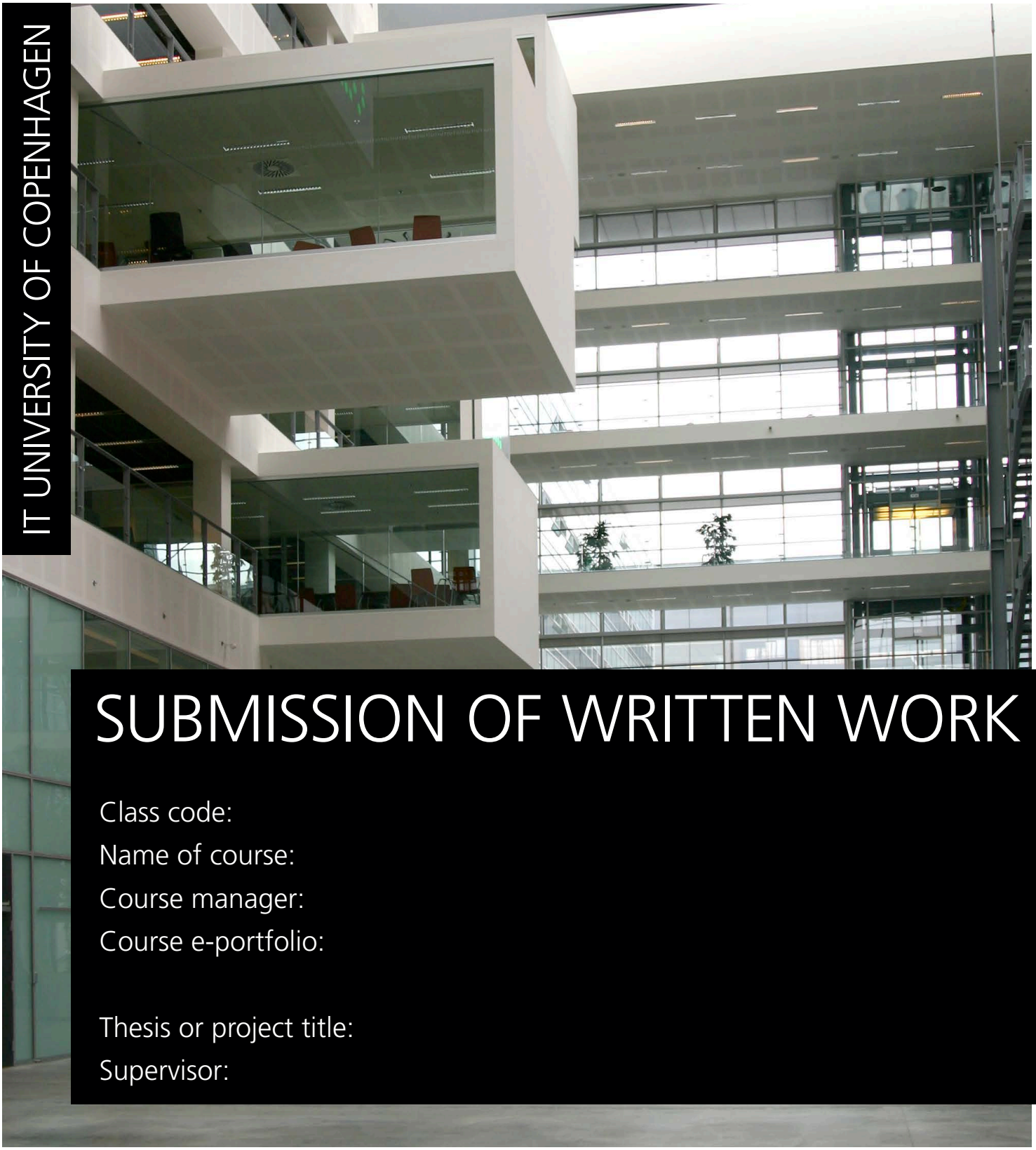
IT UNIVERSITY OF COPENHAGEN



SUBMISSION OF WRITTEN WORK

Class code:

Name of course:

Course manager:

Course e-portfolio:

Thesis or project title:

Supervisor:

Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1. _____	_____	_____@itu.dk
2. _____	_____	_____@itu.dk
3. _____	_____	_____@itu.dk
4. _____	_____	_____@itu.dk
5. _____	_____	_____@itu.dk
6. _____	_____	_____@itu.dk
7. _____	_____	_____@itu.dk

1. Forord og indledning

Gruppe 24 har udarbejdet denne rapport i forbindelse med afslutningen af kurset i Grundlæggende Programmering på IT-Universitetet i København (ITU) i 2016. Rapporten og selve programmet er udviklet under vejledning af TA's samt underviser Patrick Bahr. Projektet er udarbejdet på baggrund af problemstillingen givet i undervisningen - Biograf Booking. Vi har udviklet et java-program samt en MySQL-database, der tilsammen udgør vores løsning på problemstillingen: at bestille, opdatere og slette sædereservationer fra biografens billetluge. Dette er uden muligheden for online booking og køb af billetter via programmet, i overensstemmelse med problemstillingen.

2. Problemstilling og baggrund

I forbindelse med problemstillingen, biograf booking, er der fra undervisernes side stillet en række krav til dette bookingsystem. Programmet skal kunne håndtere de dag-til-dag opgaver, en ekspedient i en mindre biograf vil støde på. Dette indebærer opgaver såsom bestilling af billetter, ændring af en reservation, samt udvælgelsen af specifikke pladser til en kunde. Alt dette skal desuden benytte en database til lagring af data, således at information bevares mellem kørsler af programmet. Vi antager, at der ikke er mulighed for selv at booke film, f.eks. online, samt at der kun er en billetluge. Altså har vi ikke concurrency i forhold til vores database.
Specifikt skal vores implementation af løsningen kunne:

- Gemme et ID til brug ved genfindelsen af en reservation
- Ændre en reservation, f.eks. ændre antallet eller placeringen af pladser
- Slette en allerede eksisterende reservation
- Huske pladser en kunde har valgt
- Give en visuel repræsentation af salen, der vises
- Vise forestillinger, ikke kun på det præcise klokkeslæt angivet, men også nærliggende forestillinger
- Finde alle forestillinger, hvor en bestemt film vises
- Finde alle forestillinger, der vises et bestemt klokkeslæt
- Vise alle fundne forestillinger i listeform

Vores database skriver vi i MySQL, og al vores programkode vil være skrevet i Java, hvor vi bruger AWT- og Swing-komponenter til at lave vores GUI.

3. Problemanalyse

Dette afsnit vil beskrive den designprocess, der har ført til det valgte klassedesign. Vi vil også beskrive opbygningen af vores program som færdigt produkt og hvordan designprocessen har påvirket vores designbeslutninger undervejs. Dette indkluderer en teknisk beskrivelse af vores program- og databasestruktur.

3.1 Domæneanalyse

For at kunne opnå en fornuftig implementation af vores program, er det vigtigt at have en bred forståelse for problemstillingen. Vi har derfor lavet en domæneanalyse for at identificere klasser og deres interaktioner.

3.1.1 Noun/Verb - metoden

For at få en ide om hvilke klasser vores program kan være opbygget af, benyttede vi os af en metode kaldet *"the noun/verb approach"*. Denne metode går ud på at markere alle navne- og udsagnsord i opgavebeskrivelsen. Alle navneord er som udgangspunkt bud på klasser og udsagnsordene bud på metoder for de enkelte klasser. Når man skriver disse klasser og metoder op, skelner man ikke mellem ental og flertal eller bøjning af ordene. Dette giver dog langt fra et færdigt billede af opbygningen af klasser og metoder.

