1. User requirements

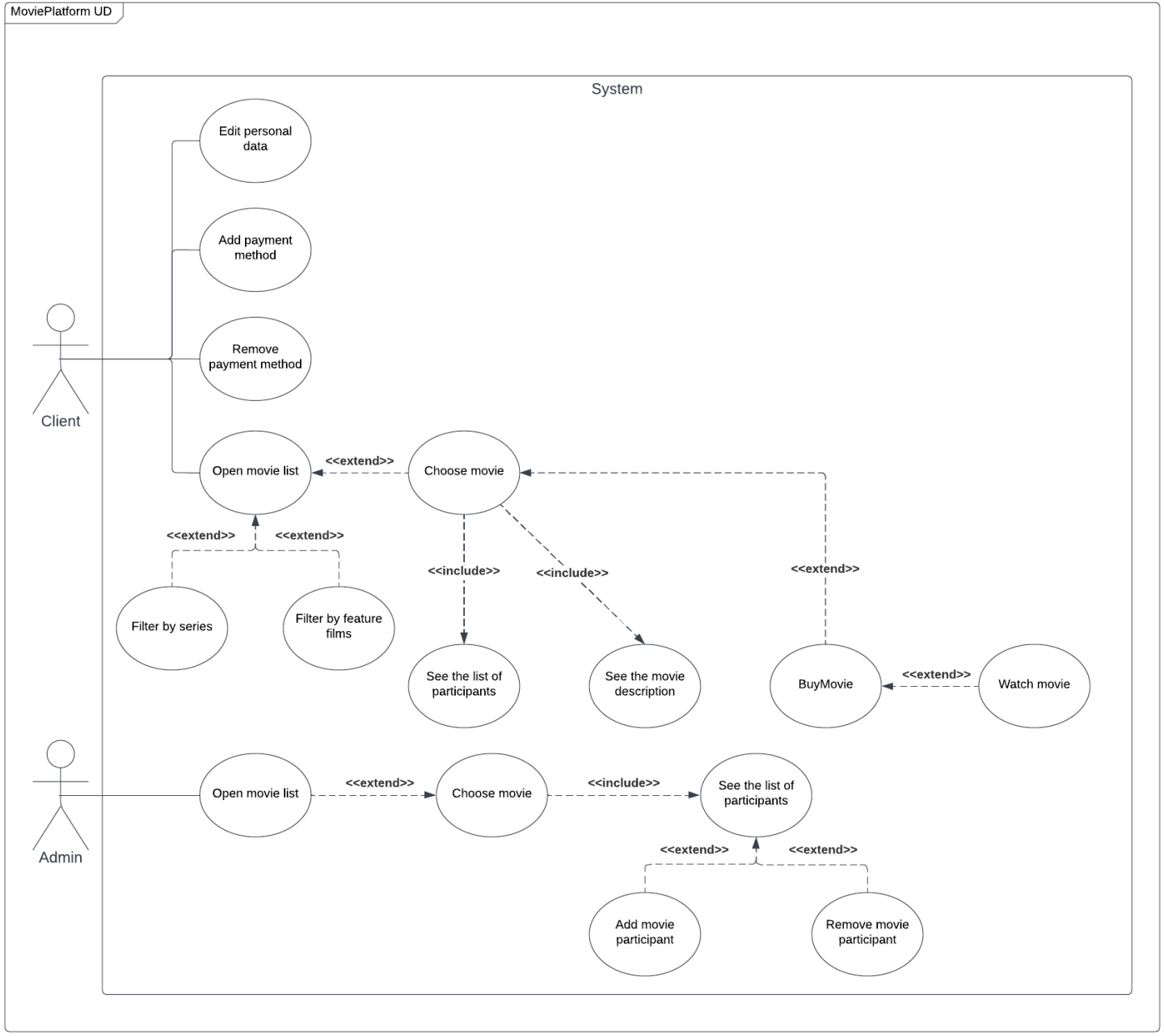
Our web application has to represent a modern movie hosting service. Firstly, we obviously want the app to provide client with movies. At this moment, we would like to see two types of movies: series and feature films. Series need to consist of series seasons which in their turn should contain series episodes. Every series season as well as series episode has to have its order number, own name and brief description or introduction. Furthermore, series episodes and seasons must be ordered by its order number to keep the proper view. Logically, it has to be specified that every series cannot exist without at least one season and the season without at least one episode. Now, the only requirement we want feature film to contain is its duration in minutes, but we consider movie to be a complete feature film if its duration is at least one hour. Both of the movie types should have information about their name, description, movie cost, release date and movie genres. In addition, there must be an option for a movie to receive a time discount which is calculated out of the time since release date. For technical reasons such as search engine, we want to have an ability to look for one or several movies by their name or part of it. Also, being movies separated on series and feature films, we want to have an ability to filter movies by type.

Going further, when choosing movie and entering its page we want to see the data about people that took part in creating this movie, their role among ("Actor", "Director", "Producer"), payment and awards received for it.

