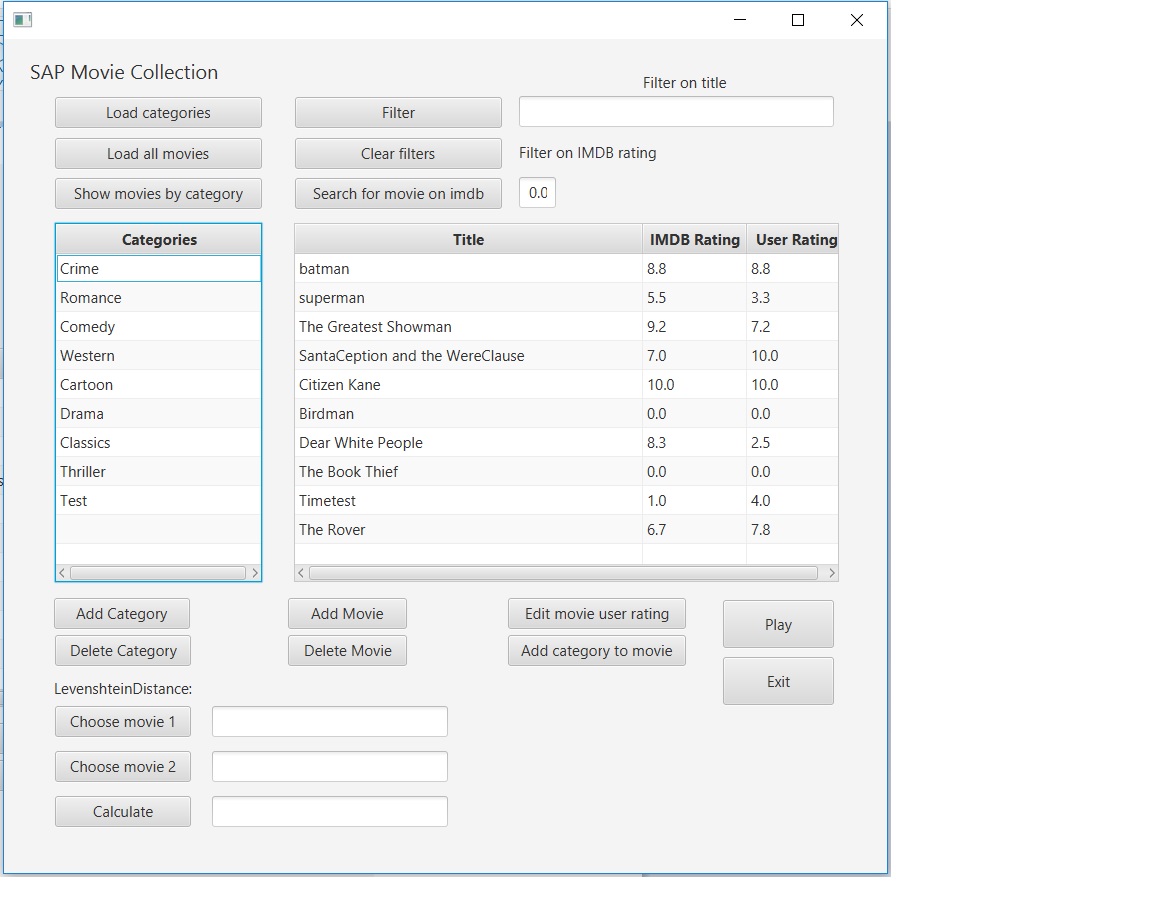
Private Movie Collection



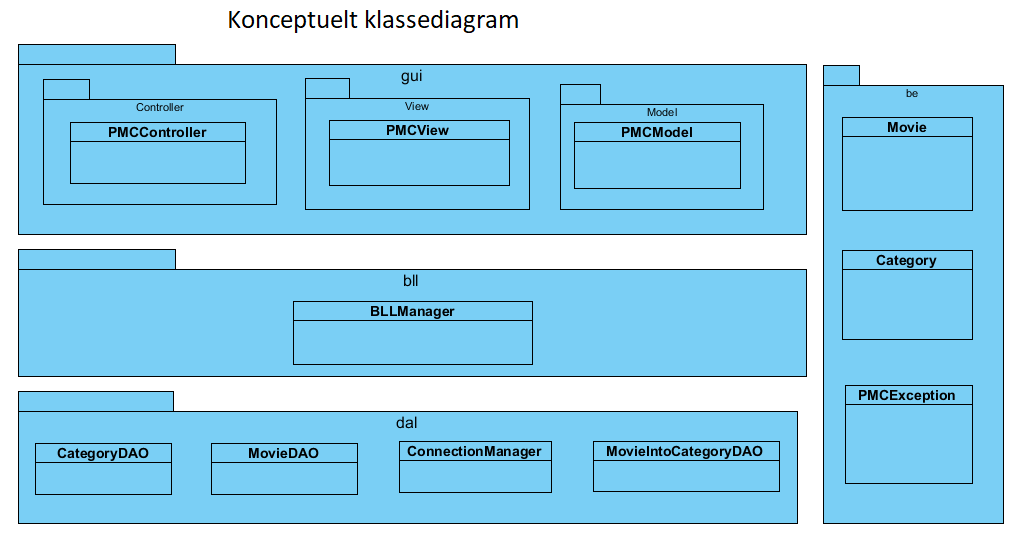
1. Make a requirement list for the system.

Vi har I programmet sikret at få alle de påkrævede funktionelle egenskaber med.

Her i blandt er følgende:

* De vigtigste fejlmeddelelser skal vises på brugerfladen.
* Man skal kunne gemme og slette film.
* Man skal kunne gemme, rette og slette genre.
* En film skal kunne have flere genrer.
* Der må ikke være 2 af den samme film i kollektionen
* Man må ikke kunne ændre på IMDb rating.
* Man skal kunne ændre sin personlige rating.
* Hvis en film har under 6 personlig rating eller ikke har været afspillet i mere end 2 år gammel, skal denne vises for brugeren ved programmets opstart, hvor det også skal være muligt at slette den.
* Man skal kunne filtrere på navn, genre og rating.
* Man skal kunne sortere på personlig rating, IMDb rating og titel i alfabetisk rækkefølge. Også mens filteret er aktivt.

1. Make a class diagram for the system.



Som det kan ses ud fra vores klassediagram, har vi gjort brug af og overholdt trelags-arkitekturen. Vi vil i fremtiden placere vores exception-klasse i en utility-package, da vi er blevet gjort opmærksomme på under projektets afslutning, at exceptions ikke er be-objekter.

1. Implement the user interface for the system.

Vi har valgt at benytte opgavens billede som skabelon og er dermed udgangspunktet for vores brugerflade. Vi har så selv slettet og tilføjet relevante funktionelle egenskaber efter behov. Vi er gået efter at lave det så let anvendeligt som muligt og vægtet funktionalitet over design. Der er blevet lagt meget vægt på fejlsikring, ikke blot exception håndtering, men også at programmets funktioner ikke kan anvendes på måder som ikke er hensigten.

