Bilkent University

CS353 DATABASE SYSTEMS

FALL 2020

TERM PROJECT DESIGN REPORT

Teaching Assistant: Arif Usta

Group No: 10

Group Members:

İbrahim Furkan Aygar 21400186

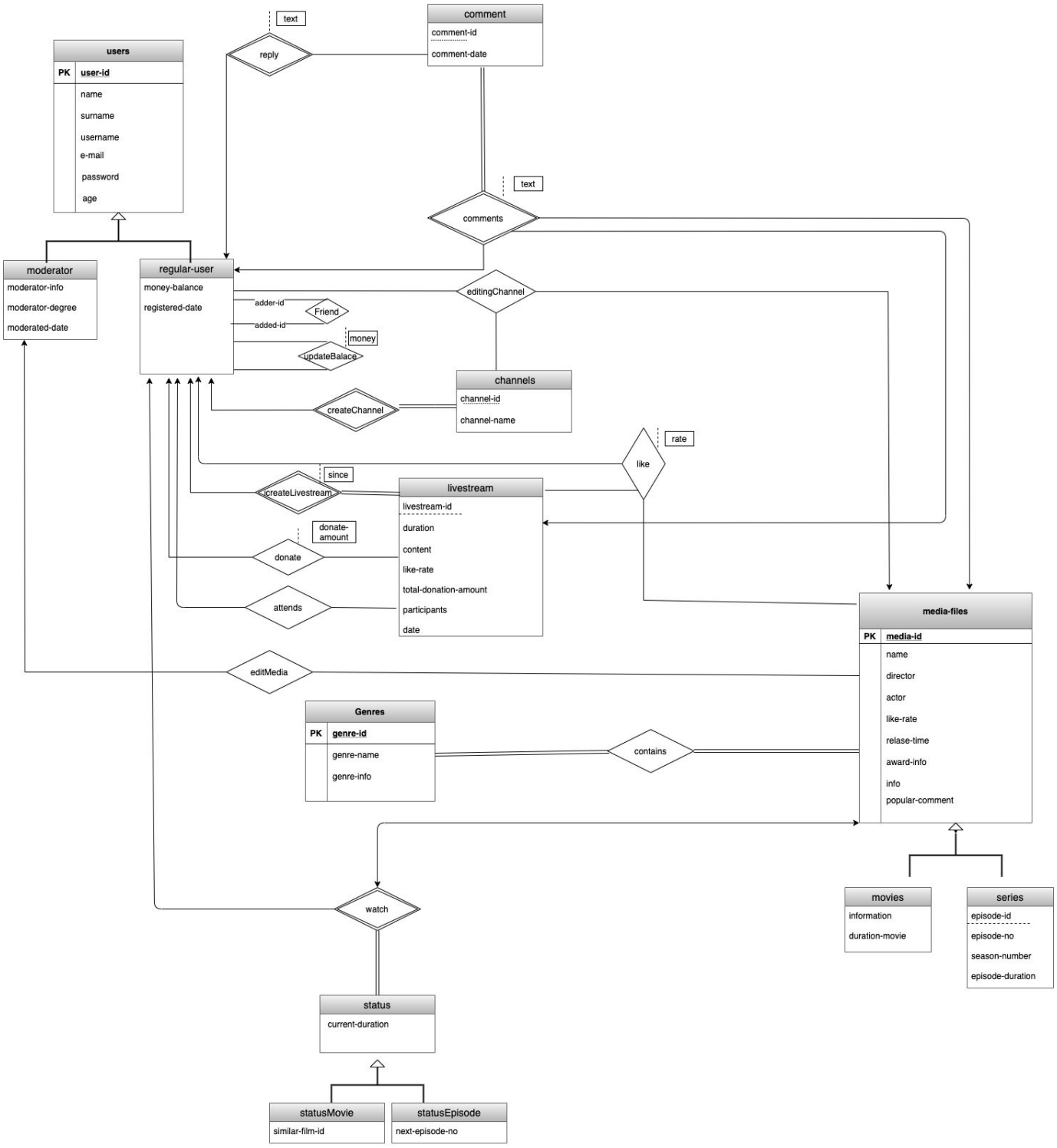
Süleyman Rahimov 21701671

Mehmet Erkin Şahsuvaroğlu 21401625

Enes Yıldırım 21602725

Table of Contents:

1. Revised E/R Diagram



2 Relations & Attributes

2.1 users

Relational Model:
users(<u>user-id</u>, name, surname, username, e-mail, password, age)

Functional Dependencies:
user-id -> name, surname, username, e-mail, password, age

Keys:

Candidate Keys: {(user-id)}
Primary Keys: user-id
Foreign Keys: NONE

Normal Forms: BCNF

Table Definition:

```
CREATE TABLE users(user-id int(7) PRIMARY KEY,
                   name  varchar(25) NOT NULL,
                   surname varchar(35) NOT NULL,
                   username varchar(20) NOT NULL,
                   e-mail  varchar(20) NOT NULL,
                   password varchar(8) NOT NULL,
                   age int(2))
```

2.2 moderator

Relational Model:
moderator(<u>user-id</u>, moderator-info,moderator-degree,moderated-date)

Functional Dependencies:
user-id -> moderator-info,moderator-degree,moderated-date

Keys:
Candidate Keys: {(user-id)}
Primary Keys: user-id
Foreign Keys: user-id
Normal Forms: BCNF
Table Definition:

```
 CREATE TABLE moderator(user-id int(7) NOT NULL,
                        moderator-info varchar(200),
```

moderator-degree int(2),
moderated-date date,
PRIMARY KEY(user-id),
FOREIGN KEY(user-id) references users(user-id))

2.3 regular-user

Relational Model:
regular-user(<u>user-id</u>, money-balance,registered-date)

Functional Dependencies:
user-id -> money-balance, registered-date
Keys:
Candidate Keys: {(user-id)}
Primary Keys: none
Foreign Keys: user-id
Normal Forms: BCNF
Table Definition:

```
CREATE TABLE regular-user(user-id int(7) NOT NULL,
                          money-balance float(10),
                          PRIMARY KEY(user-id),
                          FOREIGN KEY(user-id) references users(user-id))
```

2.4 media-files

Relational Model:
media-files(<u>media-id</u>, name, director, actor, like-rate, release-date, award-info, info, popular-comment)
Functional Dependencies:
media-id -> name, director, actor, like-rate, release-date, award-info, info,popular-comment
Keys:
Candidate Keys: {(media-id}
Primary Keys: media-id
Foreign Keys: none
Normal Forms: BCNF
Table Definition:

```
CREATE TABLE media-files(media-id int(7) NOT NULL,
                         actor varchar(255),
                         like-rate int(2),
                         release date,
                         award-info varchar(25),
                         info varchar(200),
                         popular-comment varchar(255),
                         PRIMARY KEY(media-id))
```

2.5 movies

Relational Model:
movies(media-id,information,duration-movie)
Functional Dependencies:
media-id -> information,duration-movie
Keys:
Candidate Keys: {(media-id)}
Primary Keys:media-id
Foreign Keys:media-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE movies(
        media-id   int(7) NOT NULL,
        information varchar(50),
        duration int(3),
        PRIMARY KEY (media-id),
        FOREIGN KEY (media-id) references media-files(media-id))
```

2.6 series

Relational Model:
series (media-id,episode-id,episode-no,season-number,episode-duration)
Functional Dependencies:
Media-id, episode-id -> episode-no, season-number,episode-duration
Keys:
Candidate Keys: {(media-id,episode-id)}
Primary Keys:media-id,episode-id
Foreign Keys:media-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE series(media-id int(7) NOT NULL,
                    episode-id int(7) NOT NULL,
                    episode-no int(3),
                    season-number int(1),
                    episode-duration time,
                    PRIMARY KEY(media-id, episode-id),
                    FOREIGN KEY (media-id) references media-files(media-id) )
```

2.7 channels

Relational Model:
channels(<u>user-id,channel-id</u>,channel-name)
Functional Dependencies: user-id, channel-id -> channel-name
Keys:
Candidate Keys: {(user-id,channel-id)}
Primary Keys: user-id ,channel-id
Foreign Keys: user-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE channels(user-id int(7) NOT NULL,
                      channel-id int(7) NOT NULL,
                      channel-name varchar(100),
                      PRIMARY KEY(channel-id,user-id),
                      FOREIGN KEY(user-id) references regular-user(user-id))
```

2.8 comment

Relational Model:
comment(<u>user-id,comment-id</u>,comment-date)
Functional Dependencies: user-id,comment-id -> comment-date
Keys:
Candidate Keys:{(user-id,comment-id)}
Primary Keys:user-id,comment-id
Foreign Keys:user-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE comment(user-id int(7) NOT NULL,
                     comment-id (7) NOT NULL,
                     comment-date date,
                     PRIMARY KEY(comment-id,user-id),
                     FOREIGN KEY(user-id) references regular-user(user-id))
```

2.9 comments

Relational Model:
comments(<u>user-id,comment-id,livestream-id,media-id</u>,text)
Functional Dependencies: user-id, comment-id, media-id, livestream-id-> text
Keys:
Candidate Keys: {(user-id, comment-id, media-id, livestream-id)}
Primary Keys: user-id, comment-id, media-id, livestream-id
Foreign Keys: user-id, comment-id, media-id, livestream-id

Normal Forms: BCNF

Table Definition:

```
CREATE TABLE comments(user-id int(7) NOT NULL,
                      comment-id int(7) NOT NULL,
                      livestream-id int(7),
                      media-id int(7),
                      text varchar(255),
PRIMARY KEY(user-id, comment-id, media-id, livestream-id),
FOREIGN KEY (user-id) references users(user-id),
FOREIGN KEY (comment-id) references comment(comment-id),
FOREIGN KEY(media-id) references media-files(media-id),
FOREIGN KEY(livestream-id) references livestream(livestream-id))
```

