Bilkent University

Department of Computer Engineering

CS 353 Course Project

Group 9

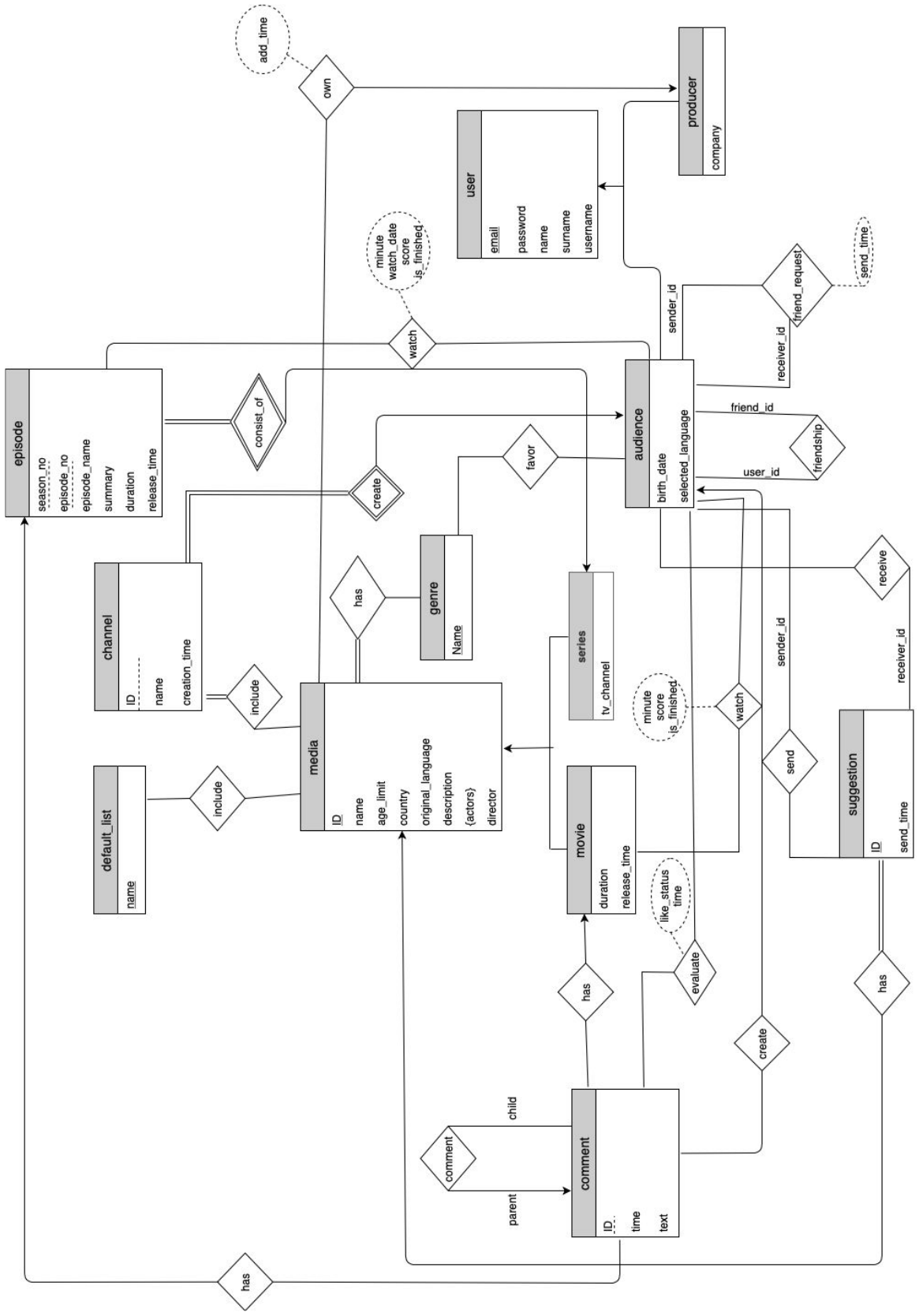
Design Report

- Yavuz Faruk Bakman
- Arda Göktoğan
- Fatih Sevban Uyanık
- Duygu Nur Yaldız

Supervisor: Arif Usta

Table of Contents

1. Revised ER Model



- According to the feedback from the TA, we have removed the notification entity because it is unnecessary to hold this information in the database.
- We represent comment and episode as not a weak entity because it causes ambiguity and it increases the number of entities and relations. Therefore, we removed comment_episode and comment_movie and created one comment entity.
- According to the functionality documentation, we do not need any subtitles in the watching simulation. Therefore we removed the subtitle entity.
- According to the functionality documentation, we add a producer entity which is responsible for adding new media. Then we rearrange the user, producer and audience entities.
- We remove the suggestions' movie_id and series_id attributes and add corresponding relations for these attributes.
- We add a watch relationship for episodes and movies. We hold the minute information, score information (rating), and watching date which is used for finding the latest watched episode.
- We removed the film_member entity and added film members to the media as attributes for the sake of simplicity.

2. Relational Schemas

2.1. User

Relational Model: `user(email, password, name, surname, username)`
Functional Dependencies: `{ (email -> email, password, name, surname, username), (username -> email, password, name, surname, username) }`
Candidate Keys: `{ {email}, {username}}`
Primary Key: `{email}`
Foreign Keys: `{}`
Normal Form: BCNF
Table Definition:

```
create table user( email varchar(50) not null,
                   password varchar(30) not null,
                   name varchar(20) not null,
                   surname varchar(20) not null,
                   username varchar(20) not null,
                   primary key (email)
            );
```

2.2. Audience

Relational Model: audience(email, birth_date, selected_language)
Functional Dependencies: { (email -> birth_date, selected_language) }
Candidate Keys: { {email}}
Primary Key: {email}
Foreign Keys: { (email -> user.email)}
Normal Form: BCNF
Table Definition:

```
create table audience( email varchar(50) not null,
                  birth_date date not null,
                  selected_language varchar(20) not null,
                  primary key (email),
                  foreign key (email) references user
            );
```

2.3. Producer

Relational Model: producer(email, company)
Functional Dependencies: { (email -> company) }
Candidate Keys: { {email}}
Primary Key: {email}
Foreign Keys: { (email -> user.email)}
Normal Form: BCNF
Table Definition:

```
create table producer( email varchar(50) not null,
                  company varchar(30) not null,
                  primary key (email),
                  foreign key (email) references user
            );
```

2.4. Friendship

Relational Model: friendship(user_id, friend_id)
Functional Dependencies: { (user_id, friend_id -> user_id, friend_id) }
Candidate Keys: { {user_id, friend_id}}
Primary Key: {{user_id, friend_id}}
Foreign Keys: { (user_id-> audience.email), (friend_id -> user.email)}

Normal Form: BCNF
Table Definition:

```
create table friendship( user_id varchar(50) not null,
                  friend_id varchar(50) not null,
                  primary key (user_id, friend_id),
                  foreign key (user_id) references audience,
                 foreign key (friend_id) references audience
            );
```

2.5. Friend_request

Relational Model: friend_request(sender_id, receiver_id, send_time)
Functional Dependencies: { (sender_id, receiver_id -> send_time) }
Candidate Keys: { {sender_id, receiver_id}}
Primary Key: {{sender_id, receiver_id}}
Foreign Keys: { (sender_id-> audience.email), (receiver_id -> audience.email)}
Normal Form: BCNF
Table Definition:

```
create table friend_request( sender_id varchar(50) not null,
                 receiver_id varchar(50) not null,
                  send_time timestamp not null,
                  primary key (sender_id, receiver_id),
                  foreign key (sender_id) references audience,
                 foreign key (receiver_id) references audience
            );
```

2.6. Suggestion

Relational Model: suggestion(id, media_id, send_time)
Functional Dependencies: { (id -> send_time) }
Candidate Keys: { {id} }
Primary Key: { {id} }
Foreign Keys: { (media_id -> media.id) }
Normal Form: BCNF
Table Definition:

```
create table suggestion( id int primary key auto_increment,
                 media_id char(30) not null,
                  send_time timestamp not null,
                  primary key (id),
                  foreign key (media_id) references media
                  );
```

2.7. Friend_suggestion

Relational Model: friend_suggestion(sender_id, receiver_id, suggestion_id)
Functional Dependencies:
{ (sender_id, receiver_id, suggestion_id-> sender_id, receiver_id, suggestion_id) }
Candidate Keys: { {sender_id, receiver_id, suggestion_id}}
Primary Key: { {sender_id, receiver_id, suggestion_id} }
Foreign Keys: { (sender_id-> audience.email), (receiver_id -> audience.email), (suggestion_id -> suggestion.id)}
Normal Form: BCNF
Table Definition:

```
create table friend_suggestion( sender_id varchar(50) not null,
                 receiver_id varchar(50) not null,
                 suggestion_id int not null,
                  primary key (sender_id, receiver_id,
           suggestion_id),
                  foreign key (sender_id) references audience,
                 foreign key (receiver_id) references audience,
                 foreign key (suggestion_id) references suggestion);
```