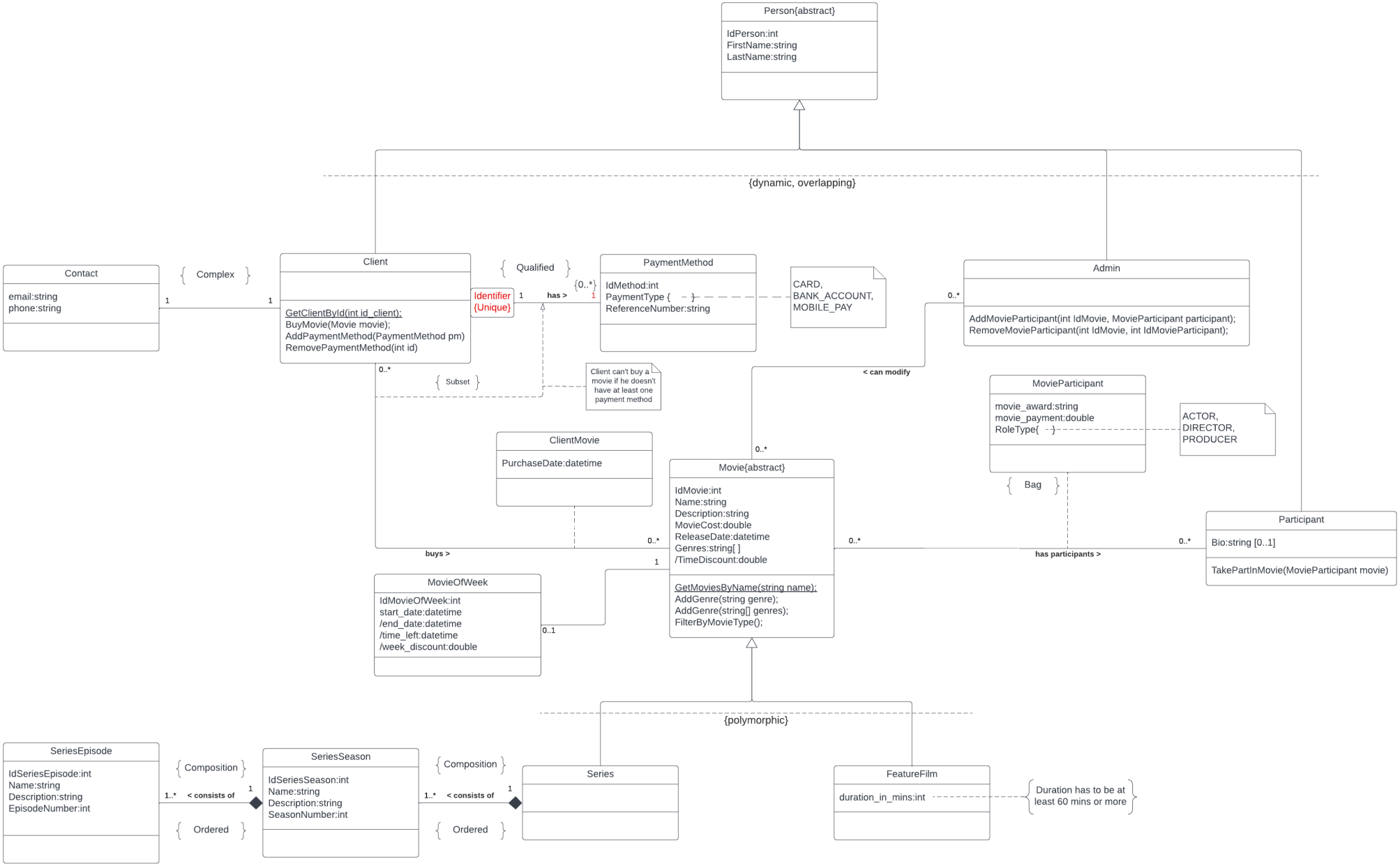
The last thing about movies, we would like web page to have a movie of the week. We want to store data about the start date. During further actions, we might want to calculate the end date and time left for this offer. The reason people would get interested in this time proposition is additional movie of the week discount which considers the movie cost and all previous discounts.

There are three types of persons: admin, client and, mentioned previously, participant. Moreover, one person can be of three types at the same time. For person we want to specify their first and last names. Their personal data has to be editable. Client additionally must have a contact attribute which consists of email address and phone number. Client can have only one contact. For client there should be an ability to add several unique payment methods. Payment method has a reference number string and specified payment type which could be one out of ("Card", "Bank account", "Mobile pay"). Before watching, client has to buy a chosen movie. During this process, additionally we want to store its purchase date. But before buying a movie, client has to add at least one payment method. For participant, the other only thing we want to know outside of its part in the movie is his bio, which could be omitted while creating participant's page but specified later. At this moment, for admin we only want him to have an ability to modify movies' details and, especially, to add and remove movie participants.

