

# SUBMISSION OF WRITTEN WORK

Class code: BSGPRO1KU  
Name of course: Grundlæggende programmering  
Course manager: Claus Brabrand, Dan Witzner Hansen  
Course e-portfolio:

Thesis or project title: RickFlix - En streamingtjeneste  
Supervisor: Signe Kyster

Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1. <u>Mikkel Lippert</u>	<u>09/11/97</u>	<u>lipp</u> @itu.dk
2. <u>Trond Pingel Anchær Rossing</u>	<u>07/03/96</u>	<u>trro</u> @itu.dk
3. <u>Veronika Raagaard Skotting</u>	<u>13/05/95</u>	<u>vesk</u> @itu.dk

Antal tegn: 34.911 (14,5 normalsider)

# Indhold

1. Indledning .....	3
2. Baggrund og problemstilling .....	3
2.1. Problemområde .....	3
2.2. Brugsmæssige krav til programmet .....	3
2.3. Systemmæssige krav til programmet .....	4
3. Problemanalyse .....	5
3.1. Design af programmets interne struktur .....	5
3.2. Design af programmets brugergrænseflade .....	6
3.3. Vedrørende behandling af data .....	7
4. Fejlmeddelelser .....	8
5. Teknisk beskrivelse .....	9
5.1. Intern struktur .....	9
5.2. Brugergrænseflade .....	12
5.2.1. Loginskærmen .....	12
5.2.2. Create User .....	14
5.2.3. Hovedvinduet .....	15
5.3. Dokumentation .....	16
6. Afprøvning .....	16
7. Refleksion .....	18
7.1. Beslutningsprocessen .....	18
7.2. Vidensdeling .....	18
7.3. Designovervejelser .....	19
8. Konklusion .....	19
9. Litteratur .....	21
10. Bilag .....	22
10.1. Substantiv-verbum metoden .....	22
10.2. Unit test .....	23
10.2.1. <i>Media</i> test .....	23
10.2.2. <i>User</i> test .....	24
10.2.3. TxtParser test .....	25
10.2.4. MediaContainer test .....	26
10.3. Brugervejledning .....	27
10.3.1. Navigation i login .....	27
10.3.2. Navigation i hovedvinduet .....	29

## 1. Indledning

Denne rapport er udarbejdet i forbindelse med et fire-ugers projekt, som er kulminationen på kurset Grundlæggende Programmering på IT-Universitetet i København. Vores gruppe består af Mikkel, Trond (1. semester-studerende på bacheloren Softwareudvikling) og Veronika (5. semester-studerende på bacheloren Global Business Informatics). Vejleder i projektet er Signe Kyster (signek@itu.dk). Vi har i gruppen udviklet et softwaresystem ved navn 'RickFlix', der fungerer som en streamingtjeneste med film og serier.

## 2. Baggrund og problemstilling

### 2.1. Problemområde

RickFlix er en streamingtjeneste, hvor en bruger kan oprette og konto, logge ind og derefter se de hundrede mest populære serier og film iflg. IMDb. Brugeren har mulighed for at søge efter specifikke titler eller bladde igennem de forskellige genrer. Efter brugeren foretager en søgning, vil denne se en oversigt over medierne bestående af plakaten og et tooltip med yderligere informationer. Disse informationer vises også på skærbilledet, når brugeren vælger et medie, ved at trykke på det. Herefter kan mediet tilføjes til brugerens personlige liste, eller der kan afspilles en animeret gif, som agerer pladsholder for en videofil, da en fuldt funktionel streamingtjeneste er uden for anvendelsesområdet for dette projekt.

### 2.2. Brugsmæssige krav til programmet

De brugsmæssige krav er baseret på de eksempler der fremgik af den udleverede projektbeskrivelse. Disse krav var oprindeligt tiltænkt som eksempler på almindelige opgaver streamingtjenester varetager, men vi har valgt at anse kravene som basis for udviklingen af vores streamingtjenestes funktionalitet.

Tabel 1: Brugsmæssige krav til programmet.

Nr.	Brugsmæssige krav
0	Brugeren vil gerne se en oversigt over alle krigsfilm.
1	Brugeren vil gerne se en oversigt over alle dramaserier.
2	Brugeren vil gerne gemme en film i "min liste".
3	Brugeren vil gerne se sin liste.
4	Brugeren vil gerne slette en film fra sin liste.
5	Brugeren vil gerne skifte til en anden bruger.
6	Brugeren vil søge efter en bestemt film eller serie.

I arbejdet med at implementere de syv krav er der løbende taget højde for at generalisere funktionaliteten, så brugeren ikke kun har mulighed for at se en oversigt over alle krigsfilm og dramaserier, men også andre genrer. Denne tankegang har været basis for at lave løsninger og strukturer, der vil gøre videreudvikling af RickFlix samt implementering af nye funktioner simpelt og strømlinet, skulle dette vise sig at være relevant.

## 2.3. Systemmæssige krav til programmet

I tabel 2 er de systemmæssige krav skrevet op. I programmets nuværende udformning skal det kun køres på en enkelt computer. Derfor ligger al nødvendig data separeret i to tekstfiler samt en mappe med billeder.

Tabel 2: Systemmæssige krav til programmet.

Nr.	Systemmæssige krav
0	Skal køres på én computer af gangen.
1	Skrives i Java.
2	Benytter JavaFX bibliotek til opbygning af brugergrænseflade.
3	Skal indlæse medie-objekter fra en 'resources' mappe der indeholder lister af film og serier samt billeder dertil.
4	Skal bruge passende fejlhåndtering, hvor bruger informeres, hvis det er relevant.
5	Til test af essentielle komponenter bruges unit tests.

For at kunne håndtere de brugsmæssige krav beskrevet i tabel 1, indeholder systemet følgende typer data (tabel 3).

Tabel 3: Data i programmet.

Data	Beskrivelse
Medie	De medier, der kan afspilles på streamingtjenesten. Medier består af en titel, et årstal, en rating samt et billede og har derudover en liste af genrer tilknyttet.
Film	Film der vises på streamingtjenesten. Er subklasse af medie.
Serie	Serier der vises på streamingtjenesten. Er også subklasse af medie. En serie har derudover et antal sæsoner samt en mængde afsnit i hver af disse.
Bruger	Brugere er medlemmer af streamingtjenesten og har et brugernavn samt et password, der bruges til at få adgang til selve programmet. Derudover har hver enkelt bruger en personlig liste af film og serier.

### 3. Problemanalyse

#### 3.1. Design af programmets interne struktur

Da programmet både skal indeholde en datamodel, der håndterer brugere og medieobjekter, samt en brugergrænseflade, har vi valgt at benytte Model-View-Controller (MVC)-princippet som grundpille i opbygningen af vores streamingtjeneste. Ved dette design pattern adskilles model-delen, der indeholder objekter som hver især bærer data, og view-delen, der indeholder brugergrænsefladen. Derimod styrer logikken i programmet (controlleren) det dataflow, der passes fra model til brugergrænseflade, når programmet kører. Hovedformålet ved dette designprincip er at forbedre strukturen i programmet samt at gøre det lettere for de individuelle programmører at arbejde parallelt i samme projekt.

Det første skridt der tages imod et færdigt produkt, er ind i analysefasen (*domæneanalysen*). Her var målet at forstå *applikationsdomænet*; det fastlægges altså hvilken sammenhæng programmet skal fungere i. Fænomener og begreber fra problemdomænet identificeres og uformelle specifikationer af begreberne bestemmes. Disse abstraheres videre til koncepter, hvilke benyttes i løsningsdomænet og fungerer som vejledere til indholdet af programmets model-del. Til at facilitere dette skrev vi, med inspiration fra opgavebeskrivelsen, en problembeskrivelse af streamingtjenesten, således at vi kunne identificere samtlige arbejdsopgaver, programmet skulle varetage. Derefter anvendte vi substantiv/verbum-metoden til at danne et overblik over systemets eventuelle klasser og metoder. Herefter dannede et overblik over sammenhængen i programmets interne struktur sig.

I forbindelse med analysen af programmets interne struktur, valgte vi at ekskludere nogle enkelte features. Eksempelvis er der mulighed for, at brugeren vælger en alder, når der oprettes en konto. Det var tiltænkt, at en bruger under 18 år ikke skulle kunne se grafisk indhold. Pga. det begrænsede tidsmæssige omfang af projektet, er dette ikke implementeret. Derudover overvejede vi hvor mange superklasser film- og serie-klasserne skulle have. Vi besluttede at bruge to; en abstrakt superklasse *Media* samt en abstrakt superklasse *Watchable* (subklasse af *Media*). Dette gjorde vi, da man så relativt let ville kunne udvide streamingtjenestens bibliotek til andre medier: lydbøger, musik, podcasts osv. I det tilfælde ville disse medier arve *Media*-klassens funktionalitet, hvor film og serier ville arve funktionalitet fra den abstrakte subklasse *Watchable*.

Til slut forestillede vi os et simpelt system til at holde styr på brugerne i programmet. Man skulle have mulighed for at logge ind på samme bruger, efter at have lukket og startet programmet igen. Dette valgte vi ikke at gøre, for derimod at oprette samtlige brugere på runtime. Dermed bliver et succeskriterie for skift mellem brugere, at programmet ikke lukkes. Hvis systemet skulle skaleres, ville det være gavnligt med et mere udførligt databasedesign, hvortil brugere gemmes og læses ved hhv. lukning og start af programmet.