Løsningen skal kunne håndtere reservation af pladser, men ikke salg (dvs. billetpriser og net-betaling skal ikke implementeres). Ved reservation skal der opgives identitet, fx i form af et telefonnummer eller navn . Biografen har kun én billetluge og tillader ikke reservation over nettet, så løsningen behøver ikke håndtere samtidige opdateringer og adgang til databasen. Biografen har flere sale og viser forskellige film på forskellige tidspunkter af dagen i de forskellige sale. En kombination af film, sal, tidspunkt og dag kaldes en forestilling. De forskellige sale kan have forskelligt antal stolerækker og antal stole per række. Antallet af sale og salenes antal stolerækker mv. skal fremgå af databasen og må ikke være indkodet i selve

3.1.2 CRC-kort

For at tage et skridt nærmere de endelige klasser i designet af programmet kan man arbejde videre med udsagnsord og navneord. Vi valgte at arbejde videre og lave en række CRC-kort. CRC står for Class/Responsibilities/Collaborators, hvilket er hvad et sådant kort beskriver. Class er et klassenavn - navneordet fra tidligere. Responsibilities er de handlinger, som klassen er ansvarlig for, hvilket er klassens metoder eller udsagnsordene til de tilhørende navneord fra programbeskrivelsen. Collaborators beskriver hvilke klasser denne klasse bruger eller arbejder sammen med.

Efter at have tegnet CRC-kort samt skrevet klassenavn og bud på metoder, brugte vi dem, sammen med de brugerscenarier og krav vi i problemstilling har

opstillet, til at spille de enkelte scenarier igennem. Mens vi gjorde dette, noterede vi Collaborators på alle kortene, hvilket gav en ide om opbygningen af programmet. Herfra trådte vi fra analyse-domænet videre til løsnings-domænet, hvor vi ville finde ud af, hvordan vi kunne udvikle vores løsning. Vi har senere valgt at tilføje et par klasser efter vores analyse af problemet ved brug af CRC-kort.
Vores klasser kan kort opsummeres således:
Billetluge: Billetlugen viser hvilke forestillinger, der vises. Derudover kan den vise salene for de enkelte forestillinger, håndtere information fra kunden og sørge for at lagre den korrekte information. En billetluge fungerer altså som formidler mellem vores data og brugergrænseflade.
MySqlConnection: Dette er en klasse vi selv har valgt at tilføje, fordi vi skal kunne skrive til vores database for at gemme informationer.
Showing: Showing lagrer information fra en enkelt forestilling, sådan at informationen kan bruges i vores brugergrænseflade.
Hall: Denne klasse beskriver en biografsal med antallet af sæder per række og antal rækker.
Reservation: Når der skal ændres eller oprettes en reservation, tænker vi at lave et reservationsobjekt, som vi kan give vores SQL-klasse.

Vi har på forhånd bestemt os for at have to klasser til vores brugergrænseflade:
BookingView: En klasse som kan vise alle forestillinger til ekspedienten i billetlugen, hvor han kan vælge en bestemt forestilling.
CinemaView: En klasse der kan vise en biografsal samt hvilke sæder, der er optagede. Derudover skal han kunne vælge nogle sæder og oprette en reservation med disse.

3.2 Design

Med en ide om hvilke klasser vores program tager udgangspunkt i, besluttede vi os for at benytte et MVC-design pattern til vores program som generel programstruktur. Vi omdøbte "billetluge" til Controller for nemmere at kunne se, hvor de enkelte klasser hørte til.

Vi har valgt MVC for at holde programmet så modulært som muligt. Derudover bliver programkoden også adskilt, hvilket reducerer coupling. Dette gør, at vi kan ændre implementeringen af vores model men samtidig beholde samme View og dermed også samme brugeroplevelse.
Model:
I vores Model har vi opbygget vores model af den virkelige verden med de klasser, vi mener at have behov for: Hall, Showing og Reservation. Ud over disse klasser er vores Model ansvarlig for kontakt til databasen. Denne er placeret i Model, fordi dataen til at beskrive programmet skal hentes gennem disse klasser.

View:
View er i vores implementering af MVC bestående af tre klasser. De er hver især ansvarlige for at vise en del af vores GUI. Vores klasse BookingView vises, når programmet åbnes. Herfra kan der navigeres videre til vores andre visninger, ReservationView og CinemaView. I CinemaView vises en bestemt forestilling, og der kan vælges sæder og oprettes en reservation. Reservationen kan ændres i ReservationView, hvor reservationer kan findes via en søgning, ændres og slettes.
Controller:
Controlleren i vores program fungerer som "talerør" mellem Model og View. Når brugeren interagerer med vores brugergrænseflade, kaldes metoder i Controlleren. Controlleren henter herefter den nødvendige information i Model, hvorefter den sendes til View, som viser informationen.

3.2.1 Overordnet design og programstruktur

Vi antager, at vores program skal betjenes af en ekspedient i en biograf, og der kun kan betjenes en kunde ad gangen. Derfor har vi designet vores brugergrænseflade, så den er nem at gå til og simpel at navigere rundt i. På denne måde skal ekspedienten ikke bruge tid på at gennemskue, hvordan han f.eks. opretter en reservation, men at det i stedet forekommer intuitivt.

Vi har valgt at lave et vindue til hver visning for at holde støjen lav i forhold til information præsenteret for ekspedienten. Derudover har vi tilføjet endnu et vindue til vores design. Dette vindue skal præsentere ekspedienten for reservationer, der allerede er oprettet og give vedkommende mulighed for at ændre eller slette en reservation.

For at kunne holde vores Model- og View-dele af programmet adskilt, har vi valgt at bruge TreeMap-datastrukturen til at give information til vores View-klasser. Vi har valgt at bruge TreeMap, fordi det lader os bruge det givne ID, til f.eks. en bestemt showing, som key. Derved kan vi bruge toString-metoden på vores objekter til at skrive en streng, der beskriver objektet med det givne ID. Dette tillader os at skelne objekter fra hinanden, selvom View ikke kender noget til selve objektet. Vi har valgt Treemap specifikt fordi, vi på den måde kan være sikre på, at vi beholder den rækkefølge vi får vores entries fra databasen i. Dette ville man ikke kunne gøre med f.eks. et HashMap, hvor ID'et stadig ville være parret med en beskrivelse, men hvor rækkefølgen ikke ville være den samme. Derudover kan der ikke med et Map opstå konflikter om et givent ID, da keys er unikke.

For at vi kan præsentere denne information til brugeren, er det vigtigt at kunne lave de relevante objekter og hente data fra dem. Dette kræver et gennemtænkt design af tabellerne i vores database, der giver anledning til fornuftige dataudtræk.

3.2.2 Database

Baseret på vores domæneanalyse har vi lavet en database, hvis tabeller beskrives i det følgende afsnit.

Vi havde tænkt, at databasen ccpbioDB skulle indeholde følgende tabeller:

- Customers
- Reservations
- Reserved_seats
- Theaters
- Movies
- Shows

Customers

Denne meget enkle tabel indeholder intet andet end de telefonnumre, der er blevet brugt til oprettelse af reservationer.

- tlf_nr: Kundens telefonnummer.

tlf_nr
0
1
2
3
4
5
6

Reservation_id	Tlf_nr	Show_id
11	123	12
122	999999999	11
128	3223	12
132	989898	14

Reservations

Denne tabel indeholder al relevant information omhandlende reservationer. Nedenfor ses en liste over hvilke kolonner tabellen skal indeholde:

- Reservation_id: Unikt ID, der gør det muligt at genfinde en given reservation.
- tlf_nr: Telefonnummer opgivet ved reservation af pladser.
- show_id: Unikt ID, der henviser til en bestemt forestilling.

reservation_id	seat_id
132	21
132	20
132	28
132	29
132	30
132	22
122	17
122	18
122	19

Reserved seats

Denne tabel indeholder alle reserverede sæder samt den tilknyttede reservation.

- Reservation_id: Oplyser hvilken reservation, der har reserveret det pågældende sæde.
- Seat_id: Sædets nummer.

Movies

Denne tabel indeholder de film, som der vises i biografen.

- Title: Titel på film
- length: Filmens spilletid.

title	length
James Bond	120
James Bond – Casino Royal	120
James Bond – Goldfinger	120
James Bond – on her majestys secret service	120

Theaters

Denne tabel indeholder information om de forskellige sale.