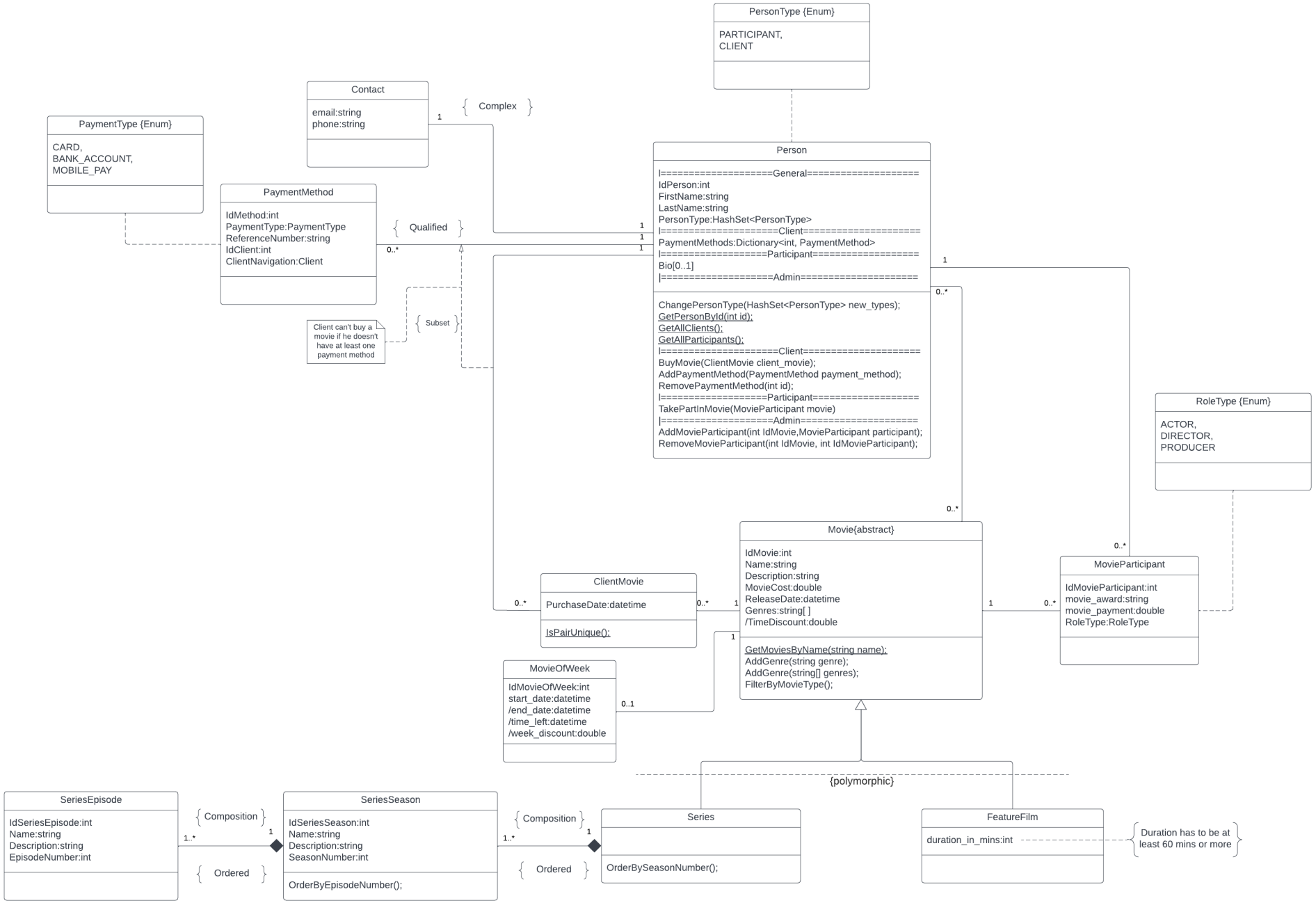
2. The use case diagram



3. The class diagram - analytical



4. The class diagram - design



5. The scenario of selected use case

1. **Use-case: Remove movie participant**
   - **1.1. Actors**
     Admin
   - **1.2. Purpose and context**
     Admin would like to remove movie participant from a specific movie.
   - **1.3. Dependencies**
     - **1.3.1. Included use-cases**
       None
     - **1.3.2. Extended use-cases**
       None
   - **1.4. Assumptions and pre-conditions**
     1.a. Admin pressed "Series" button to open movie list filtered by series.
     1.b. Admin pressed "Films" button to open movie list filtered by feature films.
   - **1.5. Initiating business event**
     Admin would like to remove movie participant from a specific movie. Admin is on movie list page.
   - **1.6. Basic flow of event**
     1. Admin chooses the movie and presses "Details" button.
     2. Repository retrieves info about movie and the list of participants from the database.
     3. Admin is shown the page with all information for the chosen movie.
     4. Admin presses on "Edit" button.
     5. Admin is shown the page with "Delete" buttons near every movie participant.
     6. Admin presses on one of the "Delete" buttons.
     7. Repository deletes the selected movie participant from the database.
     8. Repository retrieves info about movie and the list of participants from the database.
     9. Admin is shown the updated page with all information for the chosen movie.
   - **1.7. Alternative flows of event**
     - **1.7.1. Admin presses "home" button**
       4.a.1. Admin presses "home" button.

4.a.2. Admin is shown the main page.
4.a.3. The use case ends.

1.7.2. Admin does not want to remove movie participant anymore.

6.a.1. Admin presses "cancel" button.
6.a.2. Flow goes to point 3.