### 3.2. Design af programmets brugergrænseflade

Det primære mål mht. design af brugergrænsefladen var simplicitet og overskuelighed. Derfor indeholder programmet tre vinduer, der hver varetager forskellige arbejdsopgaver. Først vises et login-vindue, hvor det er muligt at logge ind på selve tjenesten eller at registrere en bruger. Vælges registrer bruger, åbner programmet et nyt vindue, hvor brugeren på normal vis kan oprette en konto. Vælges login, åbner programmet hovedvinduet, hvor størstedelen af funktionaliteten tilkendegiver sig. Vedrørende designet af hovedvinduet, tog vi en beslutning om, at alle relevante knapper og elementer skulle vises umiddelbart og uden at en bruger skulle lede efter dem.

Til start analyserede vi diverse streamingtjenester som Netflix, HBO og Spotify. Her var formålet at identificere de designbeslutninger, som de havde taget, samt vurdere positive og negative aspekter ved disse. Eksempelvis vurderede vi, at Netflix og HBOs horisontale paneler, der på forsiden fremviser diverse emner og genrer, fungerede dårligt sammenlignet med Spotifys mere overskuelige gittervisning af genrer i deres 'Browse' skærmbillede. Derimod fungerede den lignende visning af søgeresultater for både Netflix og HBO godt. Derfor søgte vi at lave et sammenligneligt udtryk, dog for alle visninger af medie-elementer. Oprindeligt forestillede vi os også at lave en forside, som skulle være det første brugeren mødte. Denne skulle til dels være lig de førnævnte streamingtjenesters forsider og indeholde diverse anbefalinger såsom en 'movie of the day' eller de fem film og serier med

højest ratings. Derfor blev RickFlix' brugergrænseflade oprindeligt bygget med en homescreen knap, der skulle kunne føre brugeren tilbage til forsiden, hvis ønsket. Forsiden blev dog ikke implementeret, da vi vendte tilbage til analysen og vurderede, at brugeren burde præsenteres for det, brugeren ønsker, kontra hvad en given tjeneste antager at brugeren vil se. Ved programstart ses derfor nu et tomt panel uden nogle film eller serier. Det er vi ikke tilfredse med, da det ikke har den ønskede visuelle effekt. Dog er det funktionelt og fungerer udemærket sammen med en kortfattet vejledende tekst.

Vi tog tidligt i processen beslutningen om, at vi ønskede at lave et vindue der har en fast størrelse. Det blev valgt på baggrund af, at det syntes for komplekst ift. projektets omfang at lave et dynamisk layout, der passede sig ind efter en brugers definition af vinduesstørrelsen. Derfor har RickFlix i sin nuværende udformning et fixed fremfor dynamisk layout. Dette sikrer os, at ved opskalering af programmet, vil den visuelle integritet bibeholdes. Eftersom ingen af gruppemedlemmerne besad forhåndserfaring med Java-programmering, er størstedelen af arbejdsindsatsen lagt i programmets funktionalitet. Vi havde i gruppen det udgangspunkt, at programmets funktionalitet var i højsædet, hvorfor vi måske ikke nåede til tilfredshed ift. visionen om brugergrænsefladens udseende. Havde vi haft længere tid til projektet forestillede vi os at modernisere programmets visuelle udtryk vha. CSS-stylesheets.

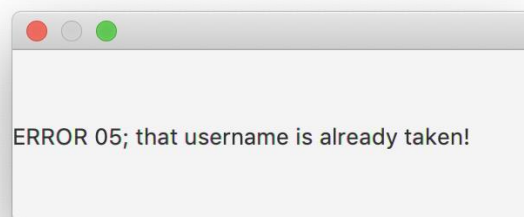
### 3.3. Vedrørende behandling af data

Da systemet i dets nuværende udformning ikke behandler store mængder data, og da systemet ikke indeholder videofiler tilhørende de forskellige film og serier, har vi valgt ikke at benytte nogen særlig databasestruktur. Derimod indeholder projektet to tekstfiler og en mappe med thumbnail-billeder til filmene og serierne der 'vises' på streamingtjenesten. Vi har valgt at placere dem i projektmappen, da filerne ikke optager meget lagerplads (ca. 6-15 KB hver). Disse filer tilgås af en metode, der initialiserer de enkelte objekter ved programstart. For at tilføje film og serier, eller ændre i de allerede eksisterende, skal man derfor blot skrive en film eller en serie ind i tekstfilen samt lægge et billede ind i den tilhørende mappe. Her er det vigtigt at konventionen for navngivning af medie-objekterne følges, da programmet ellers ikke ville kunne initialisere objekterne korrekt. Hvis man på sigt skulle skalere; altså udvide biblioteket og/eller introducere filer der kan afspilles, ville det være gavnligt med en anden databasestruktur. Dog har vi, af hensyn til den tidsmæssige horisont på projektet, valgt at benytte den førnævnte simple løsning.

## 4. Fejlmeddelelser

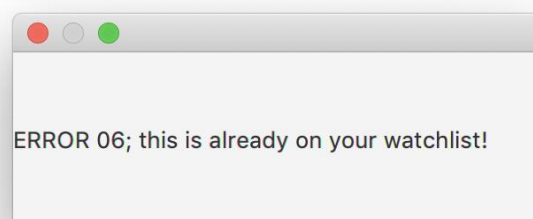
I RickFlix eksisterer seks typer af fejl en bruger kan møde, når programmet bruges (se appendix 10.3 for brugervejledning):

- 01. *Please choose a username*: Viser, hvis man forsøger at oprette en konto uden et username.
- 02. *Please choose a password*: Viser, hvis man forsøger at oprette en konto uden et password.
- 03. *Please make sure the passwords match*: Viser, hvis man forsøger at oprette en konto, hvor kodeordene i de to felter ikke matcher.
- 04. *Login failed*: Viser, hvis login forsøges med et ikke-eksisterende brugernavn eller hvis brugernavn og password ikke passer sammen.
- 05. *Username already taken*: Viser i et popup-vindue, hvis der forsøges at oprette en bruger med et allerede eksisterende brugernavn.



Figur 1: Fejlmeddelelse 05.

- 06. *Movie already on watchlist*: Viser i et popup-vindue, hvis der forsøges at tilføje en film der allerede eksisterer på brugerens watchlist.



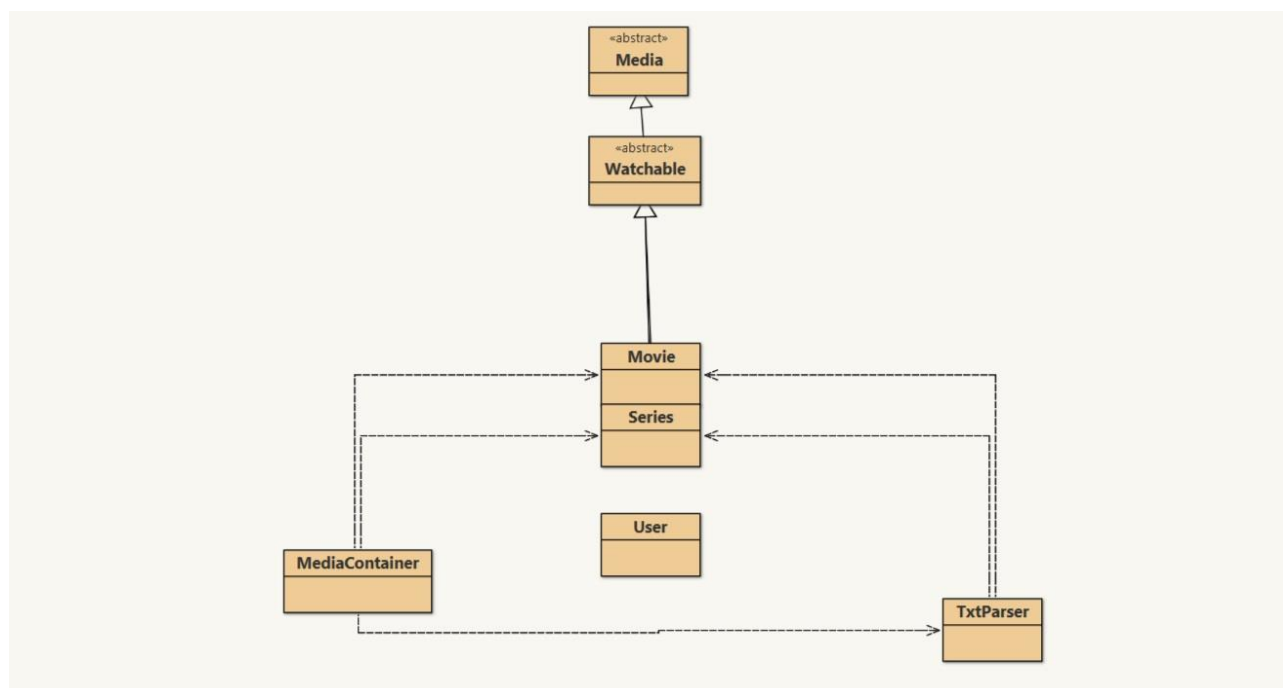
Figur 2: Fejlmeddelelse 06.



## 5. Teknisk beskrivelse

### 5.1. Intern struktur

Klassen *Main* indeholder en metode af samme navn. Når denne metode køres, startes programmet og der initialiseres et skærbillede (*login\_view*). Den tilhørende controller<sup>1</sup> initialiseres dermed også her. Ved tryk på 'Create User' vises *CreateUser\_view* skærbilledet, som også har en controller tilknyttet (*CreateUserController*). Denne controller varetager oprettelsen af *User*-objekter. Når en *User* er oprettet, tilføjes denne til en *ArrayList<User>* *activeUsers* der holdes af *LoginController*. Det er denne *ArrayList*, der løbes igennem og tjekkes for matchende brugernavn og password, når en bruger forsøger at logge ind på tjenesten. *LoginController*-, *CreateUserController*- og *MainWindowController*-klasserne sender alle data fra modeldelen af programmet til deres tilhørende *View*-klasser.