- Hall_id: Salens nummer.

hall_id	seats	rows
1	10	10
2	8	8
128	3223	12
132	989898	14

- Seats: Sæder pr række.
- Rows: Antal rækker.

Shows

Denne tabel indeholder information om de forskellige forestillinger. Nedenfor ses en liste over tabellens kolonner:

- Show_id: Unikt, auto-genereret id. Primary key.
- Date: Dato for visning
- Time: Tidspunkt for visning
- Hall_id: I hvilken hal visningen foregår
- Title: Title på film der vises

Show_id	Date	Time	Hall_id	Title
10	2017-01-02	18:00:00	2	James Bond – Casino Royal
11	2017-01-02	20:00:00	1	James Bond – Casino Royal
12	2017-01-02	20:00:00	2	James Bond – On her majestys secret service
13	2017-01-02	22:00:00	1	James Bond – On her majestys secret service
14	2017-01-01	21:00:00	2	James Bond – Goldfinger
15	2017-01-01	23:00:00	1	James Bond – Goldfinger

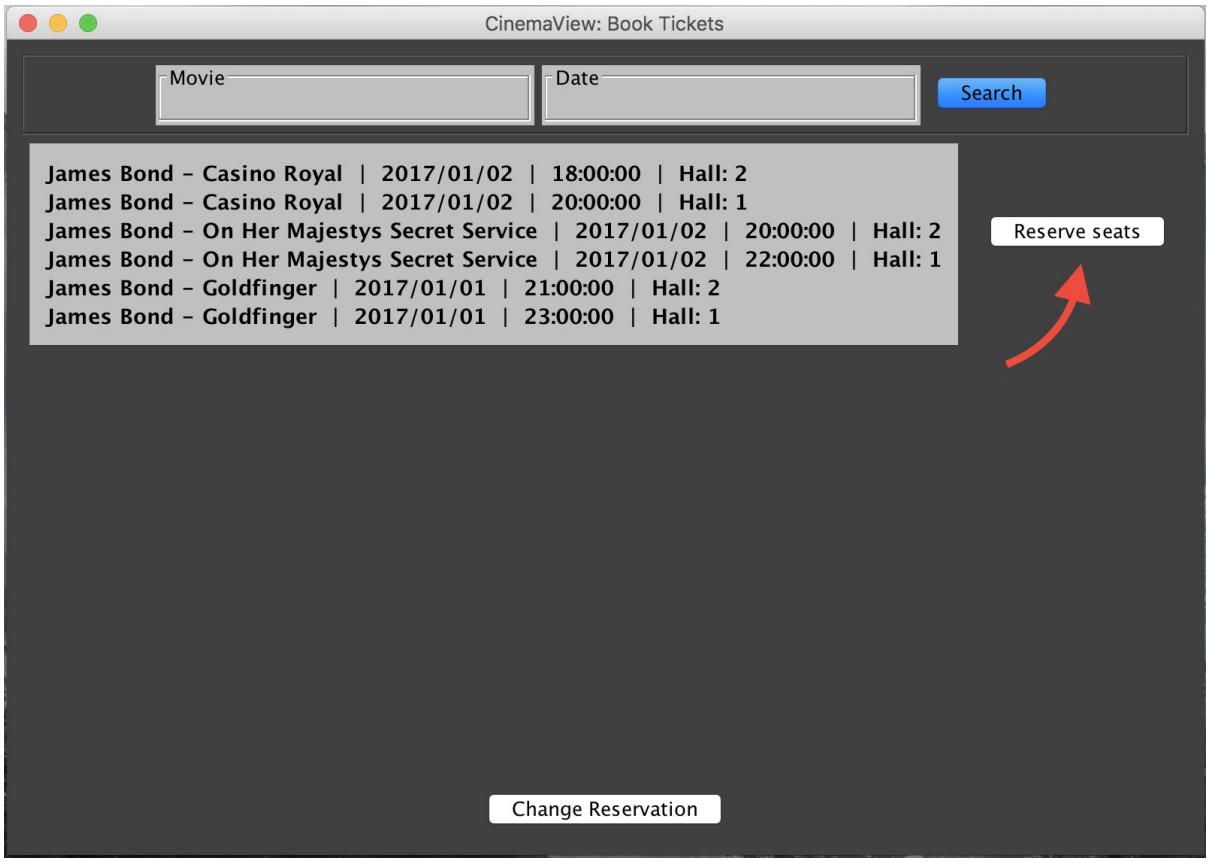
4. Brugervejledning og eksempel

Dette afsnit indeholder en detaljeret brugervejledning af det færdige program. Brugervejledningen indkluderer eksempler og use cases, for at give et bedre overblik over programmets funktioner i de forskellige situationer. Målgruppen for vejledningen er selvfølgelig brugeren af programmet, og det vil i dette tilfælde betyde ekspedienten i den pågældende biograf.

Programmet er bygget op at tre forskellige vinduer. Der er et vindue, der viser en liste over biografens forestillinger, hvor man herfra kan vælge, den der skal bookes. Der er endnu et vindue med en oversigt over biografsalene. Her ses biografsalens ledige og optagede sæder for den valgte forestilling, og det er også herfra at disse bookes. Til sidst er der et vindue, der gør det muligt at ændre allerede eksisterende reservationer. Programmet åbnes ved at åbne den vedlagte "jar" fil. Resten af brugervejledningen tager udgangspunkt i at programmet er åbent.

4.1 Oprettelse af en reservation

Når kunden ringer ind til biografen, og ekspedienten tager telefonen for at tage imod kundens reservation, åbnes programmet og det første vindue dukker op. Det er i dette vindue, at ekspedienten kan se hvilke forestillinger, der går i biografen og på hvilke dage og tidspunkter. Det er i dette vindue også muligt at søge på bestemte film eller datoer, hvis dette er relevant. Søgefelterne findes øverst i vinduet og er markeret med titlerne "Movie" og "Date". Man behøver ikke skrive hele filmens titel for at få den frem ved søgning. Vil man f.eks. finde "James Bond - Casino Royal", er det nok bare at søge på "Casino". Søgefeltet tager heller ikke højde for store og små bogstaver, så det har ingen indflydelse på søgeresultatet, om filmens titel skrives med småt eller stort. Søges der på en dato, skal det vides, at denne er opbygget som YYYY/MM/DD. Så skrives der 01/02 i søgefeltet, søges der på den 2. januar, hvor 2017/01/01 viser forestilling på den 1. januar 2017. Indtastes der noget i søgefelterne, skal der blot trykkes på "Search"-knappen eller Enter-tasten for at søge i listen af forestillinger.

