIGN=MIDDLE>");
out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">"
+ formatTime(m_tid) + "</FONT></TD>");

out.println(" <ID ALIGN=CENTER VALIGN=MIDDLE>");
out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">"
+ m_placer + "</FONT></TD>");

out.println(" <ID ALIGN=CENTER VALIGN=MIDDLE>");
out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">"
+ m_pris + "</FONT></TD>");

out.println(" <ID ALIGN=CENTER VALIGN=MIDDLE>");
out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">"
+ "<INPUT TYPE=\"Checkbox\" NAME=\"payfor\" VALUE=\"\" "
+ m_forestillingID + "\" ALIGN=\"TEXTTOP\" CHECKED "
+ "OnClick=\"calcPrice(this, " + m_pris + ")\"></FONT>");
out.println(" </TD>");
out.println("</TR>");

moreResults = rs.next();

row++;

} // --> while (moreResults)

out.println("<!-- Reservations end -->");
out.println("<TR BGOLOR=#999966>");
out.println(" <ID VALIGN=MIDDLE COLSPAN=4>");
out.println(" <FONT SIZE=3 FACE=\"Verdana, Geneva, Arial\">");
out.println(" <B>At betale i alt...</B>");
out.println(" </FONT>");

```

appendiks G – side 208 af 95



TryLoginServlet.java

```
/**
 * @(#)TryLoginServlet.java
 * @version 1.8
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (SDK2.0)
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 */

import java.io.*;
import java.util.*;
import java.text.*;

// Import Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

import Kunde;

/**
 * The <code>TryLoginServlet</code> Servlet searches through
 * database using 'NEB Navn' and 'Kodeord' as parameters. If
 * customer is found he/she will be granted access to
 * our system; otherwise the customer will be redirected to
 * a pregenerated 'do-you-want-to-register' HTML document.
 */
public class TryLoginServlet extends HttpServlet {

    /**
     * These two strings will hold username and password fetched
     * from servlets parameter string.
     */
    private String fAlias;
    private String fPassword;

    /**
     * Handles a <code>POST</code> request from client by checking
     * database for customer-existence.
     *
     * @param <code>req</code> Request from client.
     * @param <code>res</code> Response to client.
     */
    public void doPost(HttpServletRequest req, HttpServletResponse res) throws
        ServletException, IOException {

        try {

            /**
             * Read all customer informations from parameters into
             * local variables
             */
            fAlias = req.getParameter("alias");
            fPassword = req.getParameter("password");

        }
        catch (Exception e) {
            /**
             * Seems like we were unable to get all the info we wanted.
             * We don't have to inform user of this right now. Instead
             * we initializes our local variables with empty values so
             * we can't find a match in our database.
             */
            System.out.println("Error reading user info!");
            fAlias = "";
            fPassword = "";
        }

        /**
         * Try to find customer by using provided alias and password.
         * We create a 'Kunde' object which will fetch information
         * from database. If a customer were found, the 'isLoading'
         * method will return 'true'; otherwise 'false'.
         */
        Kunde k = new Kunde(fAlias, fPassword);

        /**
         * Were we able to find user?
         */
        if (k.isLoading()) {

            /**
             * Great - user was found in database and we are now going
             * to add a cookie to his/hers response using kundeID(). Then
             * we generates a HTML page telling the goooooood news.
             */

```



```

*/

Cookie c = new Cookie(CookieCheck.LOGIN, "" + k.getID());
c.setComment("Alias of NEB Login");
res.addCookie(c);

/**
 * Okay - cookie were added/updated so now we're going to
 * generate a simple HTML page.
 */

res.setContentType("text/html");
PrintWriter out = res.getWriter();

out.println("<H1>");
out.println("<HEAD><TITLE>Login gennetf%OsLash;rt</TITLE>");
out.println("<STYLE TYPE='text/css'>\" TITLE='\"Input style='\">");
out.println("    INPUT { font-size : 10pt;");
out.println("        font-weight: bold;");
out.println("        font-family : Verdana, Tahoma, Geneva, Arial;});");
out.println("</STYLE>");

out.println("</HEAD>");

out.println("<BODY BGCOLOR=#000000 TEXT=#FFFFFF LINK=#FFFFFF \"
+ \"LINK=#FFFFFF ALINK=#FFFFFF>");

out.println("    <SCRIPT LANGUAGE='\"JavaScript'>");
out.println("        parent.open(parent.BottomFrame.location = \"
+ \"\"/servlet/greetuser?kundeID=\" + k.getID()
+ \"\"");
out.println("    </SCRIPT>");

out.println("<CENTER>");
out.println("<TABLE BORDER=0 CELLSPACING=2 WIDTH=277 \"
+ \"HEIGHT=186>");

out.println("<TR VALIGN=TOP HEIGHT=21>");
out.println("<TD VALIGN=TOP WIDTH=267 HEIGHT=21 COLSPAN=2>");
out.println("<FONT FACE='\"Verdana, Geneva, Arial'\" SIZE=4 \"
+ \"COLOR=#6699CC>");
out.println("<B>LOGIN GENNEMF%OsLash;RT</B>");
out.println("</FONT>");
out.println("</TD>");
out.println("</TR>");

out.println("<TR HEIGHT=26>");
out.println("<TD WIDTH=267 HEIGHT=26 VALIGN=TOP ALIGN=CENTER>");
out.println("<FORM>");
out.println("<INPUT TYPE='\"BUTTON'\" VALUE='\" Luk \" \"
+ \"LANGUAGE='\"JavaScript'\" OnClick='\"parent.close()\">");
out.println("</FORM>");
out.println("</TD>");
out.println("</TR>");
out.println("</TABLE>");
out.println("</CENTER>");

// End HTML document
out.println("</BODY>");
out.println("</HTML>");
out.close();

} else { // -> if (k.isLoaded())

/**
 * User wasn't found in database so we have to redirect him
 * to a pregenerated 'not found' HTML page which will give
 * him the option to try again or log in as a new user.
 */

String location = "/no_user_found.html";

String scheme = req.getScheme();
String host = req.getServerName();
int port = req.getServerPort();

String redirectString = scheme + "://" + host;
if (port != 80) {
    redirectString += ":" + port;
}

redirectString += location;
res.sendRedirect(redirectString);

} // -> if (k.isLoaded()) else

} // -> public void doPost(HttpServletRequest req, HttpServletResponse res)
}

```

Klasserne

Her følger nu alle brugte klasser; ligeledes sorteret alfabetisk.



Billetpris.java

```

/*
 * @(#)Billetpris.java
 * @version 1.3
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 *
 */

import java.util.Vector;
import java.sql.*;

import JDBC;

/**
 * The <code>Billetpris</code> class handles all access to the
 * 'Billetpris' table which contains information about every
 * rows price (in danish kroner).
 */
public class Billetpris extends JDBC {

    /**
     * The ID of hall currently being accessed.
     */
    private int atSalID;

    /**
     * This string-array contains all prices in selected hall.
     */
    private int[] atPrices;

    /**
     * Constructs a vector with an element for each row. The prices
     * are read from a database using specified hall ID.
     *
     * @param fID the ID for SQL query.
     */
    public Billetpris(int fID) {

        final String mSQL = "SELECT rækkeID, pris "
            + "FROM billetpris "
            + "WHERE salID = ?";

        atSalID = fID;

        /**
         * Please note that we set a limit at max. 50 rows per hall
         * right here by defining an array of fifty int's.
         */
        atPrices = new int[50];

        try {

            PreparedStatement stmt = con.prepareStatement(mSQL);
            stmt.setInt(1, atSalID);
            ResultSet rs = stmt.executeQuery();

            try {

                while (rs.next())
                    atPrices[rs.getInt("rækkeID")] = rs.getInt("pris");

            }
            catch (Exception e) {
                System.err.println("Error setting price for a row!");
            }

            rs.close();
            stmt.close();

        }
        catch (SQLException ex) {
            System.out.println("Intern SQL fejl: Pristype ikke korrekt læst.");
            super.printStackTrace(ex);
        }

    } // -> public Billetpris(int fID)

    /**
     * Returns currently access hall ID
     *
     * @return the hall ID accessed by object.
     */
    public int getSalID() { return atSalID; }

    /**

```



```
* Returns price of specified row.
*
* @param   fRow the row to get price of/for?!
* @return  the price in danish kroner of specified row.
*/
public int getPrice(int fRow) {

    /**
     * We just read fRow directly from array, since we're not
     * using 0-index (an array starts at zero) when we read
     * prices _into_ our array.
     */

    try {
        return (atPrices[fRow]);
    }
    catch (ArrayIndexOutOfBoundsException aiocbe) { return -1; }

} // -> public int getPrice(int fRow)

}
```



BilletReservation.java

```

/**
 * @(#)BilletReservation.java
 * @version 1.4
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 *
 */

import java.sql.*;
import java.awt.Point;
import java.util.Vector;

import JDBC;

/**
 * Handles all access to 'BilletReservation' table, e.g.
 * writes a new reservation with specified attributes.
 */
public class BilletReservation extends JDBC {

    /**
     * Were we able to make the reservation?
     */
    private boolean oSuccess = false;

    /**
     * Attributes found in 'BilletReservation' table.
     */
    private int atForestillingID = -1;
    private int atKundeID = -1;

    /**
     * Default constructor - used in e.g. ShowHallServlet
     */
    BilletReservation() { atForestillingID = -1; atKundeID = -1; }

    /**
     * Writes a reservations to database by using the two specified
     * IDs and the Vector of seats.
     *
     * @param fForestillingID ID of show
     * @param fKundeID ID of customer
     * @param vSeats Vector with all seats there have to be
     * written to database.
     */
    public BilletReservation(int fForestillingID, int fKundeID, Vector vSeats) {

        final String mSQL = "INSERT INTO billetreservation (forestillingID, "
            + "raekke, saeds, kundeID) "
            + "VALUES (" + fForestillingID + ", ?, ?, " + fKundeID + ")";

        atForestillingID = fForestillingID;
        atKundeID = fKundeID;

        try {

            /**
             * Tell database *not* to update itself after every
             * 'statement'. We don't want it to do that, because
             * we have to do a rollback if we fails to reserv all
             * seats in specified Vector.
             */
            con.setAutoCommit(false);

            /**
             * Prepare the statement, since we're going to use this
             * SQL sentence vSeats.size() times in the following.
             */
            PreparedStatement stmt = con.prepareStatement(mSQL);

            for (int i = 0; i < vSeats.size(); i++) {

                stmt.setInt(1, ((Point)vSeats.elementAt(i)).y);
                stmt.setInt(2, ((Point)vSeats.elementAt(i)).x);

                /**
                 * Execute update is actually batched first. It will wait
                 * till the 'commit()' method is executed.
                 */
                stmt.executeUpdate();

            }

        }

    }

    /**

```



```

        * It's time to commit all cached updates to database.
        */
        con.commit();

    /**
     * Set AutoCommit mode back to default (true).
     */
    con.setAutoCommit(true);

    /**
     * Everything was okay if you reach to this point.
     */
    dbSuccess = true;
}
catch(SQLException ex) {

    /**
     * Seems like there was some kind of SQL Exception. Make
     * sure to tell user about that by setting our 'okay' flag
     * to false.
     */

    super.printStackTrace(ex);
    dbSuccess = false;
}
catch(Exception e) {

    /**
     * Some other kind of exception occurred! Tell user!
     */
    System.out.println("Exception in BilletReservation");
    dbSuccess = false;

}

} // -> public BilletReservation(int ...)

/**
 * Returns ID of show.
 *
 * @return ID of show.
 */
public int getForestillingID() { return atForestillingID; }

/**
 * Returns ID of customer.
 *
 * @return ID of customer.
 */
public int getKundeID() { return atKundeID; }

/**
 * Returns state of reservation.
 *
 * @return true if everything worked out; false otherwise.
 */
public boolean getSuccess() { return dbSuccess; }

/**
 * Retrieves all reserved seats from database using specified show
 * ID as search criteria.
 *
 * @param fForestillingID ID of show.
 * @return Vector with all reserved seats encapsulated in a
 *         awt.Point structure using x as column and y as row.
 */
public Vector getReservedSeats(int fForestillingID) {

    final String mSQL = "SELECT raeke, saede FROM billetreservation "
        + "WHERE forestillingID = ?";

    Vector v = new Vector();

    try {

        PreparedStatement stmt = con.prepareStatement(mSQL);
        stmt.setInt(1, fForestillingID);
        ResultSet rs = stmt.executeQuery();

        while (rs.next())
            v.addElement(new Point(rs.getInt(2), rs.getInt(1)));

        rs.close();
        stmt.close();

    }
    catch (SQLException ex) { printSQLException(ex); }

    return v;

} // -> public Vector getReservedSeats(int fForestillingID)

} // -> public class BilletReservation

```



CookieCheck.java

```
/**
 * @(#)CookieCheck.java
 * @version 1.3
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Servlet Developers Kit 2.0 (SDK2.0)
 *
 * Notes:
 * <none>
 */

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;

/**
 * Reads cookies to find out if customer has logged into our system.
 */
public class CookieCheck {

    /**
     * Name of our cookie.
     */
    public final static String LOGIN = "Login";

    /**
     * Cookies default value when no customer has logged in. Do not
     * use [ ] ( ) = , " / ? @ ; or whitespace in value to avoid
     * any 'special handling' that could happen with older browsers.
     */
    public final static String DEFAULT_VALUE = "No_User_Logged_In";

    int m_kundeID = 0;
    boolean m_LoggedIn = false;

    /**
     * Checks specified request for cookies.
     *
     * @param req A clients request with cookies.
     */
    public CookieCheck(HttpServletRequest req) {

        Cookie cookies[], c = null;

        if ((cookies = req.getCookies()) != null) {

            for (int i = 0; i < cookies.length; i++)
                if (cookies[i].getName().equals(LOGIN)) {

                    try {

                        c = (Cookie)cookies[i].clone();
                        String tmp = c.getValue();

                        if (tmp != DEFAULT_VALUE) {
                            m_kundeID = Integer.parseInt(tmp);
                            m_LoggedIn = true;
                        }

                    } catch (NumberFormatException e) {
                        m_kundeID = 0;
                        m_LoggedIn = false;
                    }

                } // --> if (cookies[i].getName().equals(LOGIN))

        } // --> if ((cookies = req.getCookies()) != null)

    } // --> public CookieCheck(HttpServletRequest req)

    /**
     * Returns state of customer log in.
     *
     * @return true if customer has logged in; false otherwise.
     */
    public boolean isLoggedIn() { return m_LoggedIn; }

    /**
     * Returns ID of customer logged in.
     *
     * @return ID of customer logged into system.
     */
    public int getKundeID() { return m_kundeID; }

}
```



Film.java

```
/**
 * @(#)Film.java
 * @version 1.2
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 *
 */

import java.sql.*;

import JDBC;

/**
 * Handles all access to 'Film' table, e.g. by reading title or
 * description of a specified movie.
 */
public class Film extends JDBC {

    /**
     * Attributes from 'Film' table.
     */
    private int ID;
    private String atTitle;
    private String atYear;
    private int atLength;
    private String atDescription;
    private String atPicture;
    private int atCategoryID;

    /**
     * Retrieves information about movie with specified ID.
     *
     * @param fID ID of movie
     */
    public Film(int fID) {

        final String SQLQuery =
            "SELECT filmtitel, aargang, spilletid, beskrivelse, billed, kategoriID "
            + "FROM film WHERE filmID = " + fID;

        ID = fID;

        try {

            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(SQLQuery);

            /**
             * Only one row's available so you don't have to use a
             * while loop for this one.
             */
            if ( rs.next() ) {

                atTitle      = rs.getString("filmtitel");
                atYear       = rs.getString("aargang");
                atLength     = rs.getInt("spilletid");
                atDescription = rs.getString("beskrivelse");
                atPicture     = rs.getString("billed");
                atCategoryID = rs.getInt("kategoriID");

            } else {

                atTitle = atYear = atDescription = atPicture = "";
                atLength = atCategoryID = -1;

            }

            rs.close();
            stmt.close();

        }
        catch (SQLException ex) { super.printStackTrace(ex); }

    } // -> public Film(int fID)

    /**
     * Returns ID of movie.
     *
     * @return ID of movie.
     */
    public int getID() { return ID; }

    /**
     * Returns title of movie.
     *

```




```
* @return Title of movie.
*/
public String getTitle() { return atTitle; }

/**
 * Returns year of movie.
 *
 * @return Year of movie.
 */
public String getYear() { return atYear; }

/**
 * Returns play length in minutes of movie.
 *
 * @return Length in minutes of movie.
 */
public int getLength() { return atLength; }

/**
 * Returns long description of movie.
 *
 * @return Description of movie.
 */
public String getDescription() { return atDescription; }

/**
 * Returns filename to screenshot of movie.
 *
 * @return filename to screenshot of movie.
 */
public String getPicture() { return atPicture; }

/**
 * Returns ID of category which this movie applies under.
 *
 * @return ID of movie's category.
 */
public int getCategoryID() { return atCategoryID; }
}
```



Forestilling.java

```

/**
 * @(#)Forestilling.java
 * @version 1.3
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 *
 */

import java.sql.*;
import java.util.*;
import java.text.*;

import JDBC;

/**
 * Handles all access to 'Forestilling' table.
 */
public class Forestilling extends JDBC {

    private int ID;
    private int atFilmID;
    private java.sql.Time atTime;
    private int atDate;
    private int atSalID;