Figur 3: Klassehierarki for model-delen af programmet.

*Media*-klassen indeholder al data om medierne i programmet, der på nuværende tidspunkt kun består af film og serier. Det er en abstrakt klasse, da der ikke skal oprettes *Media*-objekter på noget tidspunkt; derimod oprettes der objekter af subklasserne *Movie* og *Series*. *Media*-klassens hovedfunktion er i programmets nuværende udformning som parent til klassen *Watchable*, der igen er

<sup>1</sup> LoginController

superklasse til *Movie* og *Series*. *Media*-klassen eksisterer, da det så vil være lettere at tilføje andre slags medier end *Watchable* til streamingtjenesten. De vigtigste felter i *Media*-klassen er: en *String* *title*, *int* *year*, *double* *rating*, *List<String>* *genres* og et *Image* *img*. Samtlige felter er *protected*, da subclasserne skal kunne tilgå disse.

*Movie*-klassen nedarver alle felter og metoder fra *Watchable*. *Series*-klassen indeholder derimod fem metoder, som kaldes af *MainWindowController*-klassen. *Series* har en række accessor-metoder, som returnerer dataen for det enkelte objekt af klassen, samt en mutator-metode *addEpisodeCount()*, der tilføjer antallet af episoder i en bestemt sæson til en liste, som tilgås af *MainWindowController*-klassen.

*TxtParser*-klassens ansvar er at læse dataen i tekstfilerne *movies.txt* og *series.txt*, vha. af en *BufferedReader* for derefter, at splitte dem op ved semikolon vha. *BufferedReader*-klassens *split()*-metode og til slut oprette *Movie* og *Series*-objekter med den tilhørende data fra tekstfilerne (*movies.txt* og *series.txt*). Alt dette gøres med en metode *parseSeries()* for *Series*-objekter og en metode *parseMovies()* for *Movie*-objekter. Til slut i disse metoder tilføjes objekterne til *MediaContainer*'ens *ArrayList* af hhv. *Series* og *Movies*.

*MediaContainer*-klassens ansvar er at opbevare *ArrayList*erne af *Watchable*-objekter (altså film og serier). Dette er nødvendigt for, ved søgning efter- og visning af elementer i streamingtjenesten, at kunne gennemløbe listerne af diverse medier. *MediaContainer*-klassens væsentligste metoder er *loadMovies()* og *loadSeries()*. Disse kalder *TxtParser*-klassens *parseMovies()* og *parseSeries()* samt tilføjer de korrekte billeder til *Watchable*-objekterne vha. et for-loop hvor *Media.setImg()* kaldes på hvert objekt.

Ved udvidelse af streamingtjenesten til inklusion af andre medier, vil *ArrayList* af disse skulle tilføjes til *MediaContainer*-klassen.

*LoginController* indeholder kun to metoder. Den ene åbner FXML-filen *createUser\_view* og initialiserer dermed den tilhørende controller, hvorved en bruger af programmet kan oprette en konto. Den anden varetager selve login-funktionen. Her gennemløbes listen af brugere, og for hver bruger tjekkes det om teksten i *username*- og *password*-felterne stemmer overens med en brugers *username* og *password*. Dette gøres vha. *User*-klassens accessor-metoder *getUsername()* og *getPassword()*. Hvis betingelsen opfyldes, sættes programmets *currentUser* til at være den givne bruger, og hovedvinduet *mainWindow\_view* åbnes. Hvis betingelsen ikke opfyldes, sættes status-teksten øverst til venstre til at være "ERROR 04; Login Failed".

*CreateUserController*’s hovedansvar er, ved klik på vinduets OK-knap, at tjekke brugerens input for fejl og mangler. Hvis der ved oprettelse af en konto ikke er valgt et brugernavn, oplyses brugeren om at brugernavn mangler med fejlmeddelelse 01. Ligeledes bliver brugeren gjort opmærksom på manglende kodeord eller manglende overensstemmelse mellem kodeordene i de to felter (se afsnit 4.2. om fejlmeddelelser). Hvis programmet ikke finder disse fejl, bliver to booleans, *usernameChecked* og *passwordChecked*, sat til at holde værdien true, og metoden kører et sidste if-statement. Her gennemløbes listen af *Users* og der tjekkes om det valgte brugernavn allerede eksisterer. Hvis det gør, kastes en *UsernameTakenException*, og brugeren bliver gjort opmærksom på fejlen ved et popup-vindue (se evt. afsnit 4.2.). Hvis brugernavnet ikke er taget, oprettes en bruger, og denne tilføjes til *ArrayList<User> activeUsers*. Derefter lukkes *createUser\_view* og login-skærmen ses igen.

Hvis ikke der fanges en fejl eller en *Exception* i *login()* metoden, åbnes *mainWindow\_view*. Dette vindue benytter controlleren *MainWindowController*. Denne klasse har et privat felt med en *MediaContainer* medias, der kommer i spil i controllerens constructor. Her forsøges kald til *MediaContainer*’s *loadSeries()*, *loadMovies()* og *joinLists()* metoder. Hvis dette er uden succes, gribes en *Exception*. Desuden er det også denne klasse der varetager de forskellige søgemetoder. Disse metoder gennemløber, på baggrund af tre booleans *searchMovies*, *searchSeries* og *searchAll*, en af de tre kategorier, med det givne parameter (brugerens input i søgefeltet eller klik på en bestemt genre-knap fra dropdown-menuen). De tre booleans sættes af brugeren ved klik på knapperne ‘Movies’, ‘Series’ eller ‘All’. Udover dette har *MainWindowController* en metode *showWatchlist()*, *watchlistAdd()* og *watchlistRemove()*. Disse viser brugerens watchlist og hhv. tilføjer- eller fjerner et medie fra listen. *watchlistAdd()* kaster en *AlreadyOnWatchlistException* og informerer brugeren om fejlen ved et popup-vindue (se evt. afsnit 4.2.). De metoder, der viser *Media*-elementerne i det centrale panel, benytter *toRuleThemAll()* metoden.

*toRuleThemAll(Watchable m)* er grundlæggende for hele opbygningen af *MainWindowController* og dermed brugergrænsefladen i programmet, samt uden tvivl den mest komplicerede metode i systemet. Den opretter et *File*-objekt, der peger på en .jpg fil, ud fra et *Watchable*-objekts titel. Dette objekt tages som parameter til metoden. Herefter bruges denne *File*, som parameter i en constructor for et *Image*-objekt, der så kan tilføjes til et *ImageView*<sup>2</sup>-element vha. *ImageView*-klassens *setImage()*-metode. Dette *ImageView* pakkes ind i en *HBox*<sup>3</sup>, der til sidst i metoden tilføjes til gitteret

---

<sup>2</sup> javafx.scene.image.ImageView

<sup>3</sup> javafx.scene.layout.HBox

i det centrale panel. Der oprettes også et *Text*-objekt som indeholder informationer om *Watchable*-objektet vha. dets accessor-metoder. Derefter er en *setOnMouseClicked()* metode tilknyttet *ImageView*. Den har til ansvar at fremkalde *Text*-objektet samt andre relevante data for det valgte *Watchable*-objekt i info-panelet i højre side af brugergrænsefladen. Dernæst rummer *toRuleThemAll()* et if-statement, der tjekker om den nuværende brugers watchlist indeholder det valgte objekt. Hvis ja, sættes 'Remove from watchlist' knappen til at være synlig. Hvis nej, gør den ikke.

*toRuleThemAll()* tilføjer også et tooltip til *ImageView*'et med *Media*-objektets titel, rating, genrer og årstal. Metoden skelner mellem film og seriers infotekst vha. et if-statement, hvor den tjekker om metodens parameter (*Watchable*-objektet *m*) er *instanceof Movie* eller - *Series*. For *Series*-objekter opretter metoden to nye *MenuButtons*<sup>4</sup>; én til valg af sæson, og én til valg af episode. Sæsonknappen oprettes først vha. et for-loop, hvor den kalder *Series*-klassens *getSeasons()*-metode, der returnerer en *int*, som angiver antal af sæsoner. Altså oprettes der et *MenuItem*<sup>5</sup> for hver sæson.

Disse har en tilhørende *EventHandler*, som har til ansvar at oprette *MenuItems* dynamisk for hver afsnit i sæsonen. Dette gøres vha. et for-loop, hvor metoden kalder *Series*-klassens *getEpisodeCount()*-metode, som tager et index som parameter og returnerer antallet af episoder. Her kalder loopet indexet fra det ydre loop, for så at hente antallet af episoder, i den sæson, det ydre loop gennemløber.

## 5.2. Brugergrænseflade

Brugergrænsefladen i RickFlix er bygget op ved brug af JavaFX biblioteket. Vi har hovedsageligt brugt værktøjet *SceneBuilder*, hvor man på grafisk vis kan bygge FXML-koden til de forskellige skærm billeder. *SceneBuilder* fungerer ved at man lægger forskellige elementer i lag, for på denne måde at bygge skelettet til ens brugergrænseflade.

Da vi har tre forskellige *Stages*<sup>6</sup>, har vi lavet illustrationer af opbygningen for hver af disse. Dette afsnit udpensler opbygningen af disse tre skærm billeder.