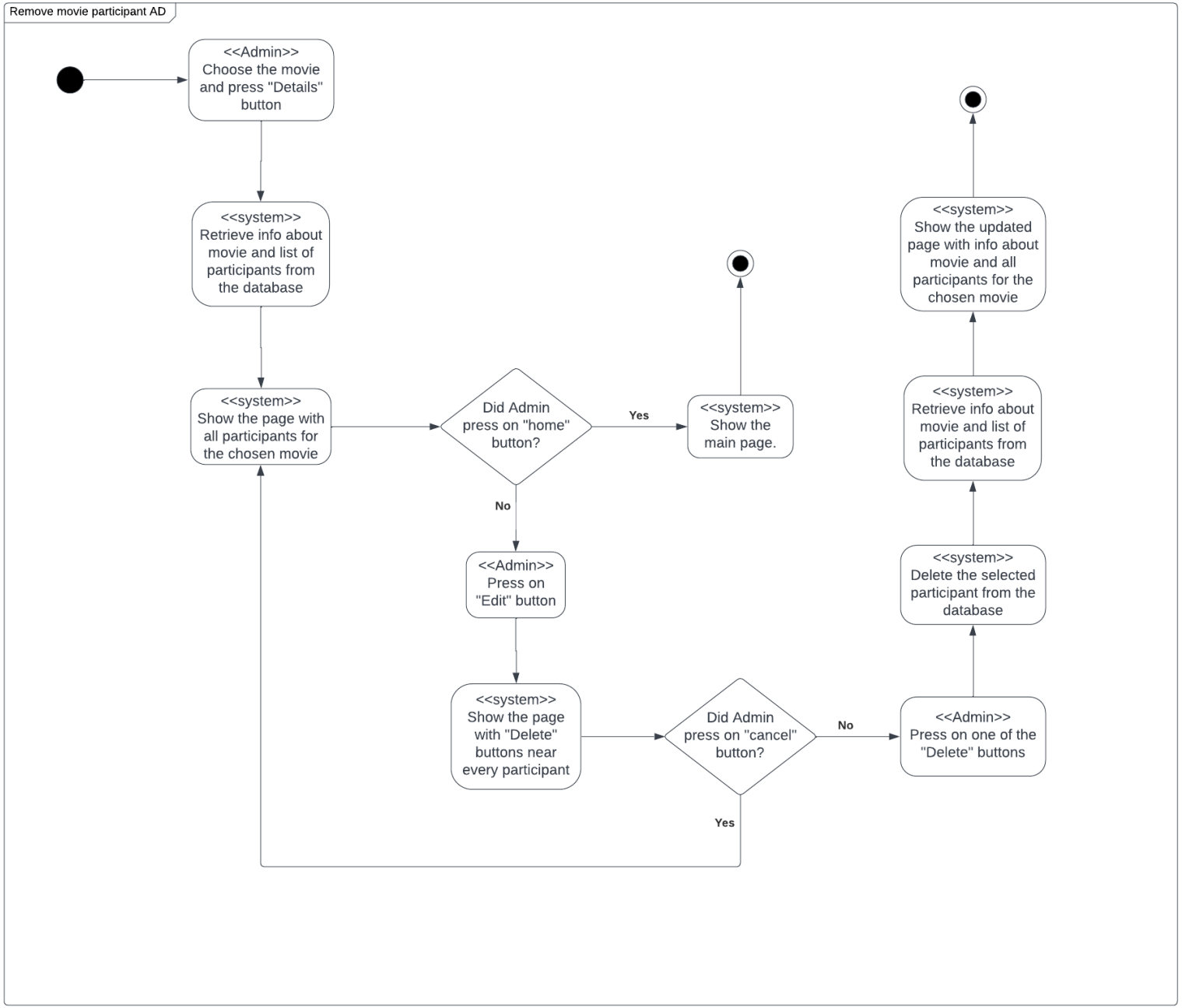
1.8. Extension points

Extend from "See the list of participants" use case

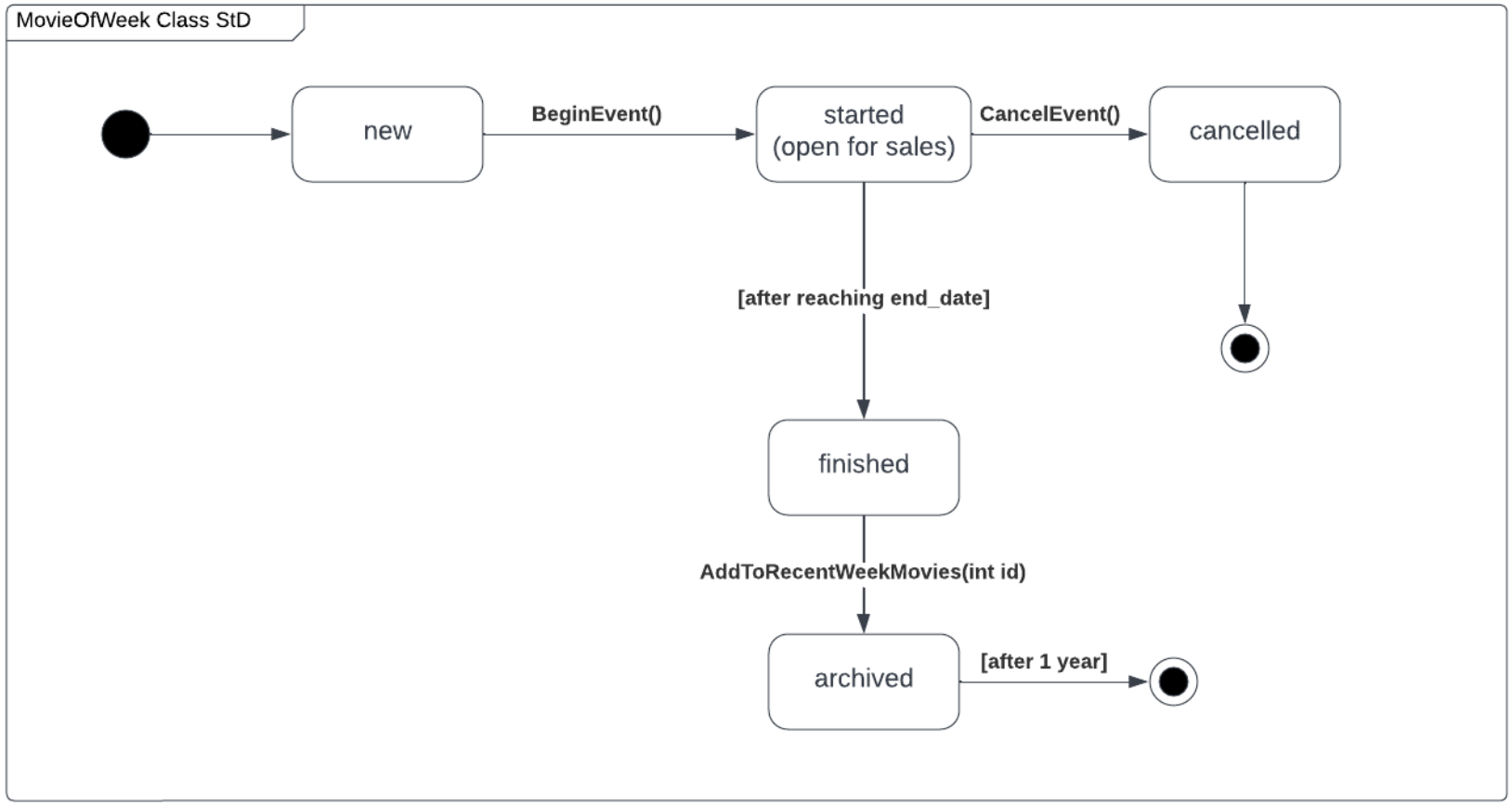
1.9. Post-conditions

