

GPP

Eksamensprojekt 2013

Flybooking

Udarbejdet af:

Anders Wind Steffensen (awis@itu.dk)
Christoffer Forup (csdf@itu.dk)
Anders Fischer-Nielsen (afn@itu.dk)

Indholdsfortegnelse

Indledning	3
Problemformulering	4
Problemstillingen	4
Reservation	4
Søgning	4
Redigering	4
Problemanalyse	5
Opstart	5
Brugeropgaver/User Tasks	5
Brugerworkflow	6
Brugergrænsefladen	6
Model-View-Controller	7
Abstraktion og modulation	8
Databaseopstilling	8
Singleton design-pattern	8
Iterativ process	9
Brugervejledning og eksempel	10
New Reservation	11
Edit Passengers	13
Confirm Booking	15
Edit Reservation	16
Eksempel 1	18
Teknisk beskrivelse af programmet	21
Systemets klasser	21
Klasser til brugergrænsefladen	22
Anvendelse af Interfaces	23
Nedarvning	23
Repræsentation og implementering af lister	24
Databasesøgning	24
Oprettelse af objekter	24
SædeID generator	25
Statisk Converter-klasse	25
Klikke på sæder	25
Singleton design-pattern	26
Brugergrænsefladen	26
Tests	27
Nødvendigheden af tests	27
Anvendelse af testdatabase	27
Debugging	27
Videreudvikling	28
Anvendelse af Factory method-pattern	28
Central View-Controller	28
Sortering af afgange	28
Nedskæring af DatabaseInterface	28
Konklusion	29
Litteraturliste	30
Litteratur	30
Hjemmesider	30
Bilag	31
Bilag 1	31
Bilag 2	32
Bilag 3	33
Bilag 4	34
Bilag 5	35
Bilag 6	37

Indledning

Denne projektrapport omhandler det endelige eksamensprojekt i kurset "Grundlæggende Programmering" fra d. 21/11 til d. 16/12, med vejledning af Tobias Grundtvig og Dan Witzner Hansen.

Formålet for denne rapport er at beskrive problemanalysen af den givne case, løsningen, designet samt implementeringen af denne er udarbejdet af Anders Wind Steffensen, Christoffer Forup og Anders Fischer-Nielsen. Derudover følger en kort beskrivelse af arbejdsprocessen under projektet.

Målet med projektet var at udarbejde et program, hvor formålet var at styre flyreservationer for et mindre flyselskab. Programmet henter tilgængelige afgange fra en MySQL-database og viser disse i en brugergrænseflade, der gør brugeren i stand til at finde afgange, reservere sæder og booke samt redigere rejser. Den færdige løsning er udviklet i IDE'en NetBeans med en database-backend på ITUs server og versionsstyring på GitHub ved hjælp af git.

Tak til MiG InfoCom for brugen af deres layoutmanager Mig Layout, der gjorde implementering af brugergrænseflade en del nemmere¹. Tak til GitHub for at kunne anvende deres services, samt en gratis opgradering af (en normalt betalingspåkrævet) service.

Rapporten medfølges af en DVD med det færdige system som en .jar Java-applikation, kildekoden for systemet, rapporten i pdf-format, samt dokumentation for implementeringen i form af JavaDoc-dokumentation.

Rapporten vil følgende gennemgå problemanalyse, brugervejledning, teknisk beskrivelse, tests og videreudvikling af programmet, og afsluttende en konklusion af rapporten.

I begyndelsen af rapportens sektioner er en kort opsummering af formålet med sektionen, så læseren kan danne sig et hurtigt overblik over denne.

¹ MiG InfoCom Ab - <http://www.miglayout.com>

Problemformulering

Følgende afsnit vil beskrive uddybe og begrænse omfanget af problemstillingen i den givne case. Derudover vil de givne brugerkrav for systemet blive beskrevet. Endelig vil systemets opbygning samt funktioner blive gennemgået.

Problemstillingen

Der ønskes et system der kan oprette og ændre reservationer for et lille flyselskab. Systemet skal kunne give brugeren overblik over tilgængelige afgange, destinationer og tidspunkter og kunne oprette en reservation ud fra disse.

Systemet skal indeholde forskellige flytyper og -afgange, herunder destinationer og tidspunkter. Kunder skal være i stand til at kunne reservere sæder til bestemte afgange eller tidsrum. En ansat skal derfor være i stand til, vha. grafiske brugergrænseflader, at opfylde disse krav. Projektets krav forudsætter, at programmet kun skal kunne køre på én enkelt maskine, og derved ikke behøver tage hensyn til samtidigheds-problemer, applets hos fjernbrugere osv..

Casen lægger op til følgende problemstillinger:

Reservation

- Reservere sæder til en afgang mellem to destinationer.
- Reservere sæder på et specifikt fly.
- Reservere sæder sammen med en specifik person.
- Reservere sæder til en afgang på et specifikt tidspunkt.
- Alle ovenstående, men ved siden af medrejsende.

Søgning

- Søge efter rejse ud fra destination(er), flynummer, person (navn) og tidspunkt.
- Søge efter ledige rejser.

Redigering

Følgende anvender alle ovenstående søge/oprettelsesopgaver, og det skal derefter være muligt at:

- Tilføje/fjerne personer til en eksisterende rejse.
- Ændre sæde til et andet sæde på samme fly.
- Slette en hel reservation.

Problemanalyse

Herunder beskrives de første tanker teamet gjorde sig om problemløsningens udseende, hvilke implementations- og designmæssige løsninger der kunne anvendes, samt en kort begrundelse for valget af disse løsninger.

Opstart

Arbejdet begyndte med at få et overblik over datastrukturen i systemet og opbygningen af databasen ved at lave et ER-diagram (se bilag 2). Herved fremkom en strukturering af data i individuelle klasser for henholdsvis reservationer, rejser, fly, sæder og passagerer.

Ved at gøre dette var det muligt at danne et overblik over hvilke variable de forskellige entiteter skulle indeholde.

Ved hjælp af dette ER-diagram er det muligt at holde objekterne separat, så en reservation ikke skal "bekymre" sig om andet end en rejse. Dermed kender en reservation f.eks. ikke til sæder i et givent fly, da flyet selv holder styr på disse. Dette sikrer en højere indkapsling.

Det blev også overvejet, hvordan samarbejdet mellem databasen, brugergrænsefladen og selve Java-systemet skulle fungere. Indarbejdning af interfaces var på tale fra start for at sørge for lavere kobling mellem forskellige dele af systemet.

Brugeropgaver/User Tasks

Herefter blev brugeropgaver gennemgået og programmets funktionalitet blev udarbejdet ud fra disse. Ifølge casen er en sekretær for det imaginære flyselskab hovedbrugeren. Dette gav mere frihed ved kontrol af brugerinput, da sekretæren ikke ville have interesse for at ødelægge eller oprette forkert information med vilje.

Derefter blev de mest normale brugerscenarier, der kunne forekomme for et flyselskab gennemgået, de opgaver der mindede om hinanden blev slæt sammen, så det kunne ses, hvilke opgaver brugeren kunne løse samtidig.

Nedenstående tabel viser et overblik over programmets hovedfunktioner:

Ny reservation	Ændre reservation
Find/Tilføj rejse ud fra tid, destination	Find booking ud fra - cpr, bookingnr, destination, tidspunkt)
Tilføj info - book sæder - person info	Rediger booking
Udskriv booking	Slet/tilføj personer
Gem booking	Ændre data - sæder
	Udskriv booking
	Gem ændringer
	Slet hele bookingen

Brugerworkflow

Valget endte på et minimalistisk workflow, hvilket gjorde, at brugeren kunne gennemføre sine opgaver på mindst mulig tid. Da brugeren ifølge problemstillingen er én receptionist, der skal oprette og redigere mange reservationer, er det hurtigt og nemt at kunne udføre den givne opgave.

Ved at gennemgå brugeropgaverne, var det muligt at opbygge et workflow, for hvor og hvornår funktionalitet skulle være til rådighed. Dette gjorde design samt implementering af brugergrænsefladen lettere, så der opstod et optimalt flow gennem systemet.

Et modifieret “wizard”-baseret workflow blev brugt som base, dog med todeling af workflowet med en opret-reservations-del og en rediger-eksisterende-reservations-del.

Et vindue med en grafisk repræsentation til valg af sæder, samt knapper til at tilføje og fjerne personer var også nødvendigt, da denne ville være lettere at overskue end ren tekst. Dette vindue ville blive brugt både for en ny reservation og for en redigering.

Et femte vindue med bekræftelse for en gennemført booking og information om reservationen er blevet tilføjet for at give et bedre overblik, da dette er nødvendigt ved redigering af reservationen senere. Hermed er brugeren ikke i tvivl om hvorvidt forkert data er oprettet.

Ved at sende brugeren forbi person-og-sædevinduet hver gang, er det muligt at genbruge koden bag, og gøre indlæring af funktionaliteten lettere. (Illustration for dette genbrug er vist på bilag 4.1).

Brugergrænsefladen

Der blev enighed om, at lave simple mock-ups af brugergrænsefladen, så det var muligt at få en generel idé om hvilken visuel retning designet skulle bevæge sig. Dette sikrede sammenhæng gennem workflowet, hvilket ville mindske eventuel brugerforvirring, samt gøre det lettere at programmere da både implementation og design, ikke skulle overvejes midt i udviklingsprocessen.

Baseret på brugeropgaverne ville drop-downs med alle destinationerne fungere til valg af afgangs- og ankomstdestinationer. Tilgængelige destinationer skulle hentes ind fra databasen. Drop-downs blev valgt, da disse er visuelt kompakte, hurtigt viser hvad indholdet er, og ved museklik giver brugeren alt information på en struktureret og overskuelig måde.

Tekstfelter blev også overvejet, men disse har ikke fornævnte fordele, da det kan være besværligt for brugeren at vide hvordan og hvilke informationer der kan skrives i feltet, og det er muligt, at brugeren skal anvende mange forsøg, før et matchende sæt af afgange og destinationer findes. Ved dato-, navne-, CPR- og adresseindtastning endte valget på tekstfelter da en drop-down-menu ikke ville kunne indeholde alle de forskellige muligheder, som en bruger kan skrive.