2.10 watch

Relational Model:
watch(<u>user-id,media-id</u>**,**current-duration**)**
Functional Dependencies: user-id, media-id -> current-duration
Keys:
Candidate Keys: {(user-id,media-id)}
Primary Keys:user-id, media-id
Foreign Keys:user-id, media-id, current-duration
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE watch(user-id int(7) NOT NULL,
    media-id int(7) NOT NULL,
    current-duration time,
    PRIMARY KEY(user-id,media-id),
    FOREIGN KEY(user-id) references regular-user(user-id),
    FOREIGN KEY(media-id) references media-files(media-id)
    FOREIGN KEY(current-duration) references status(current-duration)
```

2.11 status

Relational Model:
status(<u>user-id</u>,<u>media-id</u>,current-duration)
Functional Dependencies: user-id,media-id ->current-duration
Keys:
Candidate Keys:({user-id,media-id})
Primary Keys:user-id,media-id
Foreign Keys:user-id,media-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE status(user-id int(7) NOT NULL,
                    media-id int(7) NOT NULL,
                     current-duration int(3) NOT NULL,
                    PRIMARY KEY(user-id,media-id),
                    FOREIGN KEY(user-id) references regular-user(user-id),
                    FOREIGN KEY(media-id) references media-files(media-id))
```

2.12 statusFilm

Relational Model:
statusFilm(<u>user-id</u>,<u>media-id</u>,current-duration,similar-film-id)
Functional Dependencies:user-id,media-id ->current-duration,similar-film-id
Keys:
Candidate Keys:{(media-id, user-id)}
Primary Keys:user-id,media-id
Foreign Keys:user-id,media-id, current-duration
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE statusFilm(media-id int(7)  NOT NULL,
        user-id int(7) NOT NULL,
        current-duration int(3) NOT NULL,
        similar-film-id int(7) NOT NULL,
        PRIMARY KEY(user-id,media-id),
        FOREIGN KEY(media-id) references media-files(media-id),
        FOREIGN KEY(user-id) references regular-user(user-id),
        FOREIGN KEY(current-duration) references status(current-duration))
```

2.13 statusSeries

Relational Model:
statusSeries(user-id,media-id,current-duration,next-episode-no)
Functional Dependencies: user-id,media-id ->current-duration,next-episode-no
Keys:
Candidate Keys:{(media-id,user-id)}
Primary Keys: media-id,user-id
Foreign Keys: media-id,user-id, current-duration
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE statusSeries(media-id int(7) NOT NULL,
        user-id int(7) NOT NULL,
        current-duration int(3) NOT NULL,
        next-episode-no int(3) NOT NULL,
        PRIMARY KEY(user-id,media-id),
        FOREIGN KEY(media-id) references media-files(media-id),
        FOREIGN KEY(media-id) references regular-user(user-id),
        FOREIGN KEY(current-duration) references status(current-duration))
```

2.14 Genres

Relational Model:
genres(genre-id,genre-name,genre-info)
Functional Dependencies:genre-id -> genre-name, genre-info
Keys:
Candidate Keys:{(genre-id)}
Primary Keys:genre-id
Foreign Keys: none
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE Genres(genre-id int(7) NOT NULL,
        genre-name varchar(20),
        genre-info varchar(255),
        PRIMARY KEY(genre-id))
```

2.15 editingChannel

Relational Model:
editingChannel(user-id,channel-id, media-id)
Functional Dependencies: none
Keys:

Candidate Keys:{(user-id,channel-id, media-id)}
Primary Keys: user-id,channel-id, media-id
Foreign Keys: user-id,channel-id, media-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE editingChannel(user-id int(7) NOT NULL,
        channel-id int(7) NOT NULL,
        PRIMARY KEY(user-id,channel-id, media-id)
        FOREIGN KEY(user-id) references regular-user(user-id),
        FOREIGN KEY(channel-id) references channels(channel-id),
        FOREIGN KEY(media-id) references media-files(media-id))
```

2.16 livestream

Relational Model:
livestream(<u>user-id,livestream-id</u>,duration,context,like-rate,total-donation-amount,participants,date)
Functional Dependencies:
user-id,livestream-id -> duration,context,like-rate,total-donation-amount,participants,date

Keys:
Candidate Keys:{(user-id,livestream-id)}
Primary Keys:user-id,livestream-id
Foreign Keys:user-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE livestream(user-id int(7) NOT NULL,
        livestream-id int(7) NOT NULL,
        duration time,
        context varchar(25) NOT NULL,
        like-rate int(2),
        total-donation-amount float,
        participants varchar(255),
        date date,
        PRIMARY KEY(user-id, livestream-id),
        FOREIGN KEY(user-id) references regular-user(user-id))
```

2.17 contains

Relational Model:
contains(media-id,genre-id)
Functional Dependencies: none
Keys:
Candidate Keys:{(media-id,genre-id)}
Primary Keys:media-id, genre-id
Foreign Keys:media-id, genre-id
Normal Forms:BCNF

Table Definition:

```
CREATE TABLE contains(media-id int(7) NOT NULL,
        genre-id int(7) NOT NULL,
        PRIMARY KEY(media-id, genre-id)
        FOREIGN KEY(media-id) references media-files(media-id),
        FOREIGN KEY(genre-id) references Genres(genre-id))
```

2.18 editMedia

Relational Model:
editMedia(user-id,media-id)
Functional Dependencies: none
Keys:
Candidate Keys:{(user-id,media-id)}
Primary Keys:user-id,media-id
Foreign Keys:user-id,media-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE editMedia(user-id int(7) NOT NULL,
        media-id int(7) NOT NULL,
        PRIMARY KEY(media-id, media-id)
        FOREIGN KEY(user-id) references moderator(user-id),
        FOREIGN KEY(media-id) references media-files(media-id))
```

2.19 like

Relational Model:
like(user-id,media-id,livestream-id,rate)
Functional Dependencies:user-id,media-id,livestream-id -> rate
Keys:
Candidate Keys: {(user-id,media-id,livestream-id)}
Primary Keys:user-id,media-id,livestream-id
Foreign Keys:user-id,media-id,livestream-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE like(user-id int(7) NOT NULL,
media-id int(7),
livestream-id int(7),
rate int(2),
PRIMARY KEY(user-id,media-id,livestream-id),
FOREIGN KEY(user-id) references regular-user(user-id),
FOREIGN KEY(media-id) references media-files(media-id),
FOREIGN KEY(livestream-id) references livestream(livestream-id))
```

2.20 attends

Relational Model:
attends(user-id,livestream-id)
Functional Dependencies: none
Keys:
Candidate Keys: {(user-id,livestream-id)}
Primary Keys:
Foreign Keys: user-id, livestream-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE attends(user-id int(7) NOT NULL,
livestream-id int(7) NOT NULL,
PRIMARY KEY(user-id,livestream-id),
FOREIGN KEY(user-id) references regular-user(user-id),
FOREIGN KEY(livestream-id) references livestream(livestream-id))
```

2.21 donate

Relational Model:
donate(<u>user-id,livestream-id</u>,donate-amount)
Functional Dependencies:user-id,livestream-id -> donate-amount
Keys:
Candidate Keys:{(user-id,livestream-id)}
Primary Keys:user-id,livestream-id
Foreign Keys: user-id,livestream-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE donate(user-id int(7) NOT NULL,
        livestream-id int(7) NOT NULL,
        donate-amount float,
        PRIMARY KEY(user-id,livestream-id)
        FOREIGN KEY(user-id) references regular-user(user-id),
        FOREIGN KEY(livestream-id) references livestream(livestream-id))
```

2.22 createLivestream

Relational Model:
createLivestream(<u>user-id,livestream-id</u>,since)
Functional Dependencies:user-id,livestream-id -> since
Keys:
Candidate Keys:{(user-id,livestream-id)}
Primary Keys:user-id,livestream-id
Foreign Keys:user-id,livestream-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE createLivestream(user-id int(7) NOT NULL,
    livestream-id int(7) NOT NULL,
    since date,
    PRIMARY KEY(user-id,livestream-id)
    FOREIGN KEY(user-id) references regular-user(user-id),
    FOREIGN KEY(livestream-id) references livestream(livestream-id))
```

2.23 reply

Relational Model:
reply(<u>user-id,comment-id,</u>text)
Functional Dependencies:user-id,comment-id -> text
Keys:
Candidate Keys:{(user-id,comment-id)}
Primary Keys:user-id,comment-id
Foreign Keys:user-id,comment-id
Normal Forms:BCNF

Table Definition:

```
CREATE TABLE reply(user-id int(7) NOT NULL,
                   comment-id int(7) NOT NULL,
                   text varchar(255),
                   PRIMARY KEY(user-id,comment-id)
                   FOREIGN KEY(user-id) references regular-user(user-id),
                   FOREIGN KEY(comment-id) references comment(comment-id))
```

2.24 Friend

Relational Model:
friend(<u>adder-id,added-id</u>)
Functional Dependencies: none
Keys:
Candidate Keys:{(adder-id,adder-id)}
Primary Keys:adder-id,added-id
Foreign Keys:adder-id,added-id
Normal Forms:
Table Definition:

```
CREATE TABLE Friend(adder-id int(7) NOT NULL, adder-id int(7),
                   PRIMARY KEY(adder-id,added-id),
                   FOREIGN KEY(adder-id) references regular-user(user-id)
                   FOREIGN KEY(added-id) references regular-user(user-id))
```

2.25 createChannel

Relational Model:
createChannel(<u>user-id,channel-id)</u>
Functional Dependencies: none
Keys:
Candidate Keys:{(user-id,channel-id)}

Primary Keys:user-id,channel-id
Foreign Keys:user-id,channel-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE createChannel(user-id int(7) NOT NULL,
channel-id int(7) NOT NULL,
PRIMARY KEY(user-id,channel-id),
FOREIGN KEY(user-id) references regular-user(user-id),
FOREIGN KEY(channel-id) references channels(channel-id)
```

2.26 updateBalance

Relational Model:
updateBalance(<u>user-id</u>,money)
Functional Dependencies:user-id ->money
Keys:
Candidate Keys: {(user-id)}
Primary Keys: user-id
Foreign Keys: user-id
Normal Forms:BCNF
Table Definition:

