Grundlæggende programmering

Course code: BSGRPRO1KU

Course manager: Claus Brabrand – Brabrand@itu.dk

Project title: playIT

Supervisor: Signe Kyster – Signek@itu.dk

Group AQ:

Damien Støvring Rolighed – damr@itu.dk

Sebastian Krogh Olsen – skol@itu.dk

Tobias Emil Birch – tbir@itu.dk

IT-Universitet i København

Afleveringsdato: 20-12-2019

Indholdsfortegnelse

Forord og indledning

Denne rapport er udarbejdet i et 4-ugers projektforløb i kurset Grundlæggende Programmering på 1. semester af Softwareudvikling bacheloren på IT Universitet. Vejleder i projektet har været Signe Kyster. I projektforløbet har vi udviklet playIT, som er en streamingtjeneste, hvor man kan se film og serier.

2. Problemstilling og baggrund

2.1. Problemområde

Vores program skal fungere som en streamingtjeneste. Brugeren kan logge ind på tjenesten med sit brugernavn og kodeord. Brugeren vil herefter blive ledt hen til film- og serieoversigten, hvor man kan se film og serier samt sortere i disse. Film og serier er inddelt i kategorier som fx krimi, drama og action. Sortering kan ske ved at vælge kategorier eller ved at skrive i søgefeltet.

2.2. Brugermæssige krav til programmet

Ud fra projektbeskrivelsen har vi formuleret nogle brugermæssige krav til programmet. Disse krav er formuleret som handlinger, en bruger ville kunne lave i programmet.

ID	Brugerhandlinger
1	Brugeren vil gerne sortere i film og serier ud fra kategorier.
2	Brugeren vil gerne søge på film og serier ud fra mediets navn.
3	Brugeren vil gerne gemme og fjerne film og serier i “Min liste”.
4	Brugeren vil gerne skifte bruger.

5	Brugeren vil gerne se information om filmen, serien, sæsonen eller episoden.

Tabel 1: Brugermæssige krav til programmet

De største krav til vores program er, at søgefunktionen fungerer optimalt. Således vil en bruger kunne få lige præcis de medier frem, som der bliver søgt efter. Hvis dette lykkedes, vil mange af brugerkravene blive opfyldt, som det ses ud fra tabel 1.

2.3. Systemmæssige krav

I tabel 2 er der oplistet de systemmæssige krav, der er til vores program. Programmet skal køre på en computer, hvor al dataen allerede ligger på, og hvor databehandlingen også sker.

ID	Systemmæssige krav
1	Skal programmeres i Java.
2	Gør brug af Java Swing eller JavaFX til opbygning af brugergrænsefladen.
3	Skal hente data om film og serier fra .txt-filer

Tabel 2: Systemmæssige krav.

De data som hjælper med at løse de brugermæssige krav i afsnit 2.2. er beskrevet i tabel 3.

Data	Beskrivelse
Film	De film som er tilgængelige i vores program
Serier	De serier som er tilgængelige i vores program
Bruger	Brugeren af programmet har sin egen brugerprofil.
Min Liste	Brugerens egen favoritliste af film og serier.

Tabel 3: Beskrivelse af data i programmet

3. Problemanalyse

3.1. Designbeslutninger vedrørende programmets struktur

Da vi skulle analysere vores givne problem, startede vi ud med at læse opgavebeskrivelsen igennem, og derefter gjorde vi brug af verb/noun-metoden på baggrund af denne. Ved brug af metoden kunne vi få et indblik i hvilke klasser og metoder, der eventuelt kunne være brugbare i vores program. De navneord vi fandt frem til skulle svare til klasserne, og udsagnsordene til metoderne som disse klasser skulle have.[1] For at undersøge hvordan de forskellige klasser interagerede med hinanden, brugte vi CRC-kort (Class/Responsibilities/Collaborators). Vha. disse kort gennemløb vi nogle scenarier, som man kunne forestille sig, at der kunne finde sted, hvis en bruger brugte programmet. Dette gav os bl.a. et godt indblik i, hvilke klasser der skulle arbejde sammen med hinanden, hvordan et foreløbigt klassediagram kunne opstilles og hvilke klasser og metoder der var nødvendige.[2] Gennem disse to øvelser fandt vi frem til, at vi skulle bruge en *search engine,* en *media controller* og en *GUI* som vores udgangspunkt til at lave programmet.[3]
Derfor har vi valgt at opbygge vores program efter MVC-strukturen (*Model-View-Controller*). Det betyder at vi får adskilt de store dele af programmet. De forskellige dele i MVC har hver sit ansvarsområde. *Model*, står for at holde og behandle vores database af film, men aldrig snakker med brugeren. *View*, står for at vise brugeren filmene, og lytter efter brugerens interaktioner. *Controller*, står for at håndtere brugerens interaktioner med programmet. *Controller* fungerer samtidigt som bindeled mellem *Model* og *View* delene af vores program. Således kommer *View* altså aldrig til at snakke med vores database af film (*Model*).

MVC-strukturen bringer flere fordele. For det første gør den det nemt for dem, der udarbejder programmet at fordele arbejdsopgaver. Siden programmet er så vel opdelt er det desuden nemt at teste individuelle dele af programmet, og til sidst at binde delene sammen. Således bliver det lettere at overskue komplekse problemer i programmet, ved at opdele dem i små bidder.[4]

[1] Objects first s. 558.
[2] Objects first s. 559
[3] Se eventuelt bilag 2.
[4] Brabrand, C. 2019.

3.2. Designbeslutninger vedrørende brugergrænsefladen

Umiddelbart efter vi havde planlagt programmets struktur, begyndte vi at planlægge brugergrænsefladens udseende og funktioner.

Vi startede ud med at lave en skitse på papir over brugergrænsefladen. Skitsen havde først og fremmest til opgave at hjælpe os med at få et overblik og kortlægge hvilke funktioner, vores program skulle indeholde. Det vil sige, at skitsen gav os et billede af hvilke mulige *sorteringsmetoder og -værktøjer*, vi senere ville skulle implementere i de andre dele af vores program. Denne første skitse nogle tanker i gang i forhold til hvordan, brugergrænsefladen skulle opbygges.[5]

Det næste skridt i processen var at fastslå, om vi skulle arbejde med Java Swing eller JavaFX. Som udgangspunkt havde ingen af os tre gruppemedlemmer kodet med hverken Swing eller JavaFX i forbindelse med undervisningen, og det var derfor blot at vælge én af de to at holde fast i og arbejde videre med. Dog viste det sig, at vores grundbog i grundlæggende programmering *Objects First with Java: A Practical Introduction Using BlueJ* benytter Java Swing til at introducere grundlæggende *GUI*-teori, hvilket endte med at være årsagen til, at vi valgte Java Swing. Efter vi valgte Java Swing, lavede vi en ny, mere detaljeret skitse til brugergrænsefladen, hvor vi inkluderede placering af *Layout Managers*, *JPanels,* andre Swing komponenter, mediernes billedikoner og så videre.[6]

Vi startede umiddelbart med at vælge, at have to forskellige "frames" - et frame med en login-skærm, og en anden frame, som indeholdt selve streamingtjenesten.
På login-skærmen valgte vi, at der blot skulle være to tekstfelter til brugernavn og kodeord samt en login-knap. Desuden valgte vi undervejs i processen af tidsmæssige årsager umiddelbart at fokusere på selve streamingtjenesten, da vi ikke så funktionen med en unik brugerprofil som en del af minimumskravene for programmet, men som et tilvalg.