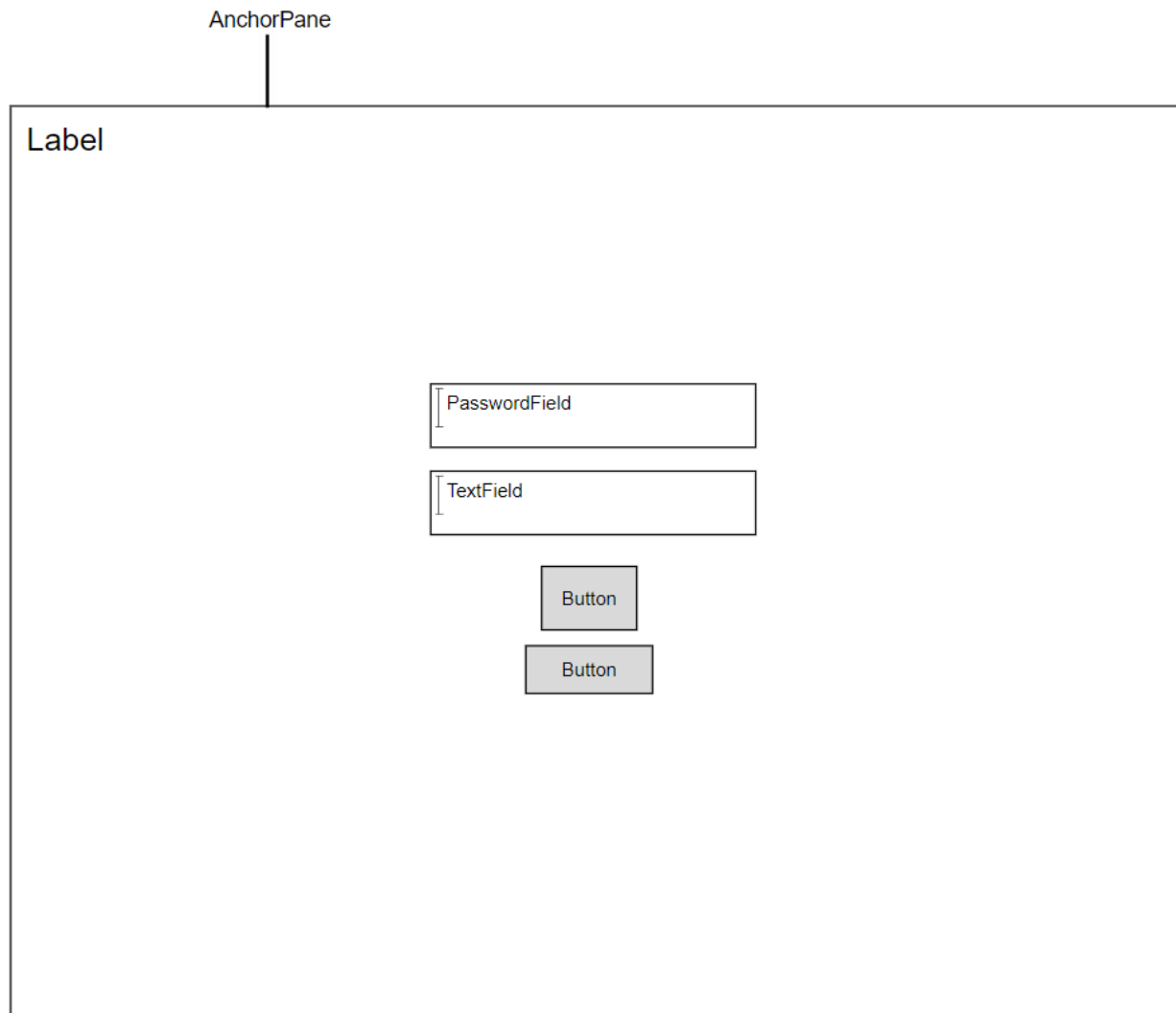
### 5.2.1. Loginskærmen

---

<sup>4</sup> javafx.scene.control.MenuButton

<sup>5</sup> javafx.scene.control.MenuItem

<sup>6</sup> javafx.scene.Stage



Figur 4: *login\_view*

I figur 8 er applikationens *login\_view* illustreret. Bagest i vinduet er et *AnchorPane*<sup>7</sup>, hvilken vi har placeret et *Label*<sup>8</sup> foran i øverste venstre hjørne. Et *AnchorPane* er brugt, da det gør det let at placere objekter med faste pladser på skærbilledet. I dette tilfælde indeholder det en tekststreng med en login-status, som ændres på runtime; et *TextField*<sup>9</sup>, hvor brugeren indtaster sit brugernavn, og et *PasswordField*<sup>10</sup>, hvor brugeren indtaster sit kodeord. Der er også indsat to knapper (*Button*<sup>11</sup>), som tilknyttes unikke fx:id. Disse ID'er muliggør ændringer af objekternes tilstande, på runtime, i Java-koden.

<sup>7</sup> javafx.scene.layout.AnchorPane

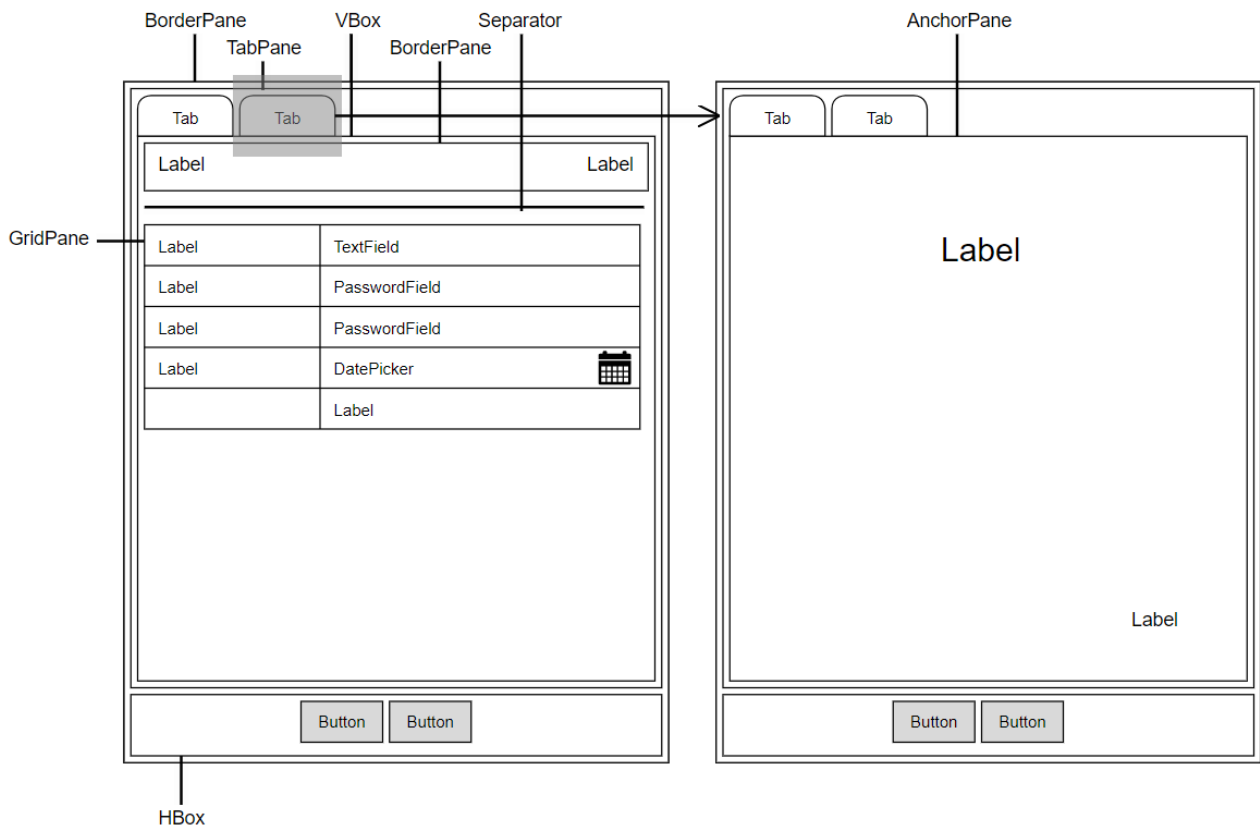
<sup>8</sup> javafx.scene.control.Label

<sup>9</sup> javafx.scene.control.TextField

<sup>10</sup> javafx.scene.control.PasswordField

<sup>11</sup> javafx.scene.control.Button

## 5.2.2. Create User



Figur 5: *createUser\_view*

Figur 9 er en illustration af *createUser\_view* stadiet, som er bygget op af et *BorderPane*<sup>12</sup>, der sætter de ydre grænser. Foran er der et *TabPane*<sup>13</sup>, som i dette tilfælde indeholder to tabs. Den venstre tab er den vigtigste i dette stadie. Her eksisterer endnu et *BorderPane*, som øverst indeholder to *Labels* placeret hhv. til venstre og højre. Under disse er der en *Seperator*, som adskiller de to *Labels* fra et *GridPane*<sup>14</sup>, der opdeles i fem rækker og to kolonner. Som det vises på billedet, indeholder de fire første rækker alle et *Label* i venstre kolonne, som viser en tekststreng der informerer om, hvad brugeren skal indsætte i *TextField*, *PasswordFields* og *DatePicker*<sup>15</sup>. Disse ses i højre kolonne. Under *GridPanet* er indsat en *HBox*; dette element placerer child-elementer horisontalt. Child-elementerne er i dette tilfælde to knapper, grafisk adskilt fra resten af stadiet, så brugeren intuitivt kan finde dem (Syzonenko, 2019). *DatePicker*en åbner en kalender, hvor det er muligt at vælge år og fødselsdato. Den anden tab indeholder pt et *AnchorPane* og et *Label*, som blot skal informere om, at der stadig arbejdes på funktionaliteten i denne tab. Der er ikke indsat andet her. Stadiet bibeholder i