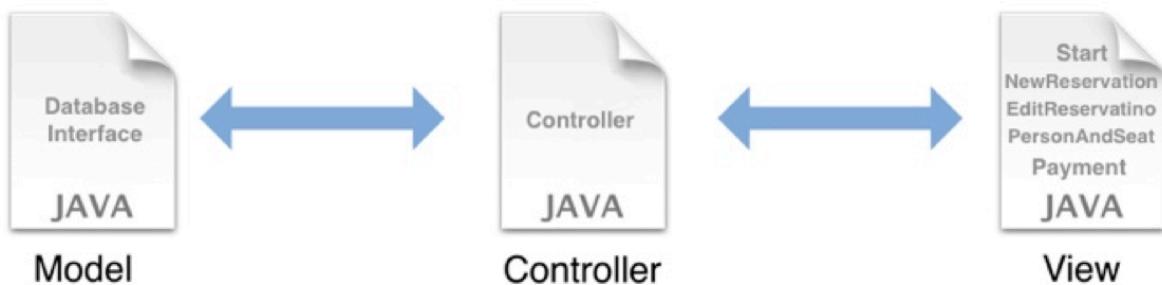
Inddeling af workflowet i flere individuelle vinduer blev baseret på, at det skulle være nemt for brugeren at fortryde midtvejs i workflowet og det skulle være nemt at se hvor i programmet han/hun befandt sig, ved at kunne overskue hele brugerfladen på én gang. En rød tråd gennem brugergrænsefladen skulle dog opretholdes, så skiftene mellem vinduer ikke ødelagde brugeren opmærksomhed fra det igangværende workflow.

Ved at anvende gestaltlovene tidligt i projektet, er det muligt at signalere hvilke elementer, der har sammenhæng. De tidlige brugergrænseflade-mockups viser anvendelsen af 16/12/2013

gestaltlovene, med vindueselementerne sat tæt sammen for eksempelvis valg af destinationer. Der er også anvendt whitespace i layoutet, der gør brugergrænsefladen overskuelig og fremmer kontrasten mellem grupperede og ikke-grupperede elementer.

Det blev besluttet at optage en stor del af vinduespladsen på listen med søgningsresultater, hvorved opmærksomheden hurtigt trækkes hen mod disse. Derudover kan der vises nok informationer om resultaterne, så disse ikke virker uoverskuelige. Hermed overholdes kravet om et hurtigt og simpelt workflow.

Der er herudover også fokuseret på programmets brugerfladedesign, så systemet hjælper brugeren til at danne en korrekt Mental Model. På denne måde ved brugeren, hvad der foregår i programmet, hvad der vil ske når brugeren trykker på en bestemt funktion, og at denne udfører hvad der forventes af den. Dette er igen for at undgå brugerfejl og -forvirring.



Model-View-Controller

Efter disse overvejelser kom en model frem, hvor man ved hjælp af interfaces holder databasen adskilt fra en central controller-klasse, der skulle styre oprettelse af objekter i systemet, modtage og videregive information fra disse objekter og databasen, samt videresende dette til brugergrænsefladen (se bilag - første version på 1.1, ses i detaljer på bilag 2.1).

Dette viste sig i løbet af processen at være et Model-View-Controller design-pattern. Denne metode sikrer lav kobling i programmet og kan derfor være meget effektiv.

Ved MVC ændres brugergrænsefladen ud fra data i en model ved hjælp af en controller-klasse. Data ændres også via brugergrænsefladen, igen med en Controller-klasse. Controlleren håndterer altså al input fra model og brugergrænseflade og videresender denne til resten af systemet.

Den færdige implementation minder til en vis grad om dette, dog er der stadig kommet ufordelagtige koblinger mellem GUI'en og modellen. Denne kobling uddybes i afsnittet under videreudvikling sidst i rapporten.

Abstraktion og modulation

Fra begyndelse af projektet var målet, at en klasse kun har ét formål, og hver metode gerne kun udfører én eller meget få opgaver - såkaldt responsibility-driven design.

Kald af Controller-klassens søgeretoder bliver derfor sendt videre til databasens søgeretoder. Herved videresender Controller-klassen kun information, og søger ikke direkte

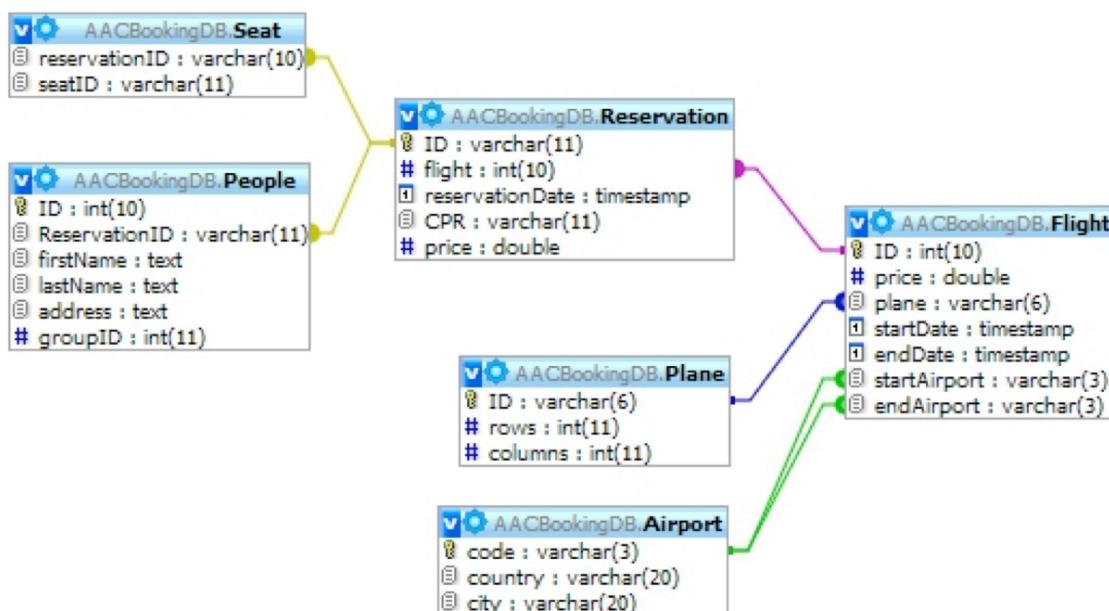
selv. Controllerens søgemetoder har ét ansvar, og udfører kun dette. Databasens søgemetoder har også ét ansvar, og søger kun med henblik på dette. Brugergrænsefladen er derudover også opdelt i flere vinduer, med hver deres ansvar og opgave.

Dette gør det lettere at indkapsle information i de enkelte klasser, da de kun bearbejder information vedrørende dem selv og deres ansvar. Dette fremmer også lav kobling.

Databaseopstilling

I databasen var separering af individuelle tabeller og oprettelse af tabeller entiteterne i ER-diagrammet et krav. Så kan de forskellige unikke keys trækkes over fra tabellerne, hvor de er nødvendige. En entry i Reservation-tabellen holder f.eks. henvisninger til et Flight, der indeholder en unik reference til et Plane, og to unikke referencer til to Airports. Dermed er al "indre" data i hver sin tabel, og kun henvisningen trækkes over mellem tabeller.

Unikke keys til alle entries i de forskellige tabeller i databasen er derfor nødvendige, da søgning ellers ville blive uhensigtsmæssigt kompliceret. Den endelige databases relations diagram kan ses nedenfor.



Separationen af database fra controller tidligt i forløbet gav også mulighed for at indsætte en lokal testdatabase, indtil basisfunktionaliteten af systemet var på plads. Herefter kunne den endelige SQL-database implementeres. Det fungerende system blev derfor hurtigere færdigt, og derefter kunne der fokuseres på sværere opgaver.

Singleton design-pattern

Anvendelse af Singleton design-pattern sikrer, at der kun er én database med én controller i spil på et hvilket som helst givet tidspunkt. Dermed undgås duplikerede reservationer og modstridende overskrivninger i databasen, hvis der f.eks. er flere vinduer åbne på én gang.

Søgning og sikkerhedsovervejelser

Under implementering af søgefunktionaliteten var målet at søgningen var så effektiv som muligt. Derfor skulle al indeksering og søgning holdes i MySQL-databasen, da denne er indekseret, og derfor ville finde givne reservationer mere effektivt end Java-siden af systemet.

Dette betød dog, at systemet blev afhængig af korrekt brugerindtastning eller at systemet skulle forudsætte, at SQL-injection ikke var muligt.

SQL-injection ville gøre det muligt ved f.eks. at skrive “DROP TABLE Reservation”, og herefter ville reservationstabellen blive slettet. Da det er en ansat i flyselskabet, der skal indtaste data, blev det set som en mindre risici, og sikring mod dette blev erklæret som en ekstra feature, der kan implementeres i videreudvikling af systemet. Implementation af sikkerhedstjek i brugerindtastningen ville være uhensigtsmæssigt tidskrævende og kompliceret at implementere, taget mængden af tid for projektet i betragtning.

Iterativ process

En hurtig implementering af et skelet af basisfunktionalitet var en prioritet. Herefter var der enighed om at implementere den mere avancerede funktionalitet på en “as-needed”-basis. Derfor er brugergrænsefladen, Controller-implementation samt database-implementation lavet iterativt indtil den aftalte deadline for code-freeze. Dette betød hvis et problem blev spottet, blev dette tilføjet som et såkaldt Issue på GitHub. Disse mangler eller rettelser blev herefter gennemgået, indtil listen var tom og processen begyndte forfra med nye Issues. Systemet har derfor gennemgået optimerende ændringer undervejs, da det endelige resultat aldrig var helt fastlagt.

Det var et naturligt valg at anvende Git, da arbejdsprocessen var iterativ. Herved ville der hele tiden blive tilføjet funktionalitet og lavet ændringer i flere forskellige filer.

Git håndterer såkaldte merge-konflikter, og muliggør at flere medlemmer i gruppen kan redigere i samme klasse samtidig, hvilket passede til arbejdsformen.

Brugervejledning og eksempel

Dette afsnit gennemgår en brugervejledning af systemet. Brugervinuerne samt dets funktioner vil blive uddybet, og endeligt vil der gives et eksempel, som læseren kan følge i programmet selv (ønskes der et redigerings-eksempel, henvises der til bilag 5).

Når programmet eksekveres, præsenteres brugeren for Flight Booking-vinduet. Her præsenteres brugeren for to valgmuligheder.