2.8. Media

Relational Model: media(id, name, age_limit, country, original_language ,description,director)
Functional Dependencies: { (id -> name, age_limit, country, original_language ,description, director) }
Candidate Keys: { {id} }
Primary Key: {id}
Foreign Keys: {}
Normal Form: BCNF
Table Definition:

```
create table media( id  int primary key auto_increment,
                  name varchar(30) not null,
                  age_limit int not null,
                  country varchar(20),
                  original_language  varchar(20) not null,
                  description  varchar(200) not null,
                  primary key (id)
            );
```

2.9. Actors_media

Relational Model: actors_media(media_id, actor)
Functional Dependencies: { (media_id, actor -> media_id, actor) }
Candidate Keys: { {media_id, actor} }
Primary Key: {media_id, actor}
Foreign Keys: {media_id -> media.id}
Normal Form: BCNF
Table Definition:

```
create table media( media_id  int not null,
                  actor varchar(30) not null,
                   foreign key(media_id) references media,
                  primary key (media_id, actor)
            );
```

2.10. Series

Relational Model: series(id, tv_channel)
Functional Dependencies: { (id -> tv_channel) }
Candidate Keys: { {id} }
Primary Key: {id}
Foreign Keys: { id -> media.id}
Normal Form: BCNF
Table Definition:

```
create table series( id  int not null,
                  tv_channel varchar(50) not null,
                  primary key (id),
                  foreign key (id) references media
            );
```

2.11. Movie

Relational Model: movie(id, duration, release_time)
Functional Dependencies: { (id -> duration, release_time) }
Candidate Keys: { {id} }
Primary Key: {id}
Foreign Keys: { id -> Media.id}
Normal Form: BCNF
Table Definition:

```
create table movie( id  int not null,
                  duration float not null,
                  release_time date not null,
```

```
                primary key (id),
                foreign key (id) references media
            );
```

2.12. Episode

Relational Model: episode(series_id, season_no, episode_no, episode_name, summary, duration, release_time)
Functional Dependencies: { (series_id, season_no, episode_no -> episode_name, summary, duration, release_time) }
Candidate Keys: { {series_id, season_no, episode_no} }
Primary Key: {series_id, season_no, episode_no}
Foreign Keys: { series_id -> series.id}
Normal Form: BCNF
Table Definition:

```
create table episode( series_id  int not null,
                season_no  int not null,
                           episode_no  int not null,
                episode_name varchar(50) ,
                summary varchar(150),
                duration float not null,
                release_time date not null,
                primary key (series_id, season_no, episode_no),
                foreign key (series_id) references series
            );
```

2.13. Watch_episode

Relational Model: watch_episode(series_id, season_no, episode_no, audience_id, minute, watch_date,score,is_finished)
Functional Dependencies: { (series_id, season_no, episode_no, audience_id -> minute, watch_date,score,is_finished) }
Candidate Keys: { {series_id, season_no, episode_no, audience_id } }
Primary Key: {series_id, season_no, episode_no, audience_id}
Foreign Keys: { series_id -> series.id, (series_id, season_no, episode_no) -> (episode.series_id, episode. season_no, eposide.episode_no), audience_id -> audience.email }
Normal Form: BCNF

Table Definition:

```
create table watch_episode( series_id  int not null,
                  season_no  int not null,
                            episode_no  int not null,
                            audience_id  varchar(50) not null,
                  minute int not null,
                  watch_date timestamp not null,
                  score int,
                   is_finished bit not null,
                  primary key (series_id, season_no, episode_no,
           audience_id),
                  foreign key (series_id) references series,
                foreing key (series_id, season_no, episode_no)
           references episode,
                foreign key (audience_id) references audience
            );
```

2.14. Media_owner

Relational Model: media_owner(media_id, producer_id, add_time)
Functional Dependencies: { (media_id, producer_id -> add_time) }
Candidate Keys: { {media_id, producer_id} }
Primary Key: {media_id, producer_id}
Foreign Keys: { producer_id -> producer.email, media_id -> media.id}
Normal Form: BCNF
Table Definition:

```
create table media_owner( media_id  int not null,
                  producer_id  varchar(50) not null,
                  add_time timestamp,
                  primary key (media_id, producer_id),
                  foreign key (media_id) references media,
                 foreign key (producer_id) references producer
            );
```

2.15. Watch_movie

Relational Model: watch_movie(movie_id, audience_id, minute ,score,is_finished)
Functional Dependencies: { (movie_id, audience_id -> minute ,score,is_finished) }
Candidate Keys: { {movie_id, audience_id } }

Primary Key: {movie_id, audience_id}
Foreign Keys: { movie_id -> movie.id, audience_id -> audience.email }
Normal Form: BCNF
Table Definition:

```
create table watch_movie( movie_id int not null,
                            audience_id  varchar(50) not null,
                  minute int not null,
                  score int,
                 is_finished bit not null,
                  primary key (movie_id, audience_id),
                  foreign key (movie_id) references movie,
                foreing key (audience_id) references audience
            );
```

2.16. Genre

Relational Model: genre(name)
Functional Dependencies: { (name -> name) }
Candidate Keys: { {name } }
Primary Key: {name}
Foreign Keys: { }
Normal Form: BCNF
Table Definition:

```
create table genre(name varchar(30) not null,
                  primary key (name)
             );
```

2.17. Media_genre

Relational Model: media_genre(media_id , genre_name)
Functional Dependencies: { (media_id , genre_name-> media_id , genre_name) }
Candidate Keys: { {media_id , genre_name } }
Primary Key: {media_id , genre_name}
Foreign Keys: { media_id -> media.id , genre_name -> genre.name }
Normal Form: BCNF
Table Definition:

```
create table media_genre(genre_name varchar(30) not null,
                                media_id int not null,
                      foreign key (media_id) references media,
```

```
                foreign key (genre_name) references genre,
                  primary key (media_id , genre_name)
            );
```

2.18. Favor_genre

Relational Model: favor_genre(audience_id , genre_name)
Functional Dependencies: { (audience_id , genre_name-> audience_id , genre_name) }
Candidate Keys: { {audience_id , genre_name } }
Primary Key: {audience_id , genre_name}
Foreign Keys: { audience_id -> audience.email , genre_name -> genre.name }
Normal Form: BCNF
Table Definition:

```
create table favor_genre(genre_name varchar(30) not null,
                              audience_id varchar(50) not null,
                     foreign key (audience_id) references media,
                foreign key (genre_name) references genre,
                  primary key (audience_id , genre_name)
            );
```

2.19. Channel

Relational Model: channel(id , owner_id, name, creation_time)
Functional Dependencies: { (id , owner_id -> name, creation_time) }
Candidate Keys: { {id , owner_id } }
Primary Key: {id , owner_id}
Foreign Keys: { owner_id -> audience.email}
Normal Form: BCNF
Table Definition:

```
create table channel(id int primary key auto_increment,
                     owner_id varchar(50) not null,
                name varchar(30) not null,
                creation_time date not null,
                      foreign key (owner_id) references audience,
                     primary key (id , owner_id)
            );
```

2.20. Channel_media

Relational Model: channel_media(channel_id , owner_id, media_id)
Functional Dependencies: { (channel_id , owner_id, media_id -> channel_id , owner_id, media_id) }
Candidate Keys: { {channel_id , owner_id, media_id } }
Primary Key: {channel_id , owner_id, media_id}
Foreign Keys: { owner_id -> audience.email , (channel_id, owner_id) -> (channel.id, channel.owner_id)}
Normal Form: BCNF
Table Definition:

```
create table channel_media(channel_id int not null,
                                owner_id varchar(50) not null,
                media_id int not null,
                      foreign key (owner_id) references audience,
                      foreign key (channel_id, owner_id) references
channel,
                     primary key (channel_id , owner_id, media_id)
             );
```

2.21. Default_list

Relational Model: default_list(name)
Functional Dependencies: { (name -> name) }
Candidate Keys: { {name } }
Primary Key: {name}
Foreign Keys: { }
Normal Form: BCNF
Table Definition:

```
create table default_list(name varchar(30) not null,
                     primary key name
             );
```