Ved planlægningen af brugergrænsefladen for selve streamingtjenesten har vi taget udgangspunkt i velkendte streamingtjenester og inkluderet:

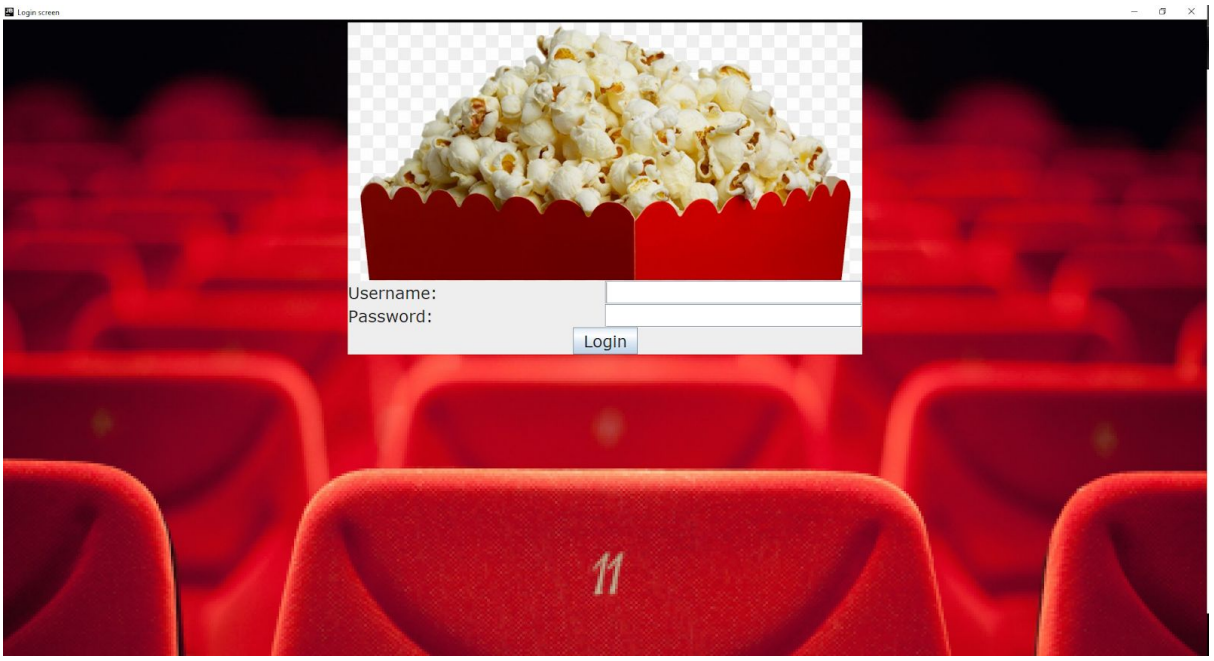
[5] Se eventuelt bilag 3.
[6] Se eventuelt bilag 4.

- Et øvre panel, som skal inkludere et søgefelt, de to overordnede kategorier “Movies” og “Series”, adgang til en personlig, brugerdefinerbar liste af medier (My List) samt en eventuel brugerprofil-knap.
- Et panel i venstre side, hvor man kan sortere biblioteket baseret på genrer. Det skal gøres med afkrydsningsfelter, så man kan få vist alle film og serier inden for én eller flere genrer.
- En bar i bunden, hvor der blot står en copyright formalitet, f.eks. “All rights reserved”.
- Et felt i midten, som fylder det meste af brugergrænsefladen og skal vise medierne med deres respektive billede og titel, og opdateres, hver gang brugeren sorterer i medierne. Når der klikkes på et billede tilhørende et medie, skal man vises information om mediet og kunne “afspille” det. Dette felt skal være dominerende.

Denne opbygning af brugergrænsefladen er valgt ved at imitere en klassisk streamingtjenestes grænseflade (“if it works, don’t change it”), dog med den ekstra funktion, at man let kan få vist medier fra flere forskellige genrer på én gang. Vi forestillede os scenariet med en online tøjbutik, hvor man kan sortere varesortimentet og kun få vist sine yndlingstøjmærker, bare med film og serier (e.g. “Jeg vil gerne se alle film og serier inde for genrene gyser, sport og komedie…”).

4. Brugervejledning

Når brugeren åbner vores program op, vil han/hun blive mødt af et login-vindue, hvorefter et succesfuldt login vil lede brugeren videre til kollektionen af film og serier. Login-vinduet ses på figur 1, og den dertil hørende vejledning ses under figuren.



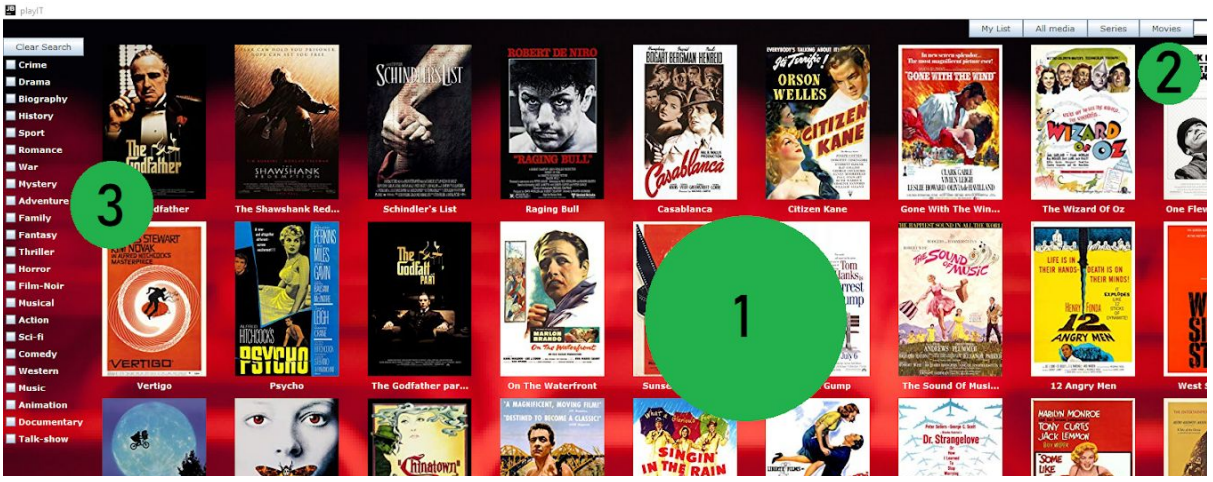
Figur 1: Screenshot af login-skærmen

For at logge ind og få adgang til programmets indhold skal følgende trin følges:

1. Brugeren bliver ved opstart af programmet mødt af en login-vindue.
2. Vinduet indeholder to tekstfelter. I det øverste tekstfelt skal brugeren skrive sit brugernavn til programmet.
3. I det nederste tekstfelt skal brugeren skrive sit kodeord til programmet.
4. Afhængigt af de informationer brugeren indtaster, vil følgende udfald ske, når brugeren trykker på login-knappen:
4.1. Hvis brugerens indtastede informationer er korrekte, vil brugeren blive ledt videre i programmet.
4.2. Hvis brugerens indtastede informationer er forkerte, vil brugeren få en meddelelse omkring dette frem på skærmen, og vil derefter kunne indtaste informationerne igen.

Når brugeren er logget ind vil programmets indhold kunne ses. Dette ses på figur 2.1 og figur 2.2., og under ses en vejledning til brug af programmet.

Figur 2.1: Screenshot af selve streamingtjenesten



Figur 2.2: Screenshot af streamingtjenesten med bobler, som viser placering af hovedkomponenter

1. I midten af skærmen ses programmets film- og serieindhold. Hvert medie har et billede og mediets titel står umiddelbart under. Ved at klikke ind på et medie vil der komme et nyt vindue frem. Vinduet indeholder informationer om mediet, har en play-knap samt to knapper, hvor man kan tilføje/fjerne film fra sin egen liste. Dette vindue ses på figur 3.

Figur 3: Screenshot af vinduet der kommer frem, når brugeren klikker ind på et medie.

2. I toppen af skærmen ses et panel med flere funktioner.

2.1. Knappen “My List” er en brugers personlige liste, hvor der kan tilføjes film og serier, som også kan fjernes igen.

2.2. Knappen “Series” viser alle serierne programmet indeholder.

2.3. Knappen “Movies” viser alle film programmet indeholder.

2.4. I tekstfeltet kan man indtaste en søgetekst og enten trykke enter eller trykke på “Search”-knappen. Dette vil bringe alle de medier frem, som indeholder en del af søgeteksten i deres titel.