<sup>12</sup> javafx.scene.layout.BorderPane

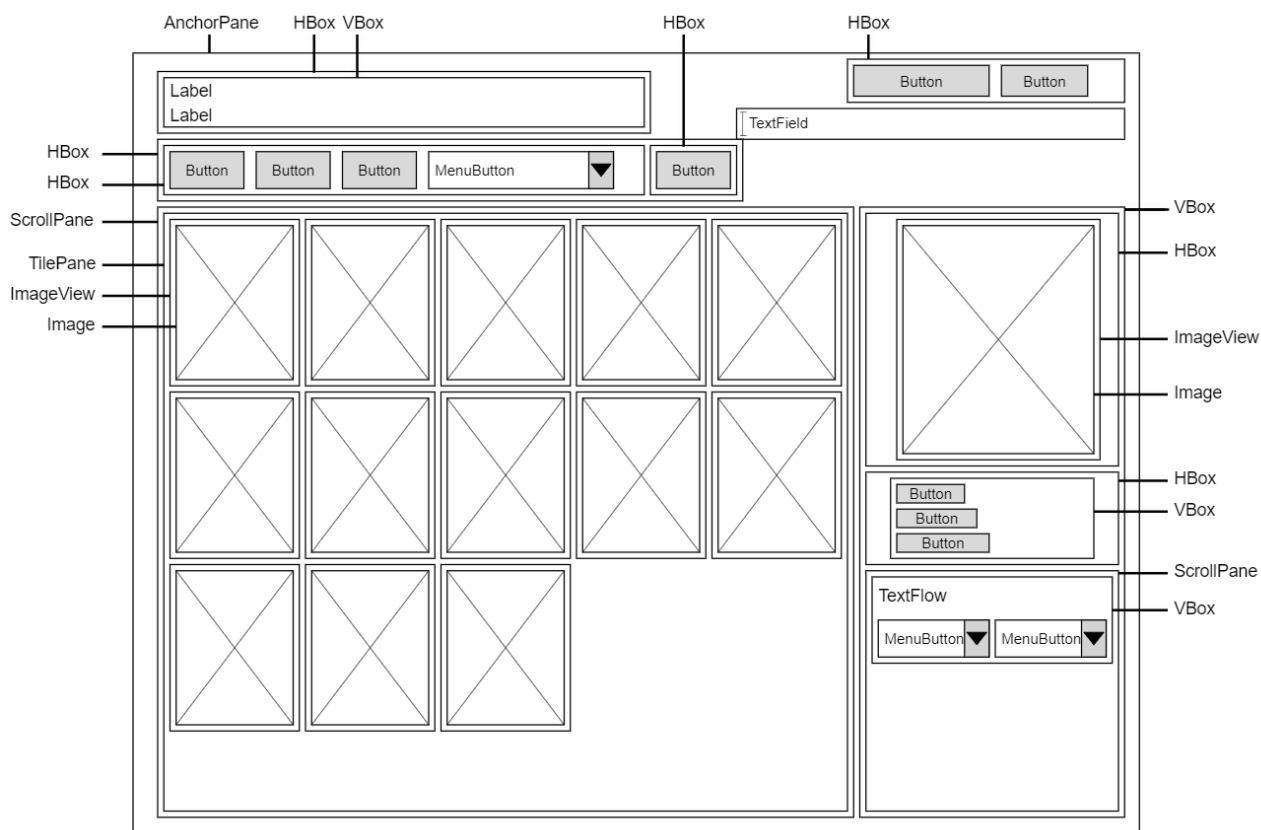
<sup>13</sup> javafx.scene.control.TabPane

<sup>14</sup> javafx.scene.layout.GridPane

<sup>15</sup> javafx.scene.control.DatePicker

denne tab sine knapper i bunden, for at mindske forvirring hos brugeren. Samtlige komponenter i de to ovenstående stadier er placeret og navngivet i SceneBuilder, da indholdet ikke skal oprettes dynamisk på runtime.

### 5.2.3. Hovedvinduet



Figur 6: mainWindow\_view

I figur 10 ses hovedvinduet. Ligesom i *createUser\_view* stadiet er et *AnchorPane* den bagerstliggende komponent. Øverst i denne er en *VBox*<sup>16</sup>, der organiserer dens to *Labels* vertikalt, pakket ind i en *HBox*. Nedenunder er en *HBox*, som omslutter to andre *HBox* komponenter. Den ene indeholder tre *Buttons* og én *MenuButton* og den anden endnu en *Button*. Disse er placeret lige over det centrale panel, da tryk på knapperne bestemmer indholdet af panelet umiddelbart nedenunder. Dette er et *ScrollPane*<sup>17</sup> som omslutter et *TilePane*, hvilket holder de enkelte *Media*-objekters *ImageView*s (oprettes med *toRuleThemAll()*-metoden, se evt. afsnit 5.1.). Disse *ImageView*s sættes i fem rækker, da dette gav det skarpeste udtryk samt bibeholdt en funktionel og brugervenlig størrelse på *Media*-objekternes plakater.

<sup>16</sup> `javafx.scene.layout.VBox`

<sup>17</sup> `javafx.scene.control.ScrollPane`

Disse *ImageViews* er de mest komplicerede elementer i programmet, eftersom de bliver oprettet dynamisk på runtime.

Scrollbaren i panelet har vi valgt at usynliggøre, da vi fandt at dette så pænest ud, og da den gængse bruger alligevel benytter et mousewheel eller et trackpad til navigation i vinduer.

*ImageView*-komponenten muliggør interaktion med det enkelte element, så brugeren ved klik kan vælge mediet. Oplysninger om mediet vil derefter blive vist til højre i stadiet i en *VBox*, der indeholder et *ImageView*, som er pakket ind i en *HBox*. Under dette har vi indsat en *VBox* og pakket den ind i en *HBox*. Denne *VBox* indeholder tre knapper, hvoraf kun to umiddelbart er synlige. Hvis mediet er tilføjet til brugerens watchlist, synliggøres den sidste knap, hvis ansvar er at fjerne det givne medie fra watchlisten.

Under dette har vi indsat et *ScrollPane*, hvis funktionalitet på nuværende tidspunkt ikke bliver udnyttet. Dog, hvis man på et senere tidspunkt ville tilføje en forklarende tekst til medierne, ville *ScrollPane* gøre det langt mere overskueligt for brugeren, uden at gå på kompromis med den visuelle integritet. *ScrollPane* indeholder et *TextFlow*, som viser information om det valgte medie. I tilfælde af, at mediet er en serie, vil der oprettes to *MenuButtons* vha. *toRuleThemAll()*-metoden. Ved valg af sæson, vil højre *MenuButton* dynamisk opdateres til at indeholde lige så mange episoder, som den valgte sæson indeholder.

## 5.3. Dokumentation

For at overskueliggøre programmet og gøre videreudvikling lettere, er der skrevet Javadoc kommentarer til essentielle klasser og metoder. Disse kommentarer udpensler de enkelte klasser og metoders ansvarsområder og kernefunktionaliteter. Desværre fejlede eksport af Java-dokumentationen til en HTML-fil, hvorfor den ikke er vedlagt i bilagene. Dog eksisterer kommentarerne i selve source-koden.

## 6. Afprøvning

Vi har løbende i implementeringsfasen gjort brug af white-box testing og skrevet og udført unit tests. Det gjorde vi for at fange kritiske fejl på et tidligt tidspunkt, og for at sikre at alle de enkelte klasser i programmet fungerede som de skulle. Hvis ikke, faciliterede disse test lokation af fejlene, hvorefter de kunne udredes og klasserne forbedres. Klassernes funktionalitet blev ved unit-testene valideret vha. *System.out.println()*-metoden, hvorved informationer om de enkelte objekter som f.eks. et *User*- eller *Media*-objekt kunne printes og verificeres. Dette sikrede at enkeltdelene i programmet virkede, inden de blev sammensat til en helhed. Vi kunne med fordel have skrevet flere negative test,



da de omtalte unit tests udelukkende er lavet som positive test. De vigtigste af disse test er vedlagt i bilag 10.2.

Efter modeldelen af programmet var testet, og hvis den fungerede til tilfredshed, fortsatte arbejdet med brugergrænsefladen og derefter controller-klasserne. Vi har testet brugergrænsefladen ved at køre programmet og dobbelttjekke, at de komponenter, der blev oprettet i SceneBuilder, sad hvor de skulle. Derefter skrev vi controller-klasserne, hvorfor disse er blevet testet til sidst. Ligesom ved test af brugergrænsefladen, kørte vi programmet for at teste disse. Vi gennemspillede i fællesskab brugsscenarier af applikationen. Herved opdagede vi fejl i controller-klassens metoder, der enten ikke oprettede de elementer de skulle, placerede disse forkert eller måske endda gjorde noget helt tredje. I denne fase af arbejdsprocessen har vi især gjort brug af at arbejde i iterationer. Dvs. vi har løbende testet, for så at vende tilbage til analysen af controller-klasserne, hvorefter klassernes metoder rettes, for så at teste igen. Dette foregik indtil vi fandt os tilfredse med vores produkt. Afslutningsvis testede vi programmet destruktivt ved at gennemgå brugervejledningen med henblik på at lave så mange fejl som muligt. I denne afsluttende test opdagede vi en logisk fejl i systemet: det ikke er muligt at benytte søgefeltet, hvis en bestemt genre er valgt, eller hvis det er brugerens watchlist, der vises. Derfor tilføjede vi disse bemærkninger til brugervejledningen for at gøre opmærksom på dette. Var det opdaget tidligere i arbejdsprocessen, ville vi med fordel kunne have rettet de relevante metoder og have forsøgt at udrede problemet.