1. Admin has successfully removed the movie participant.
2. Removed movie participant was deleted from the database.

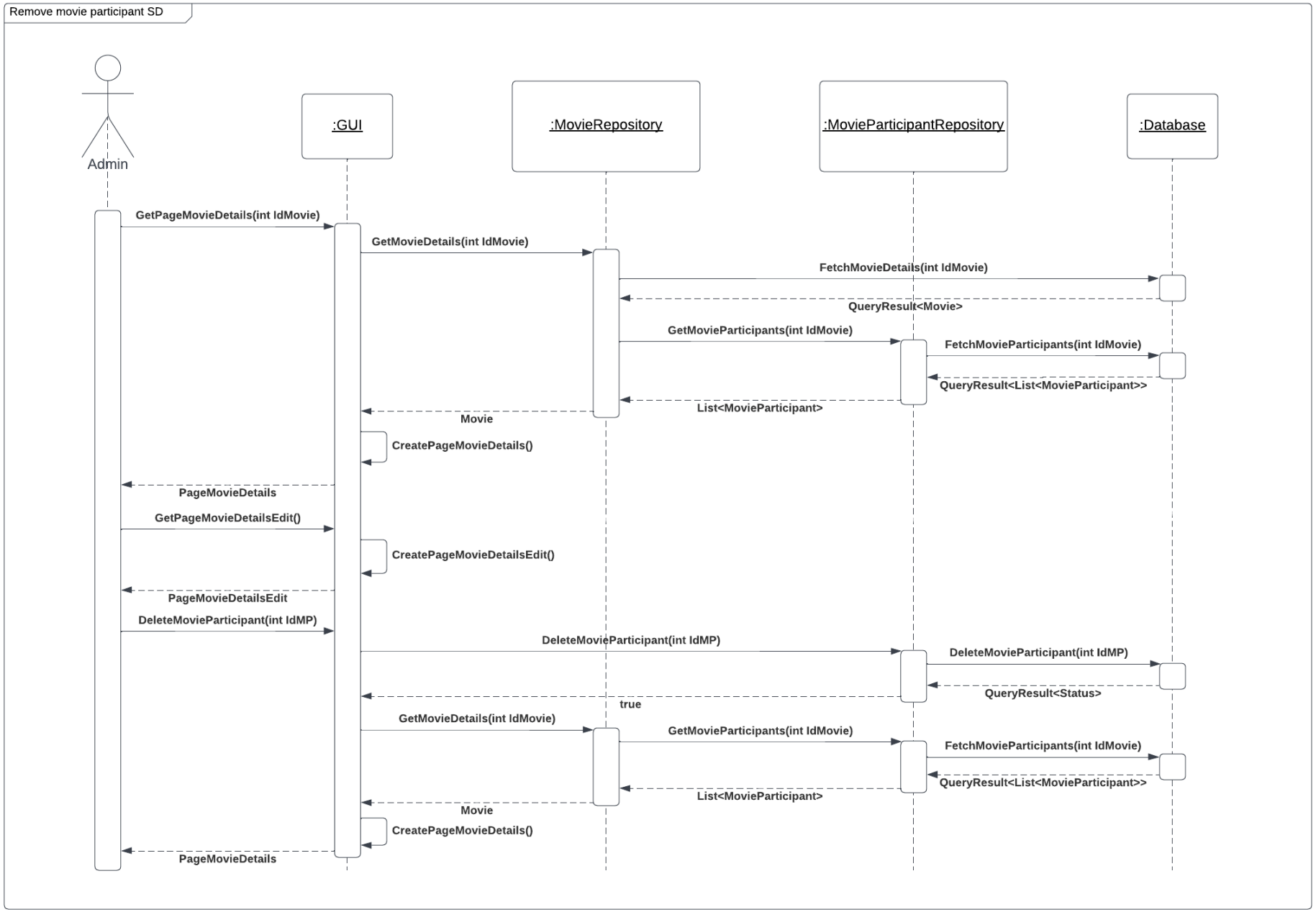
6. The activity diagram for picked use case