    /**
     * Retrieves informations (e.g. time, date, hallID) about
     * show with specified ID from database.
     *
     * @param fID ID of show you want.
     */
    public Forestilling(int fID) {

        final String fSQLQuery = "SELECT * FROM Forestilling " +
                                "WHERE forestillingID = " + fID;

        ID = fID;

        try {

            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(fSQLQuery);

            if ( rs.next() ) {

                atFilmID = rs.getInt("FilmID");
                atTime = rs.getTime("tid");
                atDate = rs.getInt("dato");
                atSalID = rs.getInt("salID");

            } else {

                atFilmID = -1;
                atTime = null;
                atDate = -1;
                atSalID = -1;

            }

            rs.close();
            stmt.close();

        }
        catch (SQLException ex) { super.printStackTrace(ex); }

    } // -> public Forestilling(int fID)

    /**
     * Returns ID of show.
     *
     * @return ID of show.
     */
    public int getID() { return ID; }

    /**
     * Returns ID of movie shown in current show.
     *
     * @return ID of movie in show.
     */
    public int getFilmID() { return atFilmID; }

    /**
     * Returns time of show.
     *

```



```

    * @return    Time of show as a sql.Time object
    */
    public java.sql.Time getTime() { return atTime; }

    /**
     * Formats sql.Time object to a string using hours:minutes
     * format.
     *
     * @return    Time in hours:minutes format
     */
    public String getFormattedTime() {

        int t = atTime.getMinutes();
        return (atTime.getHours() + ":" + (t < 10 ? "0" : "") + t);

    }

    /**
     * Returns number of days from current date.
     *
     * @return    Number of days from current date.
     */
    public int getDate() { return atDate; }

    /**
     * Formats relative DaysFromNow variable from database into a
     * absolute date by reading current date and appending
     * 'DaysFromNow' number.
     *
     * @return Absolute date in format specified by system settings.
     */
    public String getFormattedDate() {

        // Returns formatted date => currentDate + DaysFromNow (atDate)
        DateFormat fullDate = DateFormat.getDateInstance(DateFormat.FULL);
        Calendar cal = Calendar.getInstance();
        cal.add(Calendar.DATE, atDate);

        return fullDate.format(cal.getTime());

    }

    /**
     * Returns ID of hall where this show is currently playing.
     *
     * @return    ID of hall
     */
    public int getSalID() { return atSalID; }

}

```



JDBC.java – superklasse

```

/**
 * @(#)JDBC.java
 * @version 1.4
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 *
 */

import java.net.URL;
import java.util.Calendar;
import java.text.DateFormat;
import java.util.Vector;
import java.awt.Point;
import java.sql.*;

/**
 * Handles all superior database access, e.g. Finds and opens a
 * connection to the Access Database and retrieves information
 * about the database-structure. The <code>JDBC</code> is used
 * as parent-class for all the other classes handling database-
 * access.
 */
public class JDBC {

    /**
     * Needed information used when connecting to database.
     */
    final String databaseURL = "jdbc:odbc:NFB";
    final String DBuser = "svend";
    final String DBpassword = "jensen";

    // Connection to the Access 97 Database
    public Connection con;

    /**
     * Load and link JDBC/ODBC bridge to our Access Database.
     */
    public JDBC() {

        System.out.println("JDBC constructor...");

        try {

            // Load the jdbc-odbc bridge driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            // Show output on default out. (enable for debug purposes)
            DriverManager.setLogStream(System.out);

            /**
             * Attempt to connect to a driver. Each one of the
             * registered drivers will be loaded until one is found
             * that can process this URL.
             */
            con = DriverManager.getConnection(databaseURL, DBuser, DBpassword);

        }
        catch(SQLException sqle) {

            /**
             * Error occurred while trying to find appropriate driver.
             * Print out all Exception info.
             */

            System.out.println("ex: Unable to close Connection");
            printSQLException(sqle);

        }
        catch(Exception e) {
            System.out.println("Unknown problem occurred!");
            e.printStackTrace();
        }

    } // -> public JDBC()

    /**
     * Close connection to database.
     */
    protected void finalize() {

        System.out.println("JDBC finalizing()");

        try {

```



```
// Close the connection to the database
con.close();

}
catch(SQLException sqle) {

    System.out.println("Unable to close Connection");
    printSQLException(sqle);

}

} // -> protected void finalize()

/**
 * Shows information about current connection to Access database.
 */
void getInformation() {

    try {

        DatabaseMetaData dra = con.getMetaData();

        System.out.println("Connected to " + dra.getURL());
        System.out.println("Driver      " + dra.getDriverName());
        System.out.println("Version   " + dra.getDriverVersion());
        System.out.println("");

    }
    catch(SQLException sqle) {
        ;
    }

} // -> public void getInformation()

/**
 * Prints information about specified SQLException.
 *
 * @param    ex    SQLException with info about state, etc.
 */
void printSQLException(SQLException ex) {

    System.out.println("\n*** SQLException caught ***\n");

    while (ex != null) {
        System.out.println();
        System.out.println("SQLState: " + ex.getSQLState ());
        System.out.println("Message:  " + ex.getMessage());
        System.out.println("Vendor:   " + ex.getErrorCode ());
        ex = ex.getNextException();
    }

} // -> public void printSQLException(SQLException ex)

}
```



Kategori.java

```
/**
 * @(#)Kategori.java
 * @version 1.1
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 *
 */

// Import JDBC classes
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Import own JDBC handling class
import JDBC;

/**
 * Handles all access to 'Kategori' table.
 */
public class Kategori extends JDBC {

    private int atCategoryID;
    private String atCategory;

    /**
     * Retrieves name of category with specified ID from the
     * 'Kategori' table.
     *
     * @param fID ID of category
     */
    public Kategori(int fID) {

        final String mSQL =
            "SELECT kategori FROM kategori WHERE kategoriID = " + fID;

        atCategoryID = fID;

        try {

            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(mSQL);

            if (rs.next()) atCategory = rs.getString("kategori");
            else atCategory = "";

            rs.close();
            stmt.close();

        }
        catch (SQLException ex) { super.printStackTrace(ex); }

    } // -> public Kategori(int fID)

    /**
     * Returns ID of category.
     *
     * @return ID of category.
     */
    public int getID() { return atCategoryID; }

    /**
     * Returns name of category.
     *
     * @return Name of category, e.g. 'Action'.
     */
    public String getCategory() { return atCategory; }

}
```



Kunde.java

```

/**
 * @(#)Kunde.java
 * @version 1.4
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 *
 */

import java.sql.*;
import JDBC;

/**
 * Handles all access to 'Kunde' table by reading a customer
 * and adding new customers to database.
 */
public class Kunde extends JDBC {

    // All attributes in 'Kunde' table
    private int atKundeID;
    private String atAlias;
    private String atPassword;
    private String atBirthDay;
    private String atBirthMonth;
    private String atBirthYear;
    private String atPostalCode;
    private String atPhone;
    private String atGender;
    private String atEmail;
    private String atCardHolder;
    private String atCardNumber;
    private String atCardMonth;
    private String atCardYear;

    /**
     * Is true if informations have been successfully loaded from
     * database.
     */
    boolean dLoaded = false;

    /**
     * Creates new customer without setting any attributes. This
     * constructor is only (usefully) used when a new customer has
     * to be added to database or when used in LoginServlet to check
     * if user already exist in database.
     */
    public Kunde() { ; }

    /**
     * Retrieves informations about customer with specified ID. If
     * customer is found in database, all attributes will be set
     * properly and a boolean flag (dLoaded) will be true. Otherwise
     * all attributes will be initialized with 'null' values or
     * similar and flag will be set false.
     *
     * @param fID ID of customer to retrieve from database.
     */
    public Kunde(int fID) {

        final String mSQL = "SELECT * FROM kunde WHERE kundeID = " + fID;

        atKundeID = fID;

        try {

            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(mSQL);

            if (rs.next()) {

                // There IS only one row, so you don't need while loop

                atAlias = rs.getString("alias");
                atPassword = rs.getString("password");
                atBirthDay = rs.getString("foedselsdag");
                atBirthMonth = rs.getString("foedselsmaaned");
                atBirthYear = rs.getString("foedselsaar");
                atPostalCode = rs.getString("postnr");
                atPhone = rs.getString("tlf");
                atGender = rs.getString("køn"); // 0 man - 1 woman
                atEmail = rs.getString("email");
                atCardHolder = rs.getString("kortholder");
                atCardNumber = rs.getString("kortnummer");
                atCardMonth = rs.getString("kortmaaned");
                atCardYear = rs.getString("kortaar");
            }
        }
    }
}

```



```
// All information have been loaded
dLoaded = true;

} else {

    // No information loaded
    dLoaded = false;

}

rs.close();
stmt.close();

}
catch (SQLException ex) { super.printStackTrace(ex); }

} // -> public Kunde(int fID)

/**
 * Retrieves informations about customer with specified alias
 * and password. If no customer was found with these values all
 * attributes will be set to 'null' or similar and the boolean
 * dLoaded will be false. Otherwise all attributes will be set
 * with correct values and boolean flag will be true.
 *
 * This constructor is used when we want to check if a customer's
 * login is valid.
 *
 * @param fAlias Alias of customer.
 * @param fPassword Password of customer.
 */
public Kunde(String fAlias, String fPassword) {

    final String fSQL = "SELECT * FROM kunde WHERE alias = '" + fAlias
        + "' AND password = '" + fPassword + "'";

    atAlias = fAlias;
    atPassword = fPassword;

    try {

        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(fSQL);

        if (rs.next()) {

            // There IS only one row, so you don't need while loop

            atKundeID = rs.getInt("kundeID");
            atBirthDay = rs.getString("foedselsdag");
            atBirthMonth = rs.getString("foedselsmaaned");
            atBirthYear = rs.getString("foedselsaar");
            atPostalCode = rs.getString("postnr");
            atPhone = rs.getString("tlf");
            atGender = rs.getString("køn");
            atEmail = rs.getString("email");
            atCardHolder = rs.getString("kortholder");
            atCardNumber = rs.getString("kortnummer");
            atCardMonth = rs.getString("kortmaaned");
            atCardYear = rs.getString("kortaar");

            // All information have been loaded
            dLoaded = true;

        } else {

            // Initialize kundeID
            atKundeID = -1;

            // No information loaded
            dLoaded = false;

        }

        rs.close();
        stmt.close();

    }
    catch (SQLException ex) { super.printStackTrace(ex); }

} // -> public Kunde(String fAlias, String fPassword)

/**
 * Returns true if there's valid information in object, e.g.
 * after creating object using parameters.
 *
 * @return true if information has been loaded; false otherwise.
 */
public boolean isLoading() { return dLoaded; }

/**
 * Returns true if there's valid card information loaded, i.e.
 * if all four strings contains ... something.
 *
 * @return true if cardinformation was found; false otherwise.
 */
```




```

public boolean isCardLoaded() {

    return ((atCardHolder != null) && (atCardNumber != null) &&
        (atCardMonth != null) && (atCardYear != null));

} // -> public boolean isCardLoaded()

/**
 * Returns ID of customer.
 *
 * @return ID of customer.
 */
public int getID() { return atKundeID; }

/**
 * Gets the alias of this customer.
 *
 * @return the alias of the customer.
 * @see #setAlias
 */
public String getAlias() { return atAlias; }

/**
 * Sets the customers alias to be the specified string.
 *
 * @param fAlias Alias of customer.
 * @see #getAlias
 */
public void setAlias(String fAlias) { atAlias = fAlias; }

/**
 * Gets the password of this customer.
 *
 * @return the password of the customer.
 * @see #setPassword
 */
public String getPassword() { return atPassword; }

/**
 * Sets the customers password to be the specified string.
 *
 * @param fPassword Password of customer.
 * @see #getPassword
 */
public void setPassword(String fPassword) { atPassword = fPassword; }

/**
 * Gets the birthday of this customer.
 *
 * @return the birthday of the customer.
 * @see #setBirthDay
 */
public String getBirthDay() { return atBirthDay; }

/**
 * Sets the customers birthday to be the specified string.
 *
 * @param fBirthDay Birthday of customer.
 * @see #getBirthDay
 */
public void setBirthDay(String fBirthDay) { atBirthDay = fBirthDay; }

/**
 * Gets the birthmonth of this customer.
 *
 * @return the birthmonth of the customer.
 * @see #setBirthMonth
 */
public String getBirthMonth() { return atBirthMonth; }

/**
 * Sets the customers birthmonth to be the specified string.
 *
 * @param fBirthMonth Birthmonth of customer.
 * @see #getBirthMonth
 */
public void setBirthMonth(String fBirthMonth) {
    atBirthMonth = fBirthMonth;
}

/**
 * Gets the birthyear of this customer.
 *
 * @return the birthyear of the customer.
 * @see #setBirthYear
 */
public String getBirthYear() { return atBirthYear; }

/**
 * Sets the customers birthyear to be the specified string.
 *
 * @param fBirthYear Birthyear of customer.
 * @see #getBirthYear
 */
public void setBirthYear(String fBirthYear) { atBirthYear = fBirthYear; }

/**

```



```

* Gets the postalcode of this customer.
*
* @return the postalcode of the customer.
* @see #setPostalCode
*/
public String getPostalCode() { return atPostalCode; }

/**
 * Sets the customers postalcode to be the specified string.
 *
 * @param fPostalCode Postalcode of customer.
 * @see #getPostalCode
 */
public void setPostalCode(String fPostalCode) {
    atPostalCode = fPostalCode;
}

/**
 * Gets the phonenumber of this customer.
 *
 * @return the phonenumber of the customer.
 * @see #setPhone
*/
public String getPhone() { return atPhone; }

/**
 * Sets the customers phonenumber to be the specified string.
 *
 * @param fPhone Phonenumber of customer.
 * @see #getPhone
 */
public void setPhone(String fPhone) { atPhone = fPhone; }

/**
 * Gets the gender of this customer. 0 if it's a male; 1 otherwise.
 *
 * @return the gender of the customer.
 * @see #setGender
*/
public String getGender() { return atGender; }

/**
 * Sets the customers gender to be the specified string. 0 if
 * it's a male; 1 otherwise.
 *
 * @param fGender Gender of customer.
 * @see #getGender
 */
public void setGender(String fGender) { atGender = fGender; }

/**
 * Gets the email of this customer.
 *
 * @return the email of the customer.
 * @see #setEmail
*/
public String getEmail() { return atEmail; }

/**
 * Sets the customers email to be the specified string.
 *
 * @param fEmail Email of customer.
 * @see #getEmail
 */
public void setEmail(String fEmail) { atEmail = fEmail; }

/**
 * Gets the holder of this customers card.
 *
 * @return the cardholder of the customer.
 * @see #setCardHolder
*/
public String getCardHolder() { return atCardHolder; }

/**
 * Sets the cards holder to be the specified string.
 *
 * @param fCardHolder Holder of customers card.
 * @see #getCardHolder
 */
public void setCardHolder(String fCardHolder) {
    atCardHolder = fCardHolder;
}

/**
 * Gets the number of this customers card.
 *
 * @return the number of the customers card.
 * @see #setCardNumber
*/
public String getCardNumber() { return atCardNumber; }

/**
 * Sets the cards number to be the specified string.
 *
 * @param fCardNumber Number of customers card.
 * @see #getCardNumber

```



```

*/
public void setCardNumber(String fCardNumber) {
    atCardNumber = fCardNumber;
}

/**
 * Gets the expiration month of this customers card.
 *
 * @return the expiration month of the customers card.
 * @see #setCardMonth
 */
public String getCardMonth() { return atCardMonth; }

/**
 * Sets the cards expiration month to be the specified string.
 *
 * @param fCardMonth Expiration month of customers card.
 * @see #getCardMonth
 */
public void setCardMonth(String fCardMonth) { atCardMonth = fCardMonth; }

/**
 * Gets the expiration Year of this customers card.
 *
 * @return the expiration year of the customers card.
 * @see #setCardYear
 */
public String getCardYear() { return atCardYear; }

/**
 * Sets the cards expiration Year to be the specified string.
 *
 * @param fCardYear Expiration year of customers card.
 * @see #getCardYear
 */
public void setCardYear(String fCardYear) { atCardYear = fCardYear; }

/**
 * Sets all attributes in object with specified parameters.
 *
 * @param fAlias Alias of customer.
 * @param fPassword Password of customer.
 * @param fBirthDay BirthDay of customer.
 * @param fBirthMonth BirthMonth of customer.
 * @param fBirthYear BirthYear of customer.
 * @param fPostalCode PostalCode of customer.
 * @param fPhone Phone of customer.
 * @param fGender Gender of customer.
 * @param fEmail Email of customer.
 * @param fCardHolder Holder of card.
 * @param fCardNumber Number of card.
 * @param fCardMonth Expiration month of card.
 * @param fCardYear Expiration year of card.
 *
 * @see #setAlias
 * @see #setPassword
 * @see #setBirthDay
 * @see #setBirthMonth
 * @see #setBirthYear
 * @see #setPostalCode
 * @see #setPhone
 * @see #setGender
 * @see #setEmail
 * @see #setCardHolder
 * @see #setCardNumber
 * @see #setCardMonth
 * @see #setCardYear
 */
public void setAll(String fAlias, String fPassword, String fBirthDay,
String fBirthMonth, String fBirthYear, String fPostalCode,
String fPhone, String fGender, String fEmail, String fCardHolder,
String fCardNumber, String fCardMonth, String fCardYear) {

    atAlias = fAlias;
    atPassword = fPassword;
    atBirthDay = fBirthDay;
    atBirthMonth = fBirthMonth;
    atBirthYear = fBirthYear;
    atPostalCode = fPostalCode;

    if (fPhone.length() != 0)
        atPhone = fPhone;

    atGender = fGender;

    if (fEmail.length() != 0)
        atEmail = fEmail;

    if (fCardHolder.length() != 0)
        atCardHolder = fCardHolder;

    if (fCardNumber.length() != 0)
        atCardNumber = fCardNumber;

    if (fCardMonth.length() != 0)
        atCardMonth = fCardMonth;

```



```

        if (fCardYear.length() != 0)
            atCardYear = fCardYear;