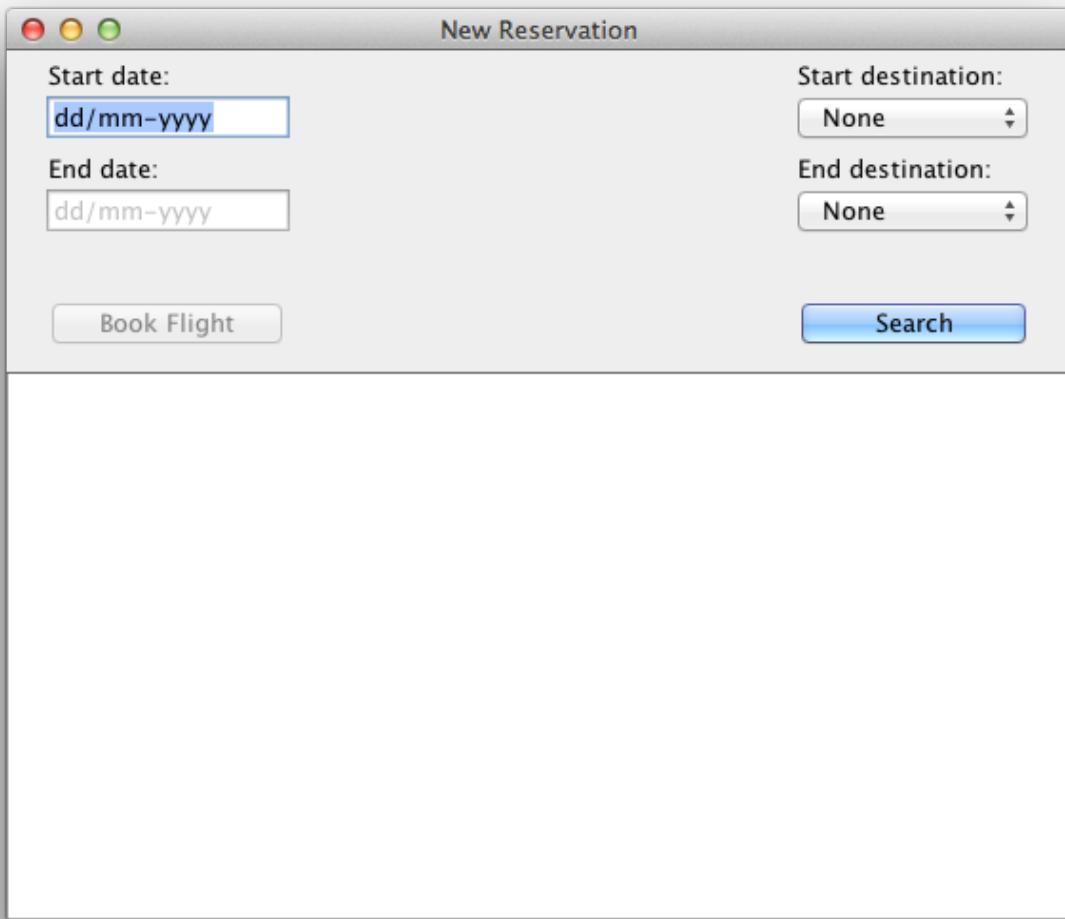
- 1. New Reservation**
- 2. Edit Reservation**

Vælger brugeren at trykke på knappen New Reservation, åbnes et nyt brugervindue - **New Reservation**.



New Reservation

I dette vindue skal brugeren søge efter en rejse som kunden ønsker og dette kan gøres ud fra en start- og slutdato, start- og slutdestination. Den øverste halvdel af vinduet indeholder søgerriterierne og den nederste halvdel af vinduet har en liste med de rejser, der overholder søgerriterierne.



Start date - End date:

I start- og end date-tekstfelteterne kan brugeren indtaste to datoer i formatet dd/mm/yyyy, hvor dd står for dato, mm står for måneden og yyyy står for årstalet. Søgningen finder da alle tilgængelige rejser med afgang mellem de givne datoer. Indtaster brugeren teksten forkert, eller udfylder denne ikke begge felter, bliver felterne blot set bort fra i søgningen.

Start destination:

Her kan brugeren, ud fra en drop-down menu, vælge en af de tilgængelige lufthavne for rejsens ønskede startdestination. Vælges None søges der ikke efter startdestination.

End destination:

Her kan brugeren, ud fra en drop-down menu, vælge en af de tilgængelige lufthavne for rejsens ønskede slutdestination. Vælges None søges der ikke efter slutdestination.

Search:

Denne knap finder alle de rejser, der ligger mellem start date og end date samt flyver mellem start- og end destination. Herefter vises resultaterne på listen over rejser i nedre del af vinduet.

Book flight:

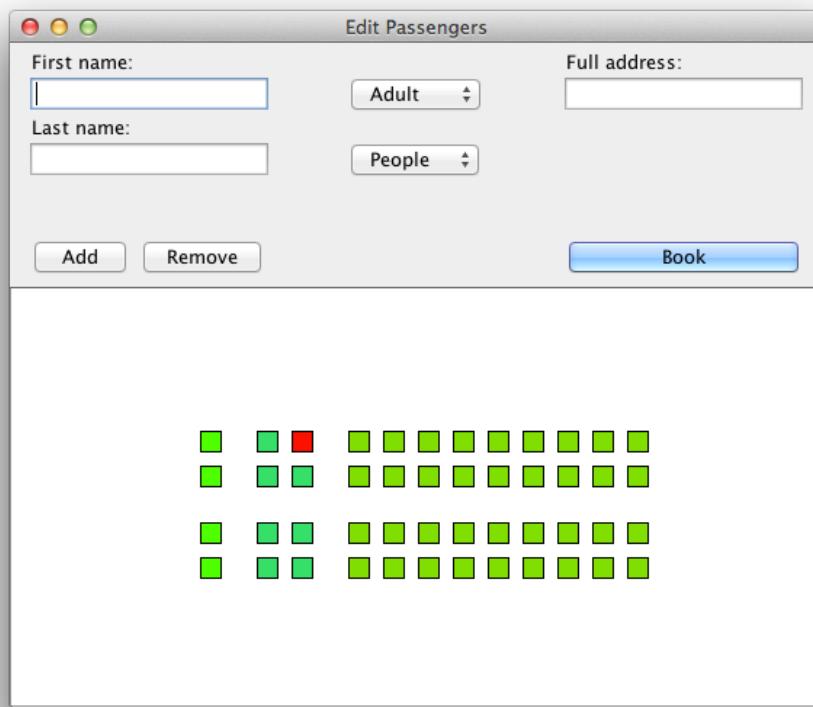
Denne knap tager den markerede rejse på listen og sender brugeren videre til næste vindue "Edit Passengers". Hvis ingen rejse er valgt, er denne knap gjort inaktiv.

New Reservation

Start date:	dd/mm/yyyy	Start destination:	Rønne
End date:	dd/mm/yyyy	End destination:	Timbuktu
<input type="button" value="Book Flight"/>		<input type="button" value="Search"/>	
11:00	Rønne – RØN > Timbuktu – TOM	H92837	
16:00	17/12-2012 to 17/12-2012	pp. 4000.00 DKK	
12:00	Rønne – RØN > Timbuktu – TOM	L02938	
17:00	18/12-2012 to 18/12-2012	pp. 4000.00 DKK	
08:00	Rønne – RØN > Timbuktu – TOM	S12947	
13:00	19/12-2012 to 19/12-2012	pp. 4000.00 DKK	
13:00	Rønne – RØN > Timbuktu – TOM	L02938	
18:00	20/12-2012 to 20/12-2012	pp. 4000.00 DKK	

Edit Passengers

Booking-vinduet er en af de væsentligste brugergrænseflader. Det er i dette vindue, at brugeren opretter passagerer på reservationen, samt vælger deres siddepladser i flyet. Den øvre halvdel af vinduet indeholder knapper og tekstmærker vedrørende passagererne, og den nederste halvdel omhandler reservering af sæder.



First name:

Tekstfelt hvor brugeren indtaster køberens fornavn.

Last name:

Tekstfelt hvor brugeren indtaster køberens efternavn.

Address:

Tekstfelt hvor brugeren indtaster køberens fulde adresse.

"Adult", "Child" eller "Elderly":

I denne funktion vælger brugeren om køberen er et barn, voksen eller ældre. Vælges Child eller Elderly får disse en rabat på prisen.

Person:

Denne drop-down-menu viser en liste over tilføjede personer der p.t. er i reservationen, og det valgte element er den nuværende aktive person.

Add/Save:

Denne funktion tilføjer en person til programmet, eller gemmer informationerne om den nuværende aktive person. Hvis der er nogle af tekstfeltene der ikke er udfyldt, bliver disse udfyldt med en fejlmeldelse, der fortæller brugeren, at de skal udfyldes.

Delete:

Denne funktion sletter den aktive person fra reservationen.

Book:

Denne knap sender den givne information videre til **Confirm Booking**-vinduet.

I den nedre halvdel af vinduet bliver brugeren præsenteret for en grafisk repræsentation af flyets sæder på rejsen. Hver farvefirkant repræsenterer ét sæde. Et sæde vælges ved at klikke på dets firkant.

Sæderne indeholder tre forskellige farvekoder, der viser deres tilgængelighed:

- **Grønne nuancer** indikerer, at sædet er ledigt og derfor kan reserveres. Farve nuancen og placeringen bestemmer hvilken klasse sædet er, first- business eller economy class. Dette er op til træningen at lære brugeren. Vælges first- eller business class tillægges ekstra pris på billetten.
- **Rød** indikerer at sædet er optaget og derfor ikke kan reserveres.
- **Blå** indikerer sædet er booket som en del af den aktuelle reservationen, altså fx hvis brugeren har klikket på sædet allerede.

Når brugeren har valgt køberens siddepladser i illustrationen trykkes herefter på knappen "Book" i vinduets øvre halvdel. Hvis brugeren ikke har booket samme antal sæder som personer, bliver brugeren bedt om at gøre dette via en popup-besked. Ellers bliver brugeren sendt videre til næste vindue - **Confirm Booking**.

Confirm Booking

Dette vindue viser en oversigt over køberens rejse detaljer, ligeledes tildeles køberen her sit reservations-ID, der kan bruges til at søge efter reservationen senere. Uover køberens reservations-ID indeholder dette vindue to væsentlige funktioner.

Payers CPR:

I dette tekstfelt skal brugeren indtaste køberens CPR nummer. Dette kan senere bruges, hvis køberen ønsker at ændre/slette sin rejse. Hvis et CPR der ikke er 10 eller 11 tegn langt, indsættes en fejlmeddeelse i feltet, når knappen "Confirm Booking" bliver trykket på.

Confirm Booking:

Denne funktion gemmer den endelige booking. Når denne funktion trykkes, gemmer det køberens rejse i systemets database, og brugeren bliver sendt tilbage til Flight Booking-vinduet.

The screenshot shows a window titled 'Confirm Booking'. At the top, there are two input fields: 'People in the reservation:' with the value '1' and 'Payers CPR:' with the value '1234567890'. Below these, a dashed line separates the input fields from the flight details. The details are listed as follows:

Reservation ID:	1279
Departure:	17/12/2012 11:00:00 RØN
Arrival:	17/12/2012 16:00:00 TOM
Passengers:	
Anders Andersen (Adult)	
Seats:	4A
Plane:	H92837
Price:	4000.0DKK

At the bottom of the window is a blue button labeled 'Confirm Booking'.