```
CREATE TABLE updateBalance(user-id int(7) NOT NULL,
money float,
PRIMARY KEY(user-id),
FOREIGN KEY(user-id) references regular-user(user-id))
```

3. Diagrams

3.1 Scenarios

Scenario 1: Create Account

Use Case: Create Account

Actors: Admin, User

Entry Condition(s): Actor opens the website in the browser.

Exit Condition(s):

- Account is successfully created

- Account creation failed

Flow of Events:

1. Actor opens the website in the browser.

2. Actor clicks the “Create Account” button.

3. “Create Account” page is shown to the actor.

4. Actor provides the necessary information (name, email, password etc.).

5. Actor clicks the “Create Account” button.

6. Account is successfully created.

Alternative Flow of Events:

A. User already exists in the system.

1. System tells the actor that the account already exists.

2.Actor can login or create an account with a different mail.

Scenario 2: Login

Use Case: Login

Actors: Admin, User

Entry Condition(s): Actor opens the website in the browser.

Exit Condition(s):

- Login is successful.

- Login is unsuccessful.

Flow of Events:

1. Actor opens the website in the browser.

2. Actor clicks the “Login” button.

3. “Login” page is shown to the actor.

4. Actor provides his/her email and password.

5. Actor clicks the “login” button.

6. Login is successful.

Alternative Flow of Events:

A. Login information is incorrect

    1. System tells the actor that the email or password provided is incorrect.

    2. Actor can retry to login.

Scenario 3: Add Media Files

Use Case: Add Media Files

Actors: Admin

Entry Condition(s): Actor is logged in as an Admin account.

Exit Condition(s): Operation is successful

Flow of Events:

1. Admin clicks the “Add Media File” button in the main menu.

2. Admin selects if he/she will add a movie or a tv show.

3. Admin provides a description for the media file.

4. Admin starts uploading the media file.

5. Admin clicks the “Complete Operation” button after the media file is uploaded.

Scenario 4: Provide feedback on media files

Use Case: Provide feedback on media files

Actors: User

Entry Condition(s): User should be in the page of a media file he/she is going to provide feedback on.

Exit Condition(s): Operation is successful.

Flow of Events:

1. User opens the media file he/she is going to provide feedback on.

2. Actor clicks the like button or writes a comment and clicks the “Add Comment” button

3. Operation is successful.

Scenario 5: Add Friends

Use Case: Add Friends

Actors: User

Entry Condition(s): Actor is logged in as a user account.

Exit Condition(s):

- Operation is successful

- User not found

Flow of Events:

1. User clicks the “Add Friend” Button in the main menu.

2. User searches the name of the person he/she wants to add as friend.

3. User clicks the “Send Friendship Request” button next to the person.

Scenario 6: Create Channel

Use Case: Create Channel

Actors: User

Entry Condition(s): Actor is logged in as a user account.

Exit Condition(s): Operation is successful

Flow of Events:

1. User clicks the “Create Channel” button in the main menu.

2. User selects the media files he/she wants to add to the channel.

3. User clicks the “Create” button to complete the operation.

Scenario 7: Specify Genre Preferences

Use Case: Specify Genre Preferences

Actors: User

Entry Condition(s): Actor is logged in as a user account.

Exit Condition(s): Operation is successful

Flow of Events:

1. User clicks the “Specify Preferences” button in the main menu.

2. User selects the genres of media files he/she wants to view.

3. User clicks the “Complete” button to complete the operation.

Scenario 8: Create Livestream

Use Case: Create Livestream