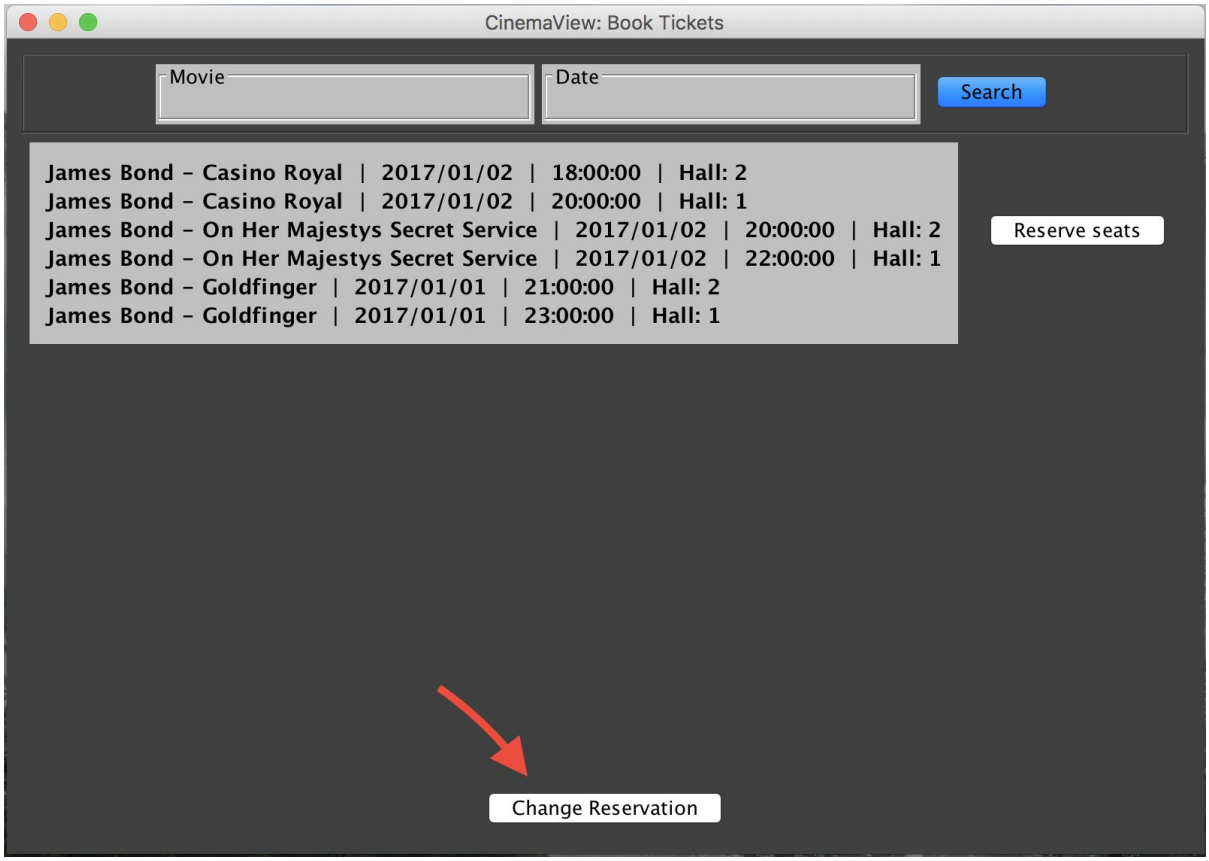


Ekspedienten modtager herefter information fra kunden om, hvilken film der skal bookes, og den forestilling, der stemmer overens med kundens ønsker, vælges. Der trykkes derefter på "Reserve seats"-knappen. Nu åbnes der et nyt vindue med en oversigt sæderne i den valgte sal.

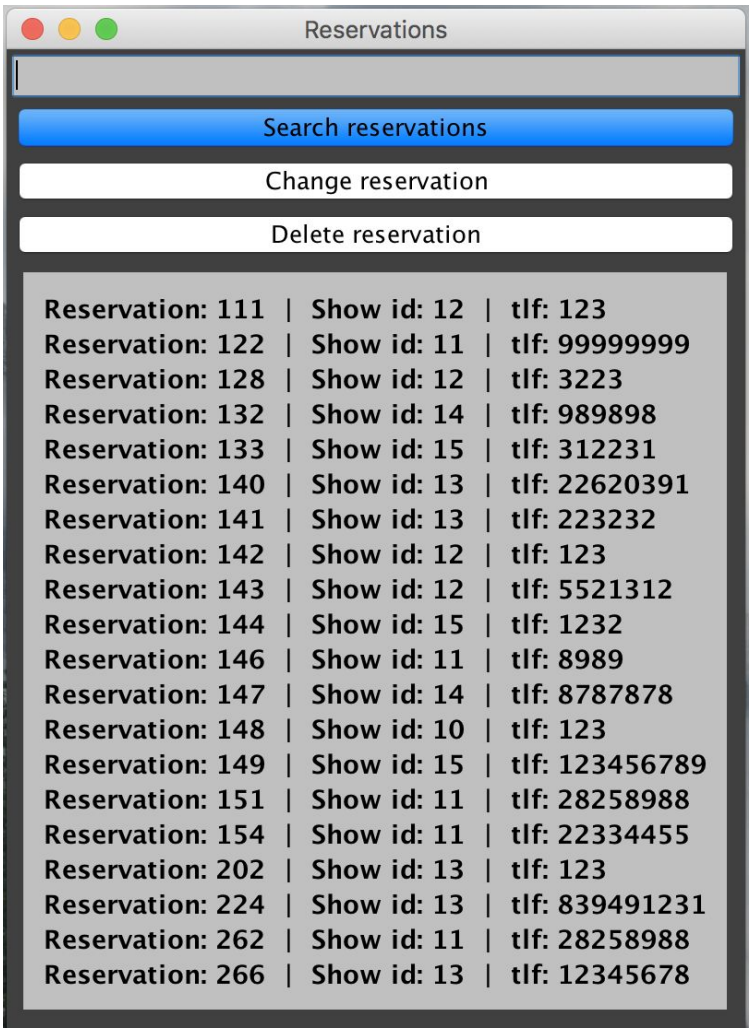
Er der allerede reserveret pladser til den valgte forestilling, vil disse være markeret med rød. De røde sæder kan ikke vælges igen. De resterende grønne sæder er alle ledige, og det er muligt at vælge disse. Ekspedienten skal nu forsøge at opfylde kundens ønsker, med hensyn til hvor i salen de helst vil placeres. Det eneste ekspedienten nu skal gøre er at trykke på de bedst mulige sæder, der opfylder kundens ønsker. De valgte sæder skifter farve fra grøn til blå, når der trykkes på dem med musen. Nu skal der bare trykkes på "Book now!"-knappen, og der vil dukke et pop-up vindue op, der beder om kundens telefonnummer. Dette indtastes af ekspedienten, og reservationen gemmes nu under det indtastede telefonnummer. Programmet vil vise en besked, der angiver, om reservationen er udført korrekt. Dette vises enten med en "Booking success" eller en "Booking failed" besked. Modtager man den førstnævnte, er reservationen fuldført, hvorimod den sidstnævnte betyder, at reservationen er udført forkert. Det er her nødvendigt at forsøge igen.

4.2 Ændring af eksisterende reservation

Ringer en kunde ind til biografen og beder om, at få en allerede eksisterende reservation ændret, så skal programmet åbnes, som hvis en ny reservation skulle laves, således at man ser det første vindue på skærmen. I stedet for at vælge en forestilling skal ekspedienten i stedet trykke på "Change Reservation"-knappen nederst i vinduet.



Når der med musen trykkes på denne knap, åbnes et nyt vindue, der hedder "Reservations". Dette indeholder en liste over de eksisterende reservationer i biografen. Det er her også muligt at søge i listen over eksisterende reservationer. Kundens telefonnummer kan nemlig indtastes i søgefeltet øverst på siden, og når der trykkes på "Search reservations"-knappen, vises der kun de reservationer, der ligger under det indtastede telefonnummer.



Det er fra dette vindue muligt at ændre en reservation på to måder:

1. Ønsker kunden at fjerne en reservation i deres telefonnummer, kan dette gøres ved at trykke på den pågældende reservation, sådan at den bliver markeret, og derefter trykke på "Delete reservation"-knappen. Reservationen er nu slettet fra systemet, og dette betyder også, at sæderne nu kan bookes igen og reserveres under et andet telefonnummer.
2. Ønsker kunden at ændre i en bestemt reservation, skal den pågældende reservation trykkes på og markeres på samme måde som i ovenstående situation. Men der skal i stedet trykkes på "Change reservation"-knappen, for at foretage ændringer. Nu åbnes et vindue med oversigten over biografsalen igen. Her er de sæder, der er knyttet til den valgte reservation, markeret med blåt. Det er disse sæder, der kan ændres. Skal der fjernes sæder fra reservationen, skal der blot trykkes på de blå sæder, for at gøre dem grønne, og dermed ledige, igen. Er det modsatte tilfældet, og skal der tilføjes sæder til reservationen, så markeres de nye, ekstra sæder, sådan at disse også bliver blå. Herefter trykkes der på "Update"-knappen, og reservationen er hermed ændret og opdateret.