I opgaveformuleringen blev en række krav til projektet fastlagt (se afsnit 2., tabel 1 og 2). Ift. kravene til almindelige arbejdsopgaver for en streamingtjeneste, opstillet i tabel 1, er vi i gruppen kommet i mål med at tage højde for at generalisere de noget specifikke krav. Derved opfylder funktionaliteten i programmet kravene opstillet i tabellen og mere til. Det kan dertil siges, at RickFlix understøtter de arbejdsopgaver som kendes fra diverse streamingtjenester.

Ift. de systemmæssige krav, opstillet i tabel 2, opfylder programmet næsten alle disse. Dog har vi ikke opfyldt kravet om at have én central 'resources'-mappe, da vi ved eksport af projektet til en executable .jar fil var nødsaget til at ændre mappestrukturen. Derfor ligger series- og movies.txt i model-mappen og 'pictures'-mappen i controller-mappen.

Havde vi haft længere tid til projektet, kunne vi med fordel have forsøgt at skrive unit tests til controller-delen af programmet også. Til gengæld har vi søgt at teste denne tilstrækkeligt ved gennemgang af diverse brugsscenarier af applikationen. Da et program har uendeligt mange valide og

invalide brugerinput, kan vi selvfølgelig ikke konkludere, at programmet er fejlfrit. Dog vurderer vi, at vi har opnået den ønskede funktionalitet samt den ønskede brugervenlighed.

## 7. Refleksion

### 7.1. Beslutningsprocessen

På grund af den stillede opgaves åbne formulering, med mulighed for at løse opgaven på utallige måder, har vi foretaget mange beslutninger undervejs; både ift. hvilke værktøjer vi har gjort brug af, men også hvordan samarbejdet mellem gruppemedlemmerne er blevet organiseret. Nogle beslutninger omkring vores gruppearbejde blev taget under vores allerførste møde, mens konsensus på andre punkter er kommet løbende. Da ingen af os før har programmeret i Java, måtte vores indledende beslutninger omkring samarbejdets overordnede struktur foretages baseret på den viden, vi nu engang havde; en mekanisme, der af James March i 1991 blev beskrevet: “Decisions in organizations involve an ecology of actors trying to act rationally with limited knowledge and preference coherence” (March, 1991: 111).

Vi så på mange måder det afsluttende projekt i faget Grundlæggende Programmering som en mulighed for at tilegne os nogle færdigheder, vi senere kunne bruge på vores uddannelser. Derfor vedtog vi at bruge de første mange mødegange på at få styr på GitHub, IntelliJ, udarbejde en rapportstruktur og snakke om MVC-princippet, som vi lærte om i lektion 19, fremfor at begynde på selve kodningen uden nogen videre plan. Derudover har vi gennem hele projektet opfordret hinanden til at komme med konstruktiv kritik på det arbejde, vi pushede efter hver implementering, for at fremme den glidende arbejdsprocess, så vi kunne tage uenigheder i opløbet.

### 7.2. Vidensdeling

I starten eksperimenterede vi med at have et Kanban board på Trello, som vi løbende opdaterede især for at undgå merge-konflikter i gruppens GitHub repository. Dog fandt vi ret hurtigt ud af, at det fungerede bedst for os bare at sidde sammen fysisk mens vi arbejdede, frem for at arbejde hver for sig, da vidensdeling og konstant dokumentation ellers ville sløve arbejdsprocessen betydeligt.

Vores arbejde har båret præg af, at vi har haft meget stiltiende viden (eng: tacit knowledge) gruppe-medlemmerne imellem, som vi udviklede når vi sad sammen, arbejdede på og snakkede om projektet. Vi har også løbende dokumenteret vores arbejdsprocess samt overvejelser og derved gjort vores viden explicit, når der har opstået refleksioner, der har haft relevans for de forskellige afsnit af

denne rapport. Eksempler på dette er commit-messages i GitHub, vores Kanban board på Trello, mens vi stadig brugte det, og kommentarer i koden.

Undervejs i arbejdet med selve kodningen har vi roteret, så alle fik erfaring med at arbejde på alle tre dele: controller, view og model. Ligeledes benyttede vi os af par-programmering, som også træ- nede os i at arbejde agilt (Noble, 2017). Disse to arbejdspraksisser gjorde, at hvis et gruppemedlem havde et problem i sin kode, kunne de andre medlemmer hurtigt overskue arbejdet - grundet deres overblik over koden - og komme med løsningsforslag.

### 7.3. Designovervejelser

I retrospekt er der nogle ting, vi gerne ville have gjort anderledes.

1. Da der kun skal oprettes én *MediaContainer*, ville det give mening at have lavet den til en singleton, så antallet af instanser af klassen var blevet restrikeret til kun én. Dette har vi ikke gjort, da vi først blev opmærksomme på dette for sent under udviklingen af RickFlix.
2. Vi ville gerne have fundet en måde automatisk at generere genreknapper på, for betydeligt at mindske kodeduplikation i *MainVindowController*-klassen.
3. Søgefeltet fungerer kun til at søge under 'Series', 'Movies' og 'All', og kan ikke bruges til at foretage søgninger i watchlisten eller under en specifik genre. Da vi planlagde controller- klassen, havde vi ikke erfaring nok til at forudse, at vores måde at skrive den på ville medføre denne svaghed ved programmet.

Vi har arbejdet i iterationer og hver dag prioriteret at blive færdige med de mest kritiske elementer, mens vi løbende har evalueret, hvad der ville være muligt at lave fremadrettet og dermed også lavet ændringer i vores oprindelige plan. Vi har dog ikke ændret ved kravene til streamingtjenesten, blot vores plan for hvordan vi opnåede dem. Vi har tilføjet en Play-knap, som på nuværende tidspunkt viser et substitueret videoklip i form af en .gif fil.

## 8. Konklusion

Vi startede ud med at få stillet til opgave at lave en streamingtjeneste. Vi analyserede problemområ- det, specificerede krav for hvad vores program skulle kunne og brugte disse krav til at lave et udkast til den interne struktur samt brugergrænseflade. Vi lavede vores projekt i Java og foretog løbende system- og unit tests, i takt med at vi blev færdige med de forskellige moduler af programmet. Denne rapport forklarer i detaljer hvordan vores program virker, hvordan man bruger programmet og vores

overvejelser omkring udviklingen af dette. Vores samarbejde er mundet ud i streamingtjenesten RickFlix. Overordnet set er vi tilfredse med udformningen af RickFlix, udover tidligere nævnte ændringer (genre-knappers kodeduplikering, manglende søgefunktion ved valg af genre). Vi har alle lært utroligt meget af dette projekt, hvilket vægtes højest for os, og vi vil derfor konkludere at vi er meget tilfredse med både arbejdsprocessen, samt det færdige produkt.

## 9. Litteratur

Hoda, R. & Noble, J. (2017) Becoming Agile: A Grounded Theory of Agile Transitions in Practice. IEEE/ACM 39th International Conference on Software Engineering

March J. (1991) How Decisions Happen in Organizations. Human-Computer Interaction: 6(2): pp 95-117.

Syzonenko, A. (2019) Buttons on the web: placement and order. UX Collective (<https://uxdesign.cc/buttons-placement-and-order-bb1c4abadfcb>) [accessed 18/12-19]

## 10. Bilag

### 10.1. Substantiv-verbum metoden

*Substantiver er grønne og verber er blå.*

The **streaming service** should **enable** a **user** to **browse**, **search** and **watch** different **movies** and **series**. A **movie** **is** a combination of a **title**, **year**, **rating** and a **thumbnail image**. A **series** **has** the same qualities, but also **has** an **end year** as well as a **number of seasons** and **episodes** in those. Before **browsing** and **streaming**, the **user** should be able to **create** a new **profile**, **log in** or **change profiles** and lastly **add movies/series** to his/her **watchlist** and **access** those.

Substantiver	Verber
Streaming service	Enable
User	Browse, search and stream, create (profile)
Movie	Is / has (title, year, rating, thumbnail)
Series	Is / has (... , # of seasons, episodes)
Profile	Log in, change, add movie/series (to watchlist)
Watchlist	Access (movie/series)
Title	
Year	
Rating	
Thumbnail image	
End year	
Number of seasons	
Episode	

## 10.2. Unit test

### 10.2.1. *Media* test

```
package test;

import sample.model.Movie;
import sample.model.Series;

public class MediaTest{

    public void movieTest(){
        String[] s = new String[2];
        s[0] = "ebin";
        s[1] = "win";
        Movie movie = new Movie("The Test of the Rings", 2069, 6.9, s);

        System.out.println(movie.getTitle() + "; " + movie.getYear() + "; "
            + movie.getGenres() + "; " + movie.getRating());
    }

    public void seriesTest(){
        String[] s = new String[2];
        s[0] = "ebin";
        s[1] = "win";
        Series series = new Series("Test of Thrones", 2069, 2070, 0.1, 8, s);

        System.out.println(series.getTitle() + "; " + series.getYear() + "-"
            + series.getEndYear() + "; " + series.getGenres() + "; " + series.getRating());
    }
}
```

### 10.2.2. *User* test

```
package test;

import sample.model.Movie;
import sample.model.Series;
import sample.model.User;
import sample.model.Watchable;

public class UserTest{

    public static void userTest(){
        User user = new User("testUser", "You shouldn't be printing this.");

        String[] s = new String[2];
        s[0] = "testgenre1";
        s[0] = "testgenre2";
        Movie movie = new Movie("testmovie", 1999999, 0.1, s);
        Series series = new Series("testseries", 199238, 123987, 0.1, 1, s);

        user.addToWatchlist(movie);
        user.addToWatchlist(series);

        System.out.println(user.getUsername() + ": " + user.getPassword());
        for(Watchable m : user.getWatchlist()){
            System.out.println(m.getTitle() + "; " + m.getYear() + "; "
                + m.getGenres() + "; " + m.getRating());
        }
    }
}
```