    } // -> public void setAll(String fAlias, ..., String fCardYear)

    /**
     * Finds customer with specified alias.
     *
     * @param fAlias Alias of customer.
     * @return true if found; false otherwise.
     */
    public boolean find(String fAlias) {

        final String mSQL = "SELECT alias FROM kunde WHERE alias = ?";

        boolean found = false;

        try {

            PreparedStatement stmt = con.prepareStatement(mSQL);
            stmt.setString(1, fAlias);
            ResultSet rs = stmt.executeQuery();

            /**
             * If you're able to do a 'rs.next()' (move to first result),
             * the customer was found in the database.
             */
            found = rs.next();

            rs.close();
            stmt.close();

        } catch (SQLException ex) { printSQLException(ex); }

        return found;

    } // -> public boolean find(String fAlias)

    /**
     * Adds customer to database by sending an INSERT SQL sentence to
     * database bridge.
     */
    public boolean addtoDatabase() {

        boolean bAdded = false;

        /**
         * You <>have</> to have information about Alias and
         * password to create a new record in DB
         */
        if ( (atAlias.length() == 0) || (atPassword.length() == 0) )
            return false;

        final String fSQL = "INSERT INTO kunde(alias, password, foedselsdag, "
            + "foedselsmaaned, foedselsaar, postnr "
            + "(atPhone != null ? ", tlf" : "")) "
            + ", køn "
            + "(atEmail != null ? ", email" : "")) "
            + "(atCardHolder != null ? ", kortholder" : "")) "
            + "(atCardNumber != null ? ", kortnummer" : "")) "
            + "(atCardMonth != null ? ", kortmaaned" : "")) "
            + "(atCardYear != null ? ", kortaar" : "")) "
            + ") "
            + "VALUES ('" + atAlias + "', '" + atPassword + "', '" + atBirthDay
            + "', '" + atBirthMonth + "', '" + atBirthYear + "' "
            + ", '" + atPostalCode + "' "
            + "(atPhone != null ? ('", '" + atPhone + "') : "")) "
            + ", '" + atGender + "' "
            + "(atEmail != null ? ('", '" + atEmail + "') : "")) "
            + "(atCardHolder != null ? ('", '" + atCardHolder + "') : "")) "
            + "(atCardNumber != null ? ('", '" + atCardNumber + "') : "")) "
            + "(atCardMonth != null ? ('", '" + atCardMonth + "') : "")) "
            + "(atCardYear != null ? ('", '" + atCardYear + "') : "")) "
            + ")";

        try {

            Statement stmt = con.createStatement();
            stmt.executeUpdate(fSQL);

            // Was successfully added to DB
            bAdded = true;

            stmt.close();

        }
        catch (SQLException ex) {
            super.printSQLException(ex);
            System.out.println("Det var ikke muligt at oprette en ny bruger!");
            bAdded = false;
        }

        return bAdded;

    } // -> public boolean addtoDatabase()

}

```



Sal.java

```
/**
 * @(#)Sal.java
 * @version 1.2
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + Java Developers Kit 1.1.7 or similar that implements JDBC 1.2
 *
 * Notes:
 * <none>
 *
 */

import java.sql.*;

import JDBC;

/**
 * The <code>Sal</code> class handles all access to the 'Sal'
 * table.
 */
public class Sal extends JDBC {

    private int ID;
    private String atMap;

    /**
     * Retrieves information about specified hall.
     *
     * @param fID the hall ID.
     */
    public Sal(int fID) {

        final String fSQLQuery = "SELECT map FROM Sal " +
                                "WHERE salID = " + fID;

        ID = fID;

        try {

            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(fSQLQuery);

            if (rs.next())
                atMap = rs.getString("map");
            else {
                atMap = "";
            }

            rs.close();
            stmt.close();

        } catch (SQLException ex) { super.printStackTrace(ex); }

    } // -> public Sal(int fID)

    /**
     * Returns ID of hall.
     *
     * @return ID of hall.
     */
    public int getID() { return ID; }

    /**
     * Returns name of file with coordinates for hall.
     *
     * @return filename to hall-mapping.
     */
    public String getMap() { return atMap; }

}
```



Sallayout.java – applet

```

/**
 * @(#)Sallayout.java
 * @version 1.79
 *
 * Copyright (c) August-November 1998 Nikolaj Norman, Sandie Petersen,
 * Morten Selmer and Peter Theill (peter@conquerware.dk).
 * All Rights Reserved.
 *
 * Requirements:
 * + JDK 1.1 compatible browser (e.g. MSTE4, NS4.Beta)
 *
 * Notes:
 * + You have to get forestillingID since it has to be passed to the
 *   Servlet checking for available seats.
 * + This document is best viewing using a 100 characters screenwidth.
 */

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.util.*;

/**
 * Applet for showing and selecting seats in a hall. This applet is
 * used by *two* different servlets: ShowHallServlet and
 * CheckSeatsServlet. When loading through ShowHallServlet the
 * applet will be able to select/deselect seats and send these to a
 * different Servlet. When using the other (CheckSeatsServlet) the
 * applet only shows selected seats and makes it impossible to
 * change selected seats and send any information from applet.
 */
public class Sallayout extends Applet {

    // Version of applet (used when debugging)
    final String dVersion = "1.79";

    int forestillingID = -1;
    boolean dChangeable = true;

    static final int SEAT_WIDTH = 10, SEAT_HEIGHT = 10;
    private final int yOffset = (33+2+18) + 10;

    /**
     * Owner defined Colors matching overall NFB webdesign.
     */
    private Color nfbBackground;
    private Color nfbGreen;
    private Color nfbBorder;
    private Color nfbGray;
    private Color nfbSelectedSeat;
    private Color nfbReservedSeat;
    private Color nfbFocusedSeat;
    private Color nfbMCR;

    /**
     * Rectangles for 'Reservér Sæder' and 'Slet Valgte' buttons.
     */
    private Rectangle rcSend;
    private Rectangle rcClear;

    /**
     * Number of currently selected seats in applet. This variable
     * could *not* exceed seven (7) since this is the maximum number
     * of seats allowed per reservation.
     */
    int SelectedSeats = 0;

    /**
     * Font used when drawing text on applet, e.g. 'Række/Sæde'
     * string and selected Column/Rows pair.
     */
    private Font font;

    /**
     * Images used for buttons – two for 'off' on two for 'on' state.
     */
    protected Image imgSend, imgClear, imgSendOn, imgClearOn;

    /**
     * This object tracks our images and make sure they're loaded
     * before they're used.
     */
    protected MediaTracker tracker;

    /**
     * We're using two soundclips; one when user selects an available
     * seat and one when user selects a reserved seat.
     */
    protected AudioClip sound, soundReserved;

```



```

/**
 * Our 'Map' object maintaining seats in hall.
 */
protected Map sal;

/**
 * Dimension (width, height) in pixels of applet canvas (this
 * should be around 420x315).
 */
Dimension dim;

/**
 * Initializes applet by creating various objects like all colors,
 * fonts, images, sounds, maps, etc.
 */
public void init() {

    // Print version of applet!
    System.out.println("Sallayout Applet Version " + dVersion);

    // Create some default colors used to draw borders, buttons, etc.
    nfbBackground = new Color(51, 51, 51);
    nfbGreen = new Color(153, 153, 102);
    nfbGray = new Color(96, 103, 96);
    nfbSelectedSeat = Color.green;
    nfbReservedSeat = Color.red;
    nfbFocusedSeat = Color.white;
    nfbBorder = new Color(204, 153, 51);
    nfbXCR = new Color(0, 85, 170);

    // Get size of entire applet
    dim = this.getSize();

    // Load all images - Reserving (on/off) and Nulstil valg (on/off)
    imgSend = this.getImage(this.getDocumentBase(), "/graphics/reserver.gif");
    imgSendOn = this.getImage(this.getDocumentBase(),
        "/graphics/reserver_on.gif");
    imgClear = this.getImage(this.getCodeBase(), "/graphics/nulstil.gif");
    imgClearOn = this.getImage(this.getCodeBase(),
        "/graphics/nulstil_on.gif");

    // Show user info
    this.showStatus("Loading images...");

    // Add to our image tracker
    tracker = new MediaTracker(this);
    tracker.addImage(imgSend, 0);
    tracker.addImage(imgSendOn, 0);
    tracker.addImage(imgClear, 1);
    tracker.addImage(imgClearOn, 1);

    // Make sure you've downloaded all images before proceeding
    try { tracker.waitForAll(); }
    catch (InterruptedException e) { ; }

    // Show user info
    this.showStatus("Loading images...Done!");

    // Load a 'click' sound
    sound = this.getAudioClip(this.getDocumentBase(), "/sounds/notswipe.au");

    // Load a 'reserved' sound (whatever that is ;)
    soundReserved = this.getAudioClip(this.getDocumentBase(),
        "/sounds/notclick.au");

    /**
     * Get ID of forestilling from servlet ShowHallServlet - you
     * have to get this since you're going to send it to
     * CheckSeatsServlet - checking if selected seats are available.
     */
    try {
        forestillingID = Integer.parseInt(getParameter("forestillingID"));
    }
    catch (NumberFormatException e) { forestillingID = -1; }

    /**
     * Create two rectangles (Rectangle objects) for the two
     * 'commit' buttons in applet. These objects are created to
     * make an 'OnMouseOver'-event possible using same method as
     * all seats (reserved or not).
     */
    rcSend = new Rectangle(166, 9, 123, 16);
    rcClear = new Rectangle(297, 9, 123, 16);

    /**
     * Create a Font object for use when writing 'Række/Sæde'-
     * string and actual selected seats.
     */
    font = new Font("Helvetica", Font.BOLD, 10);

    // Create a new map (default width, default height, default space)
    sal = new Map(10, 10, 1);

    if (sal != null) {

        String value;
        int x, y, columns, rows, sCol, sRow, cR, cG, cB;

```



```
// Show user info
this.showStatus("Laser Salen...");

// Read map-information from parameters provided in <PARAM> tags
for (int i=0; (value = getParameter("sect" + i)) != null; i++) {

    // Create 'sections' (x, y, columns, rows, sCol, sRow, cR, cG, cB)

    try {
        StringTokenizer st = new StringTokenizer(value, ",");
        x = Integer.parseInt(st.nextToken());
        y = Integer.parseInt(st.nextToken()) + yOffset;
        columns = Integer.parseInt(st.nextToken());
        rows = Integer.parseInt(st.nextToken());
        sCol = Integer.parseInt(st.nextToken());
        sRow = Integer.parseInt(st.nextToken());
        cR = Integer.parseInt(st.nextToken());
        cG = Integer.parseInt(st.nextToken());
        cB = Integer.parseInt(st.nextToken());
    }
    catch (NoSuchElementException e) { continue; }
    catch (NumberFormatException e) {
        System.out.println("reserv failed!");
        continue;
    }

    this.showStatus("Qpnetter sektor " + i + "...");

    sal.setColor(new Color(cR, cG, cB));
    sal.dbMap(x, y, columns, rows, sCol, sRow);

    this.showStatus("Qpnetter sektor " + i + "...Reddig!");

}

this.showStatus("Laser Salen...Reddig");

/**
 * Read all (max seven) selected seats from <PARAM> tags
 * provided from CheckHallServlet. Reading one of these
 * parameters makes it impossible for the Client to change
 * anything, i.e. no selection possible and no 'commit'
 * buttons visible.
 */
for (int i = 0; (value = getParameter("pick" + i)) != null; i++) {

    /**
     * Is it possible to change selected seats? This flag is
     * used when you are showing reserved seats for Client and
     * you don't want him/her to change it now. If you're able
     * to read one 'pickX' parameter you will *NOT* be able to
     * change anything in this applet.
     */
    dChangeable = false;

    int column, row;

    try {
        StringTokenizer st = new StringTokenizer(value, ",");
        column = Integer.parseInt(st.nextToken());
        row = Integer.parseInt(st.nextToken());
    }
    catch (NoSuchElementException e) { column = 0; row = 0; continue; }
    catch (NumberFormatException e) { System.out.println("reserv failed!"); column = 0; row = 0; continue; }

    ImageMapRectangle r = findSeat(column, row);
    if (r != null)
        r.selected = true;

} //--> for (int i = 0; (value = getParameter("pick" + i)) != null; i++)

/**
 * Read all booked seats.
 */
for (int i = 0; (value = getParameter("reserv" + i)) != null; i++) {

    int column, row;

    try {
        StringTokenizer st = new StringTokenizer(value, ",");
        column = Integer.parseInt(st.nextToken());
        row = Integer.parseInt(st.nextToken());
    }
    catch (NoSuchElementException e) { continue; }
    catch (NumberFormatException e) { System.out.println("reserv failed!"); continue; }

    ImageMapRectangle r = findSeat(column, row);
    if (r != null)
        r.reserved = true;

} //--> for (int i = 0; (value = ...serv" + i)) != null; i++)

}

/**
 * Add our own Mouse-events to applet. These two 'Listener'
```




```

* handles OnMouseOver-events (when moving mouse around in
* applet: it's possible to see which seat the cursor is above
* and it's possible to change images when above one of the two
* 'commit' buttons) and OnClick-events (when sending
* information to another servlet, clearing all selected seats
* or simply selecting or deselecting a seat). If you're unable
* to make a change in this applet (dChangeable=false), the
* listener will not be added to applet since it would make no
* sense.
*/
if (dChangeable) {
    this.addMouseListener(new Listener());
    this.addMouseMotionListener(new MotionListener());
}

}

public void destroy() {

    // Flush all images
    imgSend.flush();
    imgClear.flush();
    imgSendOn.flush();
    imgClearOn.flush();
}

/**
 * Paint applet on canvas.
 *
 * @param g Canvas to paint applet to.
 */
public void paint(Graphics g) {

    // Fill applet area with standard background color
    g.setPaintMode();
    g.setColor(nfbBackground);
    g.fillRect(0, 0, dim.width, dim.height);

    // Draw upper green panel with two buttons
    drawTopGreenPanel(g);

    // Draw gray 'status' panel just below the green panel
    drawTopGrayPanel(g);

    g.drawString("FilmLearnd", dim.width/2 - 26, yOffset);

    // Draw hall below the two other panels
    sal.drawMap();

    // Draw all reserved seats
    sal.drawReservedSeats();

    // Fill all selected seats in current map
    sal.drawSeats();

}

/**
 * Update canvas by repainting it.
 *
 * @param g Canvas to paint to.
 */
public void update(Graphics g) { paint(g); }

/**
 * Draw top panel which includes drawing the two 'commit'
 * buttons in their 'off' (no mouse over) state.
 */
public void drawTopGreenPanel(Graphics g) {

    // Make sure you're not XOR'ing anything
    g.setPaintMode();

    // fill area with 'green-like' color
    g.setColor(nfbGreen);
    g.fillRect(0, 0, dim.width, 33);

    // Draw orange border
    g.setColor(nfbBorder);
    g.fillRect(0, 33, dim.width, 2);

    if (dChangeable) {
        // Draw button images
        g.drawImage(imgSend, rcSend.x, rcSend.y, this);
        g.drawImage(imgClear, rcClear.x, rcClear.y, this);
    }

}

public void drawTopGrayPanel(Graphics g) {

    // ** Draw Top Gray panel **

    g.setPaintMode();

    // fill area with 'gray-like' color

```



```

g.setColor(nfbGray);

// Fill horizontal area
g.fillRect(0, 35, dim.width, 18);

// Loop through all seats and update 'seats area' (right panel)
g.setFont(font);
g.setColor(Color.white);
g.drawString("Rakke, Sake", 16, 48);

int curSeat = 1;
for (int i = 0; i < sal.getRects().size(); i++) {
    ImageMapRectangle r = (ImageMapRectangle)sal.getRects().elementAt(i);
    if (r.selected) {
        curSeat++;
        g.drawString(r.row + " ", r.column, 50+(40*curSeat), 48);
    }
} // --> for (int i = 0; i < sal.getRects().size(); i++)

} // --> public void drawTopGrayPanel(Graphics g)

protected ImageMapRectangle findSeat(MouseEvent e) {

    int x = e.getX(), y = e.getY();

    for (int i = 0; i < sal.getRects().size(); i++) {
        ImageMapRectangle r = (ImageMapRectangle)sal.getRects().elementAt(i);
        if (r.contains(x, y)) return r;
    }

    return null;
} // --> protected ImageMapRectangle findSeat(MouseEvent e)

protected ImageMapRectangle findSeat(int Column, int Row) {

    for (int i = 0; i < sal.getRects().size(); i++) {
        ImageMapRectangle r = (ImageMapRectangle)sal.getRects().elementAt(i);
        if ( (r.column == Column) && (r.row == Row) )
            return r;
    }

    return null;
} // --> protected ImageMapRectangle findSeat(int Column, int Row)

protected Rectangle findButton(MouseEvent e) {

    int x = e.getX(), y = e.getY();

    if (rcSend.contains(x, y))
        return rcSend;

    if (rcClear.contains(x, y))
        return rcClear;

    return null;
} // --> protected Rectangle findButton(MouseEvent e)