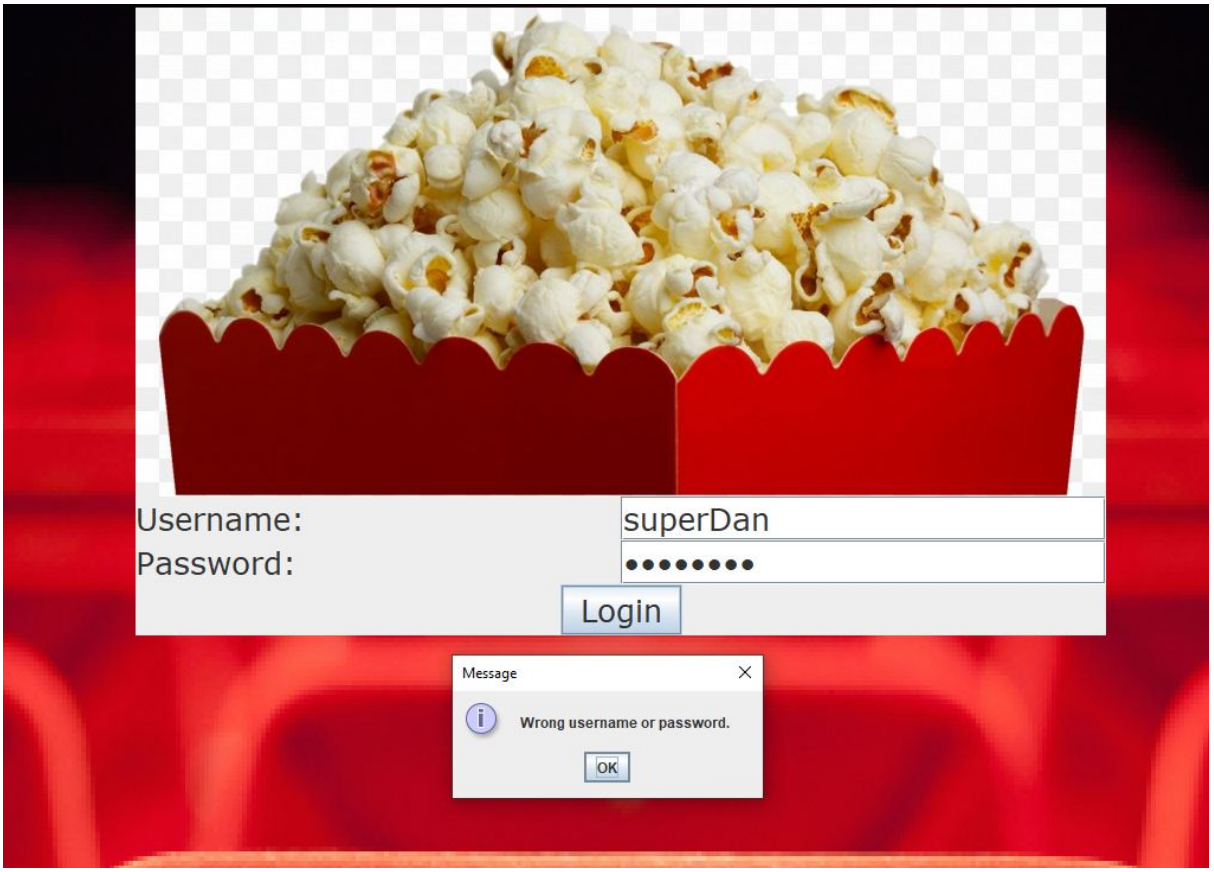
3. Det venstre panel i programmet indeholder checkboxe for alle de kategorier, som de forskellige film og serier er opdelt i. Hvis en box er checket af, vil der blive vist medier, som er af den valgte kategori. Over checkboxene er der en *clear search* knap, som rydder boxene, hvis de er checket af, og rydder den igangværende søgning sådan at hele biblioteket vises igen.

4.1. Fejlmeddelelser i programmet

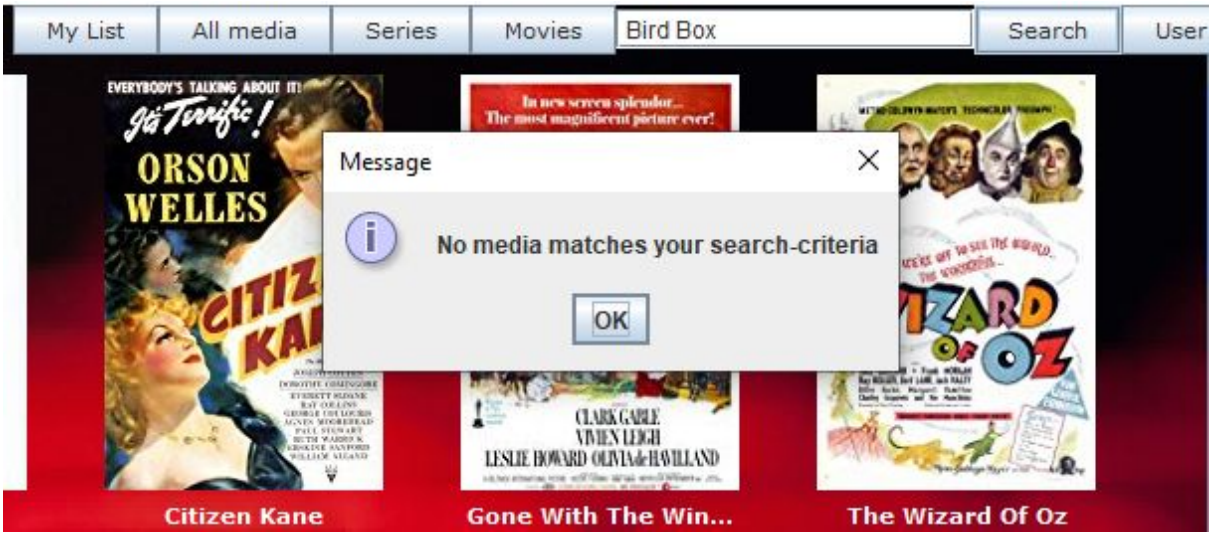
Programmet indeholder to fejlmeddelelser, som brugeren får vist, hvis der opstår en fejl.

1. *"Wrong username or password."* - vises hvis brugeren oplyser forkerte logindetaljer. Se figur 4.

2. *"No media matches your search-criteria"* - vises hvis brugeren indtaster en søgetekst, som ikke giver noget matchende resultat. Se figur 5.



Figur 4: Dialogboks, der popper op, hvis brugeren oplyser forkerte logindetaljer.

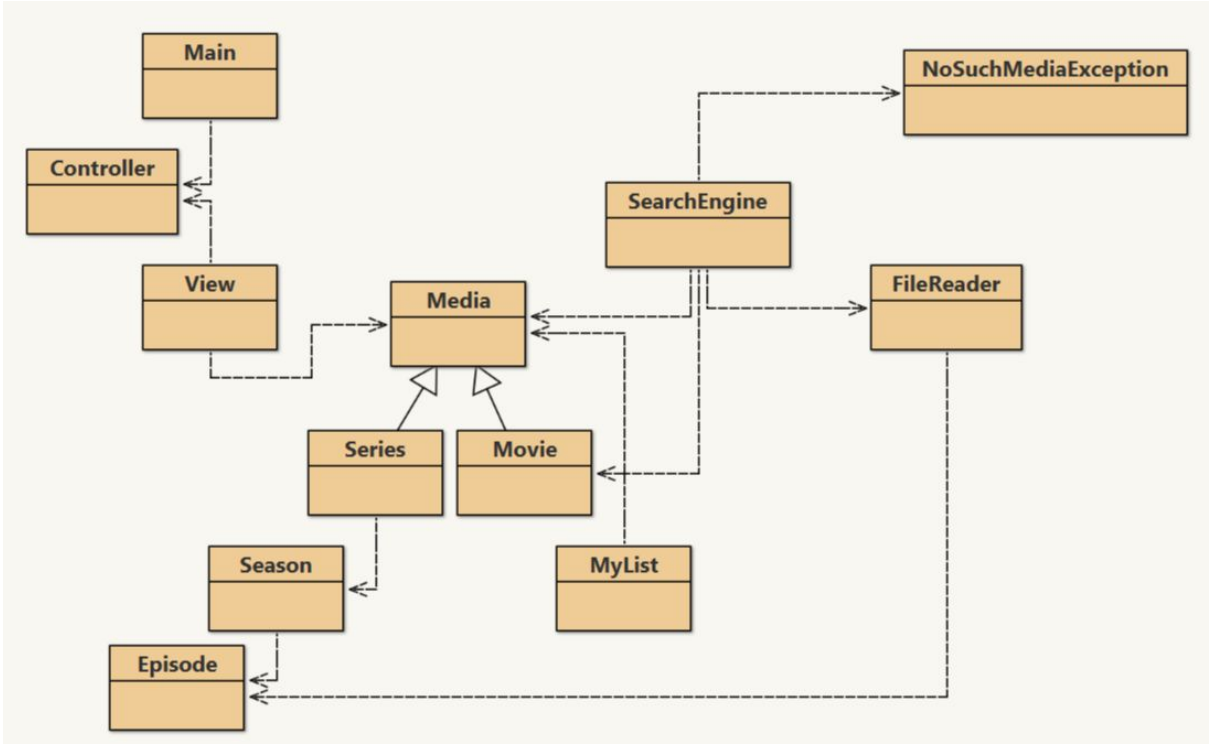


Figur 5: Dialogboks der popper op, hvis der ikke er et resultat på søgningen.

5. Teknisk beskrivelse af programmet

5.1. Programmets interne struktur

På figur 6 ses et klassediagram over vores program.