Edit Reservation

Ønsker brugeren at redigere en eksisterende reservation vælges "**Edit Reservation**" i systemets startvindue. Dette bringer vinduet "**Edit Reservation**" op.

Her kan brugeren indtaste i enten tekstfeltet "**Reservation ID:**" eller "**CPR #:**", der er nødvendige oplysninger for at finde en bestemt rejse frem ud fra en eksisterende reservation.

Endvidere kan der søges efter reservationer, hvis rejse er mellem **start-** og **end** date, og der flyves fra **start-** til **end** destination. Det er ikke nødvendigt at udfylde alle felter for at kunne søge.

Herefter klikker brugeren på "**Search**"-knappen, som returnerer de ønskede resultater på listen nedenunder.

Edit:

Giver brugeren mulighed for, at redigere informationen i en eksisterende rejse. Vælges denne funktion bliver brugeren sendt videre til det forrige Edit Passengers-vindue, og kan derfra ændre oplysningerne heri. Dette gøres ved at vælge den eksisterende køber i listen Person og alle de tidligere indskrevne data auto udfyldes samt køberens siddeplads i flyet kan redigeres.

Delete:

Giver brugeren mulighed for at slette den markerede reservation i reservationslisten.

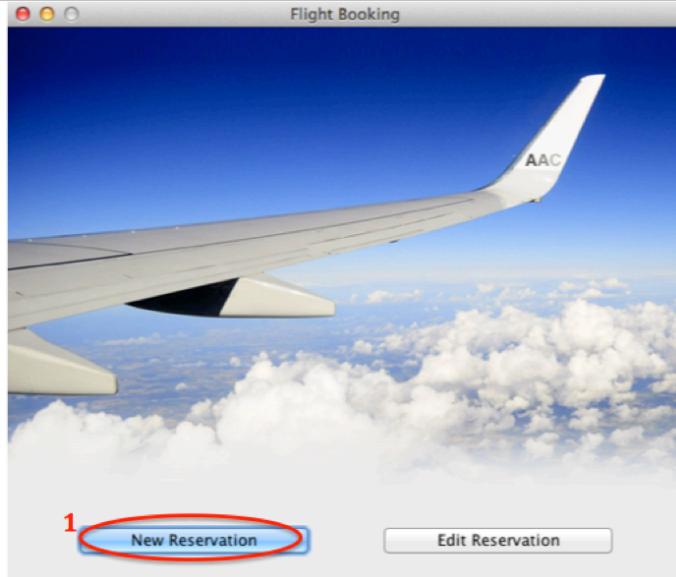
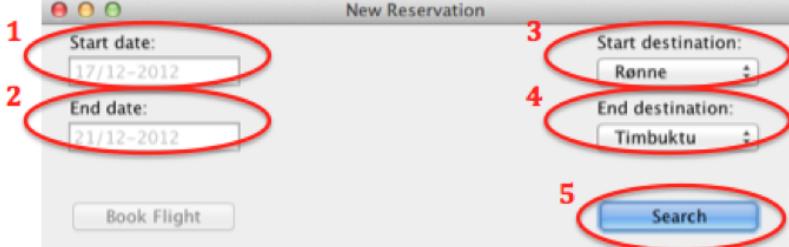
Edit Reservation

Reservation ID: <input type="text"/>	Start Date: <input type="text" value="dd/mm-yyyy"/>	Start Destination: <input type="text" value="None"/>
CPR #: <input type="text"/>	End Date: <input type="text" value="dd/mm-yyyy"/>	End Destination: <input type="text" value="None"/>
<input type="button" value="Edit"/>		<input type="button" value="Search"/>
<input type="button" value="Delete"/>		

Eksempel 1

I den kommende beskrivelse vil vi gennemgå et brugervenligt eksempel, som brugeren selv har mulighed for at følge fra start til slut.

Eksemplet er baseret på en fiktiv forbruger "Anders Andersen", der ringer ind til sekretæren og ønsker at bestille en flybillett fra Rønne til Timbuktu - gerne med en flyafgang i perioden mellem d. 17. og d. 21. december - 2012.

1. Klik "New Reservation".	
1. Indtast Start date "17/12-2012" . 2. Indtast End date "21/12-2012" . 3. Vælg Start destination " Rønne ". 4. Vælg End destination " Timbuktu ". 5. Klik på " Search ".	
	Når Search knappen aktiveres, kommer der fly frem på listen med afgang til Timbuktu der opfylder periode-kravet.

1. Vælg et passende fly og klik på flyet i listen. Herefter aktiveres "Book Flight".

2. Klik på "Book Flight" eller to gange på flyet i listen.

Flight Time	Flight Details	Flight ID	Price
11:00 16:00	Rønne - RØN > Timbuktu - TOM 17/12/2012 to 17/12/2012	H92837	pp. 4000.00 DKK
12:00 17:00	Rønne - RØN > Timbuktu - TOM 18/12/2012 to 18/12/2012	L02938	pp. 4000.00 DKK
08:00 13:00	Rønne - RØN > Timbuktu - TOM 19/12/2012 to 19/12/2012	S12947	pp. 4000.00 DKK
13:00 18:00	Rønne - RØN > Timbuktu - TOM 20/12/2012 to 20/12/2012	L02938	pp. 4000.00 DKK

Indtastning af køberens oplysninger.

1. Klik under First name og skriv fornavnet "**Anders**".

2. Klik under Last name og skriv efternavnet "**Andersen**".

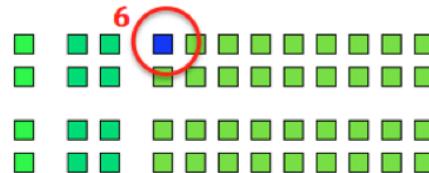
3. Klik under Full address og skriv adressen "**Anders Vej 1, 2300 KBH**".

4. Anders Andersen er en voksen person - Vælg "**Adult**".

5. Tryk på "**Add**".

6. Klik på et vilkårligt ledigt sæde blandt sæderne i det grafiske vindue.

7. Klik "**Book**".



Kvittering på reservation.
Køberens flyoplysninger fremstår i feltet og kan videregives til køberen.

1. Indtast køberens CPR-nummer **"1234567890"**.
Husk: Dette er vigtigt hvis brugeren ønsker at ændre i reservationen.

2. Noter køberens **"Reservations ID"**.
Husk: **"CPR-nummer"** og **"Reservations ID"** er vigtigt hvis brugeren ønsker at ændre i reservationen.

3. Klik **"Confirm Booking"**.

People in the reservation: 1

Payers CPR: 1234567890

Reservation ID: 2 8092

Departure: 17/12/2012 11:00:00 RØN
Arrival: 17/12/2012 16:00:00 TOM

Passengers:
Anders Andersen (Adult)

Seats: 4A

Plane: H92837

Price: 3 4000.0DKK

Confirm Booking

Du har nu gennemført eksemplet og oprettet en reservation til Anders Andersen i systemet. Ønsker du også et eksempel for redigering af en eksisterende rejse (se eks. 2 i bilag 5).

Teknisk beskrivelse af programmet

I det følgende afsnit beskrives implementeringen af problemanalysen, sammenhængen mellem klasser, samt håndtering af data. Der gennemgås også enkelte eksempler på konkrete løsninger.

Systemets klasser

Klasse	Beskrivelse
Airport	Simulerer en lufthavn med en ID, land- og bynavn.
Controller	Controller-klassen styrer programmet og er med til at holde GUI'en adskilt fra de bagvedliggende klasser. Controller-klassen henter data fra databasen ved forespørgsler, og derudover gemmer den informationen, som bliver dannet i brugergrænsefladen i løbet af en booking.
Converter	En statisk klasse der bruges til at konvertere dato objekter og strenge, samt udregner priser for reservationer. Klassen er statisk, så dens metoder kan kaldes til simple omregninger uden at skulle instantiere et nyt Converter-objekt.
Database	Database-klassen styrer forbindelsen til MySQL databasen, og kan hente informationer fra denne ud fra SQL-queries. Klassen opretter forbindelse til en given databasen ud fra givne parametre i konstruktoren. Herefter kan den slette, oprette og redigere i de tabeller MySQL-databasen indeholder.
Flight	Flight-klassen simulerer en rejse fra en lufthavn til en anden. Rejsen starter på et tidspunkt, og slutter på et senere tidspunkt. Rejsen foregår derudover på et specificeret fly. Denne klasse er derfor afhængig af givne lufthavnsobjekter, flyobjekter og Date-objekter.
Flybooking	Sætter programmet igang ved at oprette en instans af klassen StartFrame, der sætter brugergrænsefladen i gang.
Person	Simulerer en person med et fornavn, efternavn, adresse og aldersgruppe (senior, voksen eller barn), der alle gives som parametre i constructoren.
Plane	Simulerer et fly med et unikt ID, et antal rækker og et antal kolonner. Plane-klassen udregner også hvilket ID et givent seat skal have ud fra dets plads i flyet.
Printer	Printer en kvittering til brugeren med information om bookingen ud fra en given reservation.

Klasse	Beskrivelse
ProgramStorage	En lokal "database"-implementation som kan bruges som stand-in for en rigtig databaseklasse. Denne klasse er udelukkende brugt som test under udviklingen af systemet, og bruges ikke i det endelige program.
Reservation	Reservation-klassen bruges til at gemme data for en reservation: hvilken rejse, der er booket. Hermed vides også hvilket fly, hvilke lufthavne, og hvilket tidspunkt rejsen foregår på. Herudover hvilke personer, der skal med i bookingen, hvilke sæder disse personer har booket, hvilket reservations-ID reservationen er tildelt og hvilket CPR den er bestilt under.
Seat	Seat-klassen bruges til at gemme information om hvorvidt et sæde er optaget eller ledigt, og hvilket ID det individuelle sæde har, så det kan identificeres.

Klasser til brugergrænsefladen

Klasse	Beskrivelse
EditReservationFrame	EditReservationFrame udvider JFrame og er det vindue der bliver brugt, når brugeren ønsker at redigere i en reservation. Den indeholder derfor ReservationList, der viser matchende reservationer ud fra brugerens søgekriterier. Herefter kan en valgt reservation redigeres (se PersonAndSeatFrame).
FlightList	FlightList udvider Swing's JList. Den konverterer Flight-objekter til en præsentabel liste af Strings, der herefter bliver puttet ind i listen i et specielt layout.
GraphicsComponent	GraphicsComponent udvider JComponent og kan bruges til at tegne forskellige figurer med. Den bruges i dette tilfælde til at tegne oversigten over sæderne i fly.
NewReservationFrame	NewReservationFrame udvider JFrame og er det vindue der bliver brugt, når brugeren ønsker at finde en rejse i et givent tidsrum mellem givne destinationer, som kunden vil booke.
PaymentFrame	PaymentFrame udvider JFrame, og bliver vist til sidst i brugerens workflow, når brugeren skal bekræfte informationerne i reservationen. Her indtastes også kundens CPR-nummer, der fungerer som et referencenummer til reservationen.