CinemaView: Choose Seats
You have chosen James Bond - On Her Majestys Secret Service
Date: 2017-01-02. Time: 20:00:00
Hall: 2
Update
Screen

5. Teknisk beskrivelse af programmet

Dette afsnit indeholder en beskrivelse af programmets opbygning og struktur. Programmets designbeslutninger vil blive gennemgået, hvor der vil blive lagt vægt på brugergrænsefladens forskellige layouts, komponenter og tilhørende lyttere. Derudover beskrives databasen, hvor tabellernes interne relationer uddybes.

5.1 Brugergrænseflade

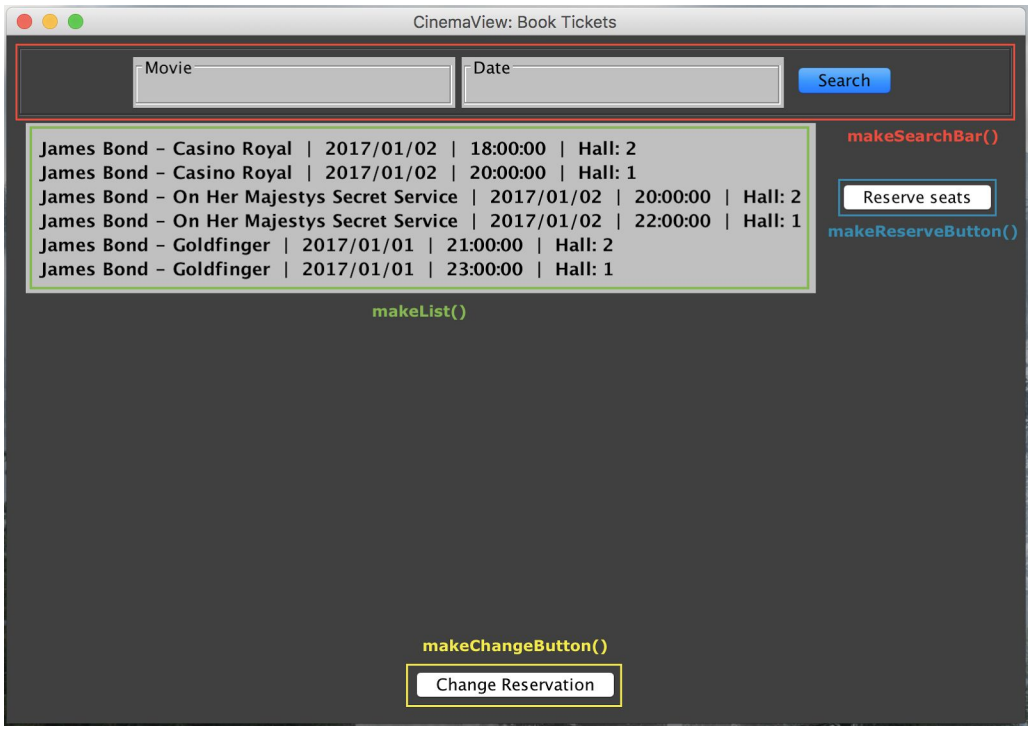
Brugergrænsefladen går under MVC-modellens View, og den består, som tidligere nævnt, af tre forskellige klasser: BookingView, CinemaView og ReservationView. Da programmet er et vinduesorienteret program, har hver brugergrænseflade-klasse en makeFrame()-metode, der har til formål at lave de enkelte vinduer med alle tilhørende komponenter. Så hver klasse kalder hver sin makeFrame()-metode, og programmet består derfor også af tre forskellige vinduer. Klassernes makeFrame()-metoder kalder også en række private metoder, der indeholder adskillige swing-komponenter, hvor nogle af dem har tilhørende lyttere for at få brugergrænsefladen til at samarbejde med programmets Controller og vise programmet, når det bliver kørt.

5.1.1 BookingView

I klassen BookingView tilføjer **makeFrame()**-metoden et JPanel contentPane til vinduet, der har et BorderLayout. Layoutets funktion er at opdele vinduets JComponents i forskellige sektioner, for at gøre brugergrænsefladen mere overskuelig. ContentPane indeholder fire brugte inddelinger, der henholdsvis er placeret i BorderLayoutets nord, syd, øst og center-sektioner:

- **makeSearchBar()**-metoden kaldes i den nordlige sektion, og denne består af et JPanel med et FlowLayout, der indeholder to JTextFields og en JButton. Søgeknappen har en ActionListener, der via if-statements undersøger, hvad der er skrevet i tekstfelterne, og på baggrund af dette opdaterer vores JList over forestillinger.
- **makeReserveButton()**-metoden er i den østlige sektion, og den returnerer en enkelt JButton men en ActionListener. Hvis intet er valgt på listen, så gør ActionListeneren intet. Hvis der er valgt en forestillingen på listen, så hentes denne forestillings ID i listModel, hvor efter det gemmes i Controlleren, og der bliver lavet en ny instans af klassen CinemaView.
- **makeChangeButton()**-metoden er i den sydlige sektion. Her laves endnu en JButton med en ActionListener. Når der trykkes på knappen, laves en instans af ReservationView-klassen.
- **makeList()**-metoden tager parameteren TreeMap<Integer, String>, og den returnerer et JPanel i center-sektionen. I denne metode tages vores TreeMap og

det deles i Strings og keys, hvor strengene bliver visualiseret i en JList, og keys bliver sat ind i en DefaultListModel. Det er JList, der bliver vist i vinduet.



5.1.2 CinemaView

Vores visning af CinemaView illustrerer for ekspedienten hvilke sæder, der til en bestemt visning af en film, er optaget.
Framen bliver bygget ved kaldet af metoden **makeFrame()**, som ud fra en række metoder laver hele vinduet. Framen er opbygget af en række containers med forskellige layouts. Det generelle layout for contentPane er et BorderLayout. ContentPane bruger fire sektioner af BorderLayoutet. Den nordlige sektion indeholder tekst, der beskriver den forestilling, ekspedienten har valgt. Denne tekst bliver genereret af metoden:

- **makeShowLabel()**: Denne metode benytter de feltvariabler, som instansen blev oprettet med. Den arrangerer dette i JPanels ved brug af BorderLayouts. Der returneres et overordnet JPanel, der tilføjes til contentPane.

I højre side af BorderLayoutet i vores contentPane findes en knap til at oprette en reservation eller opdatere en reservation. Om knappen er til at oprette eller ændre reservation er baseret på en boolean, som objektet bliver oprettet med. Der kaldes en af to metoder, som opretter hver sin knap. Den bestemmende boolean har forskellig værdi afhængig af, om objektet bliver oprettet fra BookingView eller ReservationView.

- **makeBookingButton()**: Denne metode opretter et JPanel med en knap til at oprette en ny reservation. Denne knap har en ActionListener, der kalder metoden getCustomerNumber(), der returnerer et telefonnummer. Når brugeren har givet et telefonnummer, kaldes metoden createReservation()-metoden i Controlleren med den nødvendige information.

- **makeUpdateButton()**: Metoden opretter et JPanel med en JButton. Denne knap har en ActionListener, der kalder metoden updateReservation() i Controlleren, der opdaterer den valgte reservation med nye sæder.
- **getCustomerNumber()**: Denne beder brugeren om et telefonnummer i et popup vindue. Den kaster også en exception, hvis inputtet ikke er gyldigt.

Nederst i vinduet i den sydlige sektion har vi en "screen", der skal illustrere hvor lærredet i biografen befinder sig i forhold til sæderne. Lærredet oprettes af metoden:

- **makeScreen()**: Metoden returnerer et JPanel. Dette JPanel indeholder en JLabel, hvori der blot står "Screen". JLabelen har en border, der udgør kanten på lærredet.