7. The state diagram for selected class



8. The interaction (sequence) diagram for selected use case



9. The GUI design

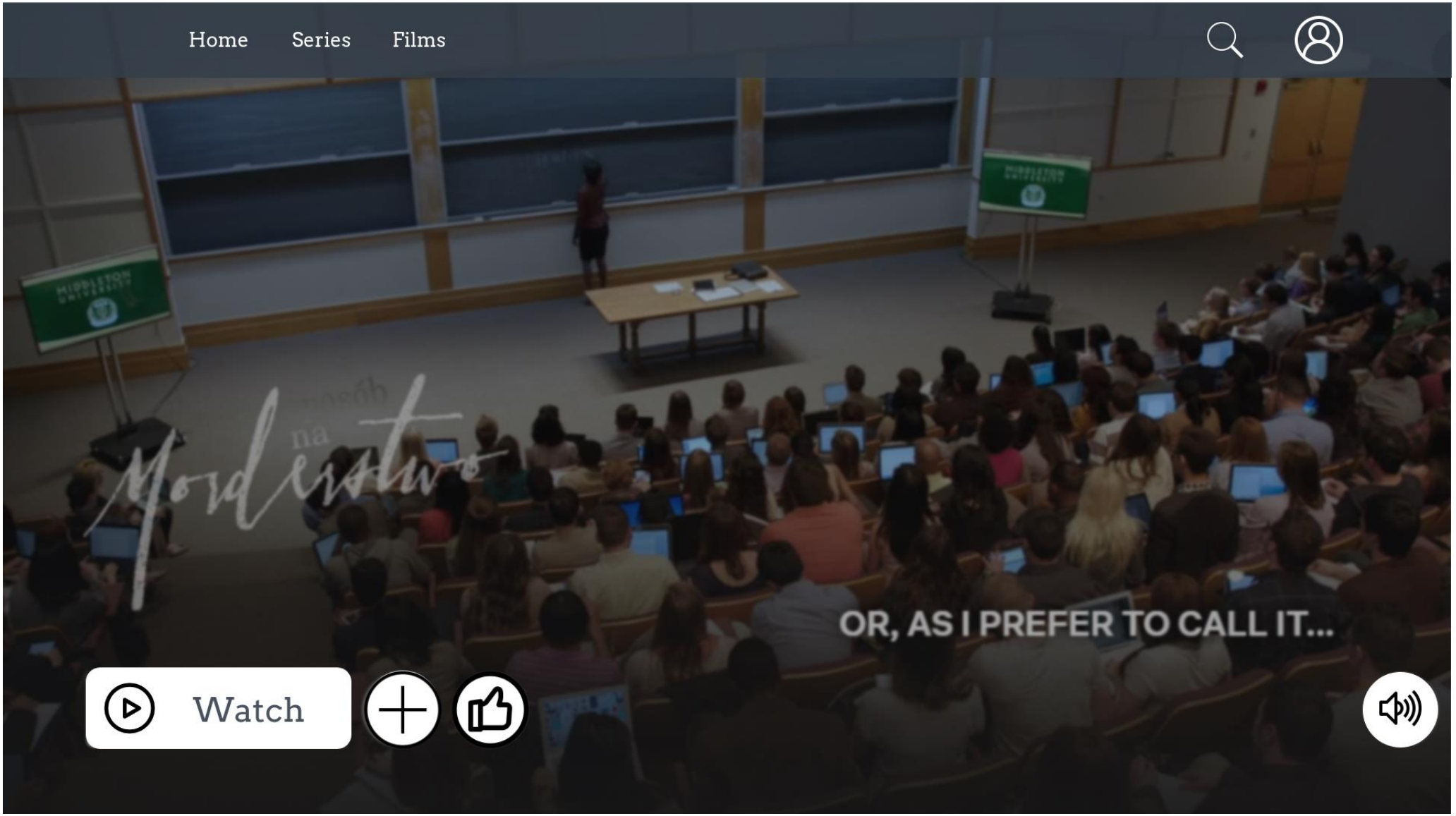


Figure 1. Main Page.