Actors: User

Entry Condition(s): Actor is logged in as a user account.

Exit Condition(s): Operation is successful

Flow of Events:

1. User clicks the “Create Livestream” button in the main menu.

2. User specifies the name of the livestream in the pop-up menu.

3. User clicks the “Create” button to create the livestream.

Scenario 9: Join Livestream

Use Case: Join Livestream

Actors: User

Entry Condition(s): Actor is logged in as a user account.

Exit Condition(s): Actor joins the livestream

Flow of Events:

1. User clicks the “View Live Streams” button in the main menu.

2. User clicks on the livestream he/she wants to join.

3. User joins the livestream.

Scenario 10: Donate

Use Case: Donate

Actors: User

Entry Condition(s): Actor is watching a livestream.

Exit Condition(s): Donation is completed.

Flow of Events:

1. User clicks the “Donate” button in the livestream.

2. User specifies the amount of donation and a note to streamer in the pop-up menu.

3. Donation is completed.

Scenario 11: Add Balance

Use Case: Add Balance

Actors: User

Entry Condition(s): Actor is logged in with a user account and is in the main menu.

Exit Condition(s): Balance is increased.

Flow of Events:

1. User clicks the “Add Balance” button in the main menu.

2. User specifies the amount of money to add to balance and credit card details in the pop-up menu.

3. Balance is increased

Scenario 12: Add to Channel

Use Case: Add to Channel

Actors: User

Entry Condition(s): Actor is logged in with a user account and is in the channel page.

Exit Condition(s): Media File is added to the channel.

Flow of Events:

1. User clicks the “Add Media File” button in the respective channel.

2. User selects the media files to add.

3. Media File is added to the channel.

Scenario 13: Remove from Channel

Use Case: Remove From Channel

Actors: User

Entry Condition(s): Actor is logged in with a user account and is in the channel page.

Exit Condition(s): Media File is removed from the channel.

Flow of Events:

1. User clicks the “Remove” button near the media file that has to be deleted.

2. Media File is removed from the channel.

Scenario 14: Remove Media Files

Use Case: Remove Media Files

Actors: Admin

Entry Condition(s): Actor is logged in with an admin account.

Exit Condition(s): Media File is removed from the platform.

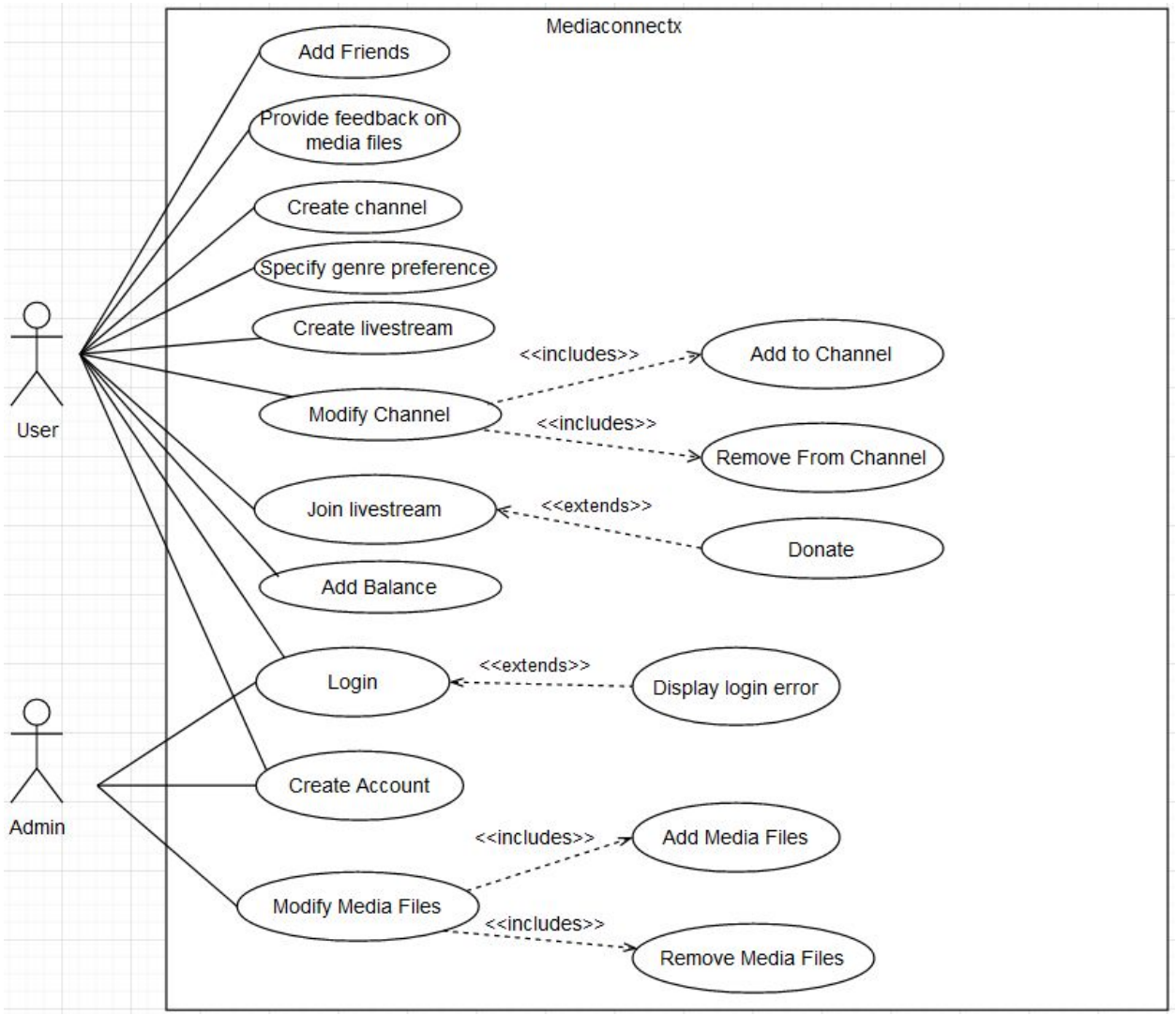
Flow of Events:

1. Admin clicks the “Remove Media File” button in the main menu.

2. Admin selects the media file that should be removed from the channel.

3. Media File is removed from the platform.

3.2 Use Case Diagram

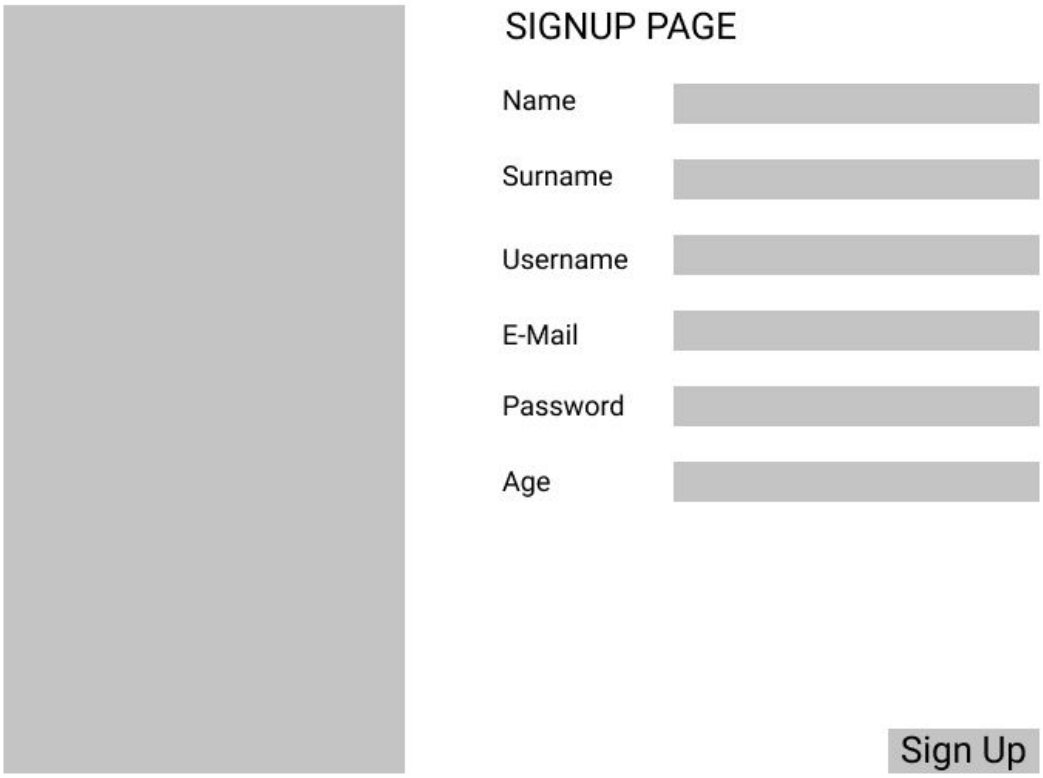


4.Functional dependencies and normalization of tables

All functional dependencies are in BCNF form, so we do not need to use any normalization and decomposition

5.User Interface Design and Corresponding SQL Statements

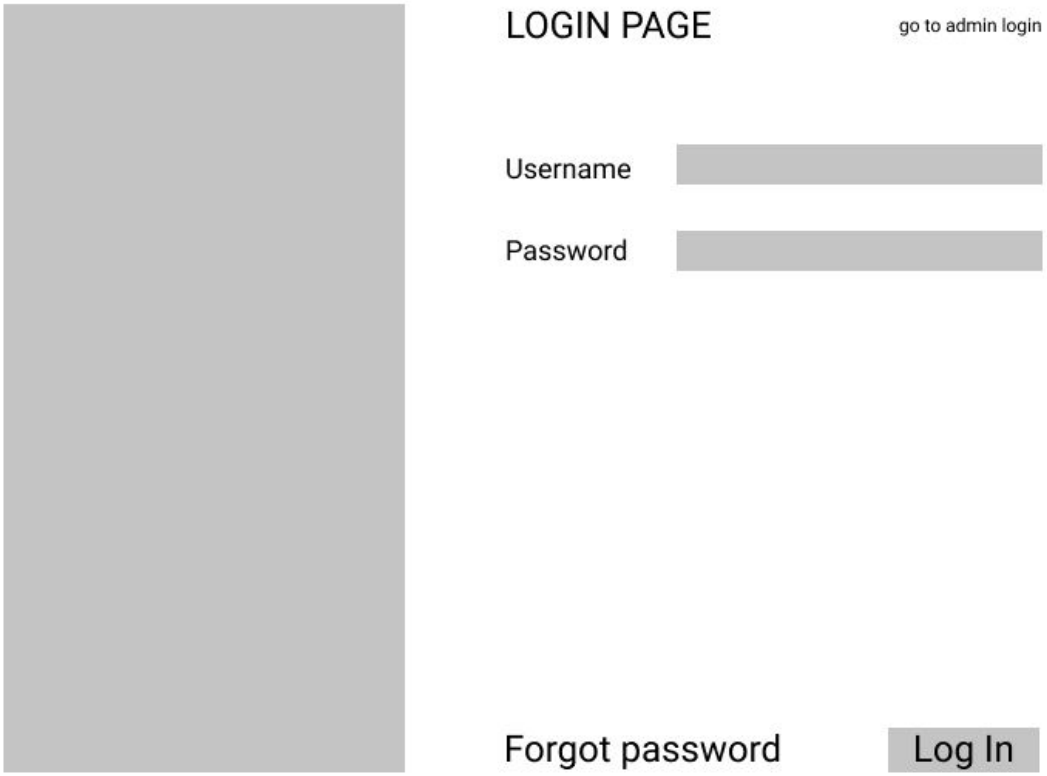
5.1 Sign Up



Inputs: @name, @surname, @username, @e-mail, @password, @age
Process: Guest users can reach this page to become a Regular user by clicking the signup button on Homepage. They need to type their Username, Name, Surname,Email, Age, Password information in order to become a user of Mediaconnectx.
SQL Statements For Sıgn Up Button:

Insert into user(user-id, name, username, e-mail, password, age) **values** (@generatedUserId,@name,@surname,@username,@e-mail,@password,@age)
Insert into regular-user(user-id,money-balance,registered-date) **values** (@generatedUserId,0, @Today'sDay)

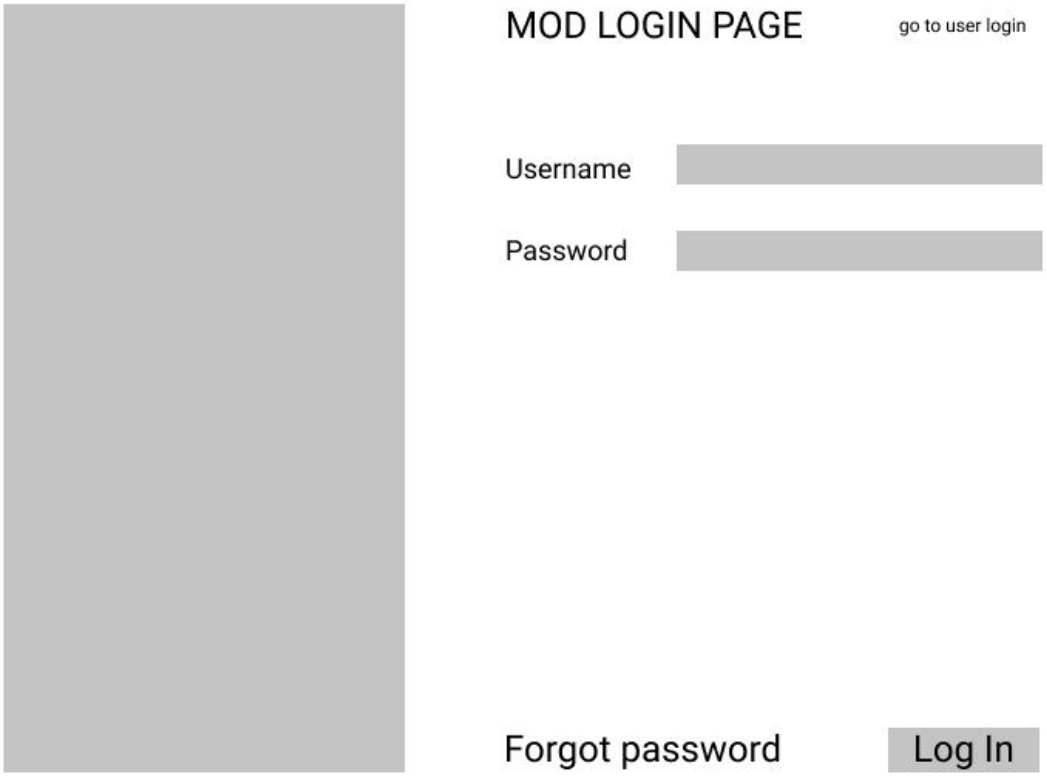
5.2 Log In



Inputs: @username, @password
Process: The user logs in with their password and username. If the user forgot their password, the option is here.
SQL Statements:

```
SELECT  *
FROM users as  U
WHERE @username =U.username AND @password =U.password
```

5.3 Moderator Log In

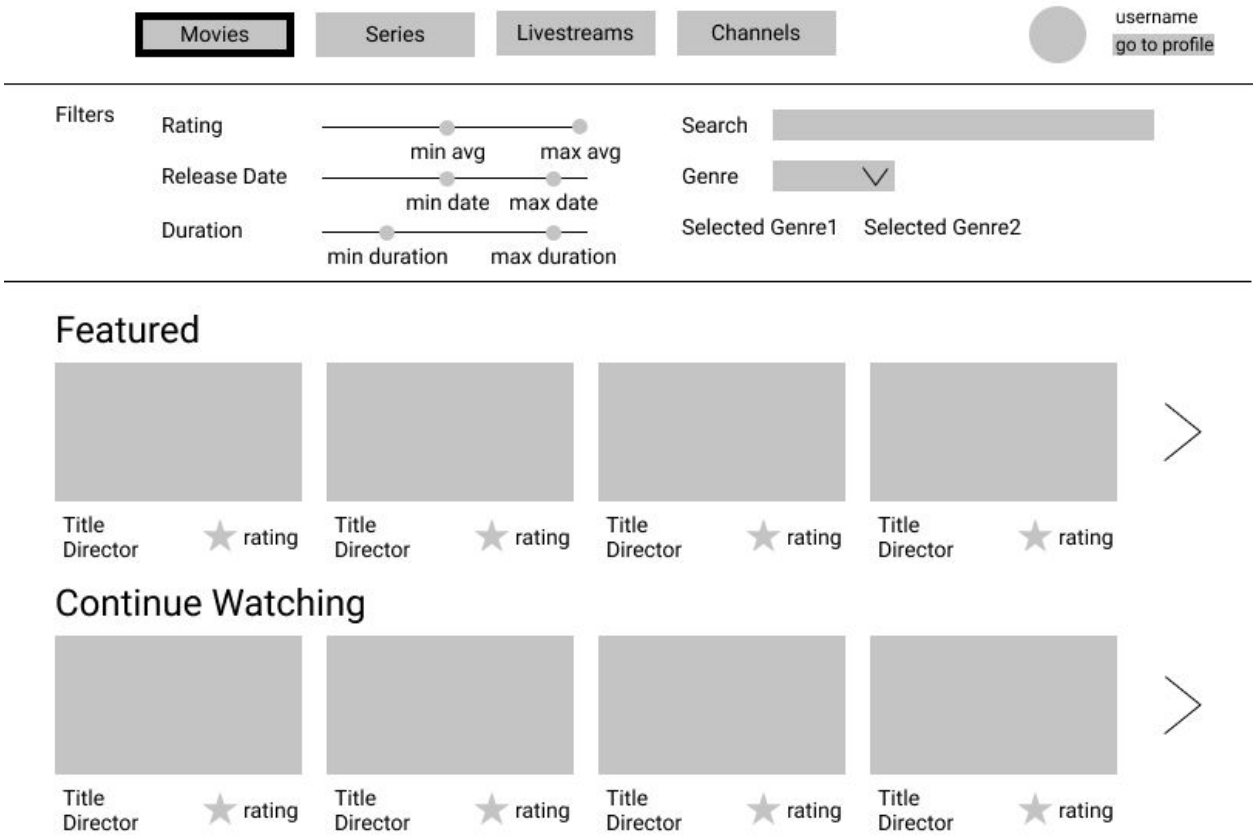


Inputs: @username, @password
Process: The moderators log in with their password and username. If the moderator forgot their password, the option is here.

SQL Statements:

SELECT *
FROM moderator as M
WHERE @username = M.username AND @password = M.password

5.4 Movies



Inputs: @maxRating, @minRating, @maxDuration, @minDuration, @minDate, @maxDate, @search, @GenreInput
Process: A galley of available content under the movie category.

SQL Statements:

<u>Display movies:</u>

Filter movies by name:

SELECT M.name, M.director, M.like-rate
FROM movies as M , media-files M1
WHERE @search = M1.name **AND** M1.media-id = M.media-id

Filter movies by genre:

SELECT C.media-id
FROM contains as C, movie as M
WHERE @GenreInput = C.genre-name **AND** C.media-id = M.media-id

Filter movies by rating:

SELECT M.media-id
FROM movies M, media-files M1
WHERE M1.like-rate < @maxRating **AND** M1.like-rate > @minRating **AND** M1.media-id = M.media-id

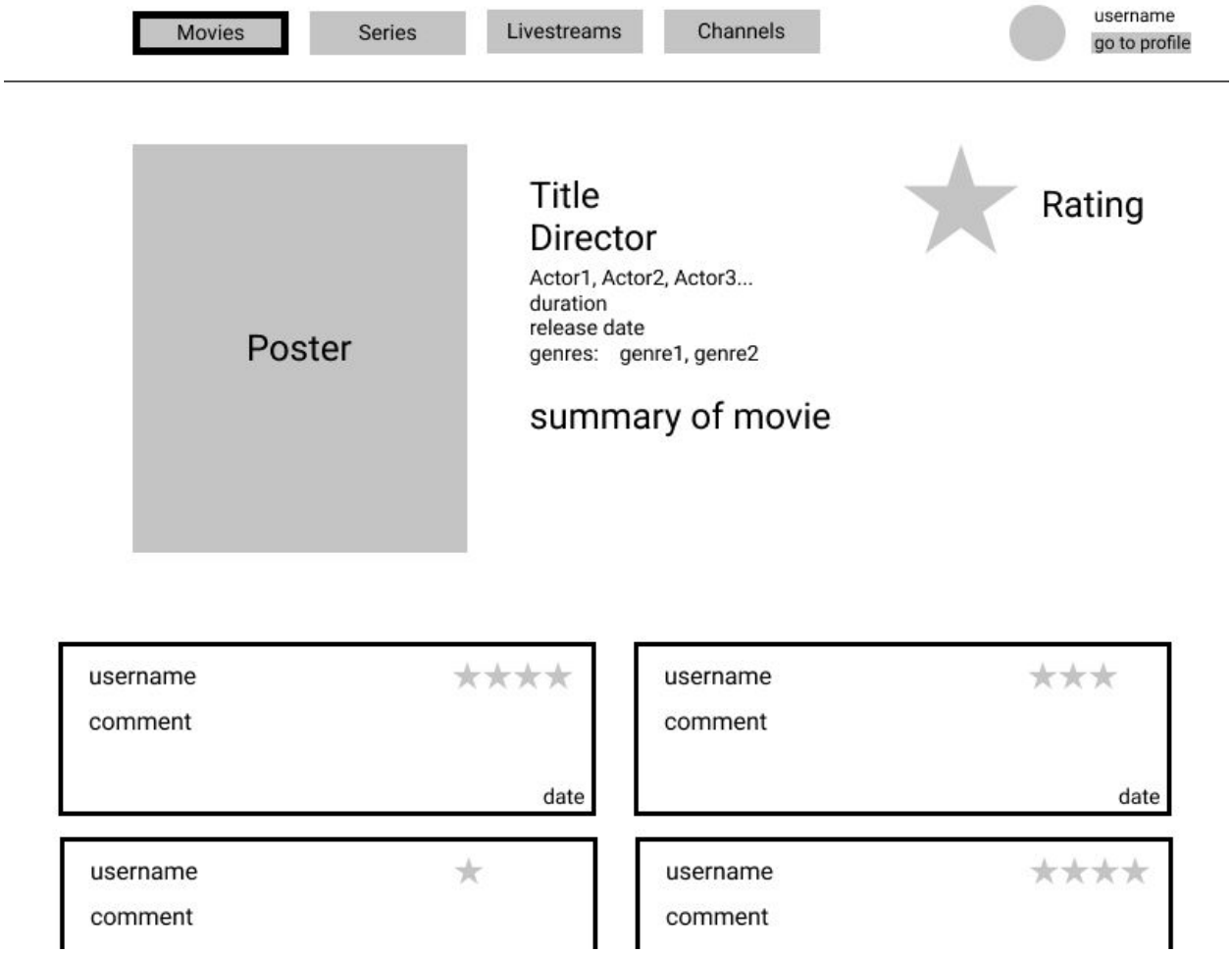
Filter movies by duration:

SELECT M.media-id
FROM movies as M , media-files as M1
WHERE M1.duration-movie < @maxDuration **AND** M.duration-movie > @minDuration **AND** M1.media-id = M.media-id

Filter movies by director:

SELECT M.media-id
FROM movies as M , media-files as M1
WHERE @search = M1.director **AND** M1.media-id = M.media-id

5.5 Movie



Inputs: @selectedMovie
Process: A preview of the movie and its critics.
SQL Statements:

Display movie info:

SELECT M1.name, M1.director, M1.{actor}, M1.like-rate, M1.release-time, M1.info, M1.popular-comment M.duration-movie
FROM movies as M , media-files as M1
WHERE @selectedMovie = M1.name **AND** M.media-id = M1.media-id

5.6 Comment

Inputs: @Comment, @Rate
Process: Adding comment into selected media file and rating it.
SQL Statements:

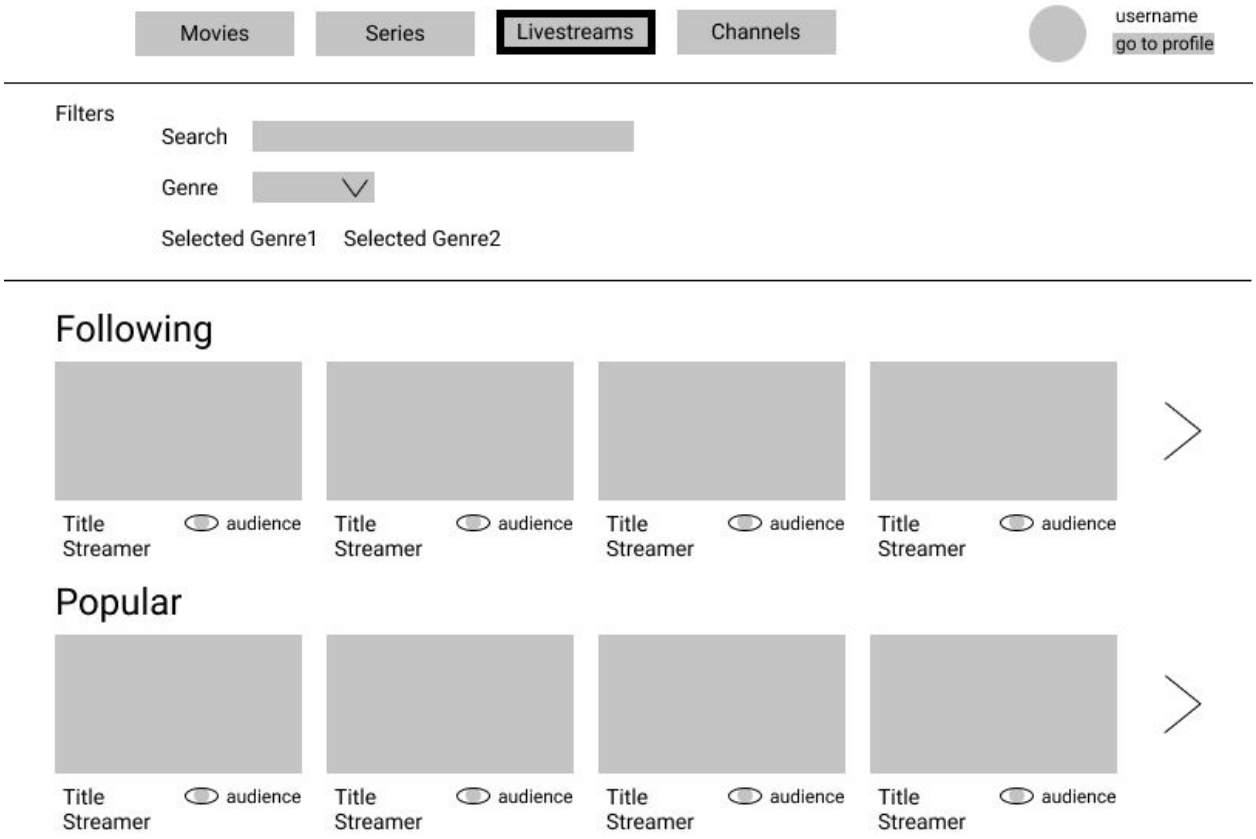
Comment for a Movie:

Insert into comments(user-id,comment-id,livestream-id,media-id,text) **values** (@generatedUserId,@generatedCommentId,NULL,@generatedMediaID,@Comment)

Like for a Movie:

Insert into like(user-id,media-id,livestream-id,rate) **values** (@generatedUserId,@generatedMediaID,generatedMediaID,NULL,@Rate)

5.7 Livestreams



Inputs: @search, @InputGenre