/**
 * The <code>Listener</code> handles events generated with the
 * mouse, e.g. a mousepress.
 */
class Listener extends MouseAdapter {

    /**
     * Handles a mousepress in window by finding clicked Rectangle
     * which probably is either a seat or one of the commit buttons
     * (send, clear).
     *
     * @param <D>MouseEvent e</D> contains (x, y) coordinates of
     * click.
     */
    public void mousePressed(MouseEvent e) {

        Rectangle rc = findButton(e);

        if (rc != null) {

            if (rc.equals(rcSend)) sendClicked();
            else clearClicked();
            // Play a little 'click' sound for the user
            sound.play();
            rc = null;
            return;
        }

        // Search array of seats
        ImageMapRectangle r = findSeat(e);

        // If user hasn't click an seat - do nothing.
        if (r == null) return;

        // If seat already is reserved - play a 'not good' sound

```



```

        if (r.reserved) {
            soundReserved.play();
            return;
        }

        if (r.selected)
            SelectedSeats--;
        else
            if (SelectedSeats < 7)
                SelectedSeats++;
            else
                return;

        // Invert selection - e.g. if seat already were reserved it's now
        // available for reservations again
        r.selected = !r.selected;

        // Get Graphics object (you're going to update some areas)
        Graphics g = Applet.this.getGraphics();

        g.setPaintMode();
        if (r.selected) g.setColor(new Color(0, 170, 170));
        else g.setColor(new Color(153, 204, 204));
        g.fillRect(r.x+1, r.y+1, r.width-2, r.height-2);

        // Play a little 'click' sound for the user
        sound.play();

        // Update areas with no. of selected seats and *which* seats
        drawTopGrayPanel(g);
    }

} // -> class Listener extends MouseAdapter

/**
 * The <code>MotionListener</code> handles events generated with
 * the mouse, e.g. a mousemove.
 */
class MotionListener extends MouseMotionAdapter {

    private Image[] rectangle;
    private Rectangle rcLast;

    /**
     * Finds seat or button mouse is located above and draws its
     * rectangle or bitmap.
     *
     * @param <code>MouseEvent e</code> contains e.g. (x, y)-
     *         coordinates of click.
     */
    public void mouseMoved(MouseEvent e) {

        Graphics g = Applet.this.getGraphics();

        // Find out if mouse is positioned above a commit button
        Rectangle rc = findButton(e);

        // There are several cases you have to check for
        // rc == null && rcLast == null => do nothing at all
        // rc != null && rcLast == null => mouse has entered button
        // rc != null && rcLast != null => mouse has moved inside button or
        //                               a new was selected
        // rc == null && rcLast != null => mouse has left button

        if ((rc != null) && (rcLast == null)) {

            // * * * Mouse has entered new button * * *

            g.setColor(nfGreen);
            g.fillRect(rc.x, rc.y, rc.width, rc.height);

            // Draw correct 'focus' image (Send or Clear)
            g.drawImage(rc.equals(rcSend) ? imgSendOn : imgClearOn, rc.x, rc.y,
                Applet.this);

            // Save Rectangle for check in further mouse movements.
            rcLast = rc;

            // Nothing more to do in this method
            return;
        }

        if ((rc != null) && (rcLast != null)) {

            // * * * Mouse has moved inside button or a new was selected * * *

            // Moved inside button, do nothing
            if (rc.equals(rcLast)) return;

            g.setColor(nfGreen);
            g.fillRect(rc.x, rc.y, rc.width, rc.height);
            g.fillRect(rcLast.x, rcLast.y, rcLast.width, rcLast.height);

            // Draw correct 'focus' and 'no focus' images (Send or Clear)
            g.drawImage(rc.equals(rcSend) ? imgSendOn : imgClearOn, rc.x, rc.y,
                Applet.this);
        }
    }
}

```



```

        g.drawImage(rcLast.equals(rcSend) ? imgSend : imgClear, rcLast.x,
            rcLast.y, Applet.this);

        // Save Rectangle for check in further mouse movements.
        rcLast = rc;

        // Nothing more to do in this method
        return;
    }

    if ((rc == null) && (rcLast != null)) {

        // * * * Mouse has left button * * *

        g.setColor(nfbGreen);
        g.fillRect(rcLast.x, rcLast.y, rcLast.width, rcLast.height);

        // Draw correct 'no focus' image
        g.drawImage(rcLast.equals(rcSend) ? imgSend : imgClear, rcLast.x,
            rcLast.y, Applet.this);

        // Save Rectangle for check in further mouse movements.
        rcLast = null;

        // Nothing more to do in this method
        return;
    }

    // If you come to this point, no commit button was hit so it's time for
    // seats checking.

    g.setXORMode(nfbXOR);

    ImageMapRectangle r = findSeat(e);
    if ((r == null) && (lastrect == null)) return;
    if (r == lastrect) return;

    if ((r != null) && (lastrect != null)) {
        // Two different seats are selected
        g.fillRect(r.x, r.y, r.width, r.height);
        g.fillRect(lastrect.x, lastrect.y, lastrect.width, lastrect.height);
        lastrect = r;
        return;
    }

    if ((r != null) && (lastrect == null)) {
        g.fillRect(r.x, r.y, r.width, r.height);
        lastrect = r;
        return;
    }

    if ((r == null) && (lastrect != null)) {
        g.fillRect(lastrect.x, lastrect.y, lastrect.width, lastrect.height);
        lastrect = null;
        return;
    }

    Applet.this.showStatus("(" + e.getX() + ", " + e.getY() + ")");

} // → public void mouseMoved(MouseEvent e)

} // → class MotionListener extends MouseMotionAdapter

/**
 * An extended Rectangle class which is used to simulate a seat;
 * extra attributes for color, row, column and selected has been
 * added.
 */
static class ImageMapRectangle extends Rectangle {

    boolean selected;    // Is seat selected for reservation?
    boolean reserved;    // Is seat taken (reserved)?
    int column, row;     // Position in hall
    Color color;         // Color of seat (price-class)

    public ImageMapRectangle(int x, int y, int column, int row, Color color,
        boolean selected) {

        super(x, y, SEAT_WIDTH, SEAT_HEIGHT);
        this.column = column;
        this.row = row;
        this.color = color;
        this.selected = selected;
        reserved = false;

    } // → public ImageMapRectangle(int x, int y, ..., boolean selected)

} // → static class ImageMapRectangle extends Rectangle

/**
 * The <code>Map</code> class maintains and draws all seats in
 * viewed hall.
 */
public class Map {

    private int width, height, space, price;

```



```

private Color color;
private Vector rects;

/**
 * Creates a new map with seat-dimensions as specified
 *
 * @param width    Width in pixels of a seat
 * @param height   Height in pixels of a seat
 * @param space    Space in pixels between seats
 */
public Map(int width, int height, int space) {

    // Initialize variables
    this.width = width;
    this.height = height;
    this.space = space;

    rects = new Vector();

} // -> public Map(int width, int height, int space)

/**
 * Returns space (in pixels) between every seat
 *
 * @return Space (in pixels) between seats.
 */
public int getSpace() { return space; }

/**
 * Returns dimension of seat in map.
 *
 * @return Dimension of every seat.
 */
public Dimension getDimension() { return new Dimension(width, height); }

public void setColor(Color color) { this.color = color; }
public Color getColor() { return color; }
public void setPrice(int price) { this.price = price; }
public int getPrice() { return price; }
public Vector getRects() { return rects; }

/**
 * Creates a part of the map by using specified group of seats
 * and create equal amounts of seats.
 *
 * @param x    X-coordinate of upper left corner in group.
 * @param y    Y-coordinate of upper left corner in group.
 * @param columns    Number of columns in group.
 * @param rows       Number of rows in group.
 * @param startColumn    Number in hall where seats have
 *                       to start with.
 * @param startRow      Number in hall where seats have
 *                       to start with.
 */
public void dMap(int x, int y, int columns, int rows, int startColumn,
int startRow) {

    int col, row, cx, cy;

    // Add to Vector and draw it
    for (col = 0; col < columns; col++) {

        cx = x + col*width + col*space;

        for (row = 0; row < rows; row++) {
            cy = y + (row*height + row*space);
            ImageMapRectangle r = new ImageMapRectangle(cx, cy,
startColumn+col, startRow+row, color, false);
            rects.addElement(r);
        }

    } // -> for (col = 0; col < columns; col++)

} // -> public void dMap(int x, int y, ..., int startRow)

/**
 * Draws complete map on Applets canvas by looping through
 * all seats in private vector.
 */
public void drawMap() {

    Graphics g = Applet.this.getGraphics();
    g.setPaintMode();

    // Draw complete map
    for (int i = 0; i < rects.size(); i++) {
        ImageMapRectangle r = (ImageMapRectangle)rects.elementAt(i);
        g.setColor(r.color);
        g.fillRect(r.x, r.y, r.width, r.height);
    }

}

```



```

/**
 * Draws all reserved seats on Applets canvas by looping through
 * all seats and checking if it's reserved.
 */
public void drawReservedSeats() {

    // Draw all reserved seats

    Graphics g = Applet.this.getGraphics();
    g.setPaintMode();
    g.setColor(nfrReservedSeat);

    for (int i = 0; i < rects.size(); i++) {
        ImageMapRectangle r = (ImageMapRectangle)rects.elementAt(i);
        if (r.reserved) g.fillRect(r.x+1, r.y+1, r.width-2, r.height-2);
    }

} // → public void drawReservedSeats()

/**
 * Draws all selected seats on Applets canvas by looping through
 * all seats and checking if it's selected. This method will be
 * especially used when updating applet after a e.g. repaint.
 */
public void drawSeats() {

    Graphics g = Applet.this.getGraphics();
    g.setColor(nfrSelectedSeat);

    for (int i = 0; i < rects.size(); i++) {
        ImageMapRectangle r = (ImageMapRectangle)rects.elementAt(i);
        if (r.selected) g.fillRect(r.x+1, r.y+1, r.width-2, r.height-2);
    }

} // → public void drawSeats()

} // → public class Map

/**
 * Creates a list of selected seats and generates a URL for use
 * with a Servlet running on our server. When URL has been
 * created, we ask the Applet to browse this, viewing output
 * (from Servlet) in a special frame.
 *
 * Make sure to check 'CheckSeatsServlet' for correct syntax of
 * the URL generated through this method.
 */
private void sendClicked() {

    String servletURL = "/servlet/checkseats?forestillingID="
        + forestillingID;
    boolean hSomeSelected = false;

    int curSeat = 0;

    // Add seats (max 7) to servletURL string
    for (int i = 0; i < sal.getRects().size(); i++) {

        ImageMapRectangle r = (ImageMapRectangle)sal.getRects().elementAt(i);
        if (r.selected) {
            servletURL += "&seat=" + curSeat + "&=" + r.column + "&=" + r.row;
            hSomeSelected = true;
            curSeat++;
        }

    }

    // → for (int i = 0; i < sal.getRects().size(); i++)

    // Don't browse URL if you haven't selected any seats
    if (!hSomeSelected) return;

    // We will have to timestamp URL to avoid using browsers cache.
    Calendar cal = Calendar.getInstance();
    servletURL += "&timestamp=" + cal.toString();

    URL url;
    try {

        // Create URL
        url = new URL(Applet.this.getDocumentBase(), servletURL);

        // Browse URL in 'MainFrame' frame
        Applet.this.getAppletContext().showDocument(url, "MainFrame");

    }
    catch (MalformedURLException male) {
        System.err.println("Unable to contact URL - server down?");
    }

} // → private void sendClicked()

/**
 * Clears all reserved (selected) seats from canvas and from

```



```
* internal Vector of selected seats.
*/
private void clearClicked() {

    // Get Graphics Canvas
    Graphics g = this.getGraphics();

    g.setColor(nfoGreen);

    // Loop through all seats and remove all the selected ones.
    for (int i = 0; i < sal.getRects().size(); i++) {

        ImageMapRectangle r = (ImageMapRectangle) sal.getRects().elementAt(i);

        if (r.selected) {

            g.fillRect(r.x, r.y, r.width, r.height);
            r.selected = false;

        }

    } // --> for (int i = 0; i < sal.getRects().size(); i++)

    // Be sure to reset our 'Number of selected seats' variable
    SelectedSeats = 0;

    // Update area with selected seats info
    drawTopGrayPanel(g);

} // --> private void clearClicked()

}
```

Pregenererede HTML sider

Den sidste del i dette appendiks indeholder de pregenererede HTML sider. Sorteret alfabetisk.

about.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
<HEAD>
<TITLE>Nordisk Film Biografer</TITLE>
</HEAD>

<!-- Background white, links blue (unvisited), navy (visited), red (active) -->
<BODY BGCOLOR="#333333" TEXT="#FFFFFF" TOPMARGIN="0" LEFTMARGIN="0" MARGINWIDTH="0" MARGINHEIGHT="0" LINK="#FFFFFF" VLINK="#FF00FF" ALINK="#FF00FF">
<IMG SRC="graphics/leftcurve.gif" WIDTH=32 HEIGHT=25 BORDER=0 ALT="Left Curve"><BR>

<CENTER>
<TABLE WIDTH="100%" BORDER="0" CELLSPACING="0" CELLPADDING="16" ALIGN="CENTER">
<TR>
<TD WIDTH=50%>
<TABLE WIDTH="100%" BORDER="2" CELLSPACING="0" CELLPADDING="8" BORDERCOLOR="#009933" BORDERCOLORLIGHT="#00AA44" BORDERCOLORDARK="#888822">
<TR>
<TD VALIGN=TOP><FONT FACE="Tahoma, Verdana, Geneva, Arial" SIZE=2 COLOR=#FFFFFF><IMG SRC="graphics/morten.gif" WIDTH=100 HEIGHT=120 BORDER=0 ALT="Image of Morten"
ALIGN="LEFT">&nbsp;<BR><B>Morten Selmer</B></NDBR><BR>&nbsp;<NDBR>27 &r</NDBR><BR>&nbsp;<A HREF="mailto:helio@post8.tele.dk">helio@post8.tele.dk</A><BR>
&nbsp;<BR><D>Dokumentering & grafiker</D></TD>
</TR>
</TABLE>

</TD>

<TD WIDTH=50%>
<TABLE WIDTH="100%" BORDER="2" CELLSPACING="0" CELLPADDING="8" BORDERCOLOR="#009933" BORDERCOLORLIGHT="#00AA44" BORDERCOLORDARK="#888822">
<TR>
<TD VALIGN=TOP><FONT FACE="Tahoma, Verdana, Geneva, Arial" SIZE=2 COLOR=#FFFFFF><IMG SRC="graphics/nikolaj.gif" WIDTH=100 HEIGHT=120 BORDER=0 ALT="Image of Nikolaj"
ALIGN="LEFT">&nbsp;<BR><B>Nikolaj Noman</B></NDBR><BR>&nbsp;<NDBR>22 &r</NDBR><BR>
&nbsp;<A HREF="mailto:noman@post10.tele.dk">noman@post10.tele.dk</A><BR>
&nbsp;<BR><D>Dokumentering & testning</D></TD>
</TR>
</TABLE>

</TD>

<TR>
<TD WIDTH=50%>
<TABLE WIDTH="100%" BORDER="2" CELLSPACING="0" CELLPADDING="8" BORDERCOLOR="#009933" BORDERCOLORLIGHT="#00AA44" BORDERCOLORDARK="#888822">
<TR>
<TD VALIGN=TOP><FONT FACE="Tahoma, Verdana, Geneva, Arial" SIZE=2 COLOR=#FFFFFF><IMG SRC="graphics/peter.gif" WIDTH=100 HEIGHT=120 BORDER=0 ALT="Image of Peter"
ALIGN="LEFT">&nbsp;<BR><B>Peter Theill</B></NDBR><BR>&nbsp;<NDBR>21 &r</NDBR><BR>
&nbsp;<A HREF="mailto:peter@conquerware.dk">peter@conquerware.dk</A><BR>
&nbsp;<BR><D>Programmering & Webdesign</D></TD>
</TR>
</TABLE>
```



```
</TD>

<TD WIDTH=50%>
<TABLE WIDTH="100%" BORDER="2" CELSPACING="0" CELLPADDING="5" BORDERCOLOR="#009933" BORDERCOLORTLIGHT="#00A044" BORDERCOLORDARK="#008822">
<TR>
<TD VALIGN=TOP><FONT FACE="Tahoma, Verdana, Geneva, Arial" SIZE=2 COLOR=#FFFFFF><IMG SRC="graphics/sandie.gif" WIDTH=100 HEIGHT=120 BORDER=0 ALT="Image of Sandie"
ALIGN="LEFT">&nbsp;<NBR><B>Sandie Petersen</B></NBR><BR>&nbsp;<NBR><22 år</NBR><BR>&nbsp;<A HREF="mailto:starwarsboots@hotmail.dk">starwarsboots@hotmail.dk</A><BR>
&nbsp;<A>Analysering & Dokumentering</A></TD>
</TR>
</TABLE>
</TD>
</TR>

</TABLE>
</CENTER>

</BODY>
</HTML>
```




error_reservation.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">

<HTML>
<HEAD>
  <TITLE>NEB - Sædereservering annulleret</TITLE>
  <STYLE TYPE="text/css" TITLE="input style">
    INPUT { font-size : 10pt;
    font-weight: bold;
    font-family : Verdana, Tahoma, Geneva, Arial;}
  </STYLE>
</HEAD>