Klasse	Beskrivelse
PersonAndSeatFrame	PersonAndSeatFrame udvider JFrame og bliver vist, når brugeren skal gemme informationer om de passagerer og sæder kunden ønsker at booke. Det er her muligt at tilføje, fjerne og redigere i personer i reservationen, samt til- og fravælge hvilke sæder der skal bookes.
ReservationList	ReservationList udvider Swing's JList. Den konverterer Reservation-objekter til en præsentabel liste af Strings, der herefter bliver puttet ind i listen i et specielt layout.
StartFrame	StartFrame udvider StartFrame og bruges som introskærm for programmet. Det er herfra du vælger, om du vil oprette en booking eller redigere, og starter resten af brugerworkflowet.

Anvendelse af Interfaces

Systemets data bliver hentet ind fra en MySQL-database via en implementering af DatabaseInterface. Alle klasser snakker så vidt muligt sammen med databasen gennem ControllerInterface, da dette giver lavere kobling i systemet, som nævnt ovenfor. Et eksempel på dette er implementeringen af saveReservation() i Controller-klassen, der gemmer den lokale reservation fra Controller-klassen i databasen.

Ved at hente data gennem interfaces på denne måde, kan elementer af programmet lettere udskiftes, hvilket muliggør at databaseimplementeringen undervejs i projektarbejdet kunne udskiftes med en lokal database (ProgramStorage-klassen), for at teste funktionaliteten af Java-systemet.

Da funktionaliteten var på plads, kunne MySQL-databasen implementeres og herved var det muligt at gemme på ITU's fjernserver. Ved at adskille brugergrænseflade fra Controller-klasse, kan brugergrænsefladen eventuelt udskiftes med en ny brugergrænseflade hvis ønsket. Controller-klassens interface gør det muligt at hente information fra en hvilken som helst given database, der implementerer DatabaseInterface og returnerer dette til brugergrænsefladen. Dette giver en god mulighed for videreudvikling af systemet.

Nedarvning

Nedarvning blev primært anvendt i nedarvning i klasserne der viser brugergrænsefladen.

Brugerfladen er skrevet ved hjælp af Swing, og ved at nedarve derfra, er det muligt at lave et særligt brugerdefineret JFrame, der bruger Swing's komponenter. Hermed har ét vindue ét formål, hvilket følger af responsibility-driven design. Dette gør også vedligeholdelse af koden lettere, da man undgår at skulle have én enkelt klasse, der skal håndtere og tegne fire individuelle JFrames.

Nedarvning er også brugt i GraphicsComponent, der så kan bruges i de forskellige frames, samtidig med at der er mulighed for at specialdesigne et komponent der kan tegnes med.

Dermed opnås høj indkapsling igen, da hver JFrame-klasse og GraphicsComponent kun bruges til én ting og ikke til noget andet. Klasserne er højt specialiserede, med én funktion og ét ansvar.

Repræsentation og implementering af lister

Et fokuspunkt for repræsentation af data for brugeren var listen over søgeresultater. En stor del af det visuelle design lægger vægt på listen, der bruges til at vise fundne rejser og reservationer. Dette betød, at listen skulle skræddersys, så den kunne vise informationer så overskueligt og effektivt som muligt. Derfor er en skræddersyet liste der arver fra Swing-klassen JList skrevet. Hermed kan den allerede skrevne og velfungerende listehåndtering fra JList anvendes, og kun den nødvendige funktionalitet udvides eller overskrives.

En skræddersyet version af DefaultListCellRenderer, en klasse der står for håndteringen af tegning af de enkelte felter i JList, er derfor implementeret.

Ved at overstyre metoden getListCellRendererComponent() i denne, er det muligt at tegne speciallavede felter i listen der viser fly- og reservationsinformation.

getListCellRenderer()-metoden returnerer et awt-komponent, der fungerer som et element i listen. Derfor kan flere elementer indsættes i dette element og give det ønskede layout. Da hver celle i listen tager imod et Flight-objekt, kan al den relevante data hentes ind i cellen ved at kalde dette objekts getter-metoder. Ved udvidelse kunne listen også simplificeres. Den tager derfor imod en ArrayList, og genererer herefter en liste ud fra denne, uden ekstra kodning er nødvendigt i de enkelte JFrames. Derved kan listen sættes ind i et givent awt-komponent og vise information om givne reservationer eller flights. Ingen har klassen ét hovedformål og ansvar. Udvidelse af klassen giver derfor højere genbrugelighed og lavere kobling.

Alternativet var at skrive listen alle steder den var nødvendig. Dette ville give kodeduplikering, lavere letlæselighed, øge muligheden for fejl og eventuelt mindske sammenhængen i brugergrænsefladen, da listen alt efter implementation kunne se forskellig ud.

Databasesøgning

Som nævnt tidligere er MySQL indekseret, og vil søge hurtigere end Java-siden af systemet uden implementering af søgemetoder i denne. Herved er søgefunktionaliteten isoleret i databasen, og derfor er andre klasser ikke afhængige af denne.

Oprettelse af objekter

Ved søgning oprettes objekter som f.eks. Person-, Airport eller Reservation-objekter. Unikke keys er anvendt i databasen, hvormed det er muligt ved et givent ID, at finde de objekter der passer sammen med eksempelvis den reservation der skal oprettes.

Objekter bliver oprettet ud fra ResultSet, der genereres ud fra en udført SQL-query. Objekterne bliver oprettet, når der er brug for dem med et new-statement i databasen, i Controller-klassen samt i Reservation-klassen.

SædeID generator

Plane-klassen genererer sæder i flyet ud fra givne variabler rows og columns. Sæderne repræsenteres i et todimensionelt array for henholdsvis columns og rows. Ud fra felterne oprettes en string, der beskriver sædets placering og dets ID, der er unikt for rejsen.

Dette ID genereres ved at tage sædets column, lægge én til og konvertere det til en string. Herefter gennemgås rows i et switch-statement og alt efter hvilken værdi denne har, bliver dette omdannet til henholdsvis A, B, C, D, E eller F. Denne værdi tilføjes til den eksisterende string.

Det genererede ID ligner det man møder på "rigtige" flyselskaber, og fungerer derudover for alle fly med et antal rækker op til seks.

Statisk Converter-klasse

Dataer fra databasen skal konverteres fra Java's Date-format til strings, der kan vises i brugergrænsefladens lister og dropdowns.

Konverteringen er en relativt simpel operation med SimpleDateFormat. I SimpleDateFormat's constructor specificeres det, hvordan den genererede String eller Date skal formateres, hvorefter det er muligt at parse fra Date til string og vice versa.

Da denne konvertering bliver brugt i korte øjeblikke, er klassen og dens metoder statiske, da de kan anvendes når der er brug for den, så en Converter-variabel ikke skal gemmes i andre klasser.

Et interface kunne ikke anvendes ved denne klasse, da det ikke er muligt at skrive abstrakte statiske klasser ved hjælp af interfaces i Java 7 (dette er understøttet i Java 8). Derfor er klassen skrevet uden brug af et interface.

Klikke på sæder

Valg af sæder udføres i et JPanel som indeholder en instans af GraphicsComponent. GraphicsComponent står for tegning af sæderne og håndtering af klik på disse. Koordinaterne fra museklik konverteres til rækker og kolonner, der derved kan fortælle hvilket sæde, der var trykket på.

Da sæderne er placeret i et todimensionelt array, er det muligt at skille x og y koordinat ad, og finde indekset i dette arrays to dimensioner. Ved at løse den ligning der blev brugt til at tegne sæderne, er det muligt at finde frem til en ligning, der kan omdanne koordinatet til et heltal, der svarer til den kolonne eller række, der er trykket på. Herefter skal der tages højde for indikatorer for flyets mellemgang og afstande mellem first-, business- og economy class. Hvis den udregnede kolonne- eller rækkeværdi er uden for antal rækker eller kolonner er negativ, bliver klikket set bort fra. Da det vides hvilken kolonne og række klikket er indenfor, er det muligt at tjekke om klikket også er inden for det sæde, og ikke blot er en del af den såkaldte padding, der ligger rundt om sædet.

Sædets ID genereres via Plane klassen, hvori det er muligt at se, om sædet er optaget eller en del af denne reservations sæder og derefter kan sædets tilstand ændres.

Singleton design-pattern

For at sikre at der kun er initialiseret én database-forbindelse, er et Singleton design-pattern anvendt i opbygningen af Database-klassen. Dette sikrer, at der kun er ét databaseobjekt

instantieret, at datakonflikter undgås, samt at den aktuelle database altid er tilgængelig fra Controller-klassen.

Da Controller-klassen også håndterer data, er denne også bygget op omkring som en singleton. Dette gør, at der under hele programmets kørsel er sikkerhed om, at der ikke er nogle datakonflikter, at disse objekter altid er instantierede og altid er til at finde.

Klasserne er Singletons ved at gøre deres konstruktør privat, og have en getInstance()-metode, der returnerer den aktuelle instance af objektet. Hvis ingen instance er til stede, oprettes én, og denne returneres.

Brugergrænsefladen

I det følgende afsnit vil opbygningen af brugergrænsefladen blive beskrævet, hvordan Action- og FocusListeners er anvendt, samt en redegørelse af brugen af tredjeparts-Layout Manageren Mig Layout.

Under arbejdet med opsætningen af de anvendte Swing-komponenter, viste det sig, at de standard Swing Layout-Managers ikke var tilstrækkelige. Tidsrammen for projektet gjorde det ikke muligt at nå at lave et færdigt layout der matchede UI-mockupsene, da man så skulle oprette unødvendigt mange komponenter inde i komponenter, og alligevel ikke have præcist det korrekte layout.

Derfor faldt valget på den gratis tredjeparts Layout-Manager kaldet MiG Layout, udviklet af MiG Infocom. MiG Layout er bygget op omkring et skema-layout, hvor hvert komponent tilføjes i hver sin kasse. Komponenterne sættes i skemaet fra venstre mod højre, indtil "wrap" skrives efter tilføjelsen af et komponent, hvorefter der springes en linje ned i skemaet. I MiG Layout-contractoren specificeres hvilken mængde spacing der skal være mellem komponenterne i skemaet. F.eks. er der i NewReservationFrame 260 pixels spacing mellem komponenterne i venstre side og komponenterne i højre side af vinduet.