Process: A gallery of available livestreams.

SQL Statements:

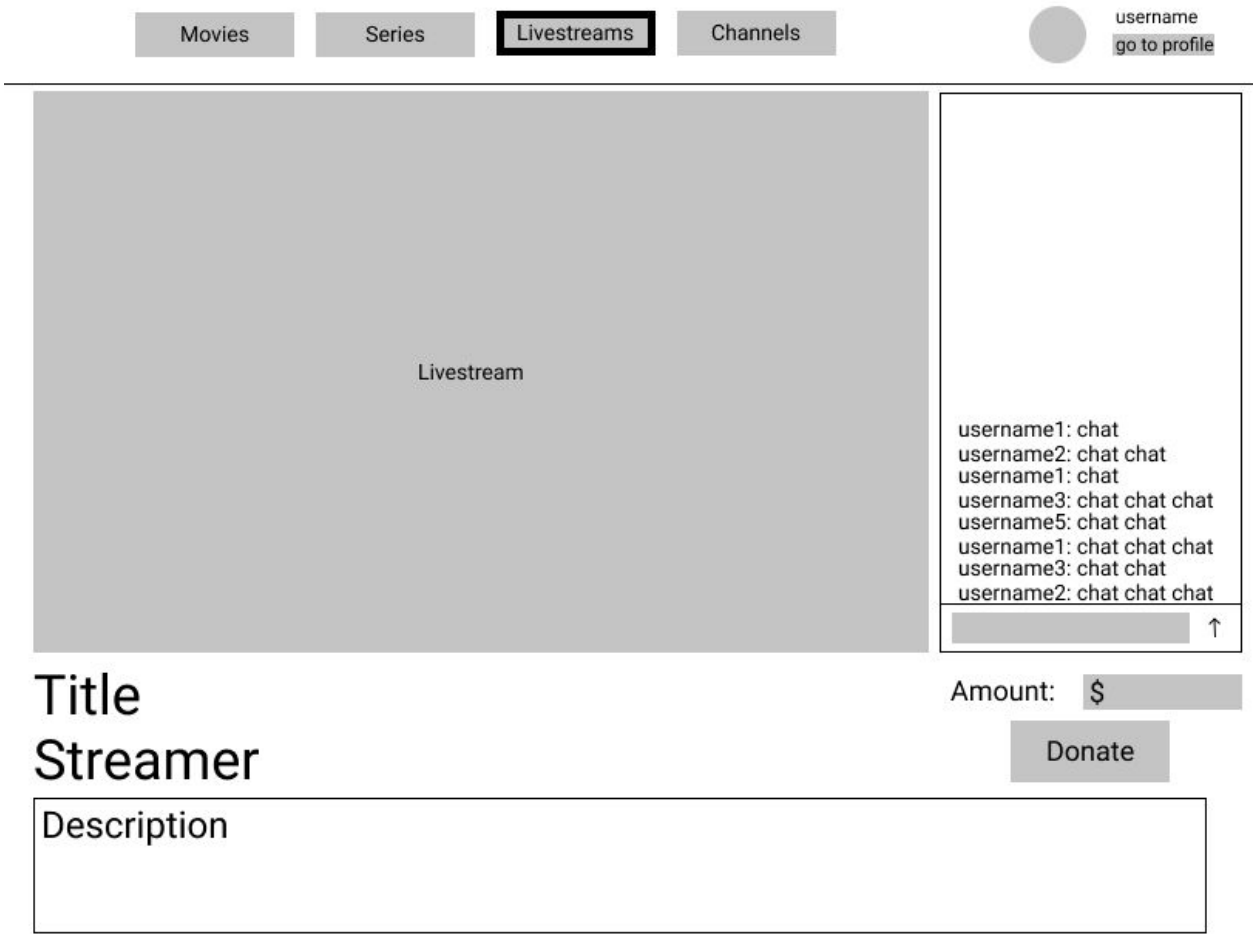
Displaying live streams:

Filter by streamer:

SELECT L.livestream-id
FROM livestream as L , createLiveStream as C , regular-user as R
WHERE @search= R.name **AND** R.user-id = C.user-id **AND** L.livestream-id = C.livestream-id

Filter by content:

SELECT L.livestream-id
FROM livestream as L
WHERE L.content =@InputGenre
5.8 Livestream



Inputs: @Time , @Donate
Process: Watching a livestream with donation option

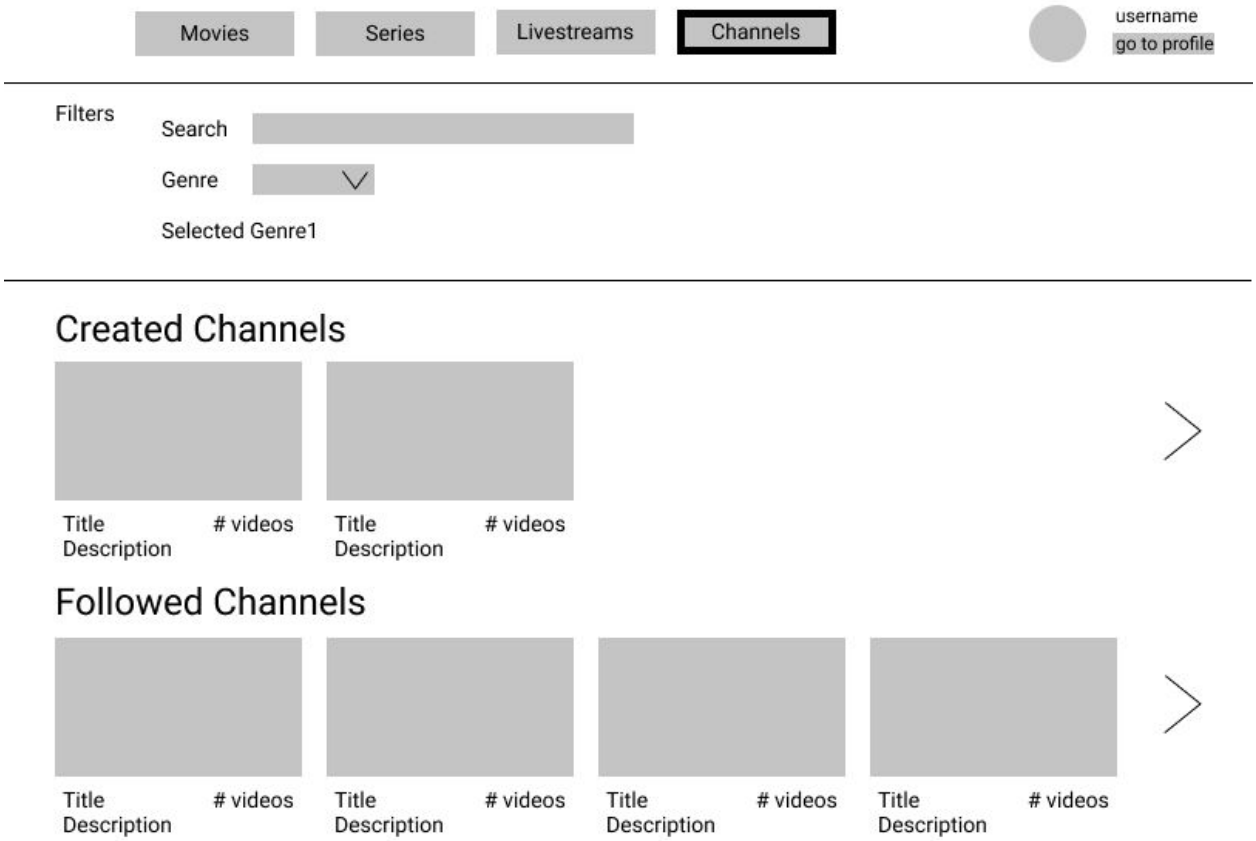
Display Livestream:

SELECT L.livestream-id, L.content , L.participants, C1.comment-id, L.total-donate-amount
FROM createLivestream as C , livestream as L , comments C1
WHERE @Time < C.since **AND** L.livestream-id = C.livestream-id **AND** C1.livestream-id = L.livestream-id

Donate:

Insert into donate(user-id, livestream-id,donate-amount) **values** (@generatedUserId,@generatedLivestream-id,@Donate)

5.9 Channels



Inputs:@Search, @Genre, @Input
Process: Searching existing channels with option which is selecting type of genre.

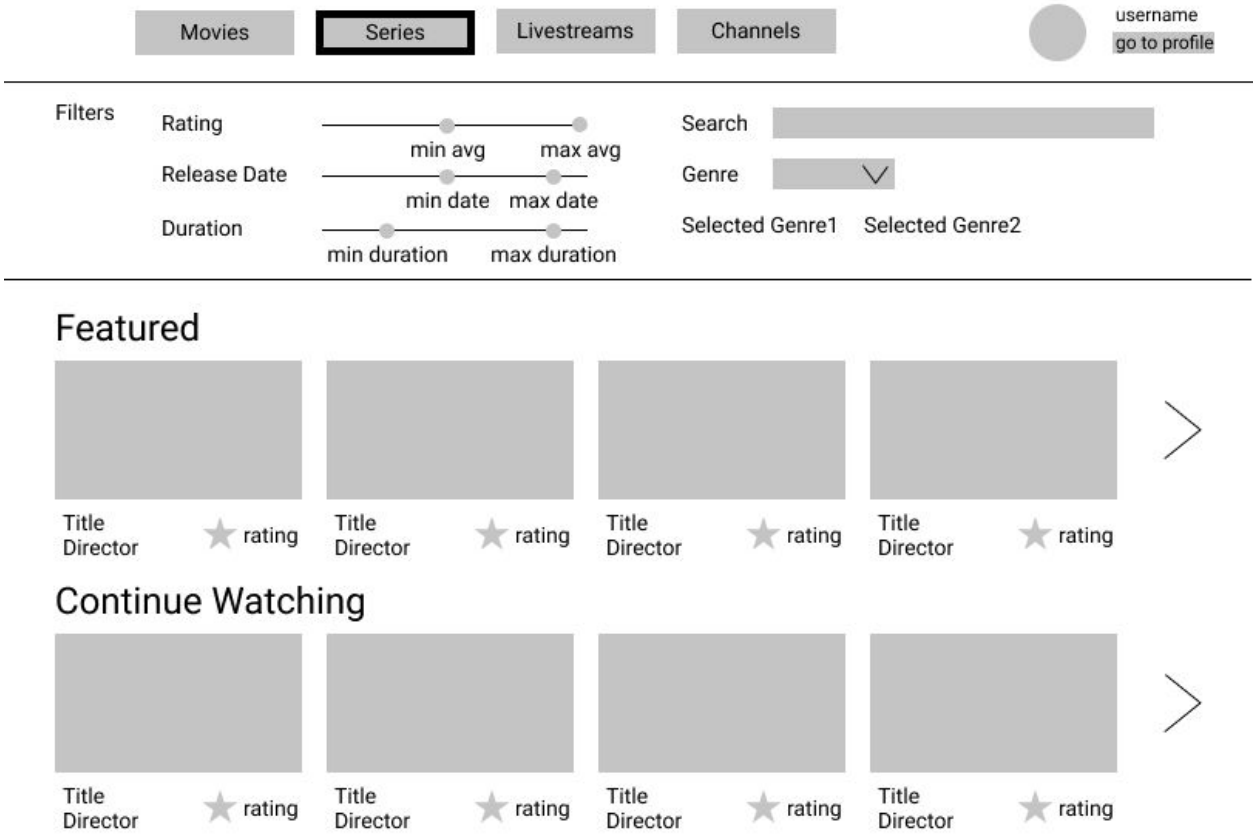
Displaying channels:

SELECT C.channel-id, C.channel name
FROM channels as C
WHERE @Input = C.name

Filter by streamer:

SELECT user-id
FROM Channel as C
WHERE C.user-id =@search

5.10 Series



Inputs: @maxRating, @minRating, @maxDuration, @minDuration, @minDate, @maxDate, @search, @GenreInput
Process: A gallery of available content under the series category.

Filter series by name:

SELECT S.name, S.director, S.like-rate
FROM series as S , media-files as M1
WHERE @search = M1.name **AND** M1.media-id = S.media-id

Filter series by rating:

SELECT S.media-id
FROM series as S, media-files as M1
WHERE M1.like-rate < @maxRating **AND** M1.like-rate > @minRating **AND** M1.media-id = S.media-id

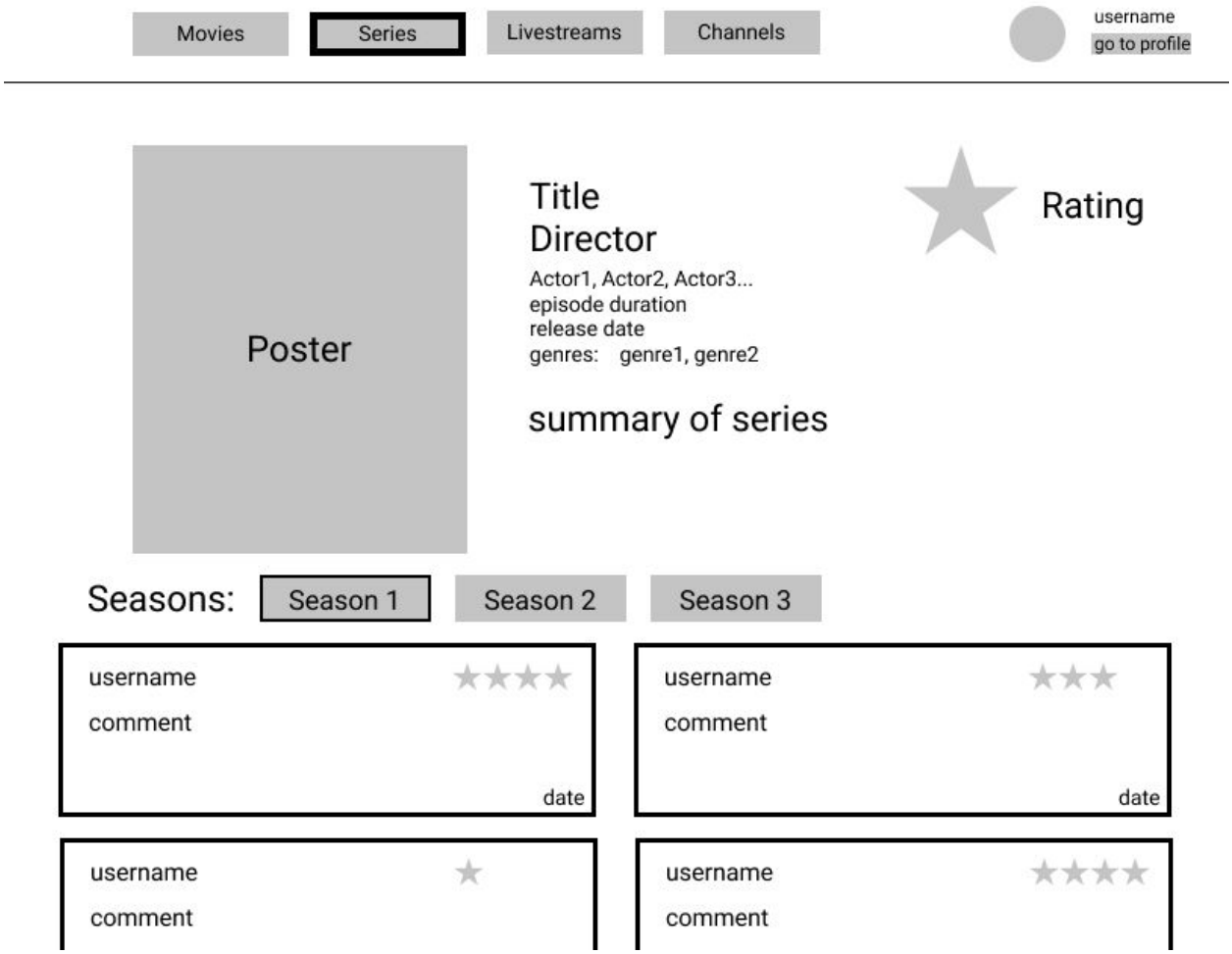
Filter series by duration:

SELECT M.media-id
FROM series as S, media-files as M1
WHERE S.episode-duration< @maxDuration **AND** S.episode-duration> @minDuration **AND** M1.media-id = S.media-id

Filter series by release date:

SELECT S.media-id
FROM series as S , media-files as M1
WHERE M1.release-time < @maxDate **AND** M1.duration > @minDate **AND** M1.media-id = S.media-id

5.11 Serie



Inputs: @selectedSerie, @selectedSeason
Process: A preview of the series and its critics.
SQL Statements:

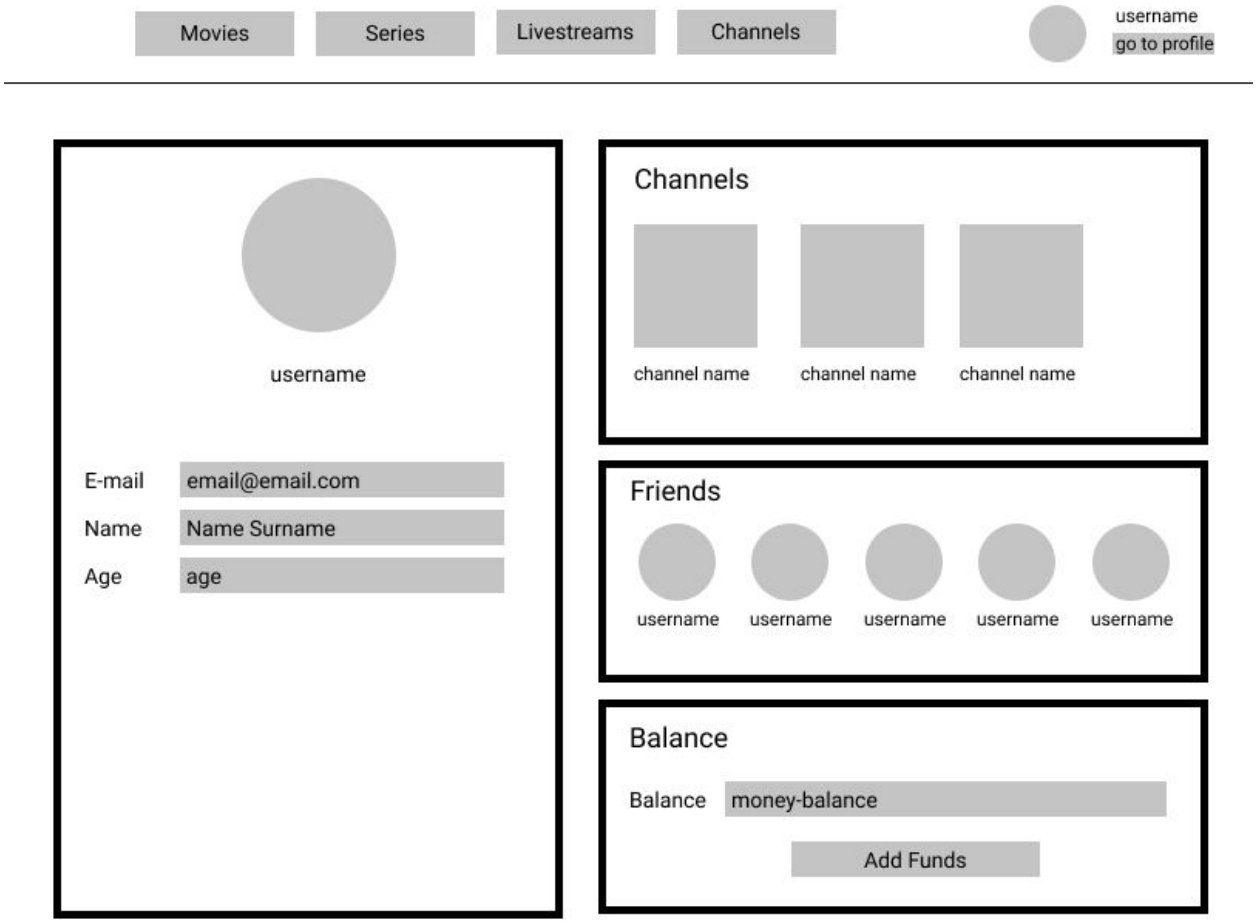
Display series:

SELECT M1.name, M1.director, M1.{actor}, M1.like-rate, M1.release-time, M1.info, M1.popular-comment
FROM series as S , media-files as M1
WHERE @selectedSerie = M.name **AND** S.media-id = M1.media-id

Display comments for each season:

SELECT comment-id
FROM comment
WHERE season-no = @selectedSeason

5.12 Profile



Inputs:
Process: Profile page that includes Channels, Friends, Balance and information about user
SQL Statements:

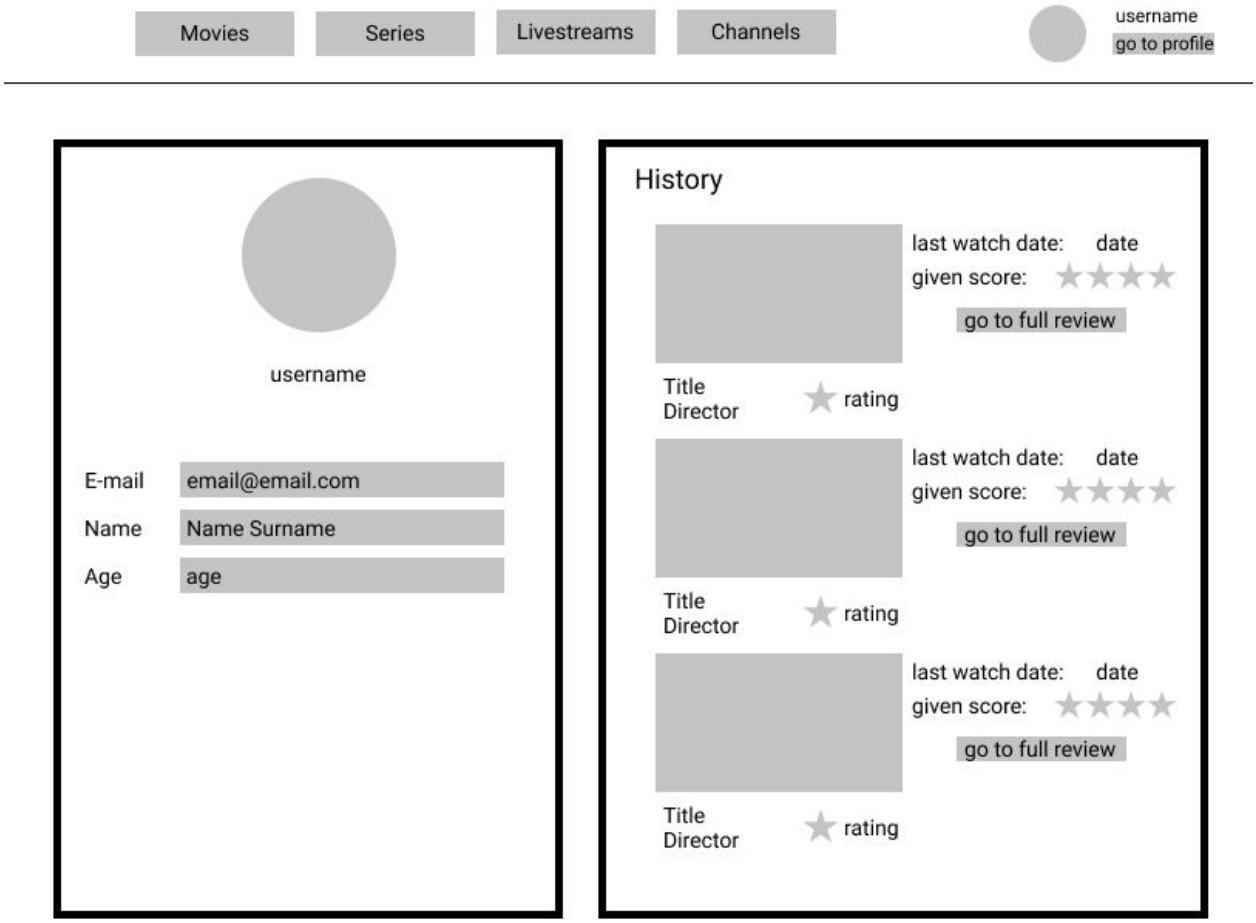
SELECT e-mail, name, username, age
FROM users

SELECT C.channel-name
FROM channels C, users U, editingChannel E
WHERE C.channel-id = E.channel-id **AND** U.user-id = E.user-id

SELECT money-balance
FROM regular-user, users
WHERE users.user-id = regular-user.user-id

SELECT U.name, F.name
FROM users U, Friend F
WHERE F.added-id = U.user-i

5.13 Profile History



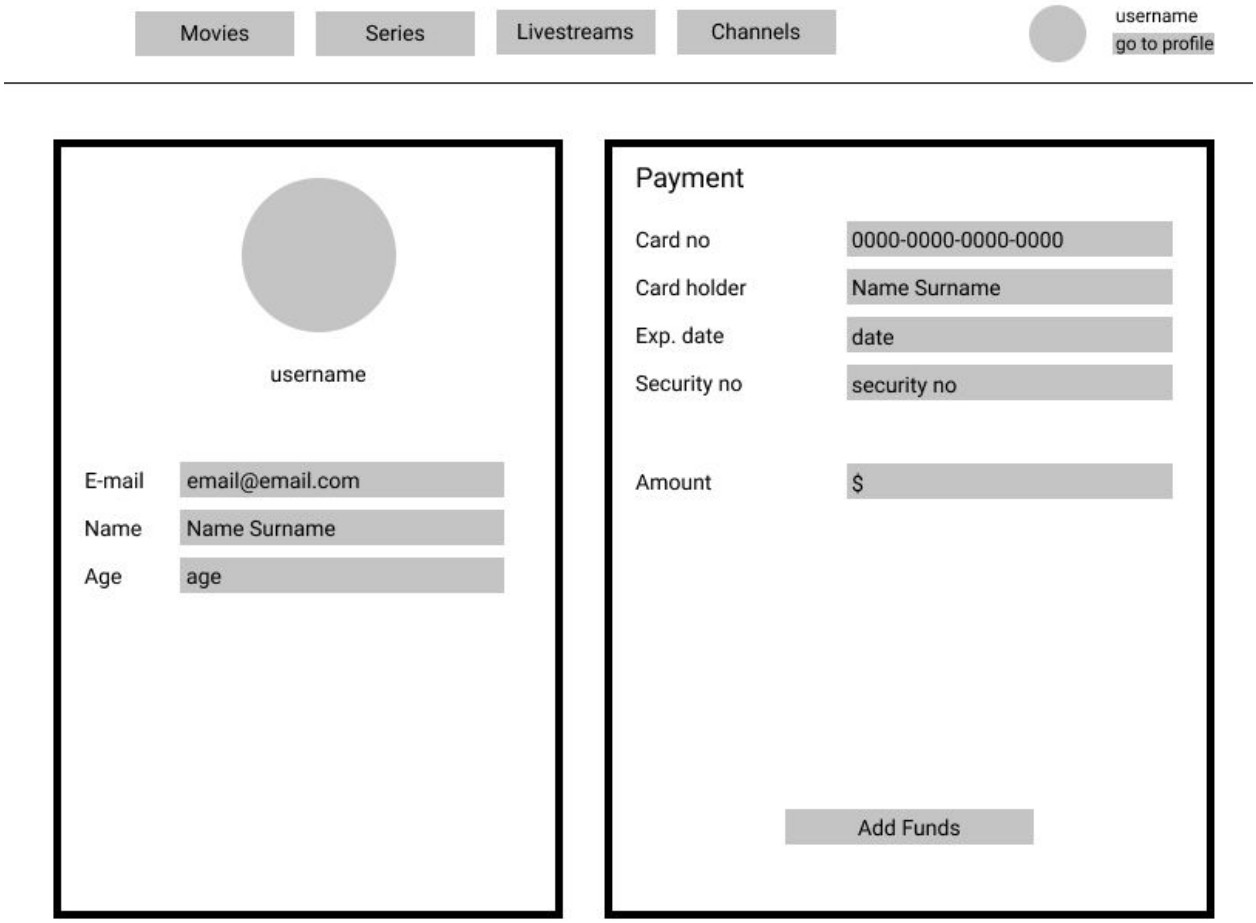
Inputs:
Process: History page that contains remaining time of the media file.
SQL Statements:

Showing the information of the user:
SELECT e-mail, name, username, age
FROM users

Showing the watched movies and series:

SELECT M.name, M.like-rate, M.director, S.current-duration
FROM media-files M, Status S, watch W, users U
WHERE U.user-id=W.user-id and W.media-id=M.media-id and S.media-id=W.media-id and S.user-id=W.user-id

5.14 Add Funds



Inputs: @CardNo, @CardHolder, @ExpDate, @SecurityNo, @Amount
Process: Adding money balance into personal account.
SQL Statements:

Showing the information of the user:
SELECT e-mail, name, username, age
FROM users

Updating balance of the user:
UPDATE regular-user
SET money-balance = @Amount

WHERE user-id = @GeneratedUserID

6.Advanced Database Components

6.1 Reports

- **Most watched movies according to age of users:**

CREATE VIEW mostWatchedMovies **as**
(**SELECT** M.name, count(W.media-id) as movieCount, age
FROM media-files M, movies M1, watch W, users
GROUP BY age
WHERE media-files.media-id = W.media-id and users.user-id = W.user-id and M1.media-id=media-files.media-id)
SELECT *
FROM mostWatchedMovies
WHERE movieCount=max(movieCount)

- **Most watched live stream in a year:**

SELECT max(mycount)
FROM(SELECT content, name, count(livestream-id) as mycount
FROM livestream, watch, users
WHERE livestream-id=watch.livestream-id and watch.user-id = users.user-id and date **BETWEEN** '01/01/year' and '12/31/year')