<BODY BGCOLOR="#333333" TEXT="#FFFFFF" LINK="#009933" VLINK="#009933" ALINK="#009933" LEFTMARGIN=0 TOPMARGIN=0 MARGINWIDTH=0 MARGINHEIGHT=0>
<IMG SRC="graphics/leftcurve.gif" WIDTH=32 HEIGHT=25 BORDER=0 ALT=""><CENTER>

<TABLE WIDTH="98%" BORDER="0" CELSPACING="0" CELLPADDING="0">
<TR>
<TD VALIGN="TOP" BGCOLOR="#999966"><IMG SRC="graphics/02left.gif" WIDTH=15 HEIGHT=15 BORDER=0 ALT="Left Curve"></TD>
<TD WIDTH="100%" ALIGN="CENTER" NOWRAP BGCOLOR="#999966"><FONT FACE="Verdana, Helvetica, Geneva, Arial, sans-serif" SIZE="3" COLOR="#FFFFFF"><B>Sædereservering
annulleret</B></FONT></TD>
<TD ALIGN="RIGHT" VALIGN="TOP" BGCOLOR="#999966" height="33"><IMG SRC="graphics/02right.gif" WIDTH=16 HEIGHT=15 BORDER=0 ALT="Right Curve"></TD>
</TR>
<TR>
<TD COLSPAN="3" BGCOLOR="#009933"><IMG SRC="graphics/pixel.gif" WIDTH=1 HEIGHT=2 BORDER=0 ALT=""></TD>
</TR>
<TR>
<TR><TD COLSPAN="3" ALIGN="JUSTIFY"><FONT FACE="Verdana, Helvetica, Geneva, Arial, sans-serif" SIZE="3" COLOR="#FFFFFF"><P ALIGN="JUSTIFY">Det var desværre ikke muligt at
reservere de valgte sæder. Det skyldes højst sandsynligt, at en anden bruger har nået at reservere sæderne inden Deres billede over reserverede sæder i salen blev
opdateret.</P>
<P>
Gå tilbage og tryk <TT>Opdater</TT> (refresh) i din browser for at se nyeste version af salen.</P>
<P>
<P>
<CENTER><FORM><INPUT TYPE="BUTTON" VALUE="%#171; Tilbage" LANGUAGE="JavaScript" OnClick="history.back()" "></FORM></CENTER>
</FONT></TD></TR>
</TABLE>
</CENTER>

</BODY>
</HTML>
```



index.html

```

<FRAMESET ROWS="*, 50" FRAMEBORDER="0" BORDER="0">
<FRAMESET COLS="121,*" FRAMEBORDER="0" BORDER="0">
  <FRAME NAME="LeftFrame" SRC="menu.html" MARGINWIDTH="0" MARGINHEIGHT="0" SCROLLING="no" FRAMEBORDER="no" NORESIZE>
  <FRAMESET ROWS="51,*" FRAMEBORDER="0" BORDER="0">
    <FRAME NAME="TopFrame" SRC="top.html" MARGINWIDTH="0" MARGINHEIGHT="0" SCROLLING="no" FRAMEBORDER="no">
    <FRAME NAME="MainFrame" SRC="main.html" MARGINWIDTH="0" MARGINHEIGHT="0" SCROLLING="auto" FRAMEBORDER="no">
  </FRAMESET>
</FRAMESET>
<FRAME NAME="BottomFrame" SRC="no_user_bottom.html" MARGINWIDTH="0" MARGINHEIGHT="0" SCROLLING="no" FRAMEBORDER="0">
</FRAMESET>

<noframes>
<?You need a frames capable browser to view this page.</?>
</noframes>

</html>

```



login.html

```
<HTML>
<HEAD><TITLE>Registrering</TITLE>
<STYLE TYPE="text/css" TITLE="Input style">
INPUT { font-size : 9pt;
        font-weight: bold;
        font-family : Tahoma, Geneva, Arial;}
SELECT { font-size : 9pt;
        font-family : Tahoma, Geneva, Arial;}
</STYLE>

</HEAD>

<BODY BGCOLOR="#000000" TEXT="#FFFFFF" LINK="#FFFFFF" VLINK="#FFFFFF" ALINK="#FFFFFF">

<SCRIPT LANGUAGE="JavaScript"><!--
function Form1_Validator(theForm)
{

    if (theForm.Alias.value == "")
    {
        alert ("Husk at udfylde NFB Navn.");
        theForm.Alias.focus();
        return (false);
    }

    if (theForm.Alias.value.length < 3)
    {
        alert ("Dit NFB Navn skal minimum være 3 bogstaver langt.");
        theForm.Alias.focus();
        return (false);
    }

    if (theForm.Alias.value.length > 20)
    {
        alert ("Der kan maksimalt være 20 bogstaver i dit NFB Navn.");
        theForm.Alias.focus();
        return (false);
    }

    var checkCK = "ABCDEFGHIJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_" ;
    var checkStr = theForm.Alias.value;
    var allValid = true;
    for (i = 0; i < checkStr.length; i++)
    {
        ch = checkStr.charAt(i);
        for (j = 0; j < checkCK.length; j++)
            if (ch == checkCK.charAt(j))
                break;
        if (j == checkCK.length)
        {
            allValid = false;
            break;
        }
    }
    if (!allValid)
    {
        alert ("Brug venligst kun bogstaver i dit NFB Navn. Ikke små, men gerne '-' og '_'");
        theForm.Alias.focus();
        return (false);
    }

    if (theForm.password.value.length == "")
    {
        alert ("Husk at udfylde dit kodeord.");
        theForm.password.focus();
        return (false);
    }

    if (theForm.gender.selectedIndex < 1)
    {
        alert ("Husk at vælge køn.");
        theForm.gender.focus();
        return (false);
    }

    if (theForm.birthday.value.length == "")
    {
        alert ("Husk at udfylde dagen for din fødselsdag.");
        theForm.birthday.focus();
        return (false);
    }

    if (theForm.birthmonth.value.length == "")
    {
        alert ("Husk at udfylde måneden for din fødselsdag.");
        theForm.birthmonth.focus();
        return (false);
    }

    if (theForm.birthyear.value.length == "")
    {

```



```

    alert("Husk at udfylde året for din fødselsdag.");
    theForm.birthyear.focus();
    return (false);
}

var checkCK = "0123456789";
var checkStr = theForm.birthday.value;
var allValid = true;
for (i = 0; i < checkStr.length; i++)
{
    ch = checkStr.charAt(i);
    for (j = 0; j < checkCK.length; j++)
        if (ch == checkCK.charAt(j))
            break;
    if (j == checkCK.length)
    {
        allValid = false;
        break;
    }
}
if (!allValid)
{
    alert("Dagen for fødselsdag må kun indeholde tal.");
    theForm.birthday.focus();
    return (false);
}

var checkCK = "0123456789";
var checkStr = theForm.birthmonth.value;
var allValid = true;
for (i = 0; i < checkStr.length; i++)
{
    ch = checkStr.charAt(i);
    for (j = 0; j < checkCK.length; j++)
        if (ch == checkCK.charAt(j))
            break;
    if (j == checkCK.length)
    {
        allValid = false;
        break;
    }
}
if (!allValid)
{
    alert("Måned for fødselsdag må kun indeholde tal.");
    theForm.birthmonth.focus();
    return (false);
}

var checkCK = "0123456789";
var checkStr = theForm.birthyear.value;
var allValid = true;
for (i = 0; i < checkStr.length; i++)
{
    ch = checkStr.charAt(i);
    for (j = 0; j < checkCK.length; j++)
        if (ch == checkCK.charAt(j))
            break;
    if (j == checkCK.length)
    {
        allValid = false;
        break;
    }
}
if (!allValid)
{
    alert("Året for fødselsdag må kun indeholde tal.");
    theForm.birthyear.focus();
    return (false);
}

if (theForm.birthyear.value != "")
{
    if (theForm.birthyear.value < 1900)
    {
        alert("Ikke et lovligt fødselsår. Du skal angive f.eks. 1968, altså også århundredet!");
        theForm.birthyear.focus();
        return (false);
    }
}

if (theForm.birthday.value != "")
{
    if (theForm.birthday.value < 1 || theForm.birthday.value > 31)
    {
        alert("Ikke en lovlig dag i din fødselsdag.");
        theForm.birthday.focus();
        return (false);
    }
}

if (theForm.birthmonth.value != "")
{
    if ((theForm.birthmonth.value < 1 || theForm.birthmonth.value > 12) && theForm.birthmonth.value != "")
    {
        alert("Ikke en lovlig måned i din fødselsdag.");
    }
}

```



```

        theForm.birthday.focus();
        return (false);
    }
}

if ((theForm.birthday.value != "") && (theForm.birthday.value != ""))
{
    if ((theForm.birthday.value>29) && (theForm.birthday.value == "2"))
    {
        alert("Ikke en lovlig dag i din fødselsdag.");
        theForm.birthday.focus();
        return (false);
    }
}

if ((theForm.birthday.value != "") && (theForm.birthday.value != ""))
{
    if ((theForm.birthday.value>30) && ((theForm.birthday.value == "4") || (theForm.birthday.value == "6") || (theForm.birthday.value == "9") ||
(theForm.birthday.value == "11"))))
    {
        alert("Ikke en lovlig dag i din fødselsdag.");
        theForm.birthday.focus();
        return (false);
    }
}

if ((theForm.birthday.value != "") && (theForm.birthday.value != "") && (theForm.birthday.value != ""))
{
    var today = new Date(); // Capture today's date.
    var thisYear = today.getFullYear() + 1900
    if (thisYear<theForm.birthday.value)
    {
        alert("Ikke en lovlig fødselsdag. Så ung kan du ikke være!");
        theForm.birthday.focus();
        return (false);
    }
}

if (theForm.postalcode.value.length == "")
{
    alert("Husk at udfylde postnummer.");
    theForm.postalcode.focus();
    return (false);
}

if (theForm.postalcode.value.length > 10)
{
    alert("Postnummer må højst være 10 tegn.");
    theForm.postalcode.focus();
    return (false);
}

var checkCK = "1234567890-firFakCK";
var checkStr = theForm.postalcode.value;
var allValid = true;
for (i = 0; i < checkStr.length; i++)
{
    ch = checkStr.charAt(i);
    for (j = 0; j < checkCK.length; j++)
        if (ch == checkCK.charAt(j))
            break;
    if (j == checkCK.length)
    {
        allValid = false;
        break;
    }
}
if (!allValid)
{
    alert("Postnummer må kun indeholde tal og evt. landekode.");
    theForm.postalcode.focus();
    return (false);
}

if (theForm.card_month.value != "")
{
    if ((theForm.card_month.value<1 || theForm.card_month.value>12) && theForm.card_month.value != "")
    {
        alert("Ikke en lovlig måned i kortets udløbsdato.");
        theForm.card_month.focus();
        return (false);
    }
}

if (theForm.card_year.value != "") {
    if (theForm.card_year.value<1900) {
        alert("Ikke et lovligt udløbsår. Du skal angive f.eks. 1968, altså også århundredet!");
        theForm.card_year.focus();
        return (false);
    }
}

return (true);
}

```



```
function why()
{
    document.frmWhy.submit();
}
//-->
</SCRIPT>
<CENTER>

<FORM ACTION="/servlet/login" METHOD="GET" ONSUBMIT="return Foml_Validator(this)" NAME="Foml">

<TABLE BORDER="0" CELLPADDING="2" CELLSPACING="2" WIDTH="277">
    <TR>
        <TD COLSPAN="3" HEIGHT="27">
            <FONT COLOR="#6699CC" SIZE="4" FACE="Verdana, Geneva, Arial, Helvetica">
                <B>REGISTERING</B>
            </FONT></TD>
        <TD>
            <TR>
                <TD COLSPAN="3"><P ALIGN="JUSTIFY">
                    <FONT SIZE="1" FACE="Verdana, Geneva, Arial, Helvetica">
                        <B>Vælg et navn/alias og kodeord til brug i NFS</B>
                    </FONT>
                </TD>
            </TR>
        </TD>
    </TR>
    <TR>
        <TD COLSPAN="4" VALIGN="middle" ALIGN="left">
            <TABLE CELLPADDING="2" CELLSPACING="0">
                <TR><TD>
                    <FONT SIZE="1" FACE="Verdana, Geneva, Helvetica, Arial">NFS</FONT>
                </TD><TD VALIGN="middle">
                    <INPUT TYPE="text" SIZE="20" MAXLENGTH="20" NAME="Alias" VALUE="" STYLE="font-family: Tahoma, Verdana; font-size: 9pt; font-weight: bold">
                </TD></TR>
                <TR>
                    <TD>
                        <FONT SIZE="1" FACE="Verdana, Geneva, Helvetica, Arial">Kodeord</FONT>
                    </TD><TD>
                        <INPUT TYPE="text" SIZE="20" MAXLENGTH="20" NAME="password" VALUE="" STYLE="font-family: Tahoma, Verdana; font-size: 9pt; font-weight: bold">
                    </TD></TR>
                <TR>
                    <TD>
                        <FONT SIZE="1" FACE="Verdana, Geneva, Helvetica, Arial">Fødselsdag</FONT>
                    </TD><TD>
                        <INPUT TYPE="text" SIZE="2" MAXLENGTH="2" NAME="birthday" VALUE="">
                    </TD><TD>
                        <INPUT TYPE="text" SIZE="2" MAXLENGTH="2" NAME="birthmonth" VALUE="">
                    </TD><TD>
                        <INPUT TYPE="text" SIZE="4" MAXLENGTH="4" NAME="birthyear" VALUE="">
                    </TD></TR>
                <TR>
                    <TD>
                        <FONT SIZE="1" FACE="Verdana, Geneva, Helvetica, Arial">Postnummer</FONT>
                    </TD><TD>
                        <INPUT TYPE="text" SIZE="4" MAXLENGTH="10" NAME="postalcode" VALUE="" STYLE="font-family: Tahoma, Verdana; font-size: 9pt; font-weight: bold">
                    </TD></TR>
                <TR>
                    <TD>
                        <FONT SIZE="1" FACE="Verdana, Geneva, Helvetica, Arial">Telefon</FONT>
                    </TD><TD>
                        <INPUT TYPE="text" SIZE="8" MAXLENGTH="10" NAME="phone" VALUE="" STYLE="font-family: Tahoma, Verdana; font-size: 9pt; font-weight: bold">
                    </TD></TR>
                <TR>
                    <TD>
                        <FONT SIZE="1" FACE="Verdana, Geneva, Helvetica, Arial">Køn</FONT>
                    </TD><TD>
                        <SELECT NAME="gender" STYLE="font-family: Tahoma, Verdana; font-size: 9pt; font-weight: bold">
                            <OPTION VALUE=""> Vælg -
                            <OPTION VALUE="0"> Mand
                            <OPTION VALUE="1"> Kvinde
                        </SELECT>
                    </TD></TR>
                <TR>
                    <TD>
                        <FONT SIZE="1" FACE="Verdana, Geneva, Helvetica, Arial">Email</FONT>
                    </TD><TD>
                        <INPUT TYPE="text" SIZE="20" MAXLENGTH="30" NAME="email" VALUE="">
                    </TD></TR>
            </TABLE>
        </TD>
    </TR>
    <TR>
        <TD COLSPAN="3" ALIGN="CENTER">
            <HR SIZE="1" NOSHDE WIDTH="85%" COLOR="#6699CC">
        </TD></TR>
    <TR>
        <TD COLSPAN="3" ALIGN="JUSTIFY">
            <FONT SIZE="1" FACE="Verdana, Geneva, Arial, Helvetica">
                <B>Hvis De ønsker at kunne betale billetter online ved brug af Deres VISA kort, skal følgende yderligere udfyldes:</B>
            </FONT>
        </TD>
    </TR>
    <TR>
        <TD>
            <FONT SIZE="1" FACE="Verdana, Geneva, Helvetica, Arial">Navn på kortet</FONT>
        </TD><TD>
            <INPUT TYPE="text" SIZE="20" MAXLENGTH="30" NAME="card_name" VALUE="">
        </TD>
    </TR>
    <TR>
        <TD>
            <FONT SIZE="1" FACE="Verdana, Geneva, Helvetica, Arial">Nummer på kortet</FONT>
        </TD><TD>
            <INPUT TYPE="text" SIZE="20" MAXLENGTH="30" NAME="card_number" VALUE="">
        </TD>
    </TR>
    <TR>
        <TD>
            <FONT SIZE="1" FACE="Verdana, Geneva, Helvetica, Arial">Udløbsmåned</FONT>
        </TD><TD>
            <INPUT TYPE="text" SIZE="2" MAXLENGTH="2" NAME="card_month" VALUE="">
        </TD>
    </TR>
    <TR>
        <TD>
            <FONT SIZE="1" FACE="Verdana, Geneva, Helvetica, Arial">Udløbsår</FONT>
        </TD><TD>
            <INPUT TYPE="text" SIZE="4" MAXLENGTH="4" NAME="card_year" VALUE="">
        </TD>
    </TR>
</FORM>
```