Figur 6: Klassediagram over applikationen

Når man kører programmet vil klassen *Main* oprette en *Controller*. *Controllers* konstruktør opretter en instans af *View* samt en instans af *SearchEngine* (*Model*). *Controller* har ansvaret for at sørge for at håndtere de input brugeren giver i GUI'en. Når et input gives anmoder sender *Controller* en anmodning til *SearchEngine*, som returnerer en mængde af data. Til sidst dataen sender *Controller* dataen videre til *View* objektet. *Controlleren* står desuden for at håndtere diverse fejl, der måtte opstå inde i *SearchEngine*.

Klassen *SearchEngine* er ansvarlig for at holde på, håndtere og sortere i dataen, efter den er blevet loadet ind af *FileReader*. Det er udelukkende *SearchEngine* der kan se de data som udgør tilgængelige film og serier. Samtidigt kun *Controller* der kan se *SearchEngine,* og benytter sig af *SearchEngine's* metoder. Således er vi sikre på at der ikke kommer rod i de data som vores *Model* holder på.

Klassen *FileReader* er overordnet set ansvarlig for at læse og indlæse de udleverede .txt-filer og oprette Collections af Movie og Series objekter. Dette gør den ved hjælp af en *BufferedReader*, som læser .txt filerne linje for linje og opretter en lang String med informationerne for det enkelte medie. Derefter kan den benytte sig af String's *split* metode på, som returnerer et String-array med hvert datapunkt som String i hvert af String-arrayets indices. Desuden har *FileReader*-klassen også til ansvar at indlæse mediernes billedfiler som BufferedImage, som tillægges Movie og Series objekterne, når de initialiseres.
Første version af *FileReader*-klassen blev udarbejdet tidligt i projektet, men under opbygningen af *View*-klassen slog det os at vi med fordel kunne udbygge *FileReader*. Vi ville i *View* give brugeren mulighed for sortering efter genre, men manglede en liste med alle de genrer som vores film og serier havde. Siden *FileReader* allerede gennemløb alle film og deres genrer ville det være oplagt at lade den generere en liste af genrer undervejs. Derfor ændrede vi metoden *readMedia( )* , således at den returnerer en ArrayList som indeholder to ArrayLists. Den første indeholder listen af medier, den anden listen af genrer.

Controller indeholder fem metoder som har til formål at sende data fra *SearchEngine*. Dette er de to metoder *searchByGenre* og *searchByString*, de to metoder til at vise film eller serier og en metode til at vise alle medier. Derudover indeholder *Controller* metoder til at tilføje og fjerne medier til og fra en brugers *myList*.

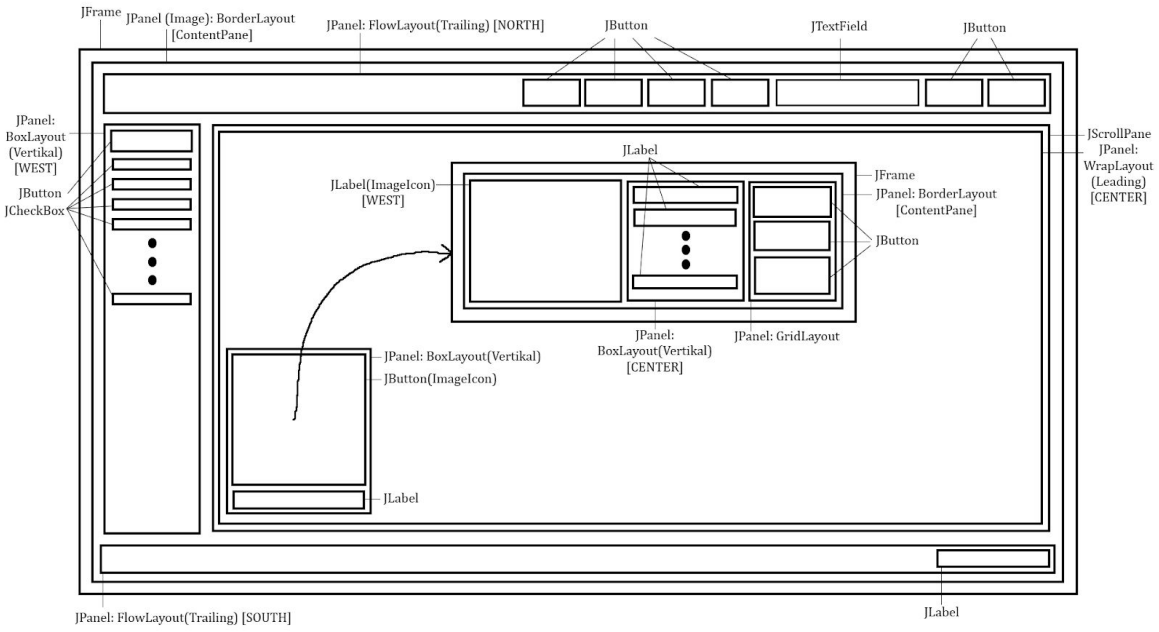
Vi har oprettet vores egne exceptions, der hvor vi synes det har givet mening. Vores exceptions bliver primært brugt til at informere brugeren, hvis vedkommende har

fremprovokeret en fejl gennem brugergrænsefladen. Det kan fx. være hvis brugeren har skrevet et forkert kodeord på startskærmen, eller søgt på en titel som ikke matcher med nogle af de film der ligger i databasen. Vores *Model- SearchEngine* - kaster en exception inden den returnerer et Array af film til *Controlleren*, hvis ArrayList'et er tomt. Det er op til *Controlleren* at håndtere den kastede exception.
En af de største udfordringer under udarbejdningen af programmet var kommunikationen mellem *View* og *Controller*. *Controller* opretter som nævnt en instans af *View*, og har derfor mulighed for at kalde de metoder der ligger inde i instansen. Problemet var bare at instansen af *View* slet ikke kunne finde instansen af *Controller*. Når nogen trykkede på en knap inde i *View* havde den altså ingen mulighed for at kommunikere det videre til instansen af *Controller*. Vi manglede altså en form for pegepind, som gav *View* instansen mulighed for at notificere *Controller* instansen om hændelser. Derfor ændrede vi i *View*-klassens constructor, således at den nu tager en instans af en *Controller* som parameter. Instansen bliver gemt i et felt i *View*. *Controller* vedlægger helt konkret sig selv ved brug af KeyWord'et *this,* idet den instantierer *View*.

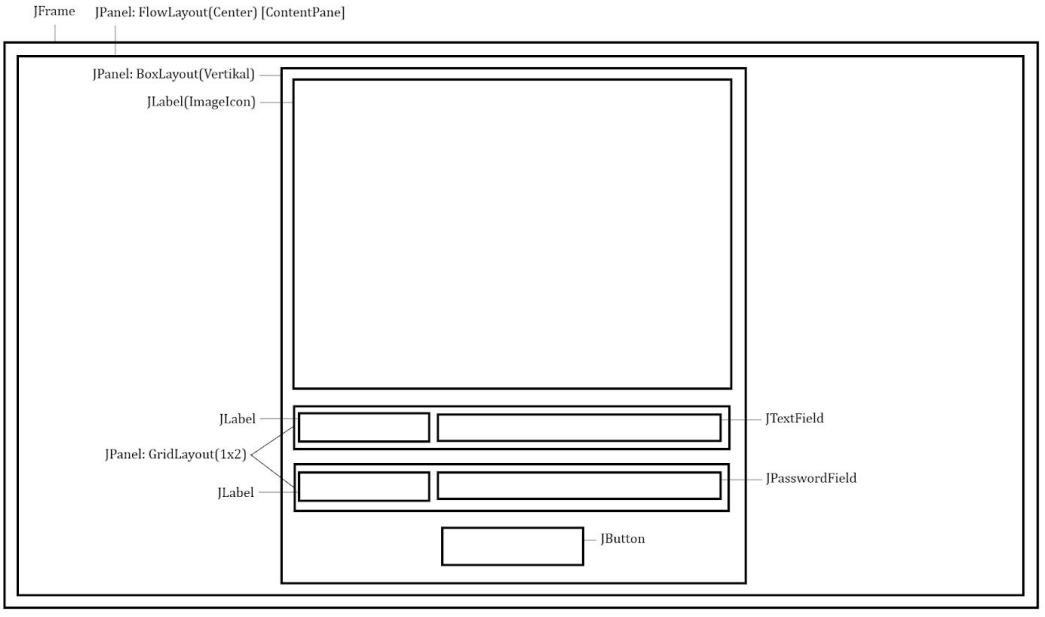
5.2. Brugergrænsefladens struktur

Brugergrænsefladen er opbygget med Java *Swing* og Java *AWT* (Abstract Window Toolkit). Den er opnået ved at tage brugergrænsefladekomponenter som *JPanels, JLabels, JButtons* mv. og putte dem ind i andre *JPanels*. Placeringen af disse elementer styres af såkaldte *Layout Managers* (f.eks. *FlowLayout* eller *BoxLayout*), som en *container* vedlægges, og som sørger for, at det enkelte panel eller element opretholder sin relative position til andre elementer - også selvom man op- eller nedskalerer applikationen, eller bruger en anden størrelse skærm.