Derudover kan et komponent også sættes til at fyldе to eller flere kasser, så det er muligt at oprette såkaldte filler-komponenter, der spænder over flere kasser, hvorved et komponent kan sættes præcist hvor ønsket.

Dette layout er anvendt konsekvent i vores skræddersyede JFrames, hvilket gør det muligt at oprette et layout, der er fleksibelt og ser ud som ønsket.

Tests

I det følgende afsnit beskrives det hvilke tests der er udført under udviklingen af systemet, hvordan tests er indarbejdet under arbejdsprocessen, samt hvordan disse tests er udført.

Nødvendigheden af tests

Testning er nødvendigt for alle programmer, da de er essentielle for at kunne være overbevist om at programmet gør det forventede, gør det korrekt og ikke crasher, hvis brugeren gør noget uforudset.

Automatiserede test er anvendt i de mest centrale klasser til testning af kode, der ligger bag brugergrænsefladens information. Her er parameter-input kontrolleret, resultatet sammenlignet med det forventede resultat. Især i Database klassen som en af de klasser med mest funktionalitet, har vi sørget for at have automatiserede tests. Her gennemgås det om databasen henter de rigtige data ud eller sætte de rigtige objekter ind i databasen. Dog kan det have problemer med sig, da tabellerne i databasen hele tiden kan ændre sig. Derfor er det endnu vigtigere at testene bliver kørt igennem som de sidste inden codefreeze.

I brugergrænsefladen er systematiske manuelle testsog brugertests anvendt for at undersøge, om programmet fungerede. Man kunne med fordel have automatiseret dette ved at bruge Java's Robot class, som bl.a. simulerer klik med musen, men dette var ikke muligt pga. tidspres.

Anvendelse af testdatabase

Ved at midtvejs i projektet at oprette en simplificeret lokal database, og sætte denne ind i stedet for den endelige MySQL-database, var det muligt at lave tests af funktionaliteten af brugergrænsefladen før systemet var færdigudviklet, og køre et komplet workflow igennem, før den egentlige database var funktionel. Dette gjorde det muligt at finde uforudsete fejl ved manuelle tests, og få disse rettet.

Den simple lokale database, der blev oprettet hver gang systemet startes, muliggjorde også at kunne genstarte tests, hvis der opstod en kritisk fejl. Herved ville hele tabeller eller større mængder data i den endelige database ikke blive slettet eller overskrevet ved et uheld.

Debugging

Et debuggingværktøj er at finde i de fleste moderne IDE'er. Debugging foregår ved, at man gennemgår koden step-by-step fra et givent breakpoint, således at man kan se objektets tilstande og de variable de indeholder på et givent tidspunkt.

Debugging blev brugt som et værktøj til at finde og rette fejl i koden, især ved databasens implementering og de dele af koden, som gruppen havde mindre træning i at programmere- herunder SQL-queries og custom Swing-implementering. Specielt under kørsel af SQL-queries blev debugging brugt meget, da disse var sværere at se sig ud blot fra en gennemlæsning af koden da compileren ikke kom med fejlmeldelser ved syntaksfejl. Her hjalp håndteringen af SQL-Exceptions også, da dette gjorde det muligt ved hjælp af Java's stack trace, at se præcis hvor fejlen opstod.

Videreudvikling

I det følgende afsnit beskrives fejl og mangler i systemet og idéer til videreudvikling af systemet for forbedring af dette.

Anvendelse af Factory method-pattern

Ved anvendelse af MVC står Controller-klassen, som nævnt tidligere, for håndtering af data. De primitive objekter i systemet oprettes ud fra denne data. Dog oprettes disse objekter af flere klasser i systemet, og hermed bliver koblingen mellem disse større.

Derfor ville anvendelse af et Factory method-pattern i Controller-klassen til generering af objekter være at foretrække. Hermed ville kun Controller-klassen kende til de primitive klasser som fly, lufthavne osv., og disse kunne videregives til f.eks. brugergrænsefladen.

Dette gør dog implementeringen af Controller og håndteringen af objektoprettelse i workflowet mere avanceret, og kræver mere tid til implementering og fejlfinding.

Dette er derfor ikke implementeret på grund af tidspres, og oprettelsen af objekter sker direkte i de højere klasser, som f.eks. brugergrænsefladen og databasen.

Central View-Controller

For at gøre anvendelsen af MVC endnu mere konsekvent, burde en View-Controller i form af et interface til tegning og opdatering af brugergrænsefladen oprettes. Hermed ville brugergrænsefladen gøres uafhængig fra Controller-klassen, og en anden brugergrænseflade ville kunne indarbejdes hvis ønsket.

Sortering af afgange

Ved at fjerne alle utilgængelige ankomstdestinationer fra brugergrænsefladen ud fra den valgte afgangsdestination inden brugeren klikker videre, kan brugeren komme hurtigere igennem sin opgave om at finde rejser. Dette mindsker også eventuel forvirring, da brugeren kunne undre sig over, hvorfor der ikke kommer nogle rejser frem ved valg af to tilsyneladende tilgængelige destinationer.

Nedskæring af DatabaseInterface

Metoder der vedrører databasen selv burde kun findes i implementeringen, da disse ikke har relevans for de andre klasser, der tilgår interfacet. Alle metoder ControllerInterface ikke kalder i DatabaseInterface ville derfor kunne fjernes, da disse er uvedkommende for MVC.

Konklusion

Ved anvendelse af ER-diagrammer, mockups og udformning af brugeropgaver, var det muligt at danne et effektivt overblik over systemets bearbejdning og håndtering af data. Dette giver en idé om de nødvendige klasser og hvad disse skulle udføre i systemet.

Systemet er designet med en stabil og overskuelig brugergrænseflade, samt en forudsigelig Mental Model, der hjælper brugeren til at gennemføre sine opgaver. Ved hjælp af Swing og herunder MiG Layout, fik systemet det ønskede design og brugervenlighed. Brugergrænsefladens funktioner løser de ønskede problemstillinger og informationsudvekslingen mellem databasen og resten af systemet er velfungerende.

Ved at anvende MVC- og Singleton design-patterns blev der opnået et relativt højt abstraktions- og modulationsniveau i systemet, og det var muligt at holde klassernes kobling forholdsvis lav og indkapslingen høj. Anvendelse af interfaces gjorde udskifteligheden af klasser betydelig mere effektiv, da hvert enkelt klasse i systemet ikke var afhængige af implementateringen af de andre klasser, dette gjorde også ændringer og tilføjelser af funktionalitet nemmere.

En bruger af systemet er i stand til at udføre de opstillede brugeropgaver, og ved hjælp af tests af brugerinput, er systemet i stand til at kunne sikre for større fejl. Systemet opdateres dynamisk ud fra information i databasen, og ved hjælp at databasesøgninger skal brugeren vente kortvarigt på feedback.

Manuelle systematiske tests hjalp til at finde uventede fejl undervejs i projektet, og ved hjælp af automatiserede JUnit-tests mod slutningen af programmet, var det muligt at sikre mod uventede fejl opstod ved hentning af resultater fra databasen.

Arbejdsprocessen er forløbet som planlagt, med en iterativ arbejdsform og ingen kriser under projektet. Målene med projektet er blevet indfriet mod de forventede krav, og visse ting er udeladt som forventet på grund af tidspres.

Der er især mulighed for forbedringer med hensyn til at mindske koblingen mellem systemets klasser at skære i de enkelte interfaces kode for at simplificere disse og tilføje ekstra funktionalitet. Med mindre kobling ville videreudvikling være simpelere, så ekstra funktionalitet kunne tilføjes nemmere og dermed hurtigere.

Litteraturliste

Litteratur:

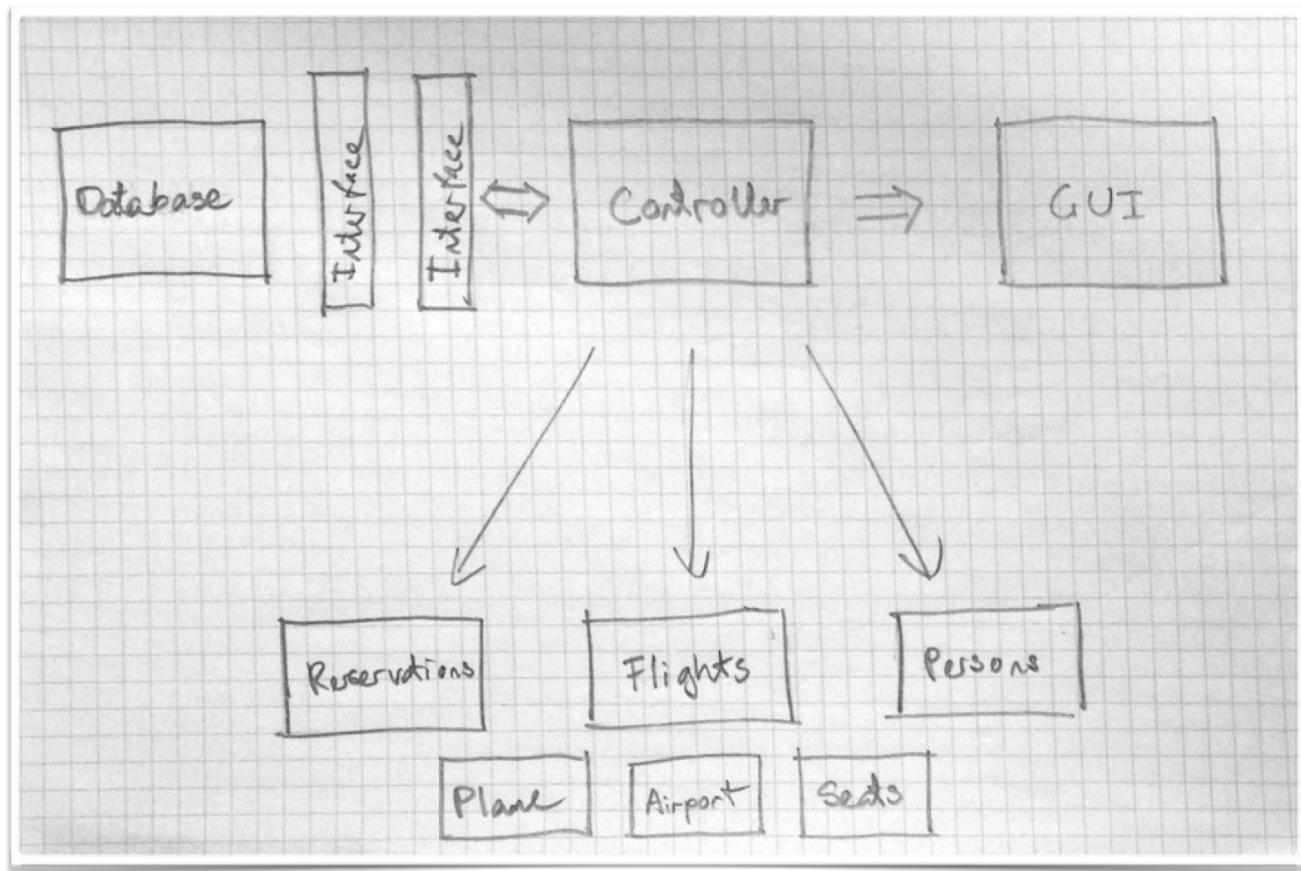
Lauesen, Soren. User Interface Design: A Software Engineering Perspective.
Addison Wesley; 1 edition (12. Nov 2004).