2.22. Default_list_include

Relational Model: default_list_include(name ,media_id)
Functional Dependencies: { (name ,media_id -> name ,media_id) }
Candidate Keys: { {name ,media_id } }
Primary Key: {name ,media_id }
Foreign Keys: { name -> default_list.name, media_id -> media.id }

Normal Form: BCNF
Table Definition:

```
create table default_list_include(name varchar(30) not null,
                media_id int not null,
                foreign key (name)  references default_name,
                foreign key (media_id) references media
                     primary key (name, media_id)
             );
```

2.23. Comment

Relational Model: comment(id, time, text, parent_id, owner_id)
Functional Dependencies: { (id -> id, time, text, parent_id, owner_id) }
Candidate Keys: { { id } }
Primary Key: { id }
Foreign Keys: { parent_id -> comment.parent_id , owner_id -> audience.email}
Normal Form: BCNF
Table Definition:

```
create table comment( id int primary key auto_increment,
                time timestamp not null,
                text varchar(300) not null,
                parent_id int,
                owner_id varchar(50),
                foreign key (parent_id)  references comment,
                foreign key (owner_id)  references audience,
                     primary key (id)
             );
```

2.24. Comment_evaluate

Relational Model: comment_evaluate(comment_id, audience_id, like_status, time)
Functional Dependencies: { (comment_id, audience_id -> like_status, time) }
Candidate Keys: { {comment_id, audience_id } }
Primary Key: {comment_id, audience_id }
Foreign Keys: { comment_id -> comment.id , audience_id -> audience.email}
Normal Form: BCNF

Table Definition:

```
create table comment_evaluate( comment_id int not null,
                audience_id varchar(50) not null,
                like_status int not null,
                time timestamp not null,
                foreign key (comment_id )  references comment,
                foreign key (audience_id )  references audience,
                     primary key (comment_id, audience_id)
             );
```

2.25. Comment_movie

Relational Model: comment_movie(comment_id, movie_id)
Functional Dependencies: { (comment_id, movie_id -> comment_id, movie_id) }
Candidate Keys: { {comment_id, movie_id } }
Primary Key: {comment_id, movie_id }
Foreign Keys: { comment_id -> comment.id , movie_id -> movie.id}
Normal Form: BCNF
Table Definition:

```
create table comment_movie( comment_id int not null,
                movie_id int not null,
                foreign key (comment_id )  references comment,
                foreign key (movie_id )  references movie,
                     primary key (comment_id, movie_id)
             );
```

2.26. Comment_episode

Relational Model: comment_episode(comment_id, series_id, season_no, episode_no)
Functional Dependencies: { (comment_id, series_id, season_no, episode_no -> comment_id, series_id, season_no, episode_no) }
Candidate Keys: { {comment_id, series_id, season_no, episode_no } }
Primary Key: {comment_id, series_id, season_no, episode_no }
Foreign Keys: { comment_id -> comment.id , (series_id, season_no, episode_no) -> (episode.id, episode.season_no, episode.episode_no)}
Normal Form: BCNF
Table Definition:

```
create table comment_episode( comment_id int not null,
                series_id int not null,
                season_no int not null,
```

```
        episode_no int not null,
        foreign key (comment_id )  references comment,
        foreign key (series_id, season_no, episode_no)
        references episode,
            primary key (comment_id, series_id, season_no,
episode_no)
  );
```

3. User Interface Design and SQL Statements

3.1. Login

Sign In

```
SELECT *
FROM user
WHERE email = @email AND password = @password
```

3.2. Forgot Password

Change Password

```
UPDATE user
SET password=@password
WHERE email=@email
```

3.3. Sign Up


Sign Up-User
LAVA
User Account
Company Member Account
E-mail:
Name:
Surname:
Username:
Birh Date:
Display Language:
English
Password:
Create Account


Sign Up-Company
LAVA
User Account
Company Member Account
E-mail:
Name:
Surname:
Username:
Company:
Password:
Create Account

Audience Sign Up

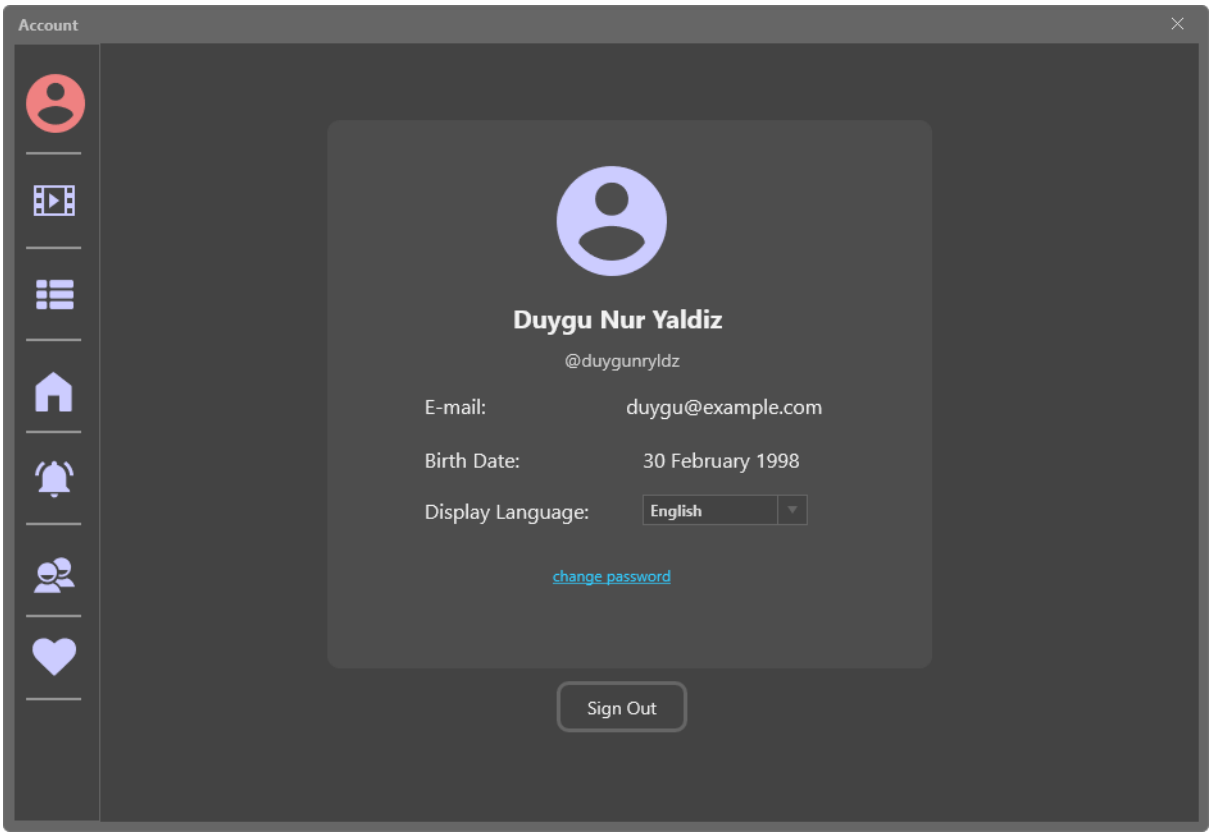
```
INSERT INTO user
VALUES (@email, @password, @name, @surname, @username)

INSERT INTO audience
VALUES (@email, @birth_date, @selected_language)
```