Vores brugergrænseflade er opdelt i to dele, eller *frames*, henholdsvis et frame for selve streamingtjenesten og et frame for login-skærmen. Oversigter med brugergrænsefladekomponenter for de to frames kan ses på Figur 7 og Figur 8.



Figur 7: En oversigt over brugergrænseflade-elementerne i framen for streamingtjenesten. OBS: Særligt er der en pil fra et af medie-ikonerne til et nyt frame. Det er dette frame, som åbnes, når man klikker sig ind på mediet.



Figur 8: En oversigt over brugergrænseflade-elementerne i framen for login-skærmen.

Kort fortalt er brugergrænsefladen for selve streamingtjenesten bygget op omkring Layout Manager'en *BorderLayout* med paneler i *NORTH*, *WEST*, *CENTER* og *SOUTH*, hvor *EAST* forbliver tomt. Herunder vil de enkelte regioners komponenter opsummeres:

- NORTH-panelet har et FlowLayout med *alignment: trailing* og indeholder en række JButtons med en række forskellige funktioner (bl.a. at kunne få vist My List, kun at få vist film eller kun at få vist serier) og et JTextField til søgefeltet.
- WEST-panelet har et BoxLayout med *alignment: Y-AXIS* og indeholder en række JCheckBoxes og en enkelt JButton. Disse JCheckBoxes repræsenterer hver sin genre og fungerer som sorteringsværktøj for genrer. JCheckBoxes'ne bliver oprettet vha. genreList[7].
- SOUTH-panelet har et FlowLayout med *alignment: trailing* og indeholder blot en enkelt JLabel med en copyright formalitet som tekst.
- CENTER-panelet har et såkaldt WrapLayout[8], og indeholder alle "medierne". *View*-klassen modtager en ArrayList af Media-instanser fra Controller-klassen, og hvert "medie" repræsenteres ved et JPanel med et vertikalt BoxLayout, som indeholder mediets tilhørende billede lagt ind i en JButton og nedenunder en JLabel med titlen på filmen.

Det mest interessante ved vores brugergrænseflade er CENTER-panelet. For det første har vi benyttet os af et WrapLayout, som tilpasser antallet af komponenter per række til størrelsen på framet. Det vil sige, at hvis man manuelt gør streamingtjenestens vindue mindre eller større, gør WrapLayoutet at komponenterne "wrapper" ned på næste række, hvis der ikke umiddelbart er plads til at vise dem på samme linje. Desuden har hvert panel, som repræsenterer et medie, mediets billede som en JButton, som opretter og viser et nyt frame, når man trykker på knappen. Dette nye frame (se Figur 7) viser information om mediet og JButtons til at "afspille" mediet samt tilføje og fjerne mediet fra My List.

Hele *View*-delen af vores *Model-View-Controller*-struktur er samlet i én klasse - netop *View*-klassen. *View*-klassen indeholder tre metoder: *run*, som opretter login-skærmen og kalder *runStreamingService*, hvis login-kredientialerne er korrekte; *runStreamingService,* som opretter framet for streamingtjenesten med alle dets komponenter; og til sidst *update*, som opretter og overskriver det nuværende CENTER-panel vha. en given ArrayList af Media. På den måde kan hele programmet initialiseres vha. *run*, og når der sker en sortering i medierne, kan *update* metoden kaldes og vise den nye, sorterede Collection af Media.

[7] *genreList* er en List af Strings (genrer), som FileReader giver til *SearchEngine*, *SearchEngine* giver til *Controller*, som *Controller* giver til *View*. Den loades direkte af fra dataene om medierne af *FileReader* og indeholder alle de mulige genrer, medierne har.
[8] En extension af *FlowLayout*. Taget fra nettet: https://biojava.org/docs/api4.2.9/src-html/org/biojava/nbio/structure/gui/WrapLayout.html

Vi har senere i projektet indset, at *View*-klassen godt kunne være delt op i mindre klasser og/eller have flere metoder. På nuværende tidspunkt er den indre struktur af *View*-klassen rodet og giver ophav til dårlig kodeskik. F.eks. kunne koden i den actionListener, som hver JButton med billede af et medie tildeles ved hjælp af et lambda-udtryk, få sin egen metode eller klasse, da den på nuværende tidspunkt blot er en 100 linjers blok kode i midten af *update*-metoden. Denne høje coupling skyldes vores manglende erfaring med kodning af grafiske brugergrænseflader samt tidspresset i forbindelse med projektet.

6. Afprøvning

Produktet lever op til næsten alle brugermæssige krav, på nær krav nr. 4 i tabel 1 som lød "Brugeren vil gerne skifte bruger.". Dette er på nuværende tidspunkt ikke muligt, da vi ikke har implementeret en funktion til at logge ud. Denne mangel skyldes at vi har fokuseret mere på de andre krav, der er blevet stillet. Det er dog muligt at logge ind på en bruger, da vi har implementeret en login-skærm. Derfor vil det også være ligetil at tilføje mulighed for at logge ud af sin bruger, under en session af programmet. For at denne funktion ville give bedre mening, skulle hele bruger-aspektet udvides lidt, da brugeren på nuværende tidspunkt ikke har nogen indflydelse på hvordan programmet kører. Brugeren kunne fx. få en egenskab som isChild, som bestemmer om der udelukkende skal vises børnefilm/familiefilm, eller om brugeren skal have adgang til alle film.
Krav nr. 5 er heller ikke opfyldt helt tilfredsstillende. På nuværende tidspunkt fungerer en serie som en film, da antallet af sæsoner og episoder ikke bliver vist, og det derfor ikke er muligt at klikke ind på enkelte sæsoner/episoder for at se afspille eller se informationer om disse. Dog har vi implementeret kode i FileReader-klassen, som opretter Season og Episode objekter ud fra dataene og placerer dem i Series - de bliver bare ikke vist noget sted i brugergrænsefladen.
Af de systemmæssige krav listet i tabel 2 i afsnit 2.3. lever programmet op til dem alle.

Under udarbejdelsen af klassen *SearchEngine* har vi løbende lavet enkelte unit-test af de forskellige metoder.[9] Nogle metoder i *SearchEngine* har krævet lidt abstraktion fra vores side og det har derfor været fordelagtigt teste enkeltdele af dem. Disse tests er blot lavet ved at kalde de forskellige metoder, og printe deres output med System.out.println()-metoden. Siden der ikke er nogle elementer af tilfældighed i nogle af *SearchEngine's* metoder, vil de

[9] Se eventuelt bilag 5 for unit tests.

hver især give samme output under samme input. Derfor vil det også være oplagt at automatisere testene og ved at lave metoder som giver et input til hver metode, og verificerer at metoden giver det forventede output. Således vil det være muligt at rette i *SearchEngine*'s metoder og samtidigt sikre sig at de fortsat virker som de skal, uden at udarbejde tests på ny. Vi har generelt i løbet af udarbejdningen af klasser i programmet benyttet os for meget at System.out.println-metoden og for lidt af 'rigtige' unit-tests. Selvom flere af vores print-tests er ganske fornuftige er det suboptimalt ikke at udarbejde dem i dedikerede test-klasser.

View-delen af programmet er hovedsageligt testet ved at køre programmet. Vi har "leget" rundt i programmet og prøvet at fremprovokere fejl samt checke at programmet opfører sig, som vi forventer det burde. Dog har vi lavet en enkelt unit test til View.[10]

7. Refleksioner over arbejdsprocessen

En af de første gange vi mødtes som gruppe udarbejdede vi en tidsplan for projektforløbet. Vi startede med at aftale dage, som vi skulle mødes på, og sætte planer for hvilke stadier i projektet der skulle bruges tid på de forskellige dage. Vi havde et håb om at bruge den iterative projektforløbsmetode, da vi gerne ville skabe os nogle gode vaner i forbindelse med projektet. Vi havde ikke afsat mere end tre dage til analyse- og designdelen. Vi følte det var lidt svært at komme videre med designet, da det er vores første programmeringsprojekt, og vi hellere ville gå videre til kodefasen. Hvis vi kigger tilbage på det, så ville vi gerne have brugt mere tid på analyse- og designfasen, da det kunne have sparet os for meget tid, når vi skulle skrive og teste koden.