```

</TR>
</TABLE>
</TD></TR>
<TR>
<TD COLSPAN=3>
<FONT SIZE="1" FACE="Verdana, Geneva, Helvetica, Arial">
</FONT></TD>
</TR>
<TR>
<TD COLSPAN=3>
<CENTER><HR SIZE="1" NOSHADE WIDTH="85%" COLOR="#6699CC"></CENTER>
</TD>
</TR>
<TR ALIGN=CENTER>
<TD COLSPAN=3>
<CENTER>
<TABLE CELLPADDING=0 CELLSPACING=0>
<TR>
<TD><INPUT LANGUAGE="JavaScript" TYPE="button" VALUE="Afbryd" ONCLICK="parent.close()" "></TD>
<TD>&nbsp;<INPUT TYPE="Submit" NAME="registrer" VALUE="Registrer"></TD>
</TR>
</TABLE>
</CENTER>
</TD>
</TR>
</TABLE>
</CENTER>
</FORM>
</BODY>
</HTML>

```



main.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
  <HEAD>
    <TITLE>Nordisk Film Biografer</TITLE>
  </HEAD>
  <!-- Background white, links blue (unvisited), navy (visited), red (active) -->
  <BODY BGCOLOR="#333333" TEXT="#000000" TOPMARGIN="0" LEFTMARGIN="0" MARGINWIDTH=0 MARGINHEIGHT=0 LINK="#000000" VLINK="#000000" ALINK="#000000">
    <IMG SRC="graphics/leftcurve.gif" WIDTH=32 HEIGHT=25 BORDER=0 ALT=""><BR>
    <CENTER><TABLE HEIGHT=90% WIDTH=100%><TR ALIGN=CENTER VALIGN=MIDDLE><TD><IMG SRC="graphics/forside_logo.gif" WIDTH=272 HEIGHT=327 BORDER=0 ALT="Nordisk Film Biografer"></TD></TR></TABLE></CENTER>

    <!-- <IMG SRC="graphics/bear.gif" WIDTH=240 HEIGHT=240 BORDER=0 ALT=""> -->

  </BODY>
</HTML>
```




menu.html

```
<HTML>
<HEAD>
<TITLE>NFB film Navigation</TITLE>
<STYLE>A:link, A:visited, A:active { text-decoration: none } a:hover {color: white;}</STYLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function PB(desktopURL, " _blank", "toolbar=no,location=no,status=no,menubar=no,scrollbars=yes,width=360,height=300,resizable=yes");
}
// -->
</SCRIPT>

</HEAD>

<BODY BGCOLOR="#009933" TEXT="#000000" TOPMARGIN="0" LEFTMARGIN="0" LINK="#000000" VLINK="#000000" ALINK="#FF0000">
<P align="center"><div align="center"><BR>

<FONT FACE="Geneva, Arial" SIZE=1><a href="/" target="_top"></a><HR SIZE="1" WIDTH="80%"
COLOR="Navy" NOSHADE>

<table Align="center" width="35">
<tr>
<td><font face="Tahoma, Geneva, Arial" SIZE="2"><br>
<NBR><nbsp;<a href="javascript:PB('trylogin.html');">Log ind</a></NBR><br>
<NBR><nbsp;<a href="/servlet/logout" target="MainFrame">Log ud</a></NBR><br>
</td>
<td>
<td align="left"><FONT FACE="Tahoma, Geneva, Arial" SIZE="2"><br>
<NBR><nbsp;<a href="/servlet/movieshow?filmID=110&DaysFromNow=2" target="MainFrame">Bestil billetter</a></NBR><br>
<NBR><nbsp;<a href="/servlet/reservations" target="MainFrame">Afbestil billetter</a></NBR><br>
<NBR><nbsp;<a href="/servlet/showorders" target="MainFrame">Bestil billetter</a></NBR><br>
<!-- <nbsp;<a href="na.html" target="MainFrame">Premierer</a><br>-->
</FONT><br>
</td>
</tr>

<!-- <tr>
<td align="left"><FONT FACE="ARIAL, HELVETICA" SIZE="1"><br>
<div><div><a href="na.html" target="MainFrame"><NBR>Ugens Nyt</NBR><br>
</div><div><a href="na.html" target="MainFrame">Debat</a></div><br>
</FONT></td>
</tr>
<tr>
<td align="left"><FONT FACE="Tahoma, Geneva, Arial, Helvetica" SIZE="2"><br>
<NBR><nbsp;<a href="/servlet/movieinfo?filmID=110" target="MainFrame">Film Info</a></NBR><br>
<NBR><nbsp;<a href="/servlet/movieactorinfo?actorID=91&partID=1" target="MainFrame">Skuespiller Info</a></NBR><br>
<!-- <div><div><a href="na.html" target="MainFrame">Tema
Arkiv </div><br>
<div><div><a href="na.html" target="MainFrame">Ugens
Quiz </div><br>
</div></div>
</FONT><br>
</td>
</tr>

<!-- <tr>
<td align="left"><FONT FACE="ARIAL, HELVETICA" SIZE="1"><br>
<div><div><a href="na.html" target="MainFrame">Anmeldelser</a></div><br>
<div><div><a href="na.html" target="MainFrame">Anmeld Selv </a></div><br>
<div><div><a href="na.html" target="MainFrame">Top 10</a>
</div><br>
</div></div>
</FONT></td>
</tr>
<tr>
<td align="center"><HR SIZE="1" WIDTH="100%" COLOR="Navy" NOSHADE><font face="Tahoma, Arial, Helvetica" size="1"><NBR><a href="about.html"
target="MainFrame"><div>Copyrightcopy; 1998</div><NBR><div>Nordisk Film Biografer</div></font></td>
</tr>
</table>

</div>
</font>
</DIV></p>

</BODY>

</HTML>
```



no_card_info.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">

<HTML>
<HEAD>
  <TITLE>NEB – Ingen VISA kort information</TITLE>
  <SCRIPT LANGUAGE="JavaScript">
    function PB(desktopURL, "_blank", "toolbar=no,location=no,status=no,menubar=no,scrollbars=yes,width=360,height=300,resizable=yes");
    }
  </SCRIPT>
</HEAD>

<BODY BGCOLOR="#333333" TEXT="#FFFFFF" LINK="#CC9933" VLINK="#CC9933" ALINK="#CC9933" LEFTMARGIN=0 TOPMARGIN=0 MARGINWIDTH=0 MARGINHEIGHT=0>
<IMG SRC="graphics/leftcurve.gif" WIDTH=32 HEIGHT=25 BORDER=0 ALT=""><CENTER><TABLE WIDTH="98%" BORDER="0" CELLSPACING="0" CELLSPACING="0" ALIGN="CENTER">
<TR>
  <TD VALIGN="TOP" BGCOLOR="#999966"><IMG SRC="graphics/02left.gif" WIDTH=15 HEIGHT=15 BORDER=0 ALT="Left Curve"></TD>
  <TD WIDTH="100%" ALIGN="CENTER" NOWRAP BGCOLOR="#999966"><FONT FACE="Verdana, Helvetica, Geneva, Arial, sans-serif" SIZE="3" COLOR="#FFFFFF"><B>Intet betalingskort fundet</B></FONT></TD>
  <TD ALIGN="RIGHT" VALIGN="TOP" BGCOLOR="#999966" height="33"><IMG SRC="graphics/02right.gif" WIDTH=16 HEIGHT=15 BORDER=0 ALT="Right Curve"></TD>
</TR>
<TR>
  <TD COLSPAN="3" BGCOLOR="#CC9933"><IMG SRC="graphics/pixel.gif" WIDTH=1 HEIGHT=2 BORDER=0 ALT=""></TD>
</TR>
<TR>
  <TD COLSPAN="3" ALIGN="JUSTIFY"><FONT FACE="Verdana, Helvetica, Geneva, Arial, sans-serif" SIZE="3" COLOR="#FFFFFF"><P ALIGN="JUSTIFY">Det var ikke muligt at finde oplysninger om Deres betalingskort. Hvis De ønsker at betale billetter online, skal De indtaste kort-informationer. Af sikkerhedsmæssige grunde, kan der ikke ændres/tilføjes i en nuværende registrering, så hvis De allerede <B>er</B> registreret, skal De oprettes som nyt medlem – <A HREF="Javascript:PB('login.html')">Registrer dig her</A>.</TD></TR>
</TABLE></CENTER>

</BODY>
</HTML>
```



no_login.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">

<HTML>
<HEAD>
  <TITLE>NEB – Brugr mængler at Log In</TITLE>
  <SCRIPT LANGUAGE="JavaScript">
    function PB(desktopURL, "_blank", "toolbar=no,location=no,status=no,menubar=no,scrollbars=yes,width=360,height=300,resizable=yes");
    }
  </SCRIPT>
</HEAD>

<BODY BGCOLOR="#333333" TEXT="#FFFFFF" LINK="#CC9933" VLINK="#CC9933" ALINK="#CC9933" LEFTMARGIN=0 TOPMARGIN=0 MARGINWIDTH=0 MARGINHEIGHT=0>
<IMG SRC="graphics/leftcurve.gif" WIDTH=32 HEIGHT=25 BORDER=0 ALT=""><CENTER><TABLE WIDTH="98%" BORDER="0" CELLSPACING="0" CELLSPACING="0" ALIGN="CENTER">
<TR>
  <TD VALIGN="TOP" BGCOLOR="#999966"><IMG SRC="graphics/02left.gif" WIDTH=15 HEIGHT=15 BORDER=0 ALT="Left Curve"></TD>
  <TD WIDTH="100%" ALIGN="CENTER" NOWRAP BGCOLOR="#999966"><FONT FACE="Verdana, Helvetica, Geneva, Arial, sans-serif" SIZE="3" COLOR="#FFFFFF"><B>Ingen bruger logget ind</B></FONT></TD>
  <TD ALIGN="RIGHT" VALIGN="TOP" BGCOLOR="#999966" height="33"><IMG SRC="graphics/02right.gif" WIDTH=16 HEIGHT=15 BORDER=0 ALT="Right Curve"></TD>
</TR>
<TR>
  <TD COLSPAN="3" BGCOLOR="#CC9933"><IMG SRC="graphics/pixel.gif" WIDTH=1 HEIGHT=2 BORDER=0 ALT=""></TD>
</TR>
<TR>
  <TD COLSPAN="3" ALIGN="JUSTIFY"><FONT FACE="Verdana, Helvetica, Geneva, Arial, sans-serif" SIZE="3" COLOR="#FFFFFF"><P ALIGN="JUSTIFY">De er ikke logget ind på systemet.
  For at kunne bestille billetter eller se dine særligdereservationer, skal du være logget ind i systemet. Vælg <A HREF="JavaScript:PB('trylogin.html')">Log in</A> fra menuen
  til venstre for at registrere dig.</TD></TR>
</TABLE></CENTER>

</BODY>
</HTML>
```



no_user_bottom.html

```
<HTML>
<HEAD>
<TITLE>NB film Navigation</TITLE>
</HEAD>

<BODY BGCOLOR=#000000 TEXT=#FFFFFF" TOPMARGIN="0" LEFTMARGIN="0" LINK="#FFFF00" VLINK="#FFFF00" ALINK="#FF00FF" MARGINWIDTH=0 MARGINHEIGHT=0 TOPMARGIN=0 LEFTMARGIN=0>
<TABLE CELLPADDING=2 CELLSPACING=2 HEIGHT=46>
<TR>
<TD VALIGN=TOP><IMG SRC="graphics/boy.gif" WIDTH=20 HEIGHT=38 BORDER=0 ALT="Boy"><IMG SRC="graphics/girl.gif" WIDTH=20 HEIGHT=38 BORDER=0 ALT="Girl"></TD>
<TD WIDTH=100% ALIGN=JUSTIFY VALIGN=MIDDLE><FONT FACE="Tahoma, Verdana, Geneva, Arial" SIZE=1 COLOR=Red><B>Ingen bruger logget ind</B></FONT></TD>
</TR>
</TABLE>

</BODY>
</HTML>
```



no_user_found.html

```
<HTML>
<HEAD>
  <TITLE>Medlem ikke fundet</TITLE>
  <STYLE TYPE="text/css" TITLE="Input style">
    INPUT { font-size : 10pt;
    font-weight: bold;
    font-family : Verdana, Tahoma, Geneva, Arial;}
  </STYLE>
  <SCRIPT LANGUAGE="JavaScript">
    <!--

function Form_Validator(theForm) {

    if (theForm.alias.value == "") {
        alert ("Husk at udfylde NEB Navn.");
        theForm.alias.focus();
        return (false);
    }

    if (theForm.password.value.length == "") {
        alert ("Husk at udfylde dit kodeord.");
        theForm.password.focus();
        return (false);
    }

    return (true);
}

//-->
</SCRIPT>

</HEAD>

<BODY BGCOLOR="#000000" TEXT="#FFFFFF" LINK="#FFFFFF" VLINK="#FFFFFF" ALINK="#FFFFFF">

<CENTER>
<FORM ACTION="/servlet/trylogin" METHOD="post" ONSUBMIT="return Form_Validator(this)" NAME="Form">

<TABLE WIDTH="277" BORDER="0" CELSPACING="2" CELLPADDING="2" HEIGHT="186">
<TR HEIGHT="21">
<TD WIDTH="267" HEIGHT="21" COLSPAN="2"><B><FONT FACE="Verdana, Geneva, Arial" SIZE="4" COLOR="#6699CC">LOGIN FORKERT</FONT></B></TD>
</TR>

<TR HEIGHT="21">
<TD WIDTH="267" HEIGHT="21" COLSPAN="2"><FONT FACE="Verdana, Geneva, Arial" SIZE="2">Prøv igen, eller registrer dig som ny bruger!</FONT></TD>
</TR>

<TR HEIGHT="21">
<TD WIDTH="59" HEIGHT="21"><FONT FACE="Verdana, Geneva, Arial" SIZE="2">NEBNavn:</FONT></TD>
<TD HEIGHT="21" WIDTH="204" ALIGN="RIGHT"><INPUT TYPE="Text" NAME="alias" SIZE="15" MAXLENGTH="20"></TD>
</TR>

<TR HEIGHT="21">
<TD WIDTH="59" HEIGHT="21"><FONT FACE="Verdana, Geneva, Arial" SIZE="2">Kodeord:</FONT></TD>
<TD HEIGHT="21" WIDTH="204" ALIGN="RIGHT"><INPUT TYPE="password" NAME="password" SIZE="15" MAXLENGTH="20" VALUE=""></TD>
</TR>

<TR>
<TD WIDTH="267" COLSPAN="2"><HR WIDTH="85%" COLOR="#6699CC" NOSHADE SIZE="1"></TD>
</TR>

<TR>
<TD COLSPAN="2" VALIGN="middle"><FONT FACE="Verdana, Geneva, Arial" SIZE="2"><CENTER><B>Ny bruger?</B> <A HREF="login.html">Registrer dig her...</A></CENTER></FONT></TD>
</TR>

<TR>
<TD WIDTH="267" COLSPAN="2"><HR WIDTH="85%" COLOR="#6699CC" NOSHADE SIZE="1"></TD>
</TR>

<TR HEIGHT="26">
<TD WIDTH="267" HEIGHT="26" COLSPAN="2" VALIGN="TOP"><CENTER><INPUT TYPE="button" VALUE="Afbryd" LANGUAGE="JavaScript" ONCLICK="parent.close()">&nbsp;<INPUT TYPE="SUBMIT" VALUE="Videre &#187;"></CENTER></TD>
</TR>

<TR>
<TD WIDTH="267" COLSPAN="2"><HR WIDTH="85%" COLOR="#6699CC" NOSHADE SIZE="1"></TD>
</TR>

</TABLE>
</FORM>
</CENTER>

</BODY>
</HTML>
```



top.html

```
<HTML>
<HEAD>
  <TITLE>NFB film Navigation</TITLE>
</HEAD>

<BODY BGCOLOR=#CC9933 TEXT=#000000 LINK=#000000 VLINK=#000000 ALINK=#FF0000 MARGINWIDTH=0 MARGINHEIGHT=0 TOPMARGIN=0 LEFTMARGIN=0>

  <TABLE WIDTH="98%" BORDER="0" CELLSPACING="4" CELLPADDING="4" ALIGN="RIGHT">
    <TR ALIGN="RIGHT" VALIGN=TOP>
      <TD VALIGN="TOP"><IMG SRC="graphics/nfb_top.gif" WIDTH=403 HEIGHT=43 BORDER=0 ALT="Nordisk Film Biografer"></TD>
    </TR>
  </TABLE>

</BODY>
</HTML>
```