I center-sektionen i BorderLayoutet for contenPane vises der en visuel repræsentation af biografsalen i vinduet. Her skildres også mellem hvilke sæder, der er valgt, hvilke der er optagede og hvilke sæder, der er ledige. Denne visning bliver lavet i metoden:

- **makeGrid()**: Her laves et JPanel, der indeholder alle sæderne i salen arrangeret i rækker. Metoden itererer gennem to for-løkker og tegner sæderne som JButtons. Knapperne tegnes forskelligt afhængig af, om sædet er optaget, highlighted eller ledigt.

5.1.3 ReservationView

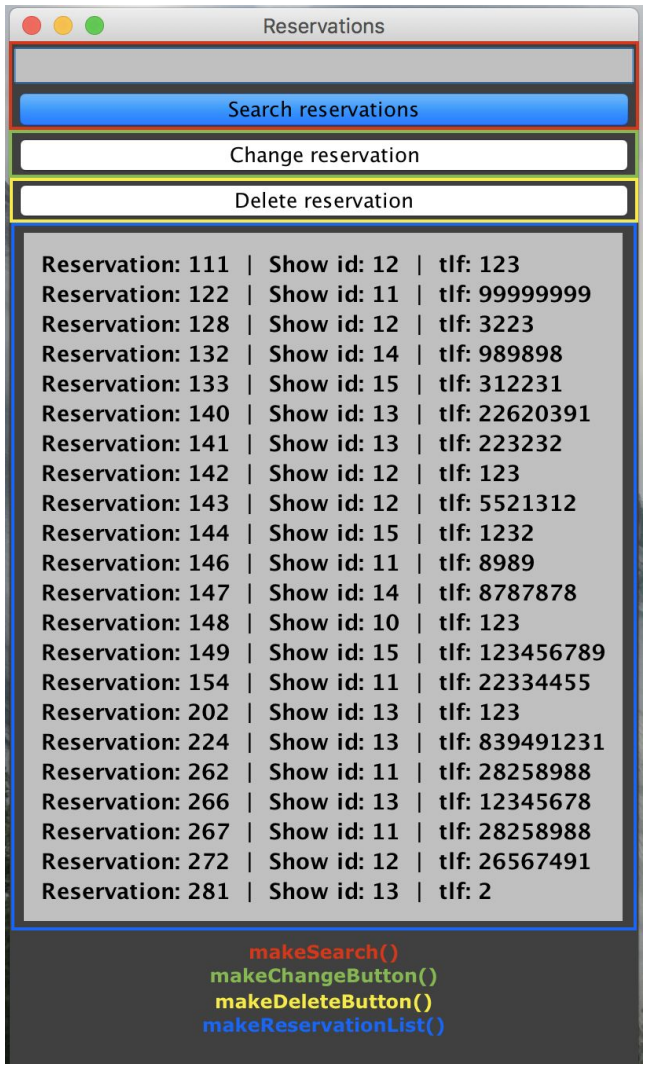
ReservationViews contentPane er opdelt i et BorderLayout. I den nordlige sektion er der indsat et JPanel(1), som også er opdelt i et BorderLayout. I dette BorderLayouts center-sektion er der indsat endnu et JPanel(2) med endnu et BorderLayout, hvor der i

dette BorderLayouts nord-sektion er placeret et JPanel(3), hvilket har et GridLayout, hvori der er placeret et JTextfield, der bruges til indtastning af kundens telefonnummer, samt en JButton med teksten "Search reservations". Denne JButton har en ActionListener, der tager input fra tekstfeltet, hvorefter metoden updateList(Controller.getReservationByPhone()) bliver kaldt, hvilket henter alle reservationer fra databasen, der er reserveret med det indtastede telefonnummer.

I center-sektionen er der placeret en JButton med teksten "Change reservation", der har en ActionListener, der kalder metoderne Controller.storeReservationID(a) og Controller.displayReservation(a), der åbner et CinemaView preloaded med de reserverede sæder tilknyttet den pågældende reservation.

I den sydlige sektion er en JButton med teksten "Delete reservation" placeret der, ligesom changeReservation()-knappen, henter det givne reservation_id og derefter kalder Controller.deleteReservation(a) med det valgte.

I JPanel(2)'s center bliver der indsat et JPanel, hvilket bliver returneret af makeReservationList(treeMap), der fungerer på samme måde som i BookingView.



5.2 Databasedesign

Under problemanalysen blev databasens opsætning kort beskrevet. I det følgende afsnit vil vi gå i dybden med tabellernes relation til hinanden, og dermed hvorledes de er oprettet. For at skabe et bedre overblik vil der blive taget udgangspunkt i SQL-kode eksemplet nedenfor, og derefter vil der blive nævnt for hvilke andre kolonner, det samme er gældende.

```
CREATE TABLE shows (
        show_id INT PRIMARY KEY AUTO_INCREMENT,
        date DATE NOT NULL,
        time TIME NOT NULL,
        hall_id INT,
        FOREIGN KEY(hall_id) REFERENCES theaters(hall_id),
        title VARCHAR(100),
        FOREIGN KEY(title) REFERENCES movies(title)
)ENGINE=InnoDB;
```

Kolonnen show_id er blevet dannet med følgende to egenskaber: PRIMARY KEY samt AUTO_INCREMENT. Vi har valgt at gøre show_id til tabellens primary key, således at vi kan referere til dette show fra andre tabeller. Dette er også gældende for følgende kolonner:

- Theaters: hall_id
- Customers: tlf_nr
- Reservations: reservation_id
- Movies: title

AUTO_INCREMENT sikrer, at hvert id er unikt, og at man ikke ved en fejl kan overskrive ældre shows eller lignende. Dette gælder også for følgende kolonner:

- Theaters: hall_id
- Reservations: reservation_id

Det sidste keyword vi har brugt er FOREIGN KEY REFERENCES. Dette betyder, at vi kan referere til primary keys, der allerede eksisterer i databasen. På denne måde undgås fejl såsom at lave reservationer i ikke eksisterende sale. Dette er gældende for:

- Shows: hall_id → theaters(hall_id)
- Shows: title → movies(title)
- Reservations: tlf_nr → customers(tlf_nr)
- Reservations: show_id → shows(show_id)
- Reserved_seats: reservation_id → reservations(reservation_id)

6. Afprøvning

Dette afsnit beskriver hvorledes programmet er testet, herunder en test af brugergrænsefladen samt udvalgte unit tests.

6.1 Afprøvning af brugergrænseflade

På baggrund af brugertest af brugergrænsefladen konkluderer vi, at programmet løser alle de specifikke krav vi bestemte i problemstillingen:

- Gemme et ID til brug ved genfindelsen af en reservation
- Ændre en reservation, f.eks. ændre antallet eller placeringen af pladser
- Slette en allerede eksisterende reservation
- Huske pladser en kunde har valgt
- Give en visuel repræsentation af salen, der vises
- Vise forestillinger, ikke kun på det præcise klokkeslæt angivet, men også nærliggende forestillinger
- Finde alle forestillinger, hvor en bestemt film vises
- Finde alle forestillinger, der vises et bestemt klokkeslæt
- Vise alle fundne forestillinger i listeform

Vi konkluderer yderligere, at programmet grafisk er simpelt og funktionelt, hvilket var, hvad vi ville opnå med vores design.

Følgende ændringer af GUI'en er stadig på vores to-do liste:

- Opsætning af kolonner i listen over henholds reservationer og showings i ReservationView og i BookingView.

6.2 Unit test

Det er ikke alle delene af programmet, der kan testes ved at afprøve GUI'en, og der kan hurtigt opstå fejl, hvis man antager, at metoder virker på en bestemt måde, men i virkeligheden ikke fungerer som forventet. Som eksempel kan nævnes off-by-one fejl, hvilket forekommer ofte. Derfor har vi valgt at lave unit tests på hovedparten af vores program. Selvom det umiddelbart virker absurd på nogle af metoderne, har vi alligevel valgt at inkludere disse i bilagene, dog vil vi her kun omtale MySqlConnectionTest().