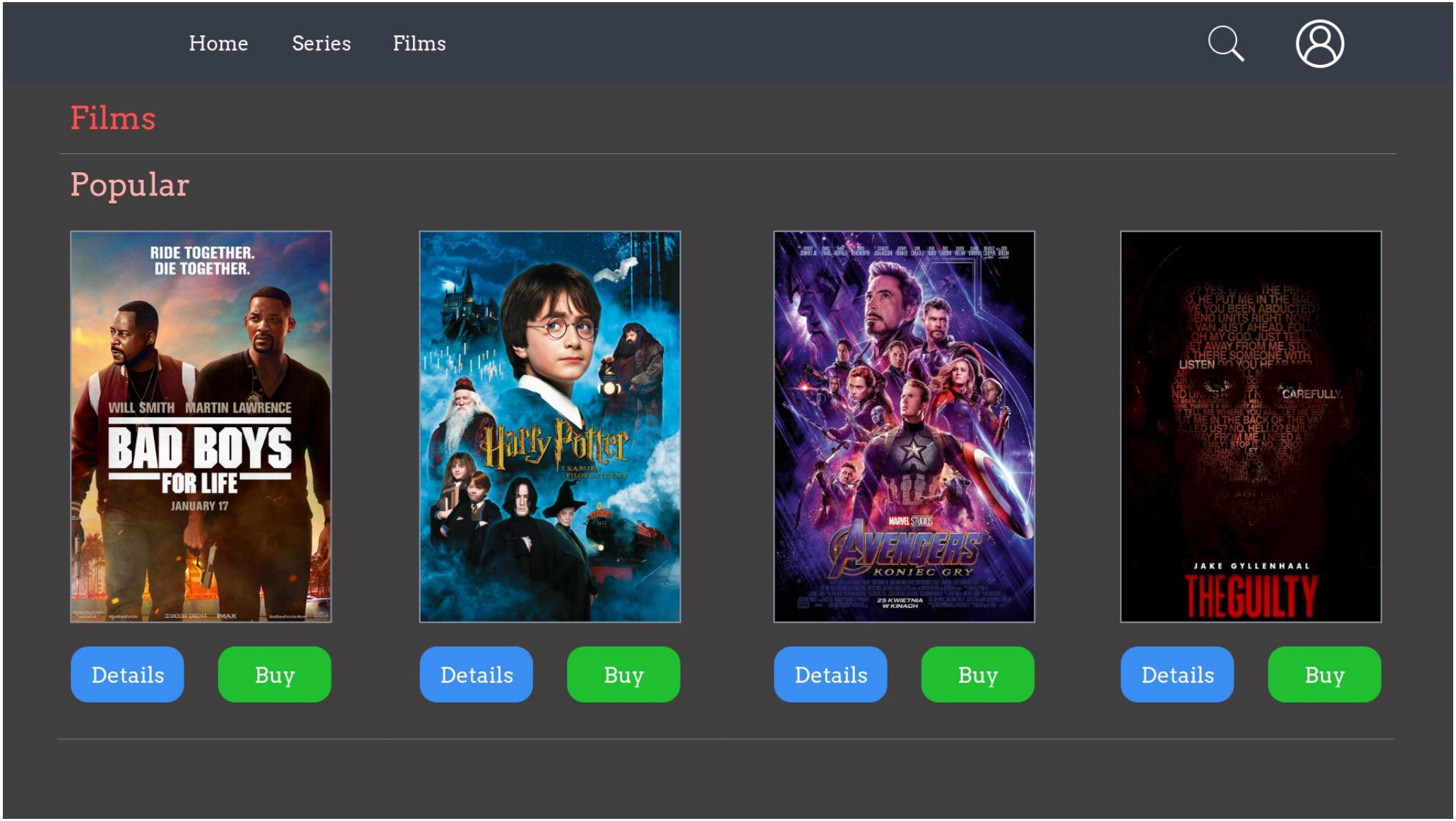


Figure 2. MovieList Page.