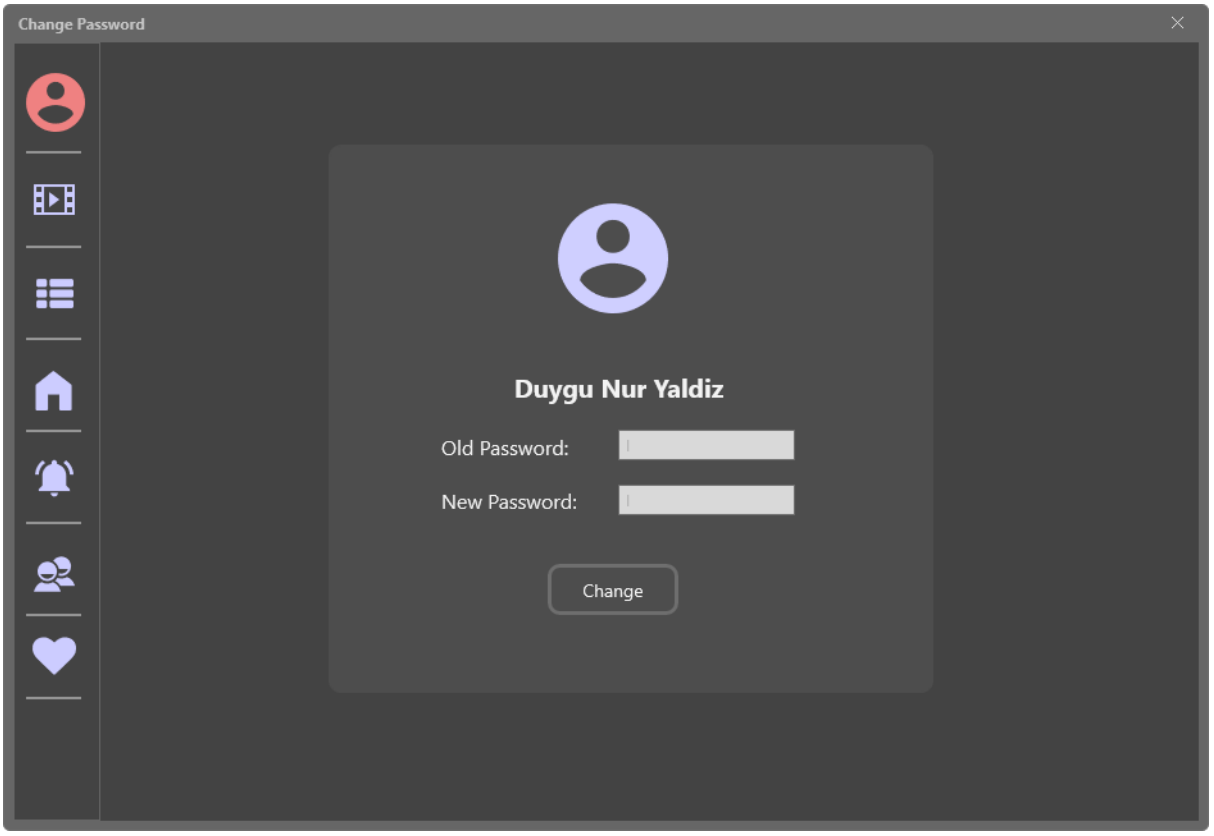
Company Member Sign Up

```
INSERT INTO user
VALUES (@email, @password, @name, @surname, @username)

INSERT INTO producer
VALUES (@email, @company)
```

3.4. Standart User

3.4.1. Account





Retrieve User Profile

```
SELECT username, surname, email, selected_language, birth_date
FROM user NATURAL JOIN audience
WHERE email=@user_email
```

Update Preferred Language

```
UPDATE audience
SET selected_language=@selected_language
WHERE email=@user_email
```

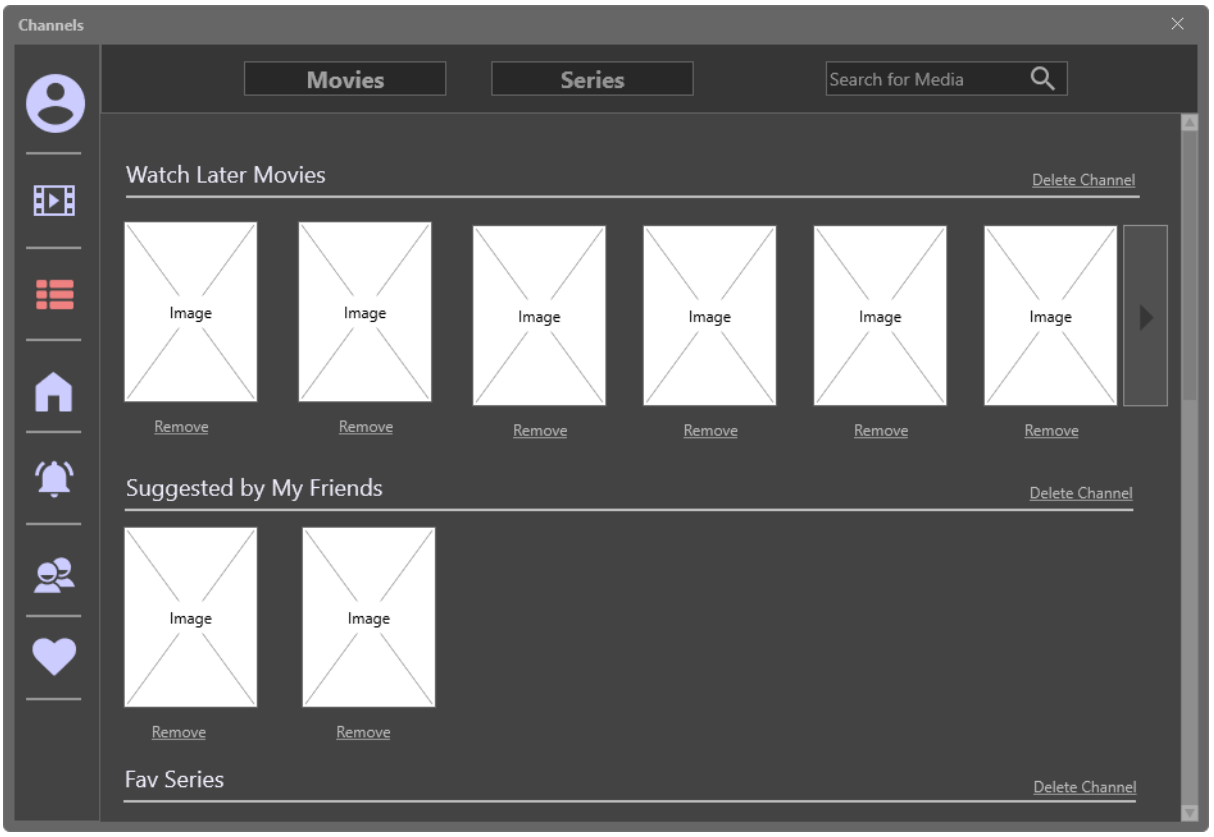
Check Whether Old Password Matches

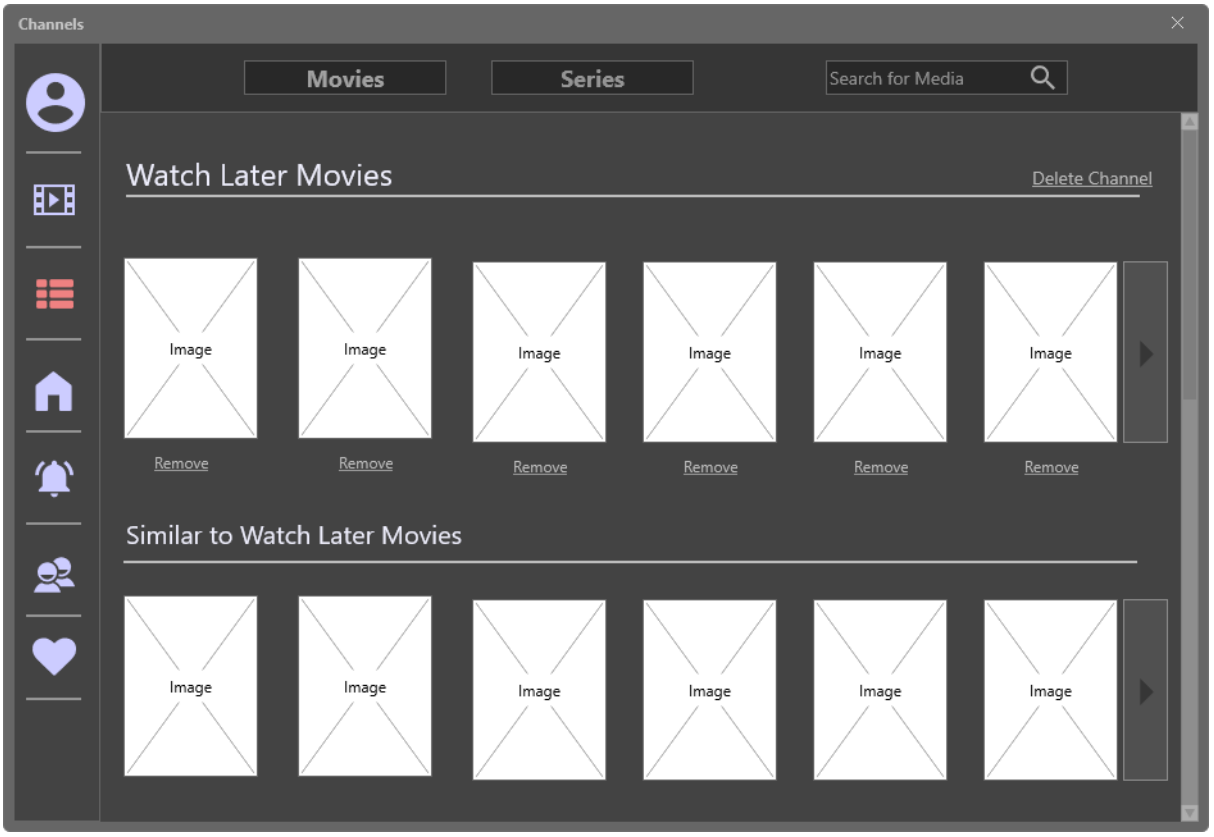
```
SELECT *
FROM user
WHERE password=@password AND email=@user_email
```