Da hovedparten af testmetoderne i MySqlConnectionTest() er en evaluering af, om vi får de forventede data tilbage fra databasen, har vi valgt at undlade disse fra dette afsnit og i stedet tage et kig på en af de mere interessante testmetoder - MySqlConnectionTest(), mere specifikt følgende testmetode, der tester følgende metoder i klassen MySqlConnection:

- createUpdateDeleteReservationTest()
    - createReservation()
    - updateReservation()
    - deleteReservation()

For at kunne udføre unit tests har vi gjort brug af "org.junit.Assert" klassen, hvor vi primært bruger klassens AssertEquals(expected value, actual value) metode, der sammenligner to værdier og vurderer om testen fejlede eller ej. Men enkelte gange bruges metoden AssertFalse(boolean expression) til tests, hvor vi forventer, at metoden der testes returnerer false.

Vi har valgt at opbygge metoderne createReservation(), updateReservation() og deleteReservation(), således at de alle returnerer en boolean, der indikerer om metodekaldet lykkedes eller ej, hvilket gør det muligt for os at teste på en simpel og overskuelig måde, selvom at hver metode har forskellige succeskriterier.

CreateUpdateDeleteReservationTest() tester også alle ovennævnte metoder ved brug af samme data, således at der oprettes en reservation. Denne reservation opdateres, og til sidst slettes den igen.

I det følgende afsnit redegøres der for, hvordan unit tests bliver brugt til at teste metoden createReservation(Reservation r).

```
69      @Test
70      public void createUpdateDeleteReservationTest() {
71          // initialise b and populate with seats we want to reserve
72          int[] b = new int [3];
73          b[0] = 50;
74          b[1] = 51;
75          b[2] = 52;
76
77          // create new reservation
78          Reservation r = new Reservation(666,10,b);
79
80          // send reservation to DB
81          assertEquals(true,MySqlConnection.createReservation(r));
82
```

Uddrag af createUpdateDeleteReservationsTest() fra MySqlConnectionTest().

Vi opretter et reservations-object, hvorefter at vi i vores unit test antager, at det lykkedes at oprette dette i databasen.

```
423          //reserved the seats related to this reservation in database
424          if(createReservedSeats(r.getReserved_seats(), reservation_id)){
425              //returns true
426              connection.close();
427              return true;
428          }
```

Uddrag af createReservedSeats() fra MySqlConnection.

Metoden createReservation() returnerer også en boolean, og den returnerer true, hvis antallet af entities created i databasen stemmer overens med antallet af elementer i det int[], der passes som argument. På denne måde evalueres vores test af createReservation() som true, hvis antallet af faktiske oprettede sædereservationer stemmer overens med det antal, vi gerne vil reservere. På samme måde evalueres updateReservation() som true, hvis både deleteReservation() og createReservedSeats() evalueres til true. DeleteReservation() evalueres også som true, hvis antallet af rows slettet i databasen er større end 0.

De andre tests i klassen MySqlConnectionTest() afhænger af data fra databasen, hvilket ikke er noget problem, så længe dataen findes i databasen. Men hvis dette ikke er tilfældet, skal de implicerede tests revideres. Dette problem kan løses ved at oprette en identisk test-database, der bruges, så længe der køres tests.

På baggrund af vores unit tests, kan vi konkludere at indskrivning af data fra programmet til database, samt fra databasen til programmet, virker efter hensigten, så længe at den data, der forsøges hentet fra databasen, eksisterer. Programmet indeholder ingen unit tests af vores View, men der kan argumenteres for, at selvom vi ikke sætter tal på om vores View fungerer efter hensigten, så kan vi visuelt bekræfte, at det er den rigtige data, der bliver visualiseret. Således konkluderer vi på baggrund vores afprøvning af programmet, at vores tests af programmet er passende, da både kommunikationen til databasen og vores visuelle repræsentation af dataen virker efter hensigten.

7. Konklusion

Baseret på problemstillingen har vi udviklet et program til brug i billetlugen i en mindre biograf. Programmet har en nem og overskuelig brugergrænseflade, der understøtter ekspedientens arbejdsopgaver.
Som en del af processen udarbejdede vi en domæneanalyse, for at anskueliggøre problemstillingen. På baggrund af domæne analysen har vi udarbejdet en løsning som vi efterfølgende har implementeret.
Programmet kan give ekspedienten et overblik over forestillingerne i biografen. For hurtigt at kunne finde en bestemt forestilling kan ekspedienten søge en forestilling frem ud fra enten titel eller dato.
Når en forestilling er valgt kan programmet vise forestillingen, i en bestemt sal. I salen viser programmet hvilke sæder der er optagede, samt giver en ekspedient muligheden for at vælge sæder. Der kan herefter oprettes en reservation med de valgte sæder.
Hvis en kunde ønsker at ændre en reservation kan ekspedienten få et overblik over eksisterende reservationer, hvor ekspedienten har mulighed for at søge på reservationer ud fra kundens telefonnummer. Derudover kan de valgte sæder der er knyttet til en bestemt reservation ændres, eller en reservation kan slettes helt.
For at gemme informationen om kunder, reservationer og forestillinger benytter programmet en relationel database. Dette gør at informationer kan gemmes mellem kørsler af programmet.
Programkoden er testet med en række unit test, for at teste at koden fungerer som tiltænkt. Da brugergrænsefladen ikke kan testes ved brug af unit test, er brugergrænsefladen blevet testet manuelt. Vores unit test har alle været succesfulde, og vores test af brugergrænsefladen viser at den lever op til kravene specificeret i problemstillingen.
Vi kan altså konkludere at vores program opfylder de ønskede krav fra projektbeskrivelsen, samt vores egne krav til projektet, beskrevet i problemstillingen.

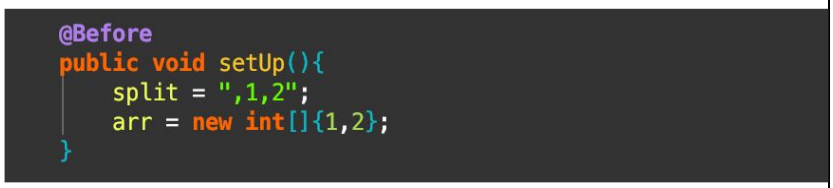
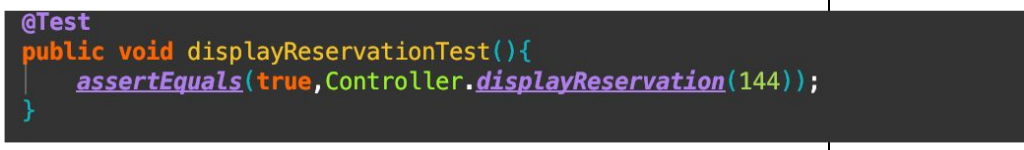
8. Refleksion over projektforløbet

Vi føler i gruppen, at vi har fået rigtig meget ud af dette forholdsvis korte projektforløb. Vi er blevet kastet ud i at programmere noget, vi ellers ikke havde nogen erfaring med at lave førhen, og hvad der i starten måske så ud til at være en stor udfordring, viste sig senere hen at være en til tider svær men lærerig og løselig opgave. Det, at gruppen kun har bestået af tre medlemmer, har gjort det væsentligt nemmere at få hele gruppen engageret i projektet. Samtidig havde ingen af os nogen egentlig erfaring med at programmere, før vi startede på uddannelsen, så det, at vi har kunnet skabe et program fra bunden, har udgjort en rigtig god læringsproces.