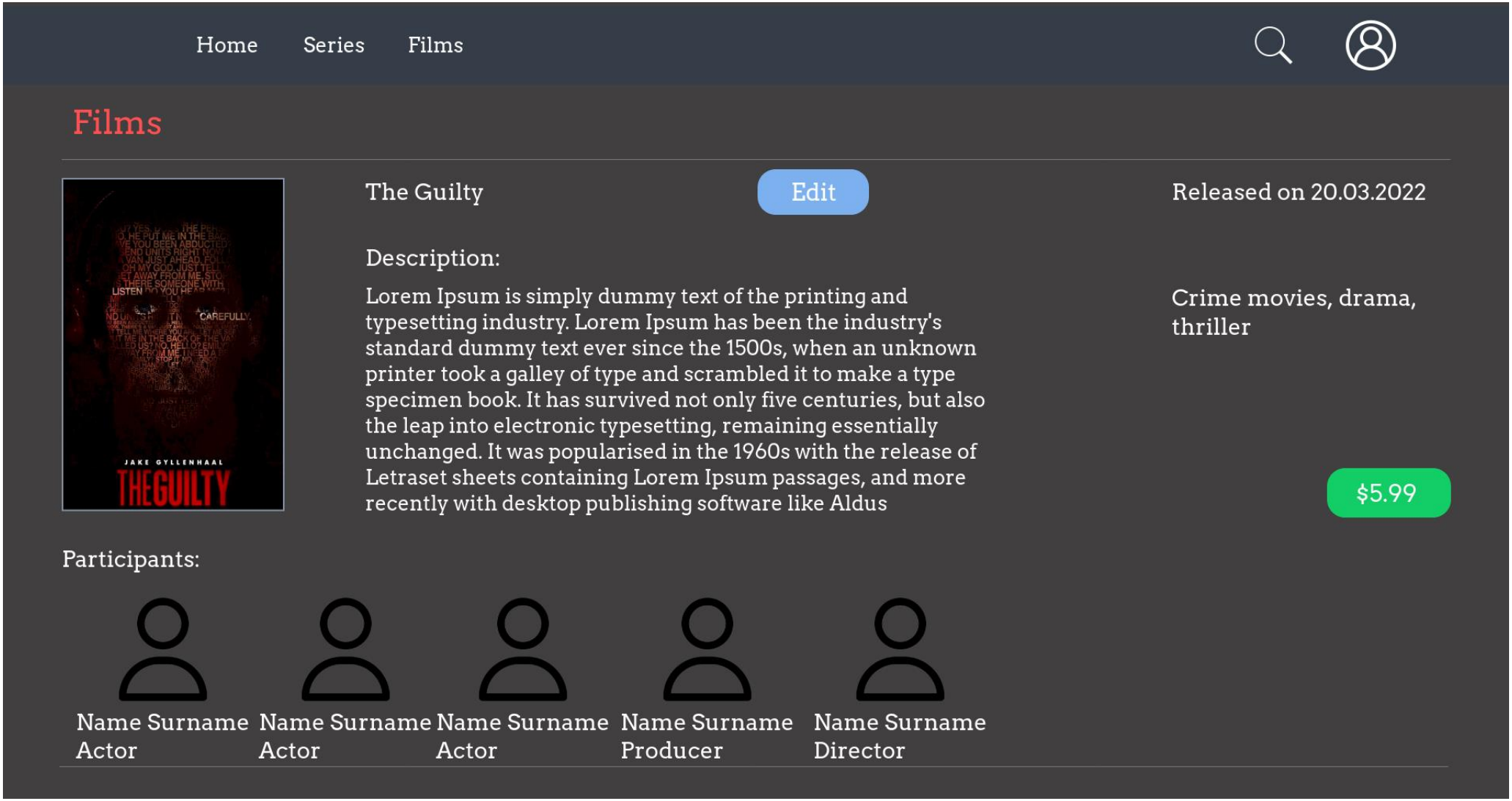


Figure 3. MovieDetails Page

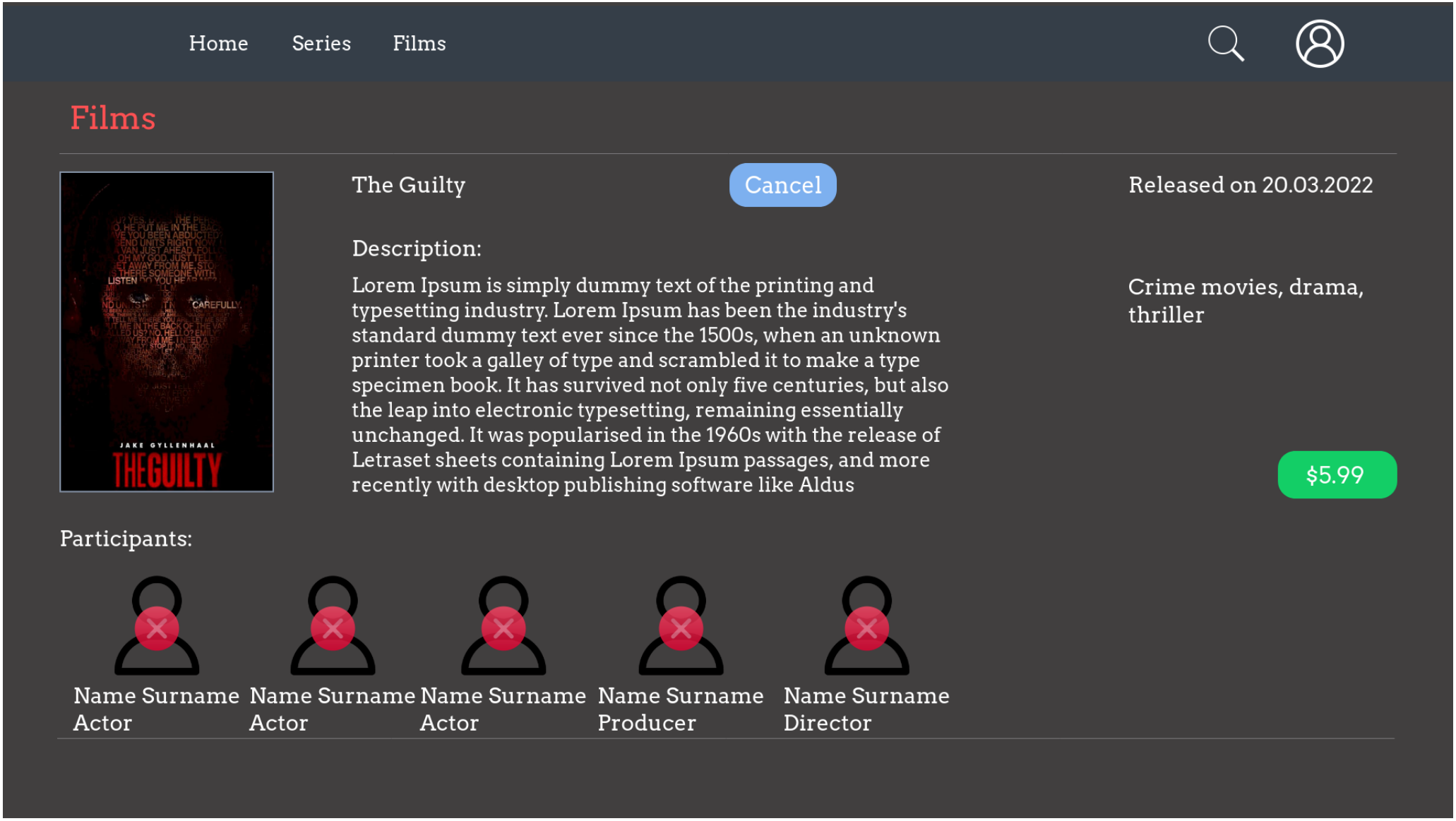


Figure 4. MovieDetailsEdit Page

10. The discussion of design decisions and the effect of dynamic analysis

For design of **overlapping inheritance** in Person class, it was decided to use only one class Person with an attribute *PersonType:HashSet<PersonType>,* which would contain a set of enum *PersonType* objects. This, combined with having separated attributes and methods only for Person itself, Admin, Client and Participant, allows me to implement overlapping inheritance between without any losses and issues.

Movie to Participant relation was implemented using separate association table with 3 attributes, which also represents ***Bag*** constraint. This gives an ability to assign the same Participant to same Movie more than once but for different *RoleType* and therefore different award and payment.

Subset constrain is represented in association Client to Movie which could not be established until there is at least one association between Client and PaymentMethod, which makes such logic: "client can't buy a movie until they add at least one payment method on the account settings".

Composition and ***Ordered*** were used between 2 relations (Series to SeriesSeason) and (SeriesSeason to SeriesEpisode). Any series can't exist without at least one season as well as season can't exist without at least one episode. Seasons and episodes has to be ordered to keep the proper presentation view.

FeatureFilm contains a **custom constraint** which doesn't allow to create a new FeatureFilm with duration or modify an old's duration to be less than 60 mins.

Dynamic analysis showed that there are 2 alternative flows for the chosen use-case **"Remove movie participant"**. First leads to the termination of the use-case as admin simply chose to exit to the main page by pressing "home" button. Second happens after admin enters "MovieDetailsEdit" page and decides not to remove movie participant but to press "Cancel" button, which redirects him back to "MovieDetails" page.

GUI was prototyped on marvelapp.com. Presented GUI contains all necessary pages to show how the chosen use-case would work. Design is not overflooded with different extra functionality and has user-friendly appealing view. At this stage, it is meant for user to be able to go to “MovieList” page, choose a movie and press “Details” button to enter “MovieDetails” page, then press “Edit” button to enter “MovieDetailsEdit” page and press “Delete” button on one of the movie participants to remove it from the page and the database. Some of additional not functional parts of the interface are being shown just to keep the proper view of the upcoming application.

Project right from the start and till the very end was done by Illia Zviezdin, s21120.