Hjemmesider:

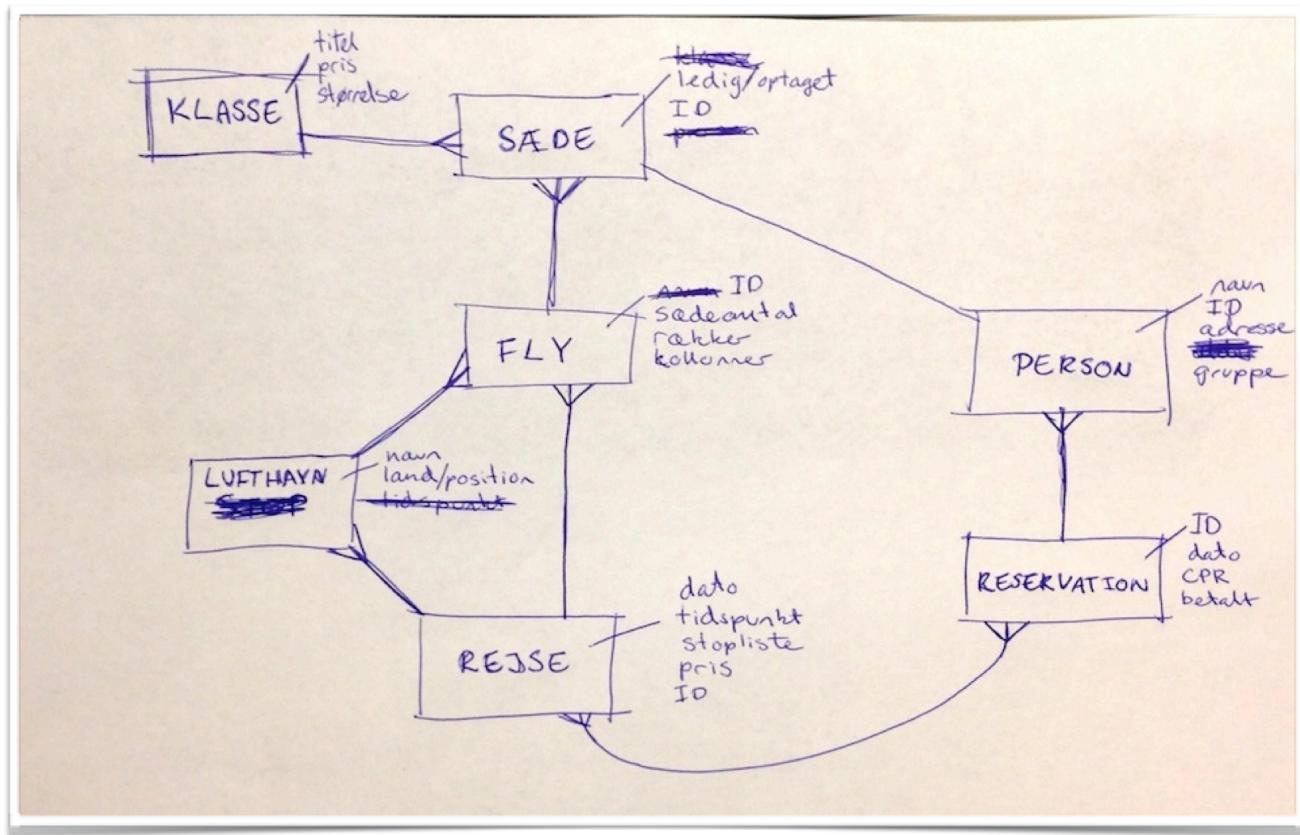
MigLayout - Java Layout Manager. Udgivet af MiG InfoCom.
Internetadresse: <http://www.miglayout.com> - Besøgt d. 12.12.2013 (Internet).