Vores fremgangsmåde i projektet har hovedsageligt været præget af vandfaldsmodellen, hvor vi er gået fra analyse- og designfasen direkte over i kodefasen uden rigtig at vende tilbage igen. Dette kunne have været undgået, hvis vi havde haft en iterativ tilgang til projektet, hvis mangel overordnet skyldes, at vi har lært mange af tingene undervejs. Eftersom det er vores første projekt, og det kun har varet 4 uger, så følte vi ikke, at vi havde tid nok til at arbejde iterativt. Dog har vi stadigvæk brugt "Divide and Conquer" på den måde, at vi har taget et delproblem ad gangen.

[10] Se eventuelt bilag 5.

Derudover har vi skrevet det meste af rapporten i den sidste uges tid af forløbet i stedet for at sprede rapportskrivningen ud over hele forløbet. Det gjorde vi, da vi følte det var svært at skrive rapporten før det meste af programmet var færdigprogrammeret. Til en anden gang vil vi begynde at skrive rapporten noget før i forløbet, så ikke den bare bliver skubbet frem i tidsplanen og ender med at ligge og vente på en til allersidst.

Med hensyn til at teste koden løbende har vi ikke været så systematiske i forhold til at lave dedikerede *Test*-klasser for vores program. Dette er noget vi fortryder, da det ville have hjulpet os med at kunne dokumentere, at programmet virker efter hensigten. Derfor vil vi helt sikkert have dette med i tankerne til fremtidige projektforløb. F.eks. testede vi ikke designet af brugergrænsefladen, før vi begyndte at kode, men har blot valgt et design og gået med det. Havde man haft mere tid, ville det havde været fordelagtigt at teste, så man kan sikre sig éns design faktisk er brugervenligt, da det nuværende design blot er baseret på anekdoter.

I kodefasen var vi gode til at få implementeret vores kode løbende og dette har Git været en stor hjælp til. Med Git blev det nemt at dele kode med hinanden selvom det har taget lidt tid at lære at bruge korrekt. Der har været et par enkelte problemer ved sammenkoblinger, men generelt har det været en positiv oplevelse at bruge.

Overordnet set er vi tilfredse med slutproduktet, men vi er ærgerlige over, at vi ikke arbejdede så iterativt, som vi havde håbet på. I fremtiden håber vi på at blive bedre til dette. Der er dele af programmet, som er rodede, hvilket skyldes, at vi ikke har planlagt tilstrækkeligt. Dette tillægger vi primært manglende erfaring.

8. Konklusion

Vi har i dette projekt udarbejdet en streamingstjeneste, playIT. Ved at analysere den udleverede projektbeskrivelse fandt vi en række bruger- og systemmæssige krav, som vi designede vores program ud fra. Vi har løbende testet med System.out.println for at sikre os at programmets enkeltdele fungerede i praksis, desuden har vi lavet enkelte unit-tests. Programmet er skrevet i Java med en brugergrænseflade opbygget af Java *Swing* og Java *AWT*.

9. Litteraturliste

Barnes, D. J. Kölling, M. (2016). Objects First with Java: A Practical Introduction using BlueJ. 6. udgave, Pearson.

Brabrand, C. (2019). Grundlæggende Programmering: OO Design Patterns. Slides fra forelæsning 6-11-2019.
https://learnit.itu.dk/pluginfile.php/252191/mod_resource/content/1/GRPRO-19.pdf.
Link sidst tilgået 19-12-2019.

10. Bilag.

10.1. Bilag 1. Verb-Noun øvelse på projektbeskrivelsen. Blå og orange ord er navneord og grønne ord er udsagnsord.

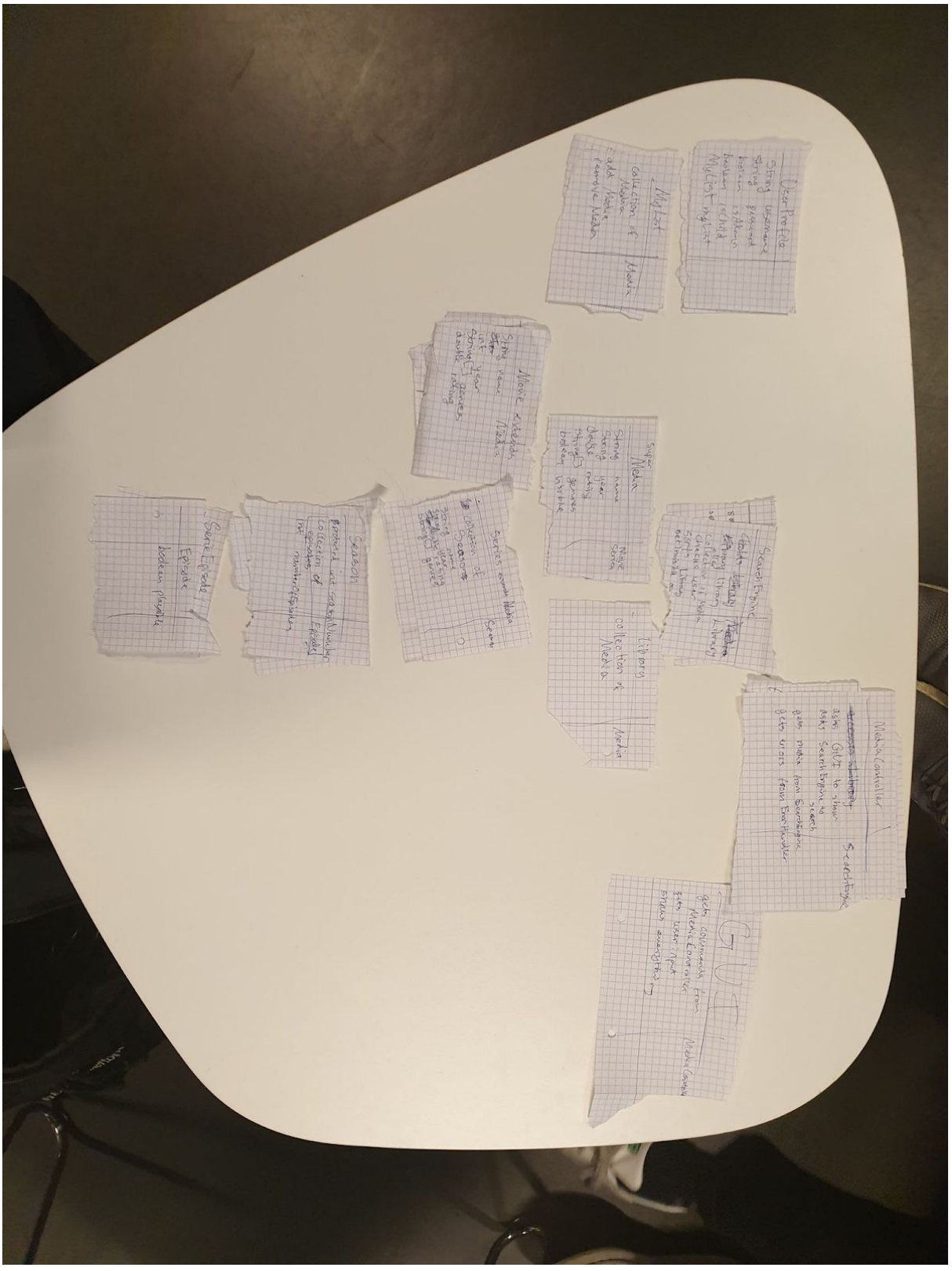
• I projektet skal I udvikle et softwaresystem til brug for streamingtjenester som Netflix, HBO, Viaplay etc.
• I systemet er der "medier", der kan afspilles. Medier kan være film eller serie-episoder. Både film og serier er inddelt i kategorier som fx crime, war, drama, family, romance og sci-fi. Serier består desuden af sæsoner og episoder.
• Løsningen skal i en grafisk brugergrænseflade vise oversigt over film, serier, sæsoner og episoder, samt give mulighed for at man klikker sig ind på den enkelte film, serie, sæson eller episode og læser om den/sætter den i gang/tilføjer den til sin egen liste. Løsningen skal ikke kunne afspille film "rigtigt" – det er fint blot at lave en play-knap, som ikke gør noget andet end fx at skifte farve når man klikker på den.