1. Implement 3-layered architecture.

Vi har fulgt MVC og 3-lags arkitekturen. Vi har så vidt muligt forsøgt at overholde funktionen af de forskellige lag, ved at vores GUI kalder BLL, som kalder DAL, som sender og modtager data fra vores SQL server. På GUI plan er det controllere der kalder modellen, som derfra kalder videre ned gennem lagene. Der kan dog forekomme logik i vores controllere, som vi godt ved hører til i BLL laget, hvilket enten skyldes tidsnød eller usikkerhed herom.

1. All errors generated and thrown as exceptions must be handled by the user interface layer.

For at gøre dette har vi lavet vores egen exception klasse I BE, som har til formål at håndtere vores exceptions. I klassen har vi constructors som vi bruger til at konvertere forskellige exceptions om til vores egne ’throwable’ exceptions.

*public class PMCException extends Exception {*

*public PMCException() {*

*}*

*public PMCException(String string) {*

*super(string);*

*}*

*public PMCException(String string, Throwable thrwbl) {*

*super(string, thrwbl);*

*}*

*public PMCException(Throwable thrwbl) {*

*super(thrwbl);*

*}*

*}*

Alle de steder i vores program hvor der forekommer exceptions, hvad end det er SQL, server eller IO, kan vi catche, logge og sende dem videre op i systemet som PMCException.

*catch (SQLException ex) {*

*Logger.getLogger(MovieDAO.class.getName()).log(Level.SEVERE, null, ex);*

*throw new PMCException("An error occurred. You cannot create a movie");*

Her ses hvordan vores exception bliver konverteret og sendt videre op i systemet med strengen. I strengen skriver vi så en kort begrundelse for hvorfor den pågældende fejl er opstået.