### 10.2.3. TxtParser test

```
package test;

import sample.model.MediaContainer;
import sample.model.Watchable;

public class TxtParserTest {

    public void txtParserTest(){
        MediaContainer medias = new MediaContainer();
        try{
            // loadMovies og loadSeries kalder metoderne parseMovies og parseSeries fra TxtPar-
            // (læser de to .txt-filer og opretter movie og series objekter samt putter d em i en MediaContai-
            // ner)
            medias.loadMovies();
            medias.loadSeries();
            medias.joinLists();
        }catch(Exception e){
            e.printStackTrace();
        }

        for(Watchable m : medias.getJoinedList()){
            System.out.println(m.getTitle() + "; " + m.getYear() + "; "
                               + m.getGenres() + "; " + m.getRating());
        }
    }
}
```

#### 10.2.4. MediaContainer test

```
package test;

import sample.model.MediaContainer;
import sample.model.Movie;
import sample.model.Series;
import sample.model.Watchable;

public class MediaContainerTest{

    public void mediaContainerTest(){
        MediaContainer medias = new MediaContainer();
        String[] s = new String[2];
        s[0] = "ebin";
        s[1] = "win";
        medias.addMovie(new Movie("testmovie", 2019, 0.1, s));
        medias.addSeries(new Series("testseries", 2019, 2020, 0.1, 1, s));
        medias.joinLists();

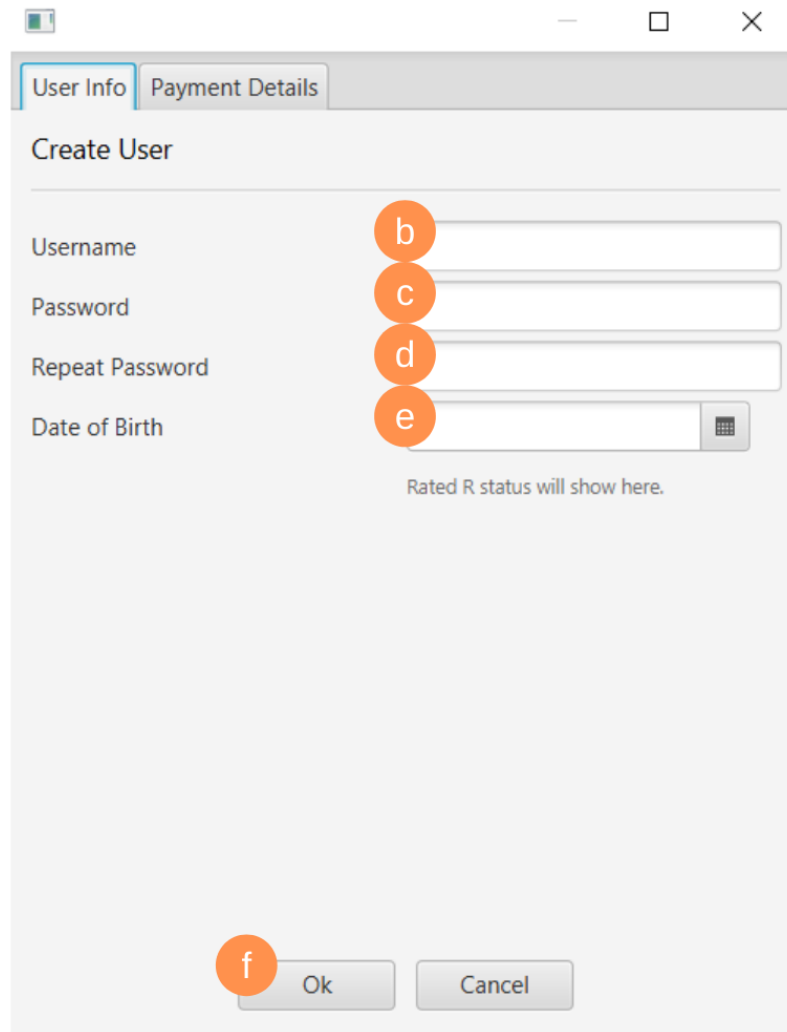
        for(Watchable m : medias.getJoinedList()){
            System.out.println(m.getTitle() + "; " + m.getYear() + "; " + m.getGenres() + "; " + m.getRating());
        }

        for(Watchable m : medias.searchTitle("test")){
            System.out.println(m.getTitle() + "; " + m.getYear() + "; " + m.getGenres() + "; " + m.getRating());
        }
    }
}
```

## 10.3. Brugervejledning

For at bruge programmet er det nødvendigt at brugeren først opretter en konto. Herefter skal brugeren logge ind for at bruge programmet. Når brugeren er logget ind, kan denne afspille, søge efter og tilføje film og serier til sin watchlist.

### 10.3.1. Navigation i login



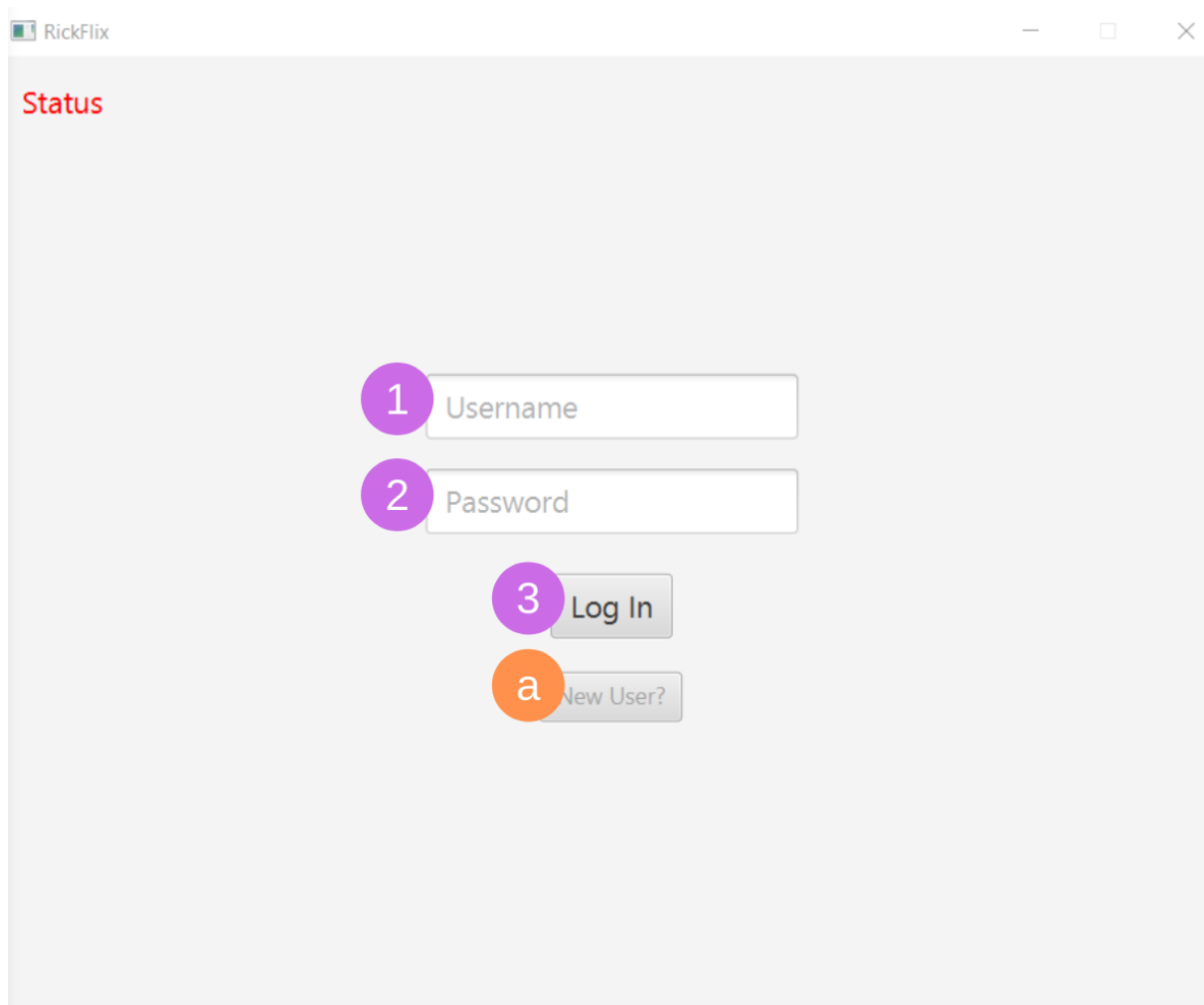
The image shows a 'Create User' dialog box with two tabs: 'User Info' (selected) and 'Payment Details'. The dialog contains four input fields: 'Username', 'Password', 'Repeat Password', and 'Date of Birth'. The 'Date of Birth' field has a calendar icon. Below the fields is a text label 'Rated R status will show here.' At the bottom are 'Ok' and 'Cancel' buttons. Annotations a-f are placed as follows: 'a' is on the 'User Info' tab, 'b' is on the 'Username' field, 'c' is on the 'Password' field, 'd' is on the 'Repeat Password' field, 'e' is on the 'Date of Birth' field, and 'f' is on the 'Ok' button.

Figur 7: Brugeroprettelsesvindue.

Opret bruger:

- Når programmet startes, vil brugeren blive præsenteret for en loginskærm. Klik på “New User?”, for at åbne brugeroprettelsesvinduet.
- Udfyld “Username” med ønsket brugernavn. Bemærk at der tages højde for store og små bogstaver.
- Udfyld “Password” med ønsket password. Der tages højde for store og små bogstaver.