- **Most watched series according to age of users:**

CREATE VIEW mostWatchedSeries as
(**SELECT** M.name, **count**(W.media-id) as seriesCount, age
FROM media-files M, series S1, watch W, users
GROUP BY age
WHERE media-files.media-id = W.media-id and users.user-id = W.user-id and S1.media-id=media-files.media-id)
SELECT *
FROM mostWatchedMovies
WHERE seriesCount=**max**(seriesCount)

6.2 Views

- **Users can see the people who donate their live streams**.

CREATE VIEW donationView as

(**SELECT** name, donate-amount
FROM donate , users, livestream
WHERE users.user-id=donate.user-id and livestream.livestream-id=@user.user-id)

- **Users can see popular media files in terms of genre**

CREATE VIEW PopularFiles as
(**SELECT** M.name, count(W.media-id) as mediaCount, g.genre-name
FROM media-files M, watch W, contains C, Genres g, users U
GROUP BY g.genre-name, mediaCount DESC
WHERE M.media-id = W.media-id and C.genre-id=g.genre-id and U.user-id=W.user-id)

6.3 Triggers

- When a media file is deleted from the system, it is deleted from the media files table and all other related tables such as Users, comment, channels etc.
- When a user account is deleted from the system, it is deleted from the related tables such as Users and Regular User.

6.4 Constraints

- Only administrators(moderators) can add new media files to the platform.
- Users, who are younger than 18, cannot view mature content.
- Administrators can remove a user if he/she violates the application usage terms and conditions.
- The user will not be able to donate money if he/she does not have enough money in the balance.
- Same media file cannot be added to the same channel twice.

6.5 Stored Procedures

- Users will be notified when other users adds them as a friend
- Users will be notified when there is a reply to their comments.
- Users will be notified when someone donates their livestream.
- Users will be informed when balance adding operation is successful.

7. Implementation Plan

We have planned to use MySQL in order to manage the database system. We will use ASP.NET core technology for development. In addition to that, we will use entity framework core to make database connection with our models. For front-end development we aim to use cshtml and css languages.

8. Website

Mediaconnectx | cs353group10 (aerk1996.github.io)