Bilag

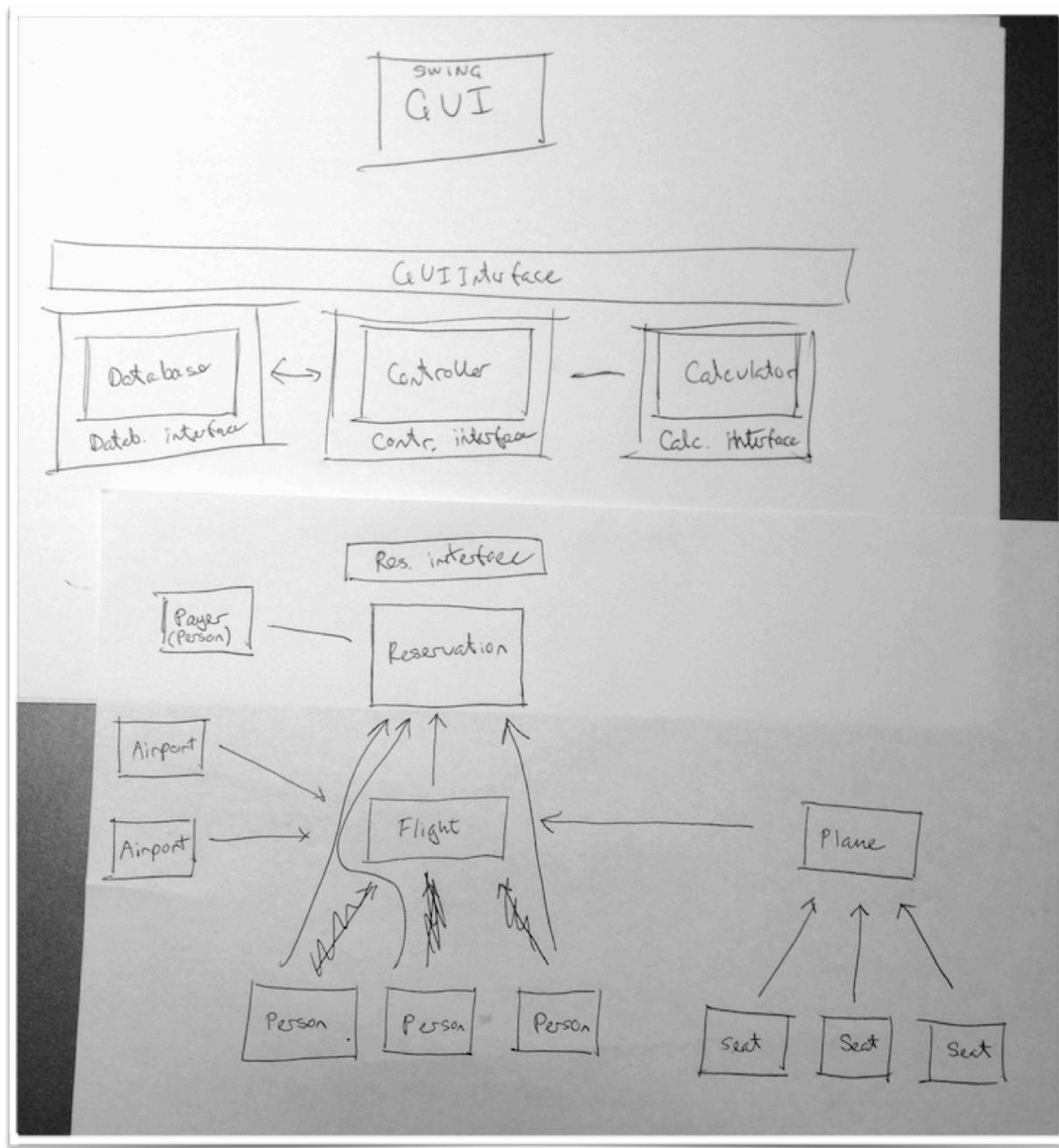
Bilag 1



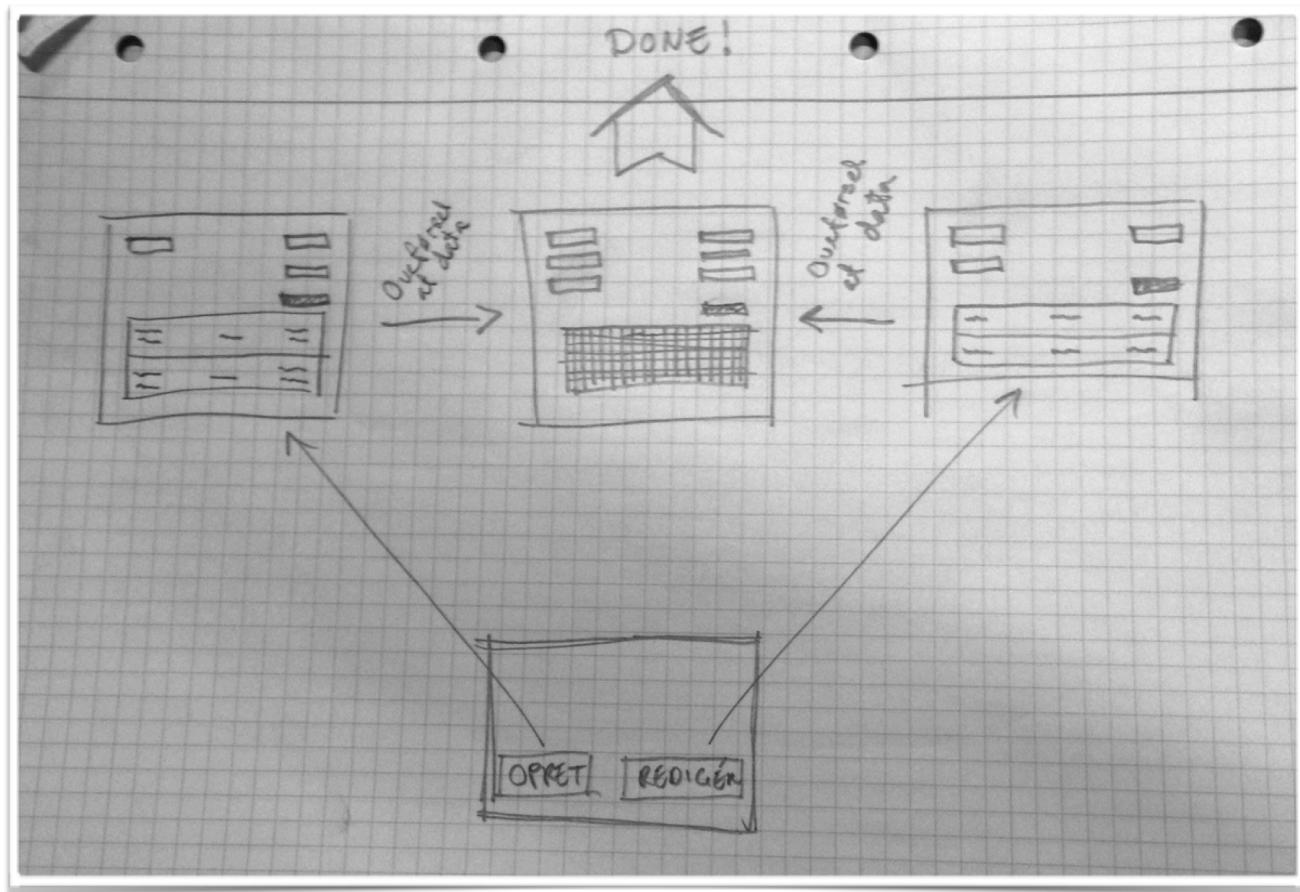
Bilag 2



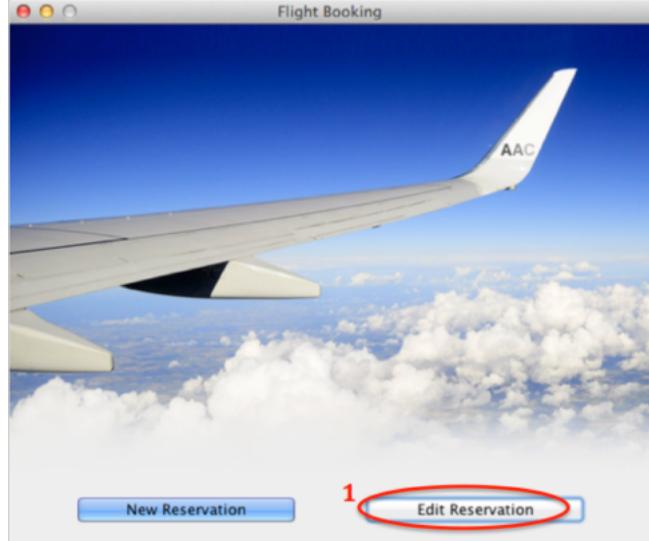
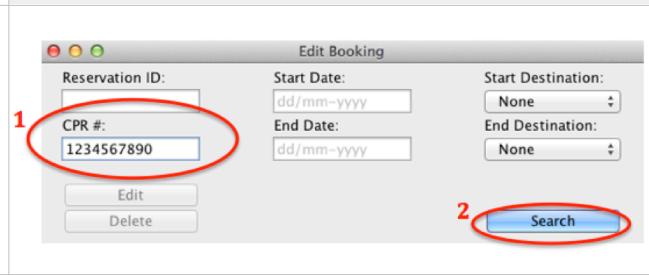
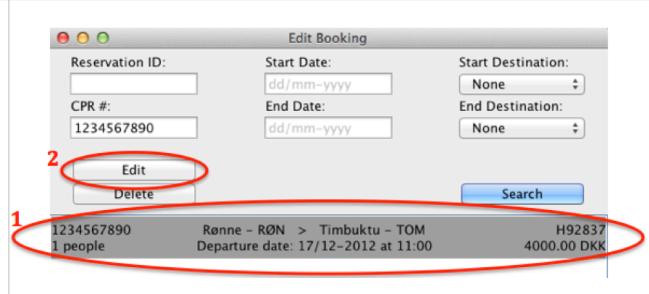
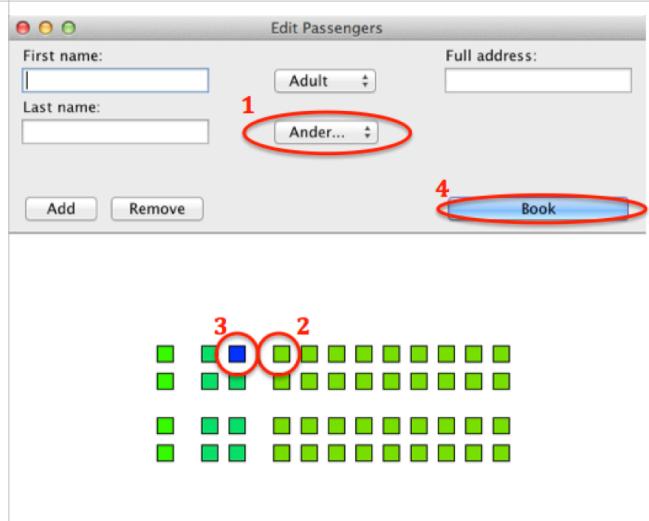
Bilag 3



Bilag 4

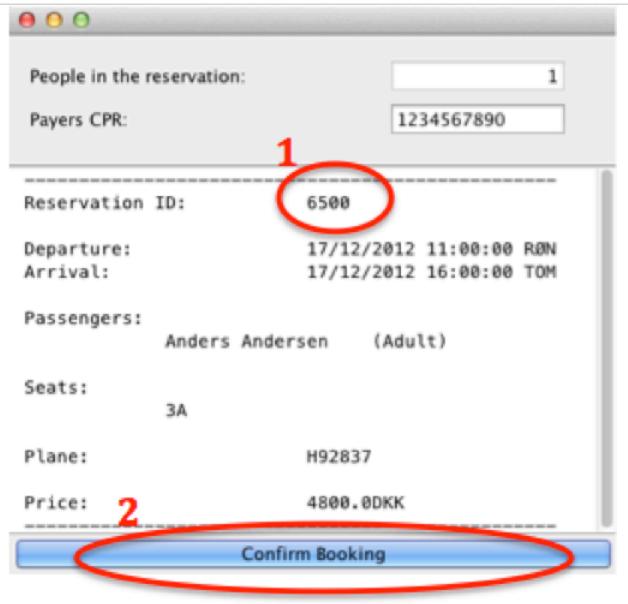


Bilag 5

<p>1. Vælg "Edit Reservation".</p>	
<p>Køberens reservations ID eller CPR indtastes.</p> <p>1. Indtast køberens CPR-nummer "1234567890".</p> <p>2. Tryk på "Search".</p>	
<p>Et fly matcher de indtastede kriterier.</p> <p>1. Check i reservationlistens venstre side og midte, at det er den rigtige reservation, der er fundet. Tryk på reservationen i listen.</p> <p>2. Tryk på "Edit".</p>	
<p>Nu ser du samme booking-vindue fra det forrige eksempel. Det blå felt repræsenterer den aktuelle siddeplads, som nu kan redigeres.</p> <p>1. Tjek at det er "Anders Andersen" under Person-dropdown-menuen.</p> <p>2. Flyt personen til et nyt sæde. Klik på det blå felt så det bliver grønt og derved ledigt igen.</p> <p>3. Vælg nu køberens nye siddeplads. Klik på et nyt vilkårligt ledigt sæde i Business Class.</p> <p>4. Tryk på "Book".</p>	

Det nye vindue sørge for at redigeringen
gemmes.

1. Læg mærke til at køberen får et nyt
"Reservation ID", og en ny **"Pris"**.
2. Tryk på **"Confirm Booking"**.



Du har nu gennemført eksemplet og redigeret en eksisterende reservation til Anders Andersen i systemet.

Bilag 6

AirportTest.java output

```
Tests run: 3, Failures: 0, Errors: 0, Time elapsed: 0,08 sec
----- Standard Output -----
getCity
getCountry
getID
-----
test-single:
BUILD SUCCESSFUL (total time: 0 seconds)
```

PlaneTest.java output

```
Tests run: 9, Failures: 0, Errors: 0, Time elapsed: 0,097 sec
----- Standard Output -----
getSeatAvailability
setSeatAvailability
getID
SeatIDGenerator - tests if the method generates correct seatIDs
getNumberOfSeats
getColumns
bookTakenSeats
getRows
getSeat - tests if the returned seat's ID matches the wanted
-----
test-single:
BUILD SUCCESSFUL (total time: 1 second)
```

ConverterTest.java output

```
Tests run: 10, Failures: 0, Errors: 0, Time elapsed: 1,967 sec
----- Standard Output -----
getAirport
getReservationList
removeReservation
getAirportCitiesAsStrings
getPlane
getAirportID
insertPerson
getFlight
getFlightList
checkForID
-----
test-single:
BUILD SUCCESSFUL (total time: 2 seconds)
```

DatabaseTest.java output

```
Tests run: 5, Failures: 0, Errors: 0, Time elapsed: 0,478 sec
----- Standard Output -----
convertDateToString
convertDateToHourString
convertStringToDate
createPersonID
getFinalPrice - tests all age groups and seat classes
-----
test-single:
BUILD SUCCESSFUL (total time: 1 second)
```

		Uge: Uge 47:		Uge 48:		Uge 49:
Kodning:	Rapport:	Torsdag (19. nov)	Onsdag (20. nov)	Torsdag (21. nov)	Mandag (25. nov)	Torsdag (26. nov)
To Do:						
Kig på orgaave-formuleringen	Brugerscenarier	Lav brugertasks	Diskussion af controller-implementation	Forelæsning om database	Dan en simpel database i til brug med databaseinterface.	Implementere flere databasemetoder
Overvej ERD	Overvej implementering	Begynd så småt på tests mht. tasks	Diskussion af GUI-mockups	Opbyg databasen med tables osv.	Find løsnings til mange-to-many-relationer med seat og person labeler	Tilføje sikkerhedsjek på brugernput
Overvej hovedproblemer	Simpel klassesætning	Kig på databaseløsninger	Kodning af "lavere klasser"	Indlære data i databasen.	Omskriv Controller-klassen, så den snakker sammen med interface	Tilføje frame til at confirme booking
Simple problem løsninger	Altal vedr. dæbogsskrivning, mødeudl osv.	Reservation Hashset/MapSet mellem pers. og saede	Til i morgen lave actionlisteners osv til frames	Hvis Id - connect database/implementation og GUI	Implementer FlightList videregående	Projektrapport - brugerekss
Startet på projektrapport - hovedframes i programmet		Projektrapport - herunder problemformulering samt problemstillinger			Raport - herunder problemformulering samt problemstillinger	