trylogin.html

```
<HTML>
<HEAD>
  <TITLE>Bruger Login</TITLE>
  <STYLE TYPE="text/css" TITLE="Input style">
    INPUT { font-size : 10pt;
    font-weight: bold;
    font-family : Verdana, Tahoma, Geneva, Arial;}
  </STYLE>
  <SCRIPT LANGUAGE="JavaScript">

  <!--
  function Foml_Validator (theForm)
  {

    if (theForm.alias.value == "") {
      alert ("Husk at udfylde NFB Navn.");
      theForm.alias.focus ();
      return (false);
    }

    if (theForm.password.value.length == "") {
      alert ("Husk at udfylde dit kodeord.");
      theForm.password.focus ();
      return (false);
    }

    return (true);
  }

  //-->
</SCRIPT>

</HEAD>

<BODY BGCOLOR="#000000" TEXT="#FFFFFF" LINK="#FFFFFF" VLINK="#FFFFFF" ALINK="#FFFFFF">

  <CENTER>
  <FORM ACTION="/servlet/trylogin" METHOD="post" OnSubmit="return Foml_Validator(this)" NAME="Foml">
  <TABLE BORDER="0" CELLPADDING="2" CELLSPACING="2" WIDTH="277" HEIGHT="186">

    <TR HEIGHT="21">
      <TD WIDTH="267" COLSPAN="2"><FONT FACE="Verdana, Geneva, Helvetica, Arial" SIZE="4" COLOR="#6699CC">BRUGER LOGIN</FONT></TD>
    </TR>

    <TR HEIGHT="21">
      <TD WIDTH="59" HEIGHT="21"><FONT FACE="Verdana, Geneva, Helvetica, Arial" SIZE="2">NFB&nbsp;Navn:</FONT></TD>
      <TD HEIGHT="21" WIDTH="204" ALIGN="RIGHT"><INPUT TYPE="TEXT" NAME="alias" SIZE="15" MAXLENGTH="20"></TD>
    </TR>

    <TR HEIGHT="21">
      <TD WIDTH="59" HEIGHT="21"><FONT FACE="Verdana, Geneva, Helvetica, Arial" SIZE="2">Kodeord:</FONT></TD>
      <TD HEIGHT="21" WIDTH="204" ALIGN="RIGHT"><INPUT TYPE="password" NAME="password" SIZE="15" MAXLENGTH="20"></TD>
    </TR>

    <TR>
      <TD WIDTH="267" COLSPAN="2"><HR WIDTH="85%" COLOR="#6699CC" NOSHADE SIZE="1"></TD>
    </TR>

    <TR>
      <TD COLSPAN="2" VALIGN="middle"><FONT FACE="Verdana, Geneva, Helvetica, Arial" SIZE="2"><CENTER><B>Ny bruger?</B> <A HREF="login.html">Registrer dig her...</A></CENTER></FONT></TD>
    </TR>

    <TR>
      <TD WIDTH="267" COLSPAN="2"><HR WIDTH="85%" COLOR="#6699CC" NOSHADE SIZE="1"></TD>
    </TR>

    <TR HEIGHT="26">
      <TD WIDTH="267" HEIGHT="26" COLSPAN="2" VALIGN="TOP"><CENTER><INPUT TYPE="button" VALUE=" Afbryd " LANGUAGE="JavaScript" OnClick="parent.close()">&nbsp;<INPUT TYPE="SUBMIT" VALUE="Videre ">&#187;"</CENTER></TD>
    </TR>

    <TR>
      <TD WIDTH="267" COLSPAN="2"><HR WIDTH="85%" COLOR="#6699CC" NOSHADE SIZE="1"></TD>
    </TR>

  </TABLE>
  </FORM>
  </CENTER>

  <SCRIPT LANGUAGE="JavaScript">
  <!--
    Foml.alias.focus ();
  //-->
</SCRIPT>

</BODY>
</HTML>
```



Indholdsfortegnelse

BRUGERVEJLEDNING	2
FORSIDEN	2
BRUGER LOGIN	3
LOGIN GENNEMFØRT.....	3
REGISTRERING AF NY BRUGER	4
TAK FOR REGISTRERINGEN	5
BESTIL BILLETTER	6
COMBO-BOKSENE FILM OG DATO.....	7
RESERVERING AF SÆDER	7
SÆDER RESERVERET	8
AFBESTIL BILLETTER	9
BETAL BILLETTER	10
FILM INFO	11
COMBO-BOKSEN FILM INFO	11
SKUESPILLER INFO.....	12
COMBO-BOKSENE SKUESPILLER OG ROLLE	12

Brugervejledning

I det efterfølgende vil de vigtigste skærbilleder fra web applikationen blive gennemgået. Denne brugervejledning henvender sig til en let øvet Internetbruger og vil derfor ikke gå i detaljer med de helt basale ting (hvad et hyperlink er, hvordan en combo-boks virker, etc.), ligesom forskellige begreber ikke vil blive forklaret nærmere.

Forsiden



Skærbillede 1: Forsiden



1. *Log ind:* Hvis man ønsker at have muligheden for at kunne bestille billetter, skal man logge ind ved at klikke på dette link. Herefter følger en række popup skærbilleder som bliver beskrevet senere.
2. *Log ud:* Efter brugeren har bestilt de ønskede pladser, logger han/hun ud af systemet ved et enkelt klik på dette link.
3. *Bestil billetter:* Ved klik på dette link, hoppes til skærbillede 6.
4. *Afbestil billetter:* Ønsker brugeren at afbestille hele eller dele af sin(e) reservation(er), kan dette link benyttes til at komme til skærbillede 9.
5. *Betal billetter:* Ønsker en bruger at betale for sin reservation via fx et VISA kort, skal dette link benyttes, hvorefter skærbillede 10 vil fremkomme.
6. *Film info:* Ved klik på dette link vises skærbillede 11, hvor man har mulighed for at søge informationer om en given film.
7. *Skuepillers info:* Klikkes på dette link, hoppes til skærbillede 12, hvor brugeren har mulighed for at søge informationer om skuespillere og hvilke film de medvirker i.
8. Dette område bruges som bruger login status, hvor man kan se om man er logget på systemet eller ej og om man har mulighed for online-betaling.

Bruger login



Skærbillede 2: Bruger Login

I dette popup skærbillede, kan en bruger logge sig på systemet. Dette gøres ved at indtaste NFB Navn, hvilket er et navn brugeren selv finder på, og derefter indtastes kodeord, som brugeren også selv har fundet på.

Efter endt indtastning kan der vælges at trykke på knappen "**Afbryd**", som afbryder login proceduren, eller at trykke på knappen "**Videre »**", som sender brugerens indtastede informationer videre.

Er brugeren i systemet vil skærbillede 3 fremkomme.

Er brugeren **ikke** registreret i systemet, skal der trykkes på linket "**Registrér dig her...**", som tager brugeren til skærbillede 4.



Login gennemført



Skærbillede 3: Login Gennemført

Her meddeles brugeren, at login er gennemført. Brugerstatuslinien, som beskrevet i skærbillede 1 pkt.8, er nu ændret til at vise, at brugeren er logget på.



Registrering af ny bruger

REGISTRERING

Vælg et navn/alias og kodeord til brug i NFB

NFB Navn:

Kodeord:

Fødselsdag: - -

Postnummer:

Telefon:

Køn:

Email:

Hvis De ønsker at kunne betale billetter online ved brug af Deres VISA kort, skal følgende ydermere udfyldes:

Navn på kortet:

Nummer på kortet:

Udløbsmåned:

Udløbsår:

Skærm billede 4: Registrering

På dette popup skærm billede, kan en ny bruger indtaste de nødvendige informationer, for at kunne få muligheden for online billet-bestilling og -betaling. Her er hvad brugeren skal indtaste:

1. *NFB Navn*: Et navn brugeren selv finder på, og som skal bruges hver gang vedkommende logger sig på systemet fremover.
2. *Kodeord*: En kombination af bogstaver eller tal, som brugeren skal bruge som sit personlige kodeord, for senere at kunne logge sig på.
3. *Fødselsdato*: Fødselsdato skal indtastes ved dag, måned og årstal, fx 20-05-1980 (kunne fx bruges til statistik).
4. *Postnummer*: Det postnummer brugeren bor under skal indtastes her (kunne bruges til statistik)
5. *Telefon*: Brugers telefonnummer skal indtastes i dette felt.
6. *Køn*: I denne combo-boks vælges, om man er mand eller kvinde (kunne bruges til statistik).
7. *Email*: Her skal brugeren indtaste sin email adresse.
8. *Navn på kortet*: Her indtastes kortholders fulde navn.
9. *Nummer på kortet*: Her indtastes kortnummeret på sit betalingskort.



10. *Udløbsmåned*: Her skal kortets udløbsmåned indtastes.

11. *Udløbsår*: Her indtastes kortets udløbsår med 4 cifre.

Efter at have indtastet alle informationerne (pkt. 7-11 er dog valgfrit), kan der nu trykkes på knappen "**Afbryd**" for at afbryde registreringen, eller på knappen "**Registrér**", som gemmer de indtastede oplysninger i databasen. Vælges sidstnævnte knap vil skærbillede 5 fremkomme.

Tak for registreringen



Skærbillede 5: Tak...

Dette popup skærbillede bekræfter, at brugeren nu er registreret i systemet. Der kan nu trykkes på knappen "**Afbryd**" for at lukke vinduet, eller på knappen "**Login**" som bruger de indtastede informationer til at logge på systemet og derved til skærbillede 3.



Bestil billetter



Skærbillede 6: Bestil Billetter

Fra dette skærbillede er det muligt at finde den forestilling, man ønsker at se. Man skal blot vælge film og den dato man ønsker at se den på. Derefter skal man trykke på **søg**, for at finde alle de forestillinger, der vises i det specificerede interval.

Nu skal man blot vælge spilletidspunktet, hvorefter skærbillede 7 vil fremkomme.

1. *Film combo-boksen:* Her kan vælges hvilken film man ønsker at se. Detaljeret billede af denne combo-boks ses på skærbillede 6a.
2. *Dato combo-boksen:* Her kan vælges en dato inden for de næste 7 dage. Et mere detaljeret billede af combo-boksen ses på skærbillede 6b.
3. *Søg:* Ved klik på denne knap, vises alle de forestillinger der findes i det specificerede interval.
4. *Film titel:* Her vises den film man har valgt. Ved klik på filmtitlen, hopper man til skærbillede 11 der giver en uddybende beskrivelse af filmen.
5. *Spilletidspunkter:* Her vises alle tidspunkter den givne film vises på, på den givne dato. Ved klik, hoppes til skærbillede 7 der giver mulighed for reservation.
6. *Brugerstatus:* Her kan man se om man har logget sig på systemet. I det viste tilfælde er brugeren logget ind.
7. *Online betalingsstatus:* Her vises, om brugeren har mulighed for at kunne betale sine reservationer med det samme (dvs. online). Har man under sin brugerregistrering *ikke* angivet de nødvendige informationer som nævnt ved



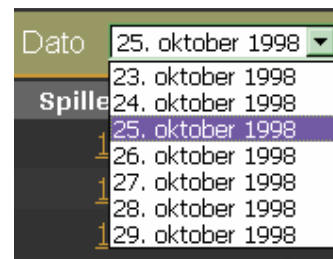
skærm billede 4 pkt. 8-11, er online betaling ikke tilgængelig. I det viste tilfælde, *er* online betaling tilgængeligt.



Combo-boksene Film og Dato

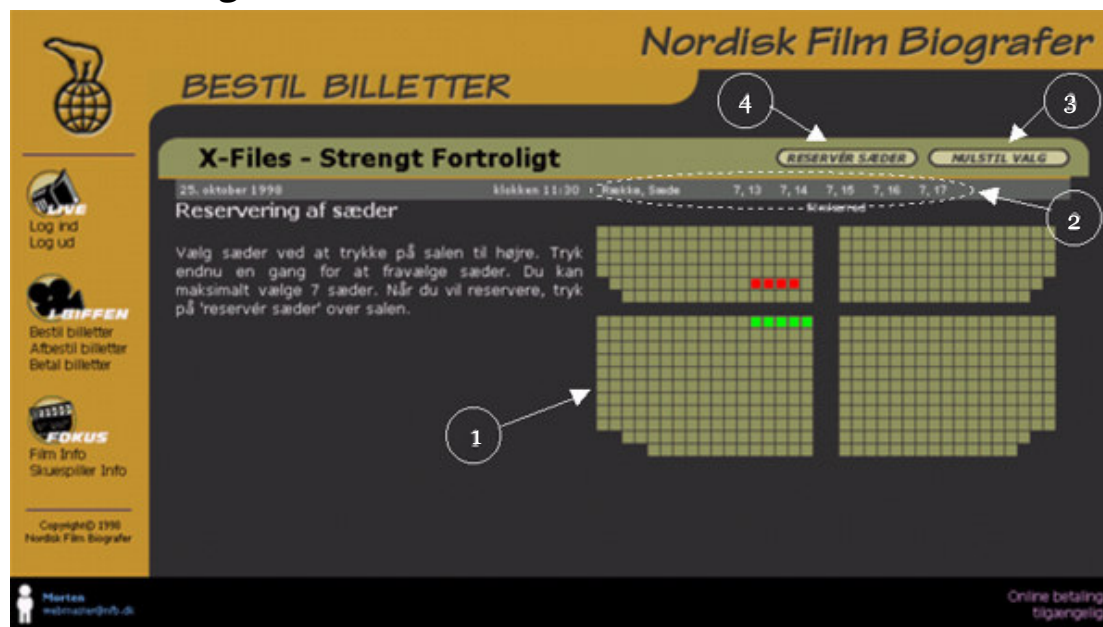


Skærbillede 6a.



Skærbillede 6b.

Reservering af sæder



Skærbillede 7: Reservering af sæder

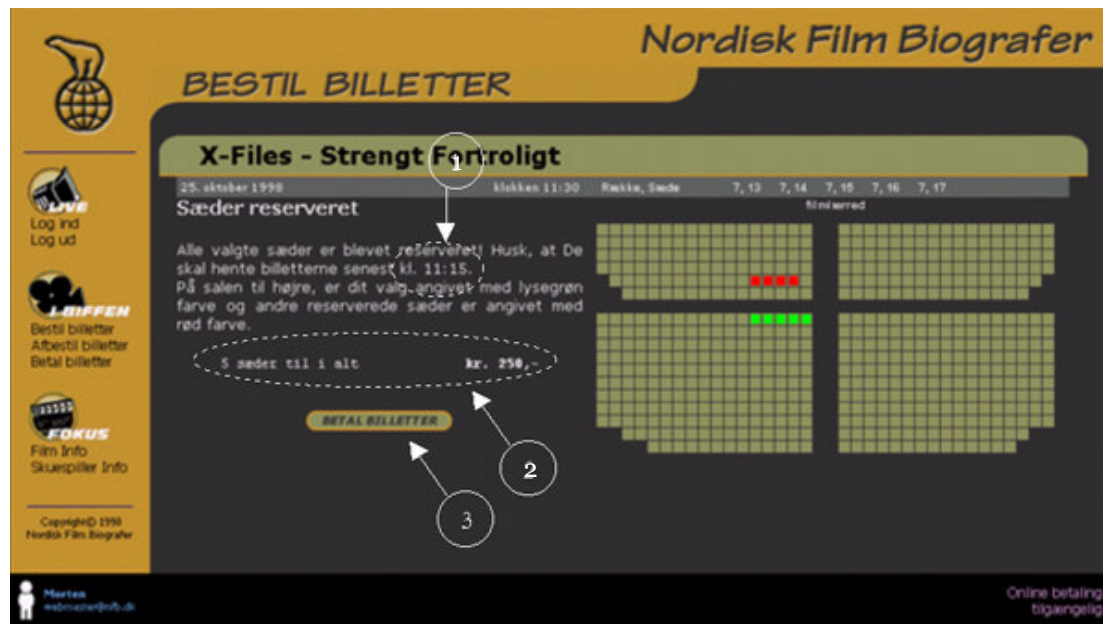
Ud fra dette skærbillede kan man (hvis man er logget ind) nu bestille de sæder man ønsker til den valgte film. Reserverede sæder er vist markeret med rødt og derfor ikke et muligt valg. Sæder der vælges af brugeren markeres med grøn.

1. *Salens layout:* Man vælger de sæder man ønsker, ved at klikke direkte på billedet af salen. Får man klikket på et forkert sæde kan man blot klikke på det igen hvorved det fravælges.
2. *Række, sæde status:* Den hvide cirkel viser det område, hvor man kan se den nøjagtige position af hver enkelt valgt sæde.



3. *Nulstil valg*: Har man valgt alle de sæde man ønsker, men opdager at det fx er på en forkert række, kan man klikke på knappen "**Nulstil valg**", som derefter fravælger alle de valgte sæder.
4. *Reservér sæder*: Når man har valgt de sæder man ønsker, skal man klikke på "**reservér sæder**" for at sende sin bestilling til systemet. Herefter fremkommer skærbillede 8.

Sæder reserveret



Skærbillede 8: Sæder reserveret

Sæderne er nu reserveret. Hvilke sæder det var, ses på billedet til venstre markeret med grønt. Man får oplyst et seneste afhentningstidspunkt samt en samlet pris for alle billetterne.

1. *Afhentningstidspunkt*: Her vises det seneste afhentningstidspunkt for billetterne.
2. *Samlet pris*: Her vises det antal af billetter, der er reserveret samt den samlede pris.
3. *Betal billetter*: Ønsker man at betale de reserverede billetter med det samme, kan man blot klikke på denne knap, hvorefter skærbillede 10 vil fremkomme.



Afbestil billetter

Nordisk Film Biografer

AFBESTIL BILLETTER

Reserverede sæder for Morten

Film Titel	Dato	Tid	Række	Sæde	Betalt	Afbestilling
X-Files - Strengt Fortroligt	25. oktober 1998	11:30	7	13	Ja	afbestil billet
X-Files - Strengt Fortroligt	25. oktober 1998	11:30	7	14	Ja	afbestil billet
X-Files - Strengt Fortroligt	25. oktober 1998	11:30	7	15	Ja	afbestil billet
X-Files - Strengt Fortroligt	25. oktober 1998	11:30	7	16	Ja	afbestil billet
X-Files - Strengt Fortroligt	25. oktober 1998	11:30	7	17	Ja	afbestil billet

1

Copyright © 1998 Nordisk Film Biografer

Online betaling tilgængelig

Skærmbillede 9: Afbestil Billetter

Er man af en eller anden grund nødt til at slette i en allerede foretaget reservation, kan man foretage det på dette skærmbillede. Alle sæderne i den pågældende reservation er listet ved én linie pr. sæde.