Når vores exception ender oppe i gui laget har vi en statisk metode hvor vi laver en ny instans af fejlbeskeden (strengen) og gemmer den i errorMsg som senere anvendes.  
Det er sat op på en sådan måde at fremkommer der en fejl, vil vores program åbne et nyt fxml vindue med fejlbeskeden.

*static void exceptionHandler(Exception ex) {*

*try {*

*String errorMsg = ex.getMessage();*

*FXMLLoader fxmlLoader1 = new FXMLLoader(ExceptionMessengerController.class.getResource("/privatemoviecollection/gui/View/exceptionMessenger.fxml"));*

*Parent root = (Parent) fxmlLoader1.load();*

*ExceptionMessengerController emc = fxmlLoader1.getController();*

*emc.setErrorMsg(errorMsg);*

*Stage stage = new Stage();*

*stage.setScene(new Scene(root));*

*stage.show();*

*} catch (IOException ex1) {*

*Logger.getLogger(PrivateMovieCollectionController.class.getName()).log(Level.SEVERE, null, ex1);*

*}*

For at kunne vise de forskellige fejl til brugeren, har vi i vores (ExceptionMessenger.fxml) et textarea, hvor den pågældende fejlmeddelelse fremvises ved at sætte errorMsg, altså fejlmeddelelsen, ind i vores textarea, som vist i nedenstående kode.

*public void setErrorMsg(String errorMSg)*

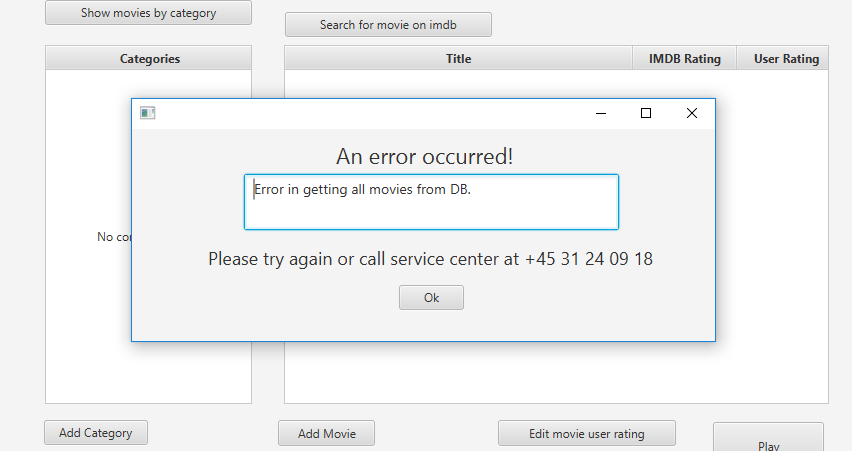
*{*

*txtErrorMsg.setText(errorMSg);*

*}*

1. Test your system by generating some test data.

Vi har gennem forløbet lavet mange tests for at sikre at programmet ikke blot virker men også giver de ønskede fejlmeddelelser. Vi har dog ikke haft i tankerne at bogføre eller på anden vis gemme nogle af de tests vi har foretaget for at kunne fremvise dem i rapporten.



Her ses hvordan vores program for eksempel viser en fejlmeddelelse, hvis man forsøger at loade alle film, men ikke er forbundet til internettet.

Ekstra opgaver:

1. Af ekstraopgaver har vi implementeret to funktioner.

I den ene kan man markere en film og ved at trykke på en knap, få vist søgeresultaterne af filmen på imdb.com.

4. Derudover har vi vist hvordan Levenshtein-algoritmen fungerer. Lige nu viser den kun et resultat ud fra en sammenligning af to ord. Den kan bruges til at sortere i eller filtrere på en liste af characterersequences. Jo lavere et resultat-ciffer, jo mere ens er de to strenge. Vi har dog kun valgt at vise hvordan den fungerer.

Bemærkninger:  
Funktionalitet:  
Filtrering: Når man vil filtrere på film, kan man selv vælge, hvor mange filtre man vil have på sin filtrering. Man kan filtrere på både IMDB-rating og title på samme tid eller en af gangen.

Hvis man vil filtrere film ud fra category, markerer man en category  
og ved tryk på den respektive knap, får man vist de film, som lægger under den valgte category.

Github link: https://github.com/pallemjensen/PrivateMovieCollection