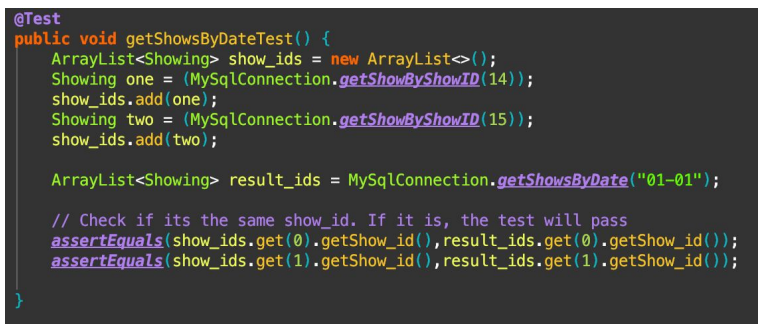
Vi har under hele forløbet arbejdet i Google Drive og SourceTree, hvilket har gjort det muligt at have både vores rapport og vores kode samlet på et sted, sådan at alle altid har været opdateret og haft et overblik over det samlede produkt. Dette har helt klart været til fordel for projektets fremdrift, da det har gjort det nemmere at arbejde på forskellige ting på samme tid og stadig kunne dele det, man har lavet, med resten af gruppen. Vi valgte i starten af forløbet at bruge et andet IDE end BlueJ, og vi har i stedet arbejdet og skrevet koden i IntelliJ. Det skift krævede dog at vi først satte os ind i, hvordan IntelliJ fungerede, og denne tid kunne jo have været sparet, hvis vi havde valgt at arbejde i BlueJ, som vi i forvejen havde kendskab til. Det har dog ikke udgjort nogen synderlig udfordring, og vi kan nu bruge vores erfaring med IntelliJ i senere forløb på uddannelsen.

9. Bilag

9.1 Tests

ControllerTest	
Variabler til test of splitSeatString	
Tester om splitSeatString virker ved at teste om den splitter strengen ",1,2" korrekt	
Tester om programmet kan åbne en given reservation i et cinemaView således denne kan rettes.	
Tester om det lykkedes at oprette en reservation, og efterfølgende slette denne igen.	
Tester om det lykkedes at oprette et nye reservationView	
Tester om det lykkedes at opdatere en reservation med nye sæder	

MySqlConnectionTest	
Tester om hall_id 1 & 2 er i databasen	
Tester om getShowByShowID returnere show_id's fra 10 til 16.	
Asserts that getReservationByShowID(10) returnere en arrayliste der indeholder 148	
Tester om getShowByReservationID(rese rvation_id) returnerer 10, så der gives reservation_id = 148.	
Denne testmetode tester om det lykkedes programmet at oprette, ændre og slette en reservation.	

Denne metode tester om det lykkedes at opdatere en eksisterende reservation med et sæt nye sæder.	
Tester om det lykkedes at hente et givent antal reservation lavet med et bestemt telefonnummer, i dette tilfælde skal metode returenre en arrayliste med (111,142,148,292) når der passes "123" som argument.	
Tester om de sædenumre der returneres fra getReservedSeats(146), er 95 og 96.	
Tester om getShowing returnere alle show_ids, hvilket gerne skulle være 10-16	
Tester at når der søges efter forestillinger den "01/01", at så returneres show 14 og show 15	

Tester at det er show med id 10 og 11 der returneres, når der søges efter "casino"

```java
@Test
public void getShowsByTitleTest() {
    // we know that the shows in the DB that will be returned when the string "casi
    ArrayList<Showing> show_ids = new ArrayList<>();
    Showing one = (MySqlConnection.getShowByShowID(10));
    show_ids.add(one);
    Showing two = (MySqlConnection.getShowByShowID(11));
    show_ids.add(two);

    ArrayList<Showing> result_ids = MySqlConnection.getShowsByTitle("casino");

    assertEquals(show_ids.get(0).getShow_id(),result_ids.get(0).getShow_id());
    assertEquals(show_ids.get(1).getShow_id(),result_ids.get(1).getShow_id());

}
```

HallTest

Simple tests, ingen forklaring

```java
private int hall_id;
private int seats;
private int rows;
private Hall h;

@Before
public void setUp(){
    hall_id = 1;
    seats = 5;
    rows = 10;
    h = new Hall(hall_id,seats,rows);
}

@Test
public void testGetRows(){
    assertEquals(rows,h.getRows());
}

@Test
public void testGetSeats(){
    assertEquals(seats, h.getSeats());
}
```

ReservationTest
Simple tests, ingen forklaring

```java
private int phone;
private int showID;
private int[] seats;
private int resID;
private Reservation r;
private Reservation rID;

/**
 * Sets up the test fixture.
 *
 * Called before every test case method.
 */
@Before
public void setUp()
{
    seats = new int[]{1,2,3};
    phone = 12346578;
    showID = 5;
    resID = 3;
    r = new Reservation(phone, showID, seats);
    rID = new Reservation(resID, phone, showID);
}

@Test
public void testGetPhone(){
assertEquals(phone, r.getTlf_nr());
}

@Test
public void testGetShowID(){
    assertEquals(showID, r.getShow_id());
}


@Test
public void testGetReservedSeats(){
    for(int i = 0; i < r.getReserved_seats().length; i++)
    assertEquals(seats[i], r.getReserved_seats()[i]);
}


@Test
public void testGetReservationID(){
    assertEquals(resID,rID.getReservation_id());
}

@Test
public void testNonEqualResID(){
    assertFalse(resID == r.getReservation_id());
}
```

ShowingTest
Simple tests, ingen forklaring

```java
int show_id;
Date date;
String time;
int hall_id;
String title;
Showing s;

@Before
public void setUp(){
    show_id = 2;
    date = new Date(2016,8,22);
    time = "1900";
    hall_id = 1;
    title = "James Bond";
    s = new Showing(show_id, date, time, hall_id, title);
}

@Test
public void testShowID(){
    assertEquals(show_id,s.getShow_id());
}

@Test
public void testGetDate(){
    assertEquals(date, s.getDate());
}

@Test
public void testGetTime(){
    assertEquals(time,s.getTime());
}

@Test
public void getHallID(){
    assertEquals(hall_id,s.getHall_id());
}

@Test
public void getTitle(){
    assertEquals(title, s.getTitle());
}


@After
public void tearDown(){}
```

9.2 CRC-kort

Billetlugen	Forestilling Database Reservation Kunde View
getShows() showTeater() createCustomer() makeReservation() addToReservation() moveReservation() removeReservation()	

Film	Billetlugen
Titel Director Length	

Reservation	Billetlugen
Kunde Forestilling Sæde(r)	

View	Billetlugen
BookingView CinemaView	

Biografsal	Billetlugen
Antal sæder Antal rækker Seat Collection	

Database	Billetlugen
SqlConnection Hente og opdatere data i databasen	

9.3 Projektlogbog

Under projektet har vi løbende skrevet en logbog, hvor vi kort har beskrevet arbejdsprocessen hver dag.

24/11-16
Projektopgave præsenteret. Derefter kort møde omhandlende hvordan opgaven gribes an.

29/11-16
Analyse af problemstilling ved brug af crc kort. Besluttet at vi benytter mvc pattern

30/11-16
Vi har i dag designet vores database, og oprettet de tilsvarende tabeller. Derudover har vi besluttet os for at bruge intellij til at skrive java, samt github til deling af filer.

1/12-16
I dag arbejdede vi videre på vores database design, og er begyndt at designe interfaces til vores database klasse. Yderligere har vi kigget på queries til databasen.

5/12 - 16
Vi har arbejdet med database udtræk, og at få vist reservationer og bestemte visninger af film i vores GUI.

8/12-16
I dag sad vi efter vores sidste forelæsning og arbejdede på projektet, vi fik søgefunktionerne til at virke, og fik lavet alle komponenterne til at kunne opdatere en reservation.

9/12-16
Da Cæcilie var syg i dag besluttede vi os for at arbejde hjemmefra i dag. Vi har skrevet på rapporten og arbejdet på småting i programmet

13/12-16
I dag har vi for alvor taget hul på den tekniske del af rapporten. Vi regner med i morgen at kunne blive færdige med rapporten, således at vi kan bruge torsdag på at læse korrektur.

14/12-16

Vi skrev i dag på rapporten, således at vi kun mangler at skrive konklusion og læse korrektur i morgen.

15/12-16
I dag har vi færdiggjort rapporten, og ordnede bilag.

16/12-16
I dag afleverede vi.