Løsningen skal understøtte almindelige opgaver, som I kender det fra streamingtjenester, fx
• Brugeren vil gerne se en oversigt over alle krigsfilm
• Brugeren vil gerne se en oversigt over alle dramaserier
• Brugeren vil gerne gemme en film i "min liste"
• Brugeren vil gerne se sin liste
• Brugeren vil gerne slette en film fra sin liste
• Brugeren vil gerne skifte til en anden bruger
• Brugeren vil søge efter en bestemt film eller serie
• Der skal være billeder knyttet til film og serier

Navneord	**Udsagnsord**
MedieViser	
Søgemaskine	
Medie	
Film	*Afspilles* (filmen afspilles)

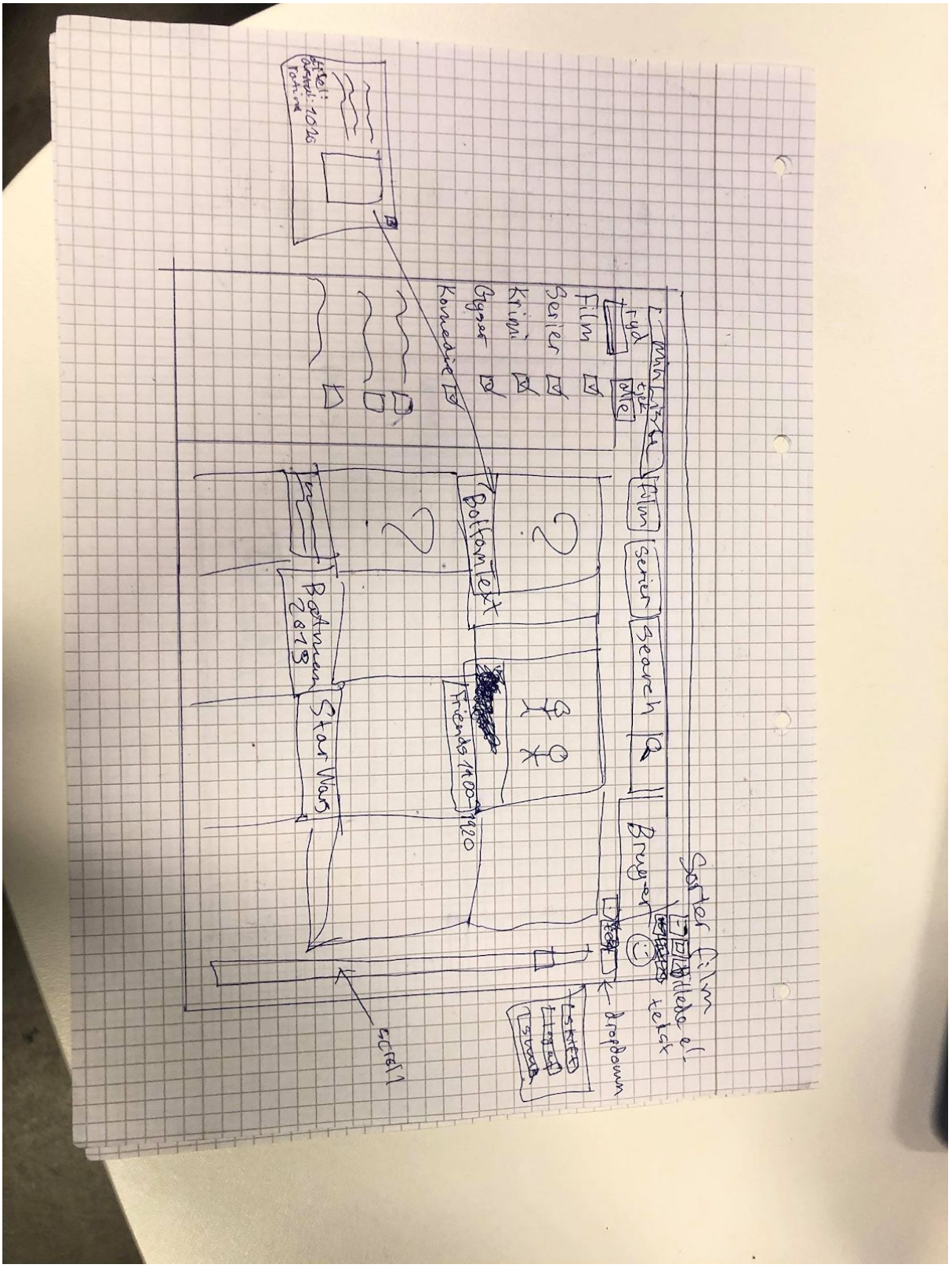
	Klikker (sig ind på en film) *Læser* (om filmen) *Tilføjer* (filmen til sin egen liste)
Serie	*Består (af sæsoner og episoder)*
Serie-episode	
Play-knap	
Brugeren	*Se* (en oversigt) *Gemme* (en film, i min liste) *Se* (sin liste) *Slette* (en film fra sin liste) *Skifte* (til en anden bruger) *Søge* (efter en bestemt film)
Kategorier	
Sæson	
Episode	
Oversigt	
Billede	
Egen liste	

10.2. Bilag 2. CRC-kort.



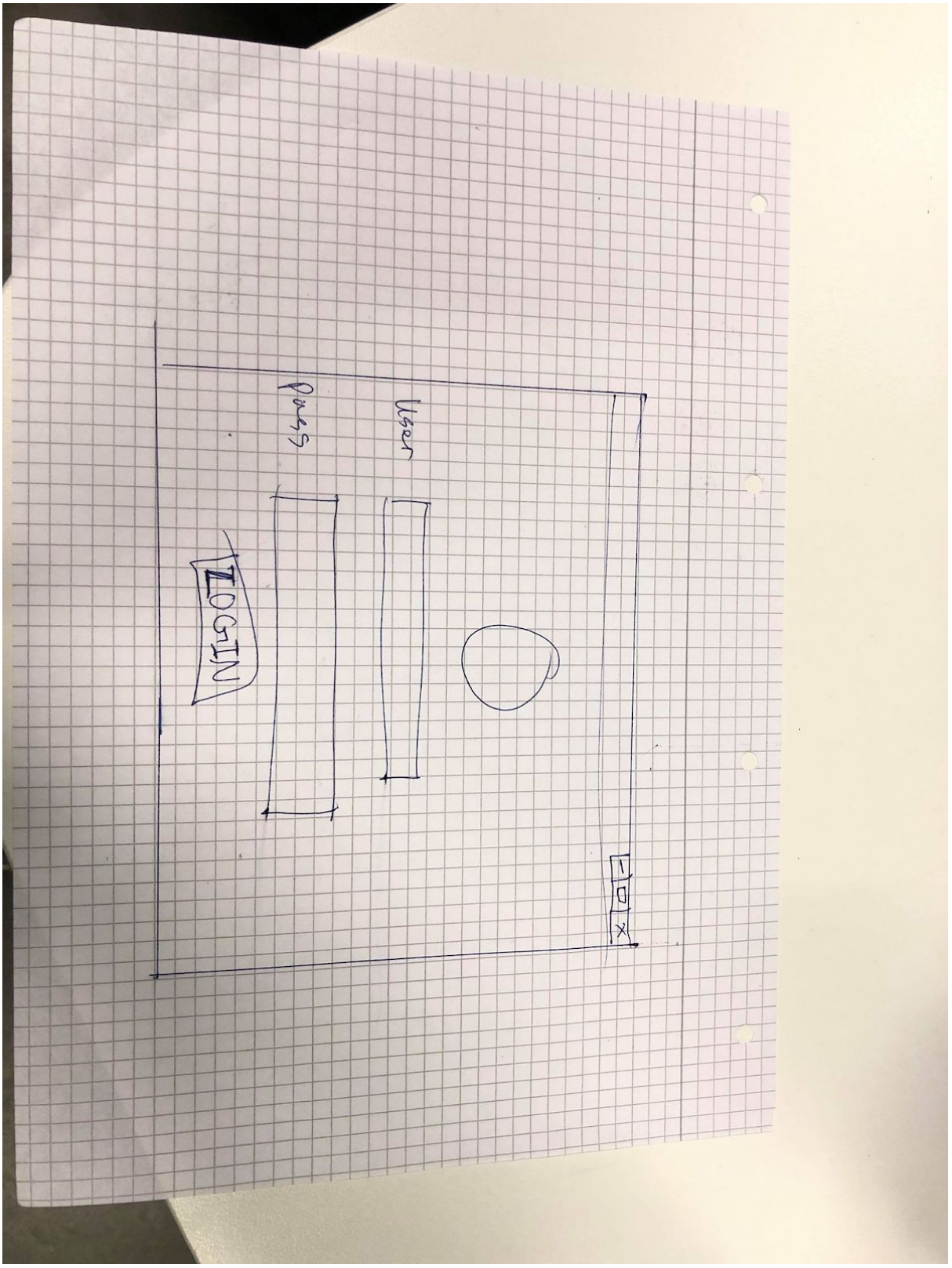
10.3.1. Bilag 3.1. Første skitse af GUI.