Update Password

```
UPDATE user
SET password=@password
WHERE email=@user_email
```

3.4.2. Channels





Retrieve Channels Belonging to User

```
SELECT id,name
FROM channel
WHERE owner_id = @user_email
```

Retrieve Media Belonging to a Channel

```
SELECT *
FROM media JOIN channel_media ON media.id = channel_media.media_id
WHERE channel_id=@channel_id AND owner_id=@owner_id
```

Delete Channel

```
DELETE FROM channel_media
WHERE       channel_id = @channel_id AND owner_id = @owner_id

DELETE FROM channel
WHERE       id = @channel_id AND owner_id = @owner_id
```

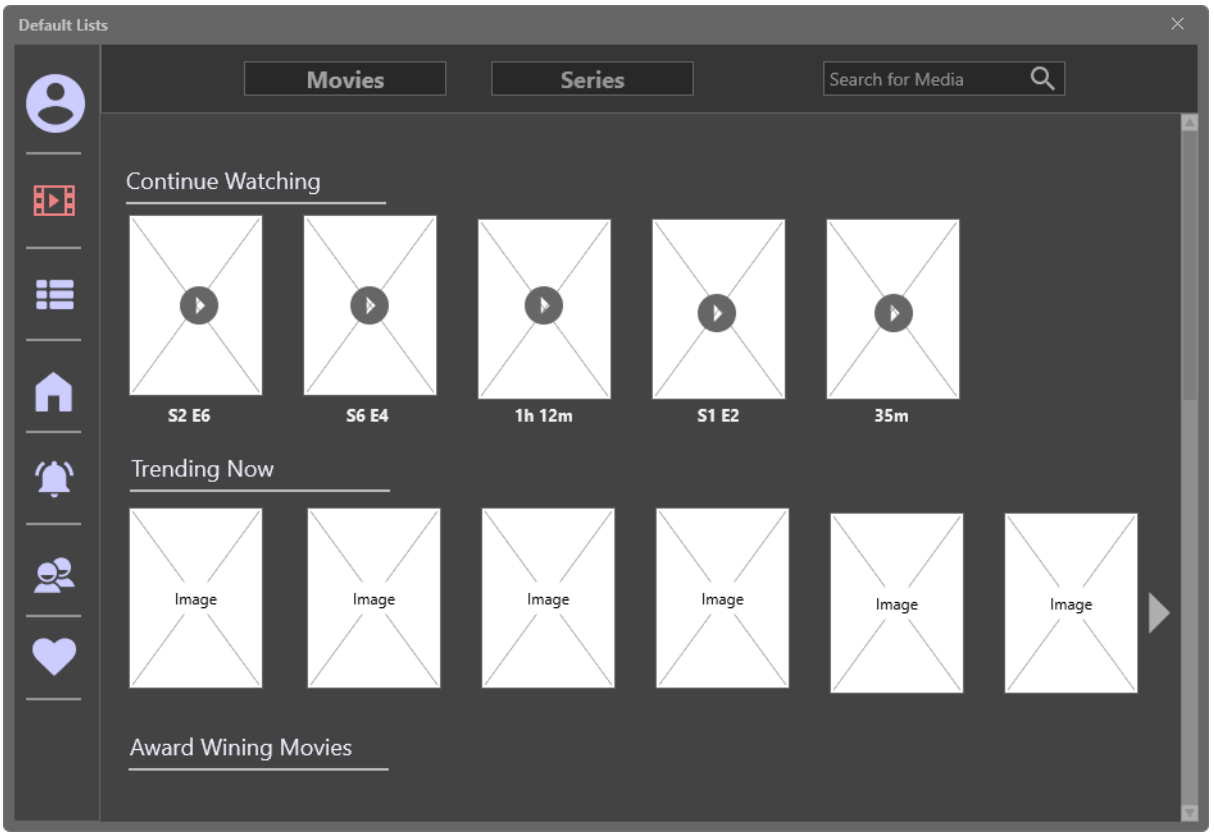
Remove Media From Channel

```
DELETE FROM channel_media
WHERE       channel_id = @channel_id AND
            owner_id   = @owner_id   AND
            media_id   = @media_id
```

Similar to Current Channel Movies

```
SELECT *
FROM media M1 JOIN media_genre G1 ON ( M1.id = G1.media_id )
WHERE G1.genre_name in (SELECT G2.genre_name
                        FROM media_genre G2, channel_media CM
                        WHERE M2.id = G2.media_id AND
                        CM.owner_id = @owner_id AND )
```

3.4.3. Default List



Continue Watching

```
SELECT *
FROM watch_movie
WHERE audience_id=@audience_id
LIMIT 5

SELECT *
FROM watch_episode
WHERE audience_id=@audience_id
LIMIT 5
```