- d) Gentag det valgte password i feltet “Repeat Password”.
- e) Brugeren kan vælge at udfylde “Date of Birth” med sin fødselsdato.
- f) Tryk på “Ok” for at oprette en bruger og gå tilbage til loginvinduet.

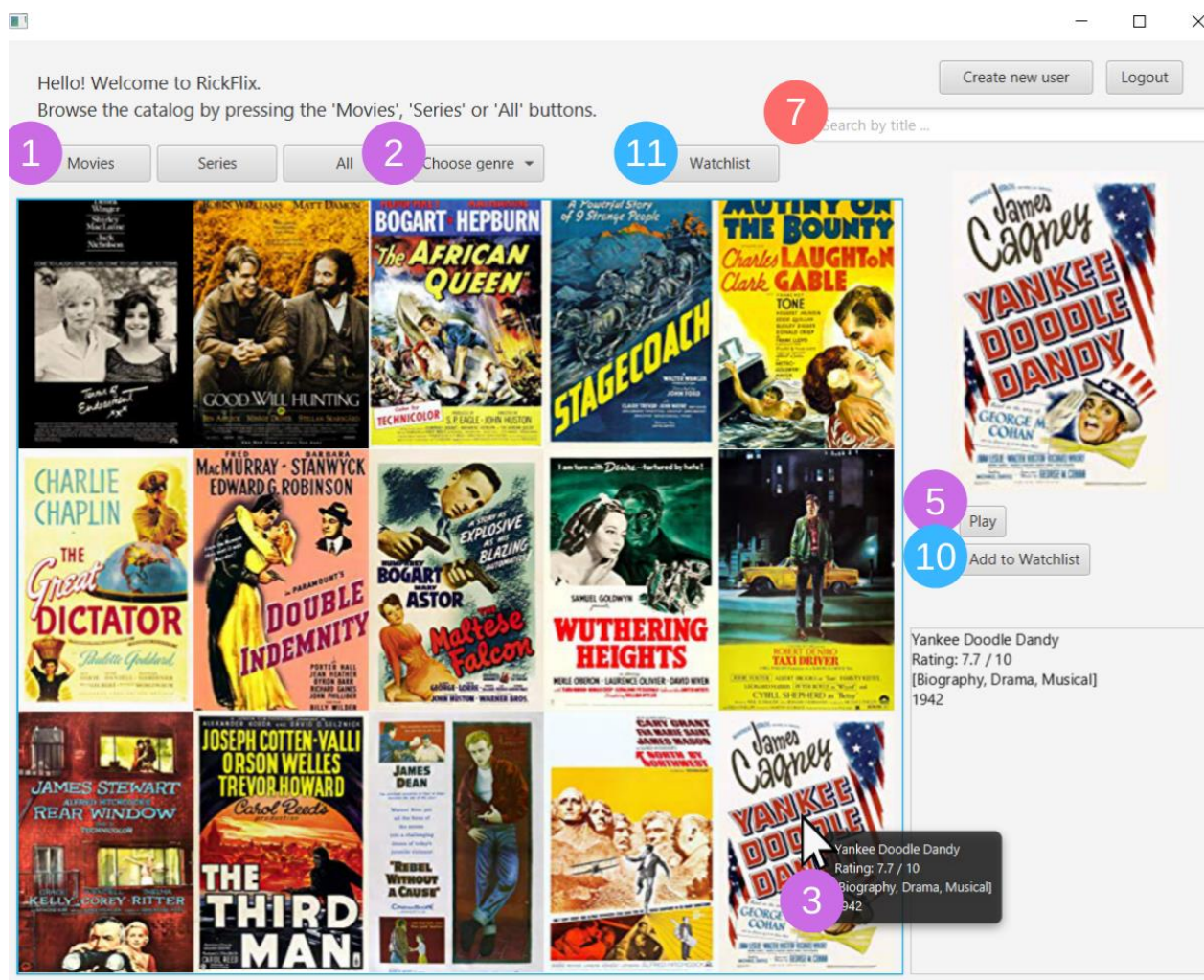


Figur 8: Login-vindue.

Log på RickFlix:

1. Udfyld “Username” med det valgte brugernavn fra punkt b. Bemærk at der tages højde for store og små bogstaver.
2. Udfyld “Password” med det valgte password fra punkt c. Der tages højde for store og små bogstaver.
3. Tryk på “Log In” for at åbne selve tjenesten.

### 10.3.2. Navigation i hovedvinduet



Figur 9: Hovedvinduet.

Afspil en video:

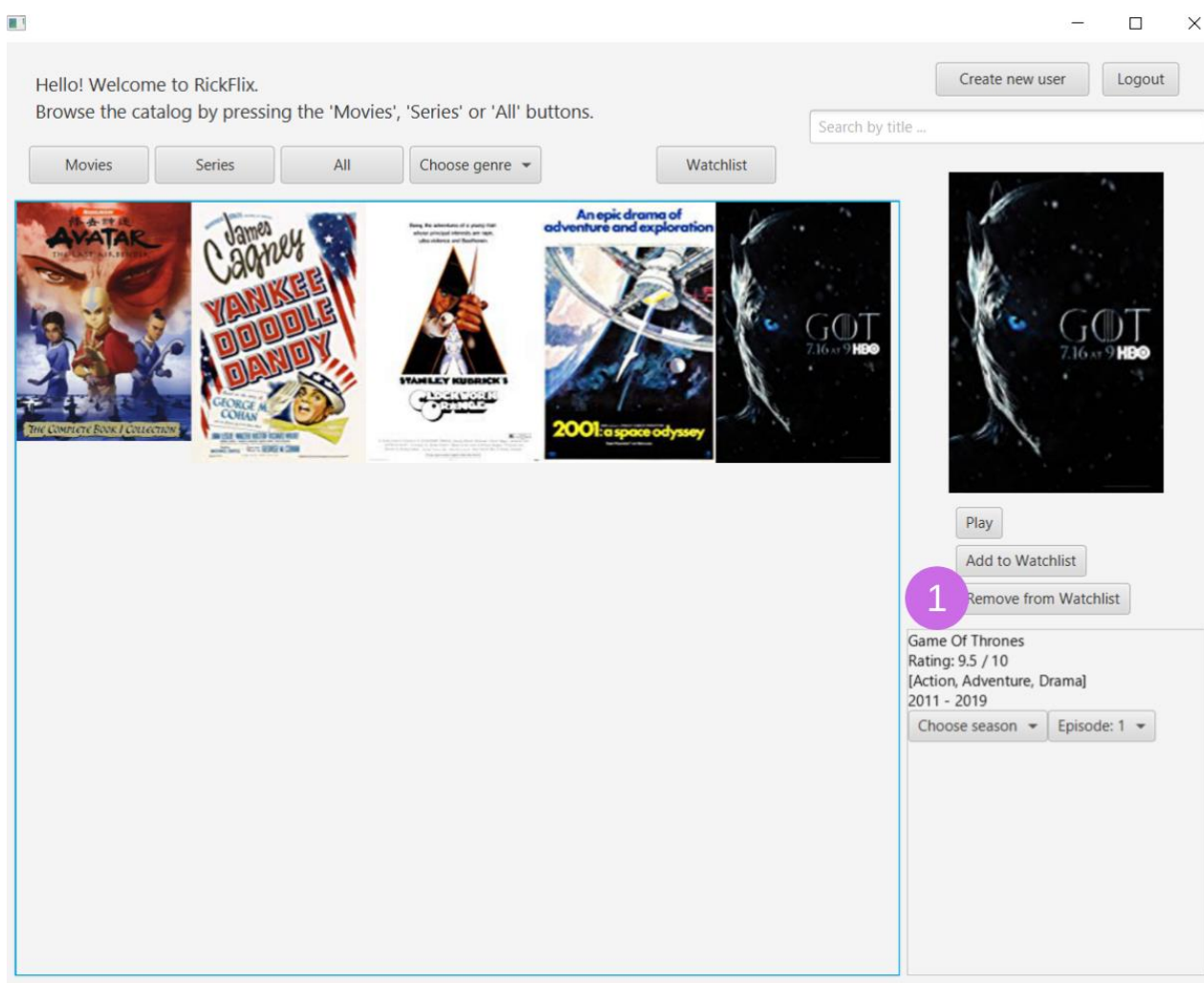
- 1) Når brugeren er logget ind trykkes der på "Movies", "Series" eller "All". Dette frembringer listen af film- og/eller serieplakater.
- 2) Begræns listen til en bestemt genre ved at klikke på "Choose genre" og vælge en genre fra drop-down menuen. Bemærk: det er ikke muligt at søge efter titel indenfor en bestemt genre.
- 3) Hold musen henover et element for at se flere detaljer.
- 4) Klik på elementet for at vælge det.
- 5) Tryk på "Play" for at afspille.
- 6) For serier er det også muligt at vælge en bestemt sæson og episode.

Søg efter film eller serie:

- 7) Klik på søgefeltet.
- 8) Begynd at skrive; listen opdateres løbende.
- 9) Begræns søgningen til film og serier ved at vælge hhv. "Movies" eller "Series" (Se punkt 1) .

Tilføj element til brugerens watchlist:

- 10) Når element er valgt, tryk på "Add to Watchlist".
- 11) Klik på "Watchlist" for at få vist alle elementer på watchlisten. Bemærk: Det er ikke muligt at søge i watchlisten.



Figur 10: Watchlisten vist i hovedvinduet..

Tilgå eller fjern element fra watchlisten:

- 1) Når watchlisten er åben kan elementer vælges ligesom i resten af programmet.
- 2) Klik på "Remove from Watchlist" for at fjerne elementet fra listen.