Første skitse der blev udarbejdet af GUI'en, hvor indholdet skulle vises.

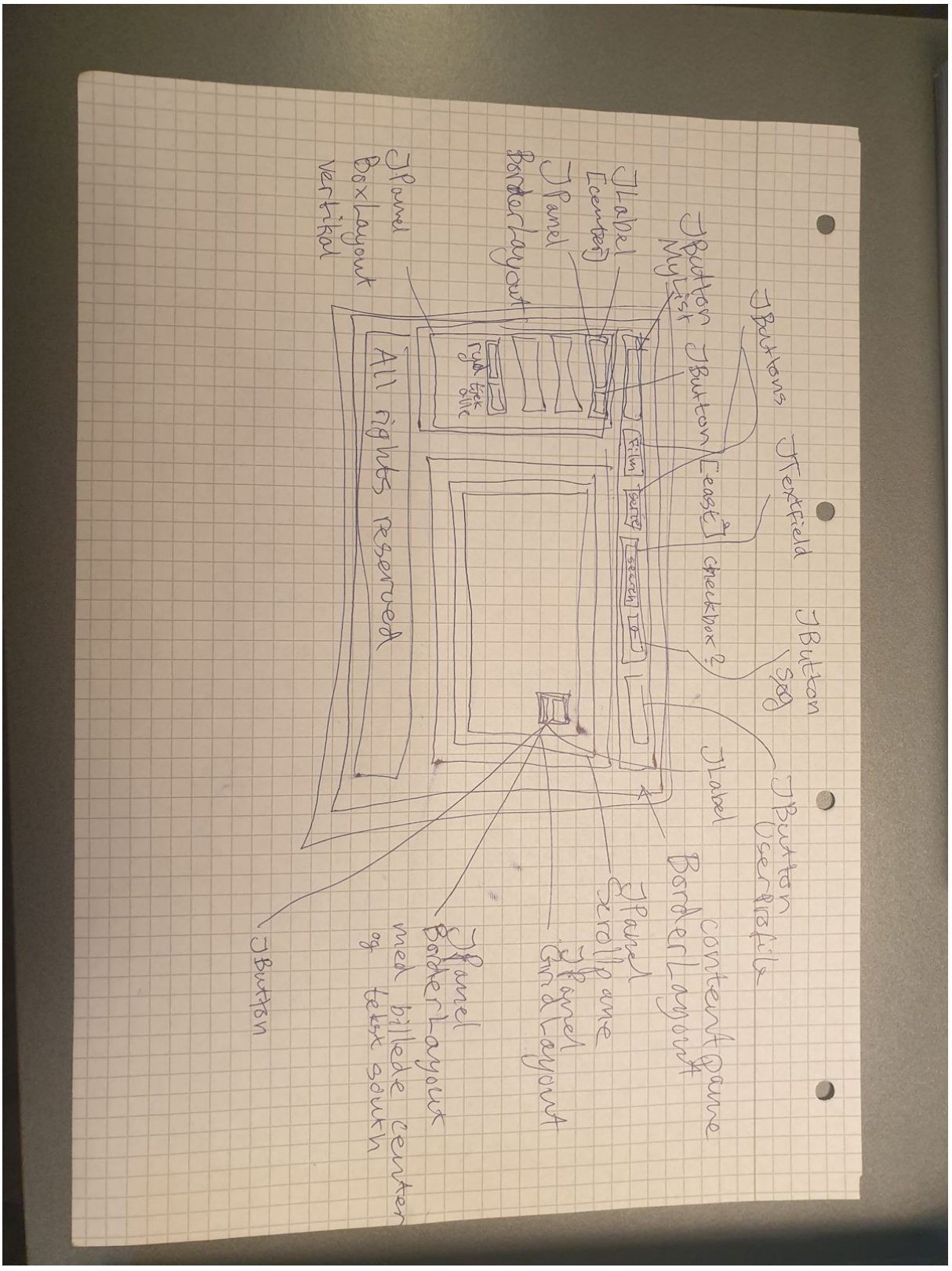


10.3.2 Bilag 3.2. Skitse af login-skærm

Skitse af login-skærmen til programmet.



10.4. Bilag 4. Anden skitse over en prototype af GUI'en.



10.5. Bilag 5. Unit Test

10.5.1. FileReader test.

```
27      @Test
28      void readMedia() {
29          //public Movie(String name, String year, String[] genres, String rating, BufferedImage pictureFile)
30          String[] genres = {"Crime", "Drama"};
31          Movie movie = new Movie( name: "The Godfather", year: "1972", genres, rating: "9,2", pictureFile: null);
32          //The Godfather; 1972; Crime, Drama; 9,2;
33
34          System.out.println(FileReader.readMedia().get(0)==null);
35          ArrayList<Media> media = FileReader.readMedia().get(0);
36          Movie godFather = (Movie) media.get(0);
37
38          assertEquals(movie.getName(), godFather.getName());
39          assertEquals(movie.getYear(), godFather.getYear());
40          assertArrayEquals(movie.getGenres(), godFather.getGenres());
41
42      }
43  }
```

FileReaderTest › readMedia()

Tests passed: 1 of 1 test – 1 s 528 ms

```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.2.4\jbr\bin\java.exe" ...

C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs\film.txt
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs\serier.txt
false
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs\film.txt
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs\serier.txt


Process finished with exit code 0
```

10.5.2. SearchEngine test.

```
38      @Test
39      void sortByCategory() {
40          SearchEngine searchEngine = new SearchEngine();
41          ArrayList<String> categories = new ArrayList<>();
42          categories.add("Sport");
43
44          assertEquals(searchEngine.sortByCategory(categories).get(0).getName(), actual: "Raging Bull");
45          assertEquals(searchEngine.sortByCategory(categories).get(1).getName(), actual: "Rocky");
46          assertEquals(searchEngine.sortByCategory(categories).get(2).getName(), actual: "GLOW");
47      }
48
49
```

SearchEngineTest › sortByCategory()

Category

Tests passed: 1 of 1 test – 2 s 56 ms

```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.2.4\jbr\bin\java.exe" ...

C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs\film.txt
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs\serier.txt
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs\film.txt
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs\serier.txt


Process finished with exit code 0
```

ersion Control 0: Messages

10.5.3. View test.

```java
@Test
void update() {
    String filePath = new File( pathname: "").getAbsolutePath();
    View view = new View(new Controller());
    ArrayList<Media> media = new ArrayList<>();
    ArrayList<String> genres = new ArrayList<>();
    String[] genreArray = {"Horror"};
    String[] genreArray2 = {"Action"};
    try {
        BufferedImage image = ImageIO.read(new File( pathname: filePath + "/Film - billeder/" + "The Godfather" + ".jpg"));
        BufferedImage image2 = ImageIO.read(new File( pathname: filePath + "/Serier - billeder/" + "The Godfather" + ".jpg"));

        Movie movie = new Movie( name: "The Godfather", year: "1972",genreArray,  rating: "9,2", image);
        // Twin Peaks; 1990-1991; Crime, Drama, Mystery; 8,8; 1-8, 2-22;
        Series series = new Series( name: "Twin Peaks",  year: "1990-1991", genreArray2, rating: "8,8", image2,  seasons: null );
        media.add(movie);
        media.add(series);
        genres.add(genreArray[0]);
        genres.add(genreArray2[0]);
        media.add(movie);
        media.add(series);
        view.runStreamingService(media, genres);
    }
    catch (IOException e) {
        System.out.println("Error");
    }

}
}
```

```
Run:  Main  ViewTest.update
Tests passed: 1 of 1 test – 2 s 663 ms

Test Results  2 s 663 ms
  ViewTest  2 s 663 ms
    update()  2 s 663 ms

"C:\Program Files\JetBrains\IntelliJ IDEA 2019.2.4\jbr\bin\java.exe" ...

C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs\film.txt
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs\serier.txt
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs\film.txt
C:\Users\Bruger\Documents\GitHub\Vinterprojekt-Progammering-2019\Juleprojekt-undervejs\serier.txt


Process finished with exit code 0
```

10.6. Bilag 6. JavaDoc

JavaDoc'en tilhørende vores program er vedlagt i den afleverede zip-fil.