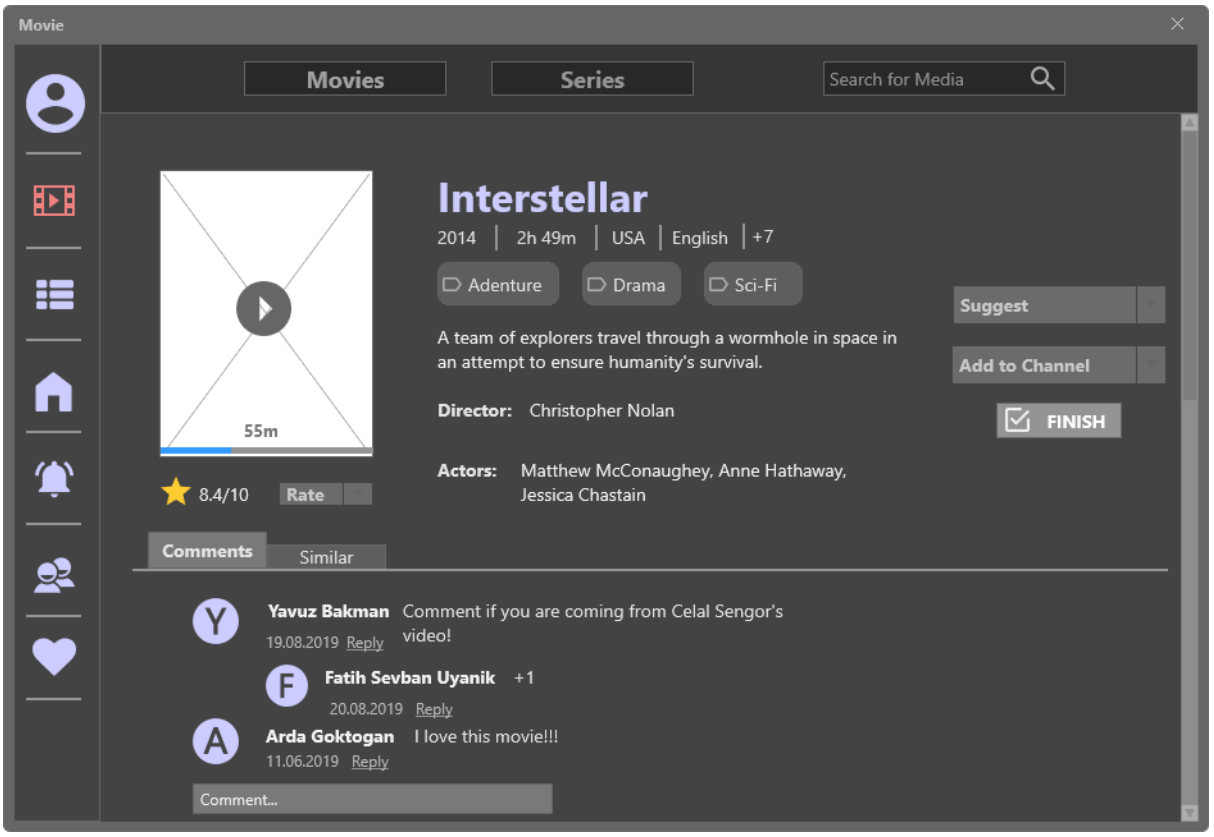
Select Default Lists

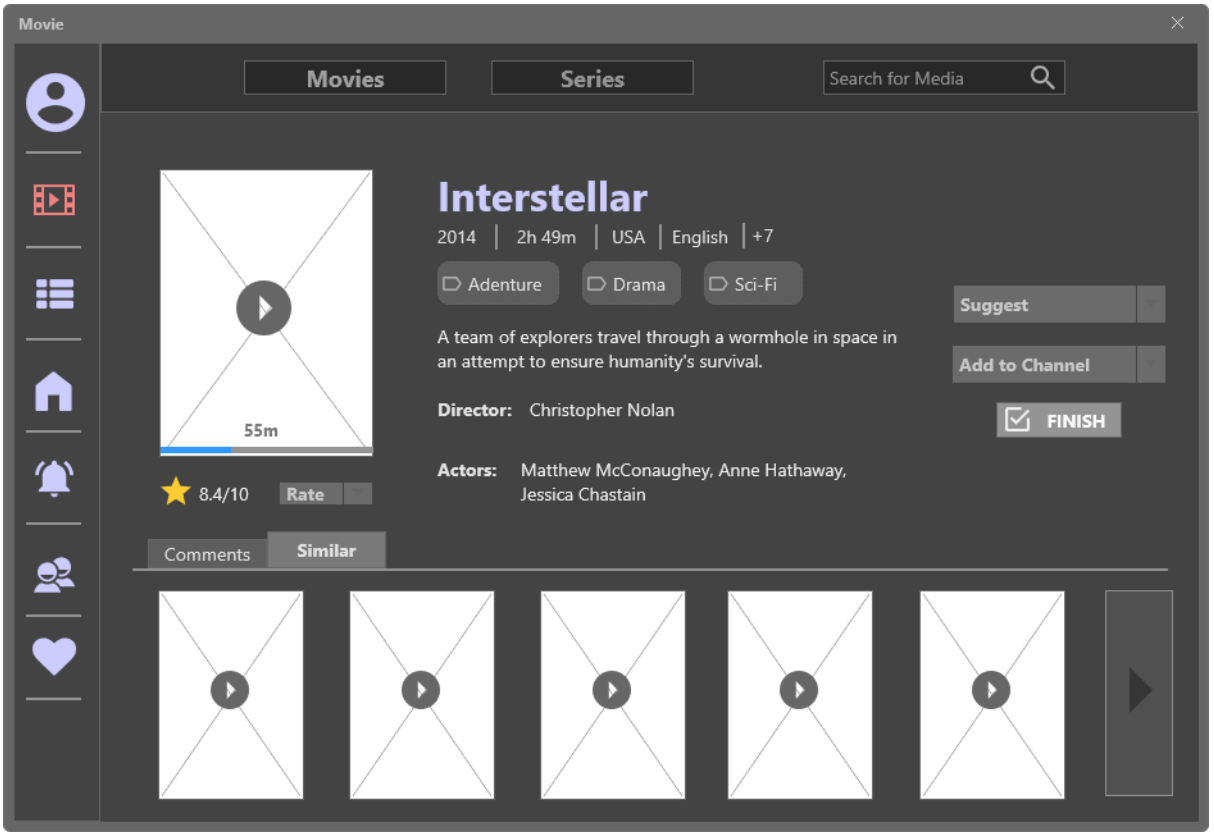
```
SELECT name
FROM default_list
```

Select Default List Content

```
SELECT *
FROM default_list_include JOIN media ON
media.id=default_list_include.media_id
WHERE name = @name
```

3.4.4. Movie Page





Retrieve Movie to be Played

```
SELECT *
FROM movie NATURAL JOIN media
WHERE id=@movie_id
```

Retrieve Actors of Movie

```
SELECT actors
FROM  actors_media
WHERE media_id = @movie_id
```

Retrieve Movie Rating

```
SELECT avg(score) as score
FROM watch_movie
WHERE movie_id = @movie_id
```

Constraint to Movie Rating

```
CREATE CONSTRAINT movie_rating
check (not exist(Select *
                FROM Movie M, watch_episode W,
            WHERE M.id = W.movie_id AND W.score <> NULL
            AND W.is_finished = 0))
```

Finish Movie

```
UPDATE watch_movie
SET    minute=0, is_finished = 1
WHERE  movie_id = @movie_id AND audience_id = @user_id
```

Retrieve Movie Comments

```
SELECT *
FROM (comment JOIN user ON ( commnet.owner_id = user.email) ) join
comment_movie  ON comment_movie.comment_id=comment.id
WHERE movie_id=@movie_id
```

Retrieve Similar Movies and Series

```
WITH  related_genres(genre_name) as (SELECT G1.genre_name
                                     FROM media_genre G1
                                     WHERE G1.media_id = @media_id),
      related_media(id) as (  SELECT G2.media_id as id
                              FROM media_genre G2
                              WHERE G2.genre_name in related_genres)
SELECT * FROM related_media NATURAL JOIN media LIMIT 10
```

Create Movie Comment

```
INSERT INTO comment
VALUES (@comment_id, @time, @text, @parent_id, @owner_id)

INSERT INTO comment_movie
VALUES (@id, @movie_id)
```

If User Has no Record on Movie, Create Watch Time

```
INSERT INTO watch_movie
VALUES(@movie_id, @audience_id, @minute, @watch_date, @score)
```

If User Has Record on Movie, Update it

```
UPDATE watch_movie
SET    minute=@minute, watch_date = @watch_date, score = @score,
is_finished = @is_finished
WHERE  movie_id = @movie_id, audience_id = @auidance_id
```