1. *Afbestil billet*: Når der klikkes på dette link, fjernes reservationen af det sæde man klikkede ud for.



Betal billetter

Nordisk Film Biografer

BETAL BILLETTER

Ubetalte reserveringer for Morten

Film Titel	Dato	Tid	Pladser	Pris	Valgt
X-Files - Strengt Fortroligt	25. oktober 1998	11:30	5	250	<input checked="" type="checkbox"/>

At betale i alt... kr. 250,- **BETAL**

1. Afkrydsningsfelt (checkbox)
2. Samlede pris (kr. 250,-)
3. Betal knappen (BETAL)

Log ind
Log ud

LOFFEN
Bestil billetter
Afbestil billetter
Betal billetter

FOKUS
Film info
Skuespiller info

Copyright © 1998
Nordisk Film Biografer

Morten
morten@nfb.dk

Online betaling
tilgængelig

Skærbillede 10: Betal Billetter

Såfremt man under sin personlige registrering har indtastet sit kortnummer og tilhørende data, kan man her foretage en egentlig betaling af sin eller sine reservationer .

1. *Afkrydsningsfelt:* Klikker man i den hvide firkant, sættes et såkaldt "flueben", som tilkendegiver at man ønsker at betale denne reservation.
2. *Samlede pris:* Har man reserveret billetter til en eller flere film, kan det samlede beløb ses i dette felt.
3. *Betal knappen:* Ved klik på denne knap udføres betalingen, og man får meddelelse om udfaldet.



Film info



Skærbillede 11: Film Info

Her kan søges informationer om de aktuelle film, med bl.a. en kort synopsis på filmen samt nogle af de medvirkende.

1. *Combo-boksen Film info*: I denne combo-boks findes titlerne på alle de aktuelle film. Et mere detaljeret billede af denne comboboks ses i skærbillede 11a.
2. *Søg knappen*: Ved klik på denne knap, søges efter filminformationer på den valgte film.
3. *Bestil billetter knappen*: Ved klik på denne knap, hoppes til skærbillede 6, hvor alle dagens forestillinger for filmen vises.

Combo-boksen Film info

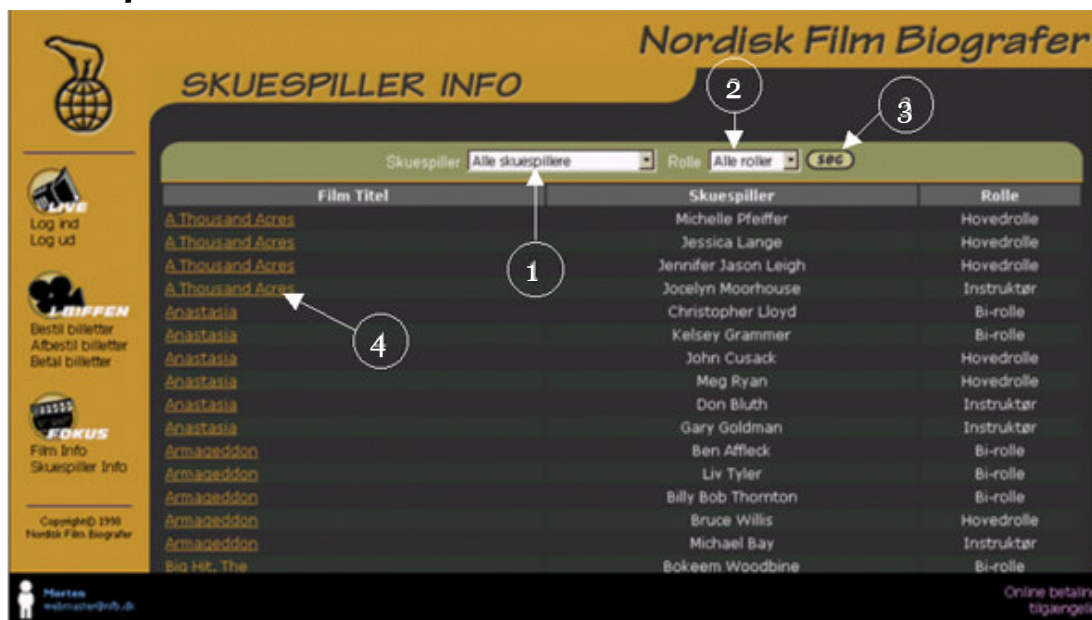


Skærbillede 11a.

I denne combo-boks kan ses alle titlerne på de aktuelle film.



Skuespiller info



Skærbillede 12: Skuespiller Info

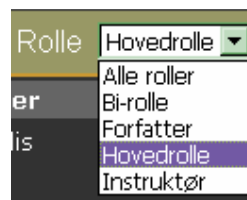
Her kan man bl.a. undersøge hvilke medvirkende der er med i hvilke film og hvilke film en bestemt medvirkende har været med i samt hvilken rolle de havde.

1. *Skuespiller combo-boksen*: Her listes de navne på aktuelle medvirkende. Se et mere detaljeret billede af denne combo-boks i skærbillede 12a.
2. *Rolle combo-boksen*: Her listes hvilke roller man ønsker at søge efter. Se et mere detaljeret billede i skærbillede 12b.
3. *Søg knappen*: Denne kan sætter søgningen i gang med de valgte data i de to combobokse.
4. *Film titel*: Ved klik på en film titel, hopper man til skærbillede 11.

Combo-boksene Skuespiller og Rolle



Skærbillede 12a



Skærbillede 12b



Indholdsfortegnelse

TIDSPLAN	2
REFERENCELINIEMØDER	3
REFERENCELINIEMØDE D. 12/8-98: FORANALYSE.....	3
REFERENCELINIEMØDE D. 21/8-98: ANALYSE.....	3
REFERENCELINIEMØDE D. 11/9-98: DESIGN.....	3
REFERENCELINIEMØDE D. 9/10-98: IMPLEMENTERING	3
REFERENCELINIEMØDE D. 16/10-98: TEST.....	5
REFERENCELINIEMØDE D. 29/10-98: AFSLUTNING	5



Referenceliniermøder

For at inddrage noget kvalitetsstyring i projektet, har gruppen afholdt referenceliniermøder efter hver endt aktivitet. På disse møder er projektets status vurderet således, at det er gjort op om resultatet af hver aktivitet opfylder de krav, som vi selv har sat.

Vi har ikke vedlagt referaterne af de enkelte vejledermøder, da vi ikke mener at disse er så væsentlige som referenceliniermøderne. På de enkelte vejledermøder er forskellige ting blevet vurderet, og hvis de ikke var helt i orden, blev de revideret. Derfor mener gruppen ikke, at disse mødereferater er væsentlige. Referaterne fra referenceliniermøderne giver udtryk for meget mere.

Korte referater af møderne ses nedenfor.

Referenceliniermøde d. 12/8-98: Foranalyse

Projektplanlægningen, den interne kontrakt samt foranalysen er færdig. Problemformuleringen, projektgrundlag samt ekstern kontrakt skal redigeres. Når disse er rettet til, er alle projektdeltagere enige om at projekttableringen er en færdig og afsluttet aktivitet. Alt ovenstående er gennemlæst af projektdeltagerne og vejleder.

De sidste ændringer skal diskuteres på et vejledermøde d. 20/8-98.

Referenceliniermøde d. 21/8-98: Analyse

Byg Model-aktiviteten med undtagelse af adfærdsdiagrammer samt Fastlæg Krav-aktiviteten med undtagelse af skitser af brugergrænseflader er færdiggjort. Begge aktiviteter indeholder de delaktiviteter, som er angivet af Lars Mathiassen. Begge dokumenterede aktiviteter er gennemlæst af projektdeltagerne og vejleder.

De sidste rettelser til analysedokumentet skal diskuteres på et vejledermøde d. 24/8-98.

Referenceliniermøde d. 11/9-98: Design

PDCen og DM Cen er færdige. Der mangler enkelte småting, hvad angår HICen. Design af de tre komponenter overholder alle Coad & Yourdons standard for OOD. Tabledesign er også færdigt.

Endelig godkendelse af designdokumentet skal ske på et vejledermøde d. 16/9-98.

Referenceliniermøde d. 9/10-98: Implementering

Efter gennemgang af referencelinie 4 for implementering har vi fundet ud af, at den ikke holder. Vi kan nu se at vi i projekttableringsaktiviteten ikke havde nok overblik over det fremtidige system til at kunne fastlægge, dels hvilke mellemprodukter som skulle udvikles og dels hvilke vurderingskriterier og – procedurer der skulle til.



Man kan stort set se indholdet for referencelinie 4 (del I - afsnit 6.5) som kasseret, da de daværende overvejelser ikke passer sammen med den måde, systemet er implementeret på og derfor slet ikke kan bruges.

På nuværende tidspunkt kan vi sige, at referencelinie 4 i projektetableringsaktiviteten burde have set således ud:

6.5 Referencelinie 4: Implementering

6.5.1 Mellemprodukter og ønskede tilstande

Databasen

Alle de klasser, som findes i implementationen skal bruge de nødvendige tabeller, og de ubrugte skal være fjernet.

Hjemmesiden

At den ligner EONs design.

6.5.2 Kriterier for vurdering af mellemprodukter

Databasen

Skal opfylde det tabeldesign, som findes i del IV - kapitel 13.

Hjemmesiden

Skal opfylde EONs designudformning (farver, layout, billeder m.v.).

6.5.3 Procedurer for vurdering af mellemprodukter

Databasen

Teknisk gennemgang af gruppen selv.

Hjemmesiden

Teknisk gennemgang af gruppen selv.

Figur I.1

Med hensyn til procedurer for vurdering af mellemprodukter skal der for den tekniske gennemgang af databasen gælde, at koden skal ses igennem, alle de klasser der læses skal være nedskrevet, de attributter som klasserne har, skal være nedskrevet og dem, som ikke bruges skal være slettet.

For teknisk gennemgang af hjemmesiden skal gælde, at den skal overholde HTML 3.2-standard.

Der er styr på, hvilke tabeller som skal slettes samt hvilke attributter, der er nødvendige; fx skal **Række_Sæde** tabellen slettes. Dette er ikke gjort på nuværende tidspunkt, men vil blive gjort indenfor den nærmeste fremtid.

Implementeringen opfylder BATOFF-kriteriet fra del II, men gruppen *har* udvidet og ønsker at udvide med noget mere funktionalitet.

Hjemmesiden opfylder på nuværende tidspunkt EONs designudformning, og implementeringsaktiviteten kan nu anses som afsluttet.



Referencelinimøde d. 16/10-98: Test

Der er udarbejdet et par fejllister mere (tabellerne 25.1 til 25.7). Disse indeholder kun en funden fejl, som kan ses i del VI - tabel 25.7.

Da der kun fandtes én fejl, har det været umuligt at prioritere, som vi ellers havde planlagt at gøre. Testpersonerne var generelt tilfredse med udformningen af vores design, så der var ingen forslag til designændringer.

Referencelinimøde d. 29/10-98: Afslutning

Selvom denne referencelinie er sat til d. 2. November 1998, har vi alligevel været nødsaget til at fremskynde denne, da rapporten printes flere dage før. Men stadig kan man sige, at vi overholder denne referencelinie, da rapporten er samlet, trykt og afleveres kl. 12⁰⁰.



Indholdsfortegnelse

ENDELIGE TABELLER.....	2
------------------------	---

Endelige tabeller

I det følgende ses det endelige tabeldesign for implementeringen.

Kunde

attributnavn	data type	format
<u>pkundeID</u>	autonummer	langt heltal
alias	tekst	50
foedselsdag	tekst	50
foedselsmaaned	tekst	50
foedselsaar	tekst	50
<u>postnr</u>	tekst	4
tlf	tekst	12
koen	tekst	50
email	tekst	50
password	tekst	12
kortholder	tekst	50
kortnummer	tekst	12
kortmaaned	tekst	2
kortaar	tekst	4

Tabel J.1

En kunde får tildelt et *kundeID*, når kunden oprettes i databasen. *Postnr* er fremmednøgle til tabellen **Postnrby**. Ved ikke at have by som attribut, har vi fjernet den redundans det ville give, hvis man skulle indtast kundens by. *Email* og *Password* er vigtigt i denne sammenhæng, da det kræves at en kunde har begge disse, for at kunne lave en reservation fra Internettet. Email adressen kan, yder mere bruges til at sende kunden en ordrebekræftelse. *Kortholder*, *kortnummer*, *kortmaaned* og *kortaar* er alle valgfrie og indeholder informationer om betalingskort.

Billettereservation

attributnavn	data type	format
<u>forestillingsID</u>	tal	langt heltal
<u>raekkeID</u>	tal	langt heltal
<u>saedeID</u>	tal	langt heltal
<u>kundeID</u>	tal	langt heltal
betalt	boolsk	true/false

Tabel J.2

forestillingID, *raekkeID* og *saedeID* angiver tilsammen den unikke nøgle. Når en plads bliver reserveret til en given forestilling, oprettes en ny tupel i tabellen. *kundeID* er ID på den kunde som lagde reservationen og *betalt* angiver om billetten er blevet betalt.

Forestilling



attributnavn	data type	format
<u>forestillingID</u>	autonummer	langt heltal
<u>filmID</u>	tal	langt heltal
<u>salID</u>	tal	langt heltal
dato	tal	byte
tid	dato og klokkeslæt	langt klokkeslætsformat

Tabel J.3

Denne centrale tabel danner relation mellem en reservation og en bestemt film på et bestemt tidspunkt i en bestemt sal. *FilmID* er fremmednøgle til tabellen **Film**. Fremmednøglen *salID* er vigtig, da den binder en bestemt forestilling til en bestemt sal vha. tabellen **Sal**. *Dato* skal angives ved dato-måned og årstal, fx 28-08-1998. *Tid* skal angives med størrelsen timer-minuter, fx 16:30.

Sal

attributnavn	data type	format
<u>salID</u>	tal	langt heltal
map	tekst	50

Tabel J.4

Map angiver stien til en tekstfil på webserverens harddisk, der indeholder informationer om en bestemt sals layout.

Biograf

attributnavn	data type	format
<u>navn</u>	tekst	25
adresse	tekst	50
<u>postnr</u>	tekst	4
tlfnr	tekst	12
faxnr	tekst	12
salantal	tal	3

Tabel J.5

Denne tabel skal indeholde de pågældende stamdata for en biograf. Vi har valgt at bruge attributten *navn* som primærnøgle, da det er yderst sjældent at to biografer hedder det samme. Vores system indeholder kun én biograf, men man ville let kunne tilføje flere biografer. *Postnr* er fremmednøgle til tabellen **Postnrby**. *Salantal* angiver hvor mange sale der findes i biografen.

Billetpris

attributnavn	data type	format
<u>raekkeID</u>	tal	langt heltal
<u>salID</u>	tal	langt heltal
pris	tekst	8

Tabel J.6

Hver række i en given sal har en given pris. *raekkeID* og *salID* angiver tilsammen den unikke nøgle, da det ikke er muligt at have to forskellige priser på samme række i samme sal. *pris* angiver prisen i danske kroner.

Film

attributnavn	data type	format
<u>filmID</u>	autonummer	langt heltal



filmtitel	tekst	50
aargang	tekst	4
spilletid	tal	langt heltal
medvirkendeID	tal	langt heltal
kategoriID	tekst	20
beskrivelse	tekst	1000

Tabel J.7

I denne tabel skrives en films stamdata. *MedvirkendeID* er fremmednøgle til tabellen **Medvirkende**, da der som regel er flere medvirkende i en film. *KategoriID* er fremmednøgle til tabellen **Kategori**. Dette er desuden også for at undgå redundans.

Kategori

attributnavn	data type	format
⌚kategoriID	tekst	20

Tabel J.8

I denne tabel ligger alle de forskellige filmkategorier, fx komedie, action, drama osv.

Medvirkende

attributnavn	data type	format
⌚medvirkendeID	autonummer	langt heltal
navnID	tal	langt heltal
rolleID	tal	langt heltal

Tabel J.9

Denne tabel er skabt for at undgå redundans. *NavnID* er fremmednøgle til tabellen **Nanveliste**, som har til formål at skabe en relation til de rigtige navne, til en bestemt film. *RolleID* slår op i tabellen **Rolle**, for at skabe relationen til de roller en medvirkende har i en bestemt film

FilmMedvirkende

attributnavn	data type	format
⌚filmID	tal	langt heltal
⌚medvirkendeID	tal	langt heltal

Tabel J.10

Denne tabel skaber mange-til-mange relation mellem tabellerne **Film** og **Medvirkende**. Den indholder begge tabellers unikke ID.

Navnliste

attributnavn	data type	format
⌚navnID	autonummer	langt heltal
navn	tekst	75

Tabel J.11

I denne tabel er listet alle navnene på alle implicerede i en film, da en impliceret kan medvirke i flere film. Dette vil mindske redundansen betydeligt ved at oprette denne specielle navnliste.

Rolle



attributnavn	data type	format
rolleID	autonummer	langt heltal
rolle	tekst	30

Tabel J.12

I denne tabel er listet alle de former for roller der findes, fx skuespillere, instruktører og forfattere.