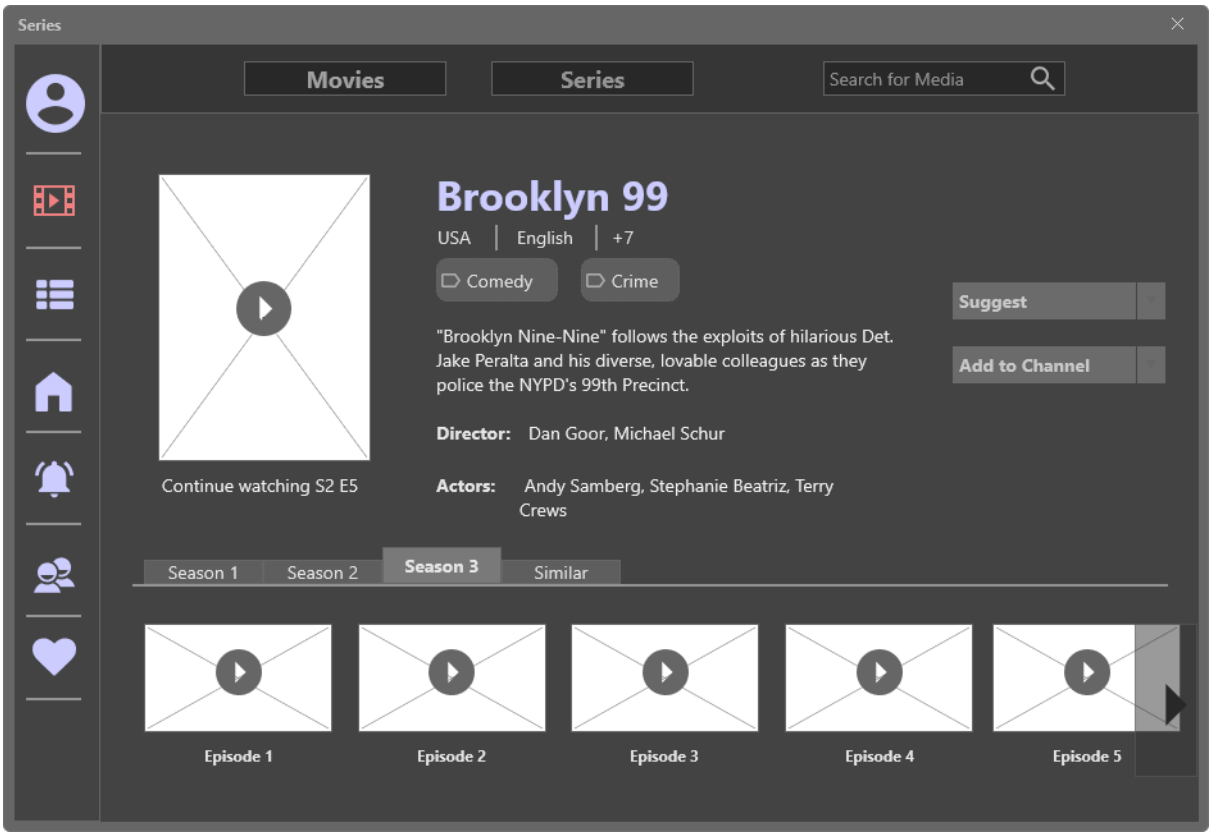
Suggest Movie

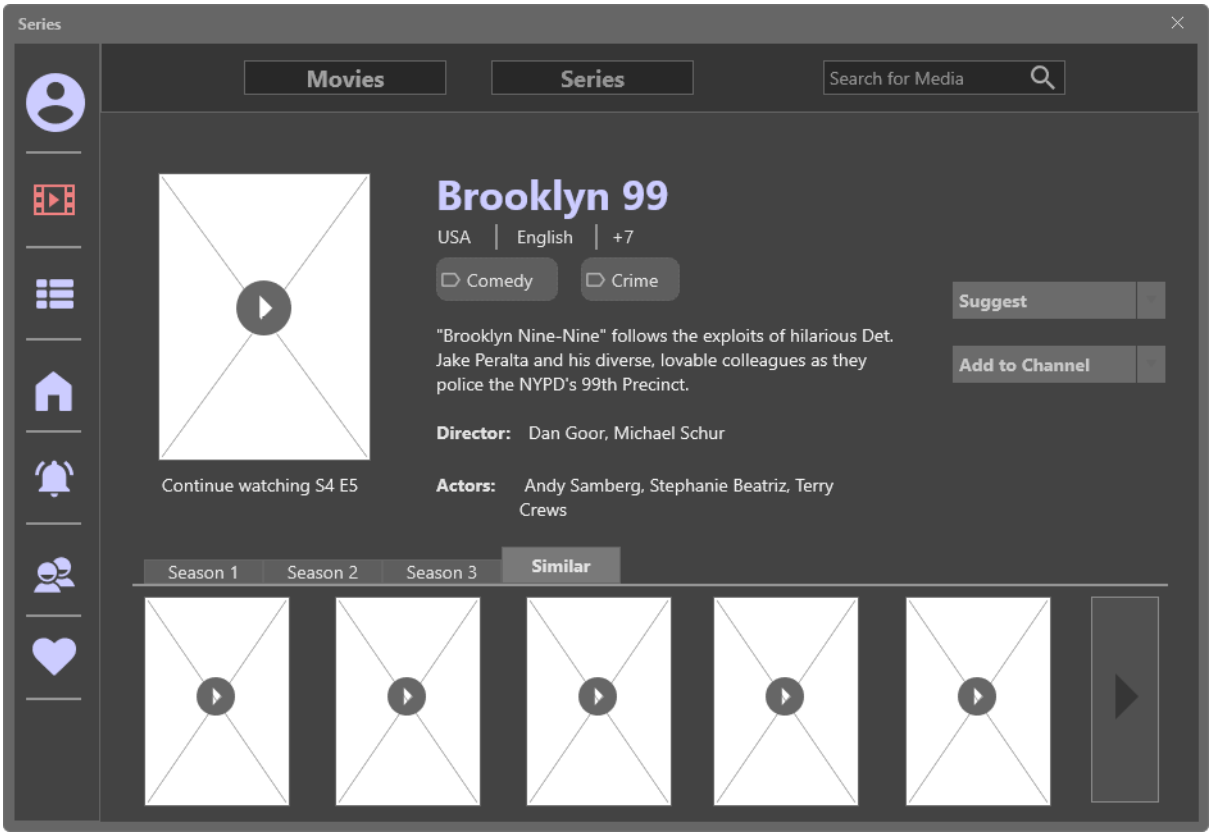
```
INSERT INTO suggestion
VALUES(@id, @media_id, @send_time)

INSERT INTO  friend_suggestion
VALUES(@sender_id, @receiver_id, @suggestion_id)
```

Add Movie to Channel

```
INSERT INTO  channel_media
VALUES(channel_id , owner_id, media_id)
```

3.4.5. Series Page





Retrieve Series

```
SELECT *
FROM series NATURAL JOIN media
WHERE id=@id
```

Retrieve Actors of Series

```
SELECT actors
FROM  actors_media
WHERE media_id = @media_id
```

Suggest Series

```
INSERT INTO suggestion
VALUES(@id, @media_id, @send_time)

INSERT INTO  friend_suggestion
VALUES(@sender_id, @receiver_id, @suggestion_id)
```

Add Series to Channel

```
INSERT INTO  channel_media
VALUES(channel_id , owner_id, media_id)
```

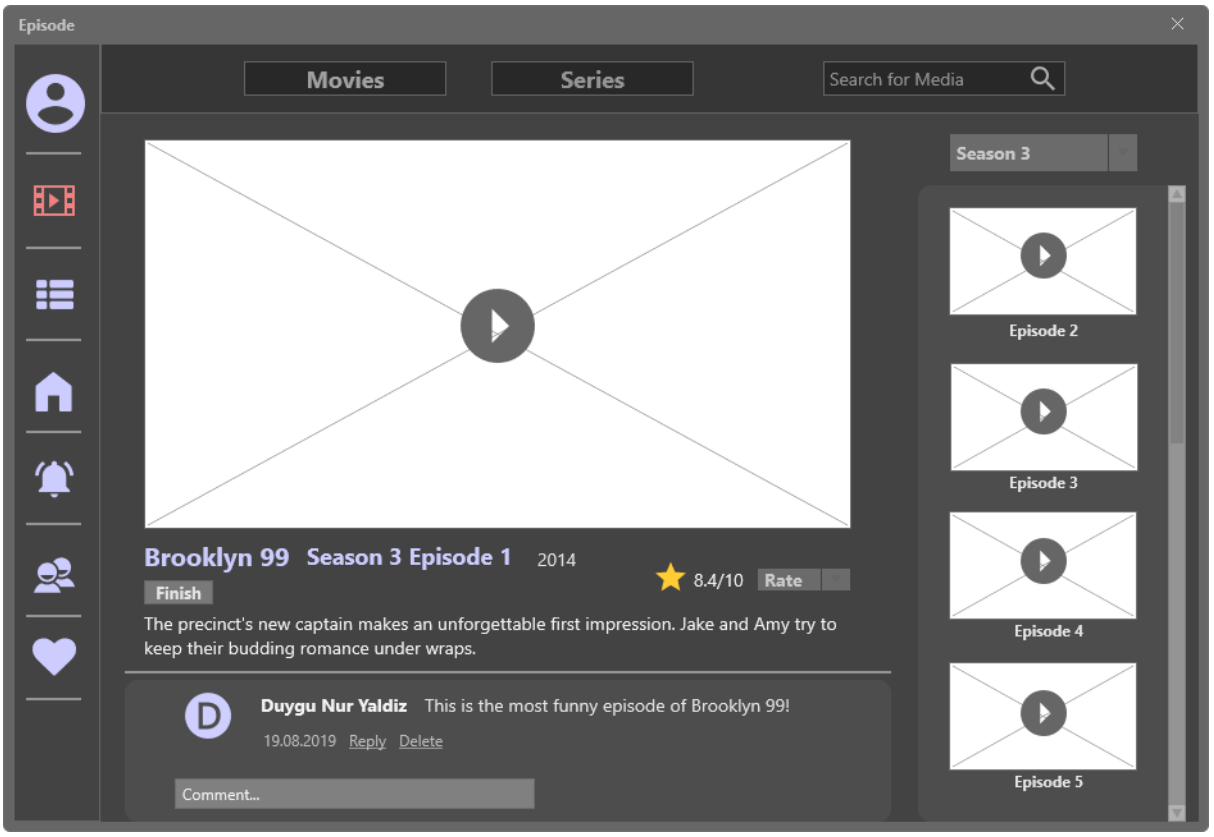
Retrieve Episodes of Particular Season

```
SELECT *
FROM episode
WHERE series_id = @id AND season_no=@season_no
```

Retrieve Similar Movies and Series

```
WITH related_genres(genre_name) as (SELECT G1.genre_name
                                        FROM media_genre G1
                                        WHERE G1.media_id = @media_id),
          related_media(id) as (SELECT G2.media_id as id
                              FROM media_genre G2
                                              WHERE G2.genre_name in
related_genres)
SELECT * FROM related_media NATURAL JOIN media LIMIT 10
```

3.4.6. Episode Page



Retrieve Episode to be Played

```
SELECT * FROM episode
WHERE series_id = @series_id AND
      season_no = @season_no AND
      episode_no= @episode_no
```

Retrieve Episode Rating

```
SELECT avg(score) as score
FROM watch_episode
WHERE series_id = @series_id AND
      season_no = @season_no AND
      episode_no= @episode_no
```

Constraint to Episode Rating

```
CREATE CONSTRAINT episode_rating
check (not exist(Select *
                FROM episode E, watch_episode W,
            WHERE E.series_id = W.series_id AND E.season_no = W.season_no
            AND E.episode_no = W.episode_no AND W.score <> NULL AND
            W.is_finished = 0))
```

Finish Episode

```
UPDATE watch_episode
SET    minute=0, is_finished = 1
WHERE  series_id = @series_id  AND
       season_no = @season_no  AND
       episode_no= @episode_no AND audience_id = @user_id
```

Retrieve Episode Comments

```
SELECT * FROM (comment JOIN user ON ( commnet.owner_id = user.email) )
             JOIN comment_episode ON comment_episode.comment_id=comment.id
WHERE series_id = @series_id AND
      season_no = @season_no AND
      episode_no= @episode_no
```

Retrieve Related Season Episodes

```
SELECT * FROM episode WHERE series_id = @series_id AND
season_no=@season_no AND episode_no <> @episode_no
```

Create Episode Comment

```
INSERT INTO comment VALUES (@comment_id, @time, @text, @parent_id,
@owner_id)
INSERT INTO comment_episode VALUES (@id, @series_id, @season_no,
@episode_no)
```

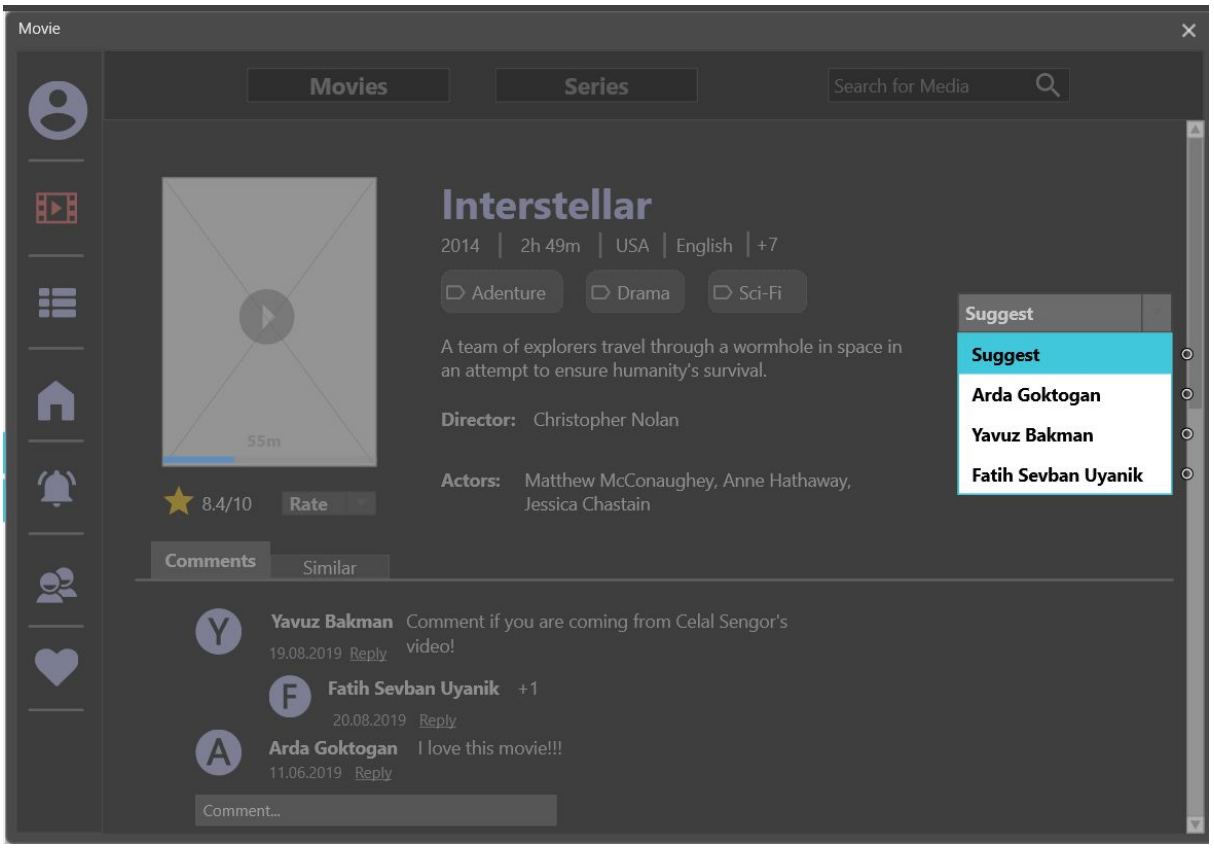
If User Has no Record on Episode, Create Watch Time

```
INSERT INTO watch_episode
VALUES(@series_id, @season_no,
       @episode_no, @audience_id,
       @minute, @watch_date, @score)
```

If User Has Record on Episode, Update it

```
UPDATE watch_episode
SET    minute=@minute, watch_date = @watch_date, score = @score,
is_finished = @is_finished
WHERE  series_id  = @series_id  AND
       season_no  = @season_no  AND
       episode_no = @episode_no AND
       audience_id= @auidience_id
```

3.4.7. Media Suggestion



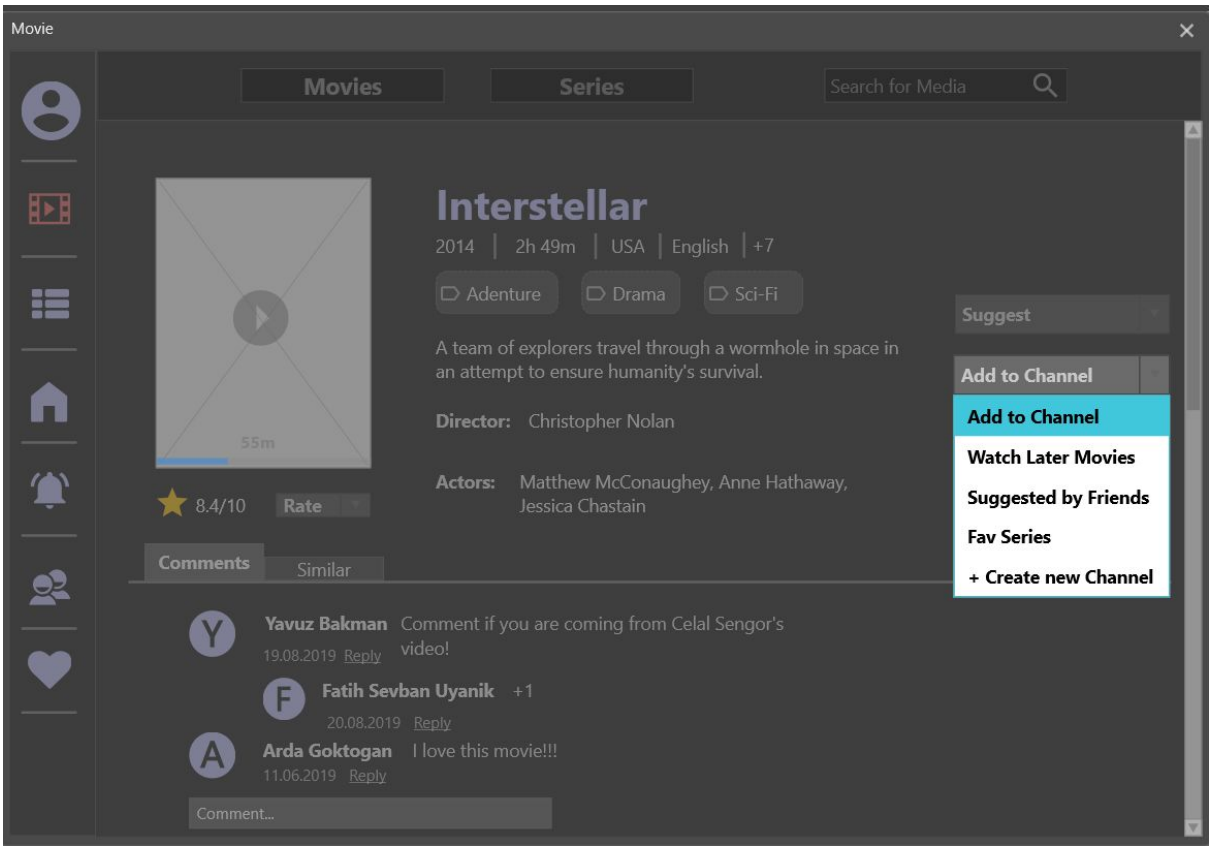
List of Friends for Suggestion

```
SELECT      name,surname,email
FROM        ( audience NATURAL JOIN user ) JOIN
            friendship ON (frienship.user_id = email)
WHERE       friendship.user_id = @auidience_id
```

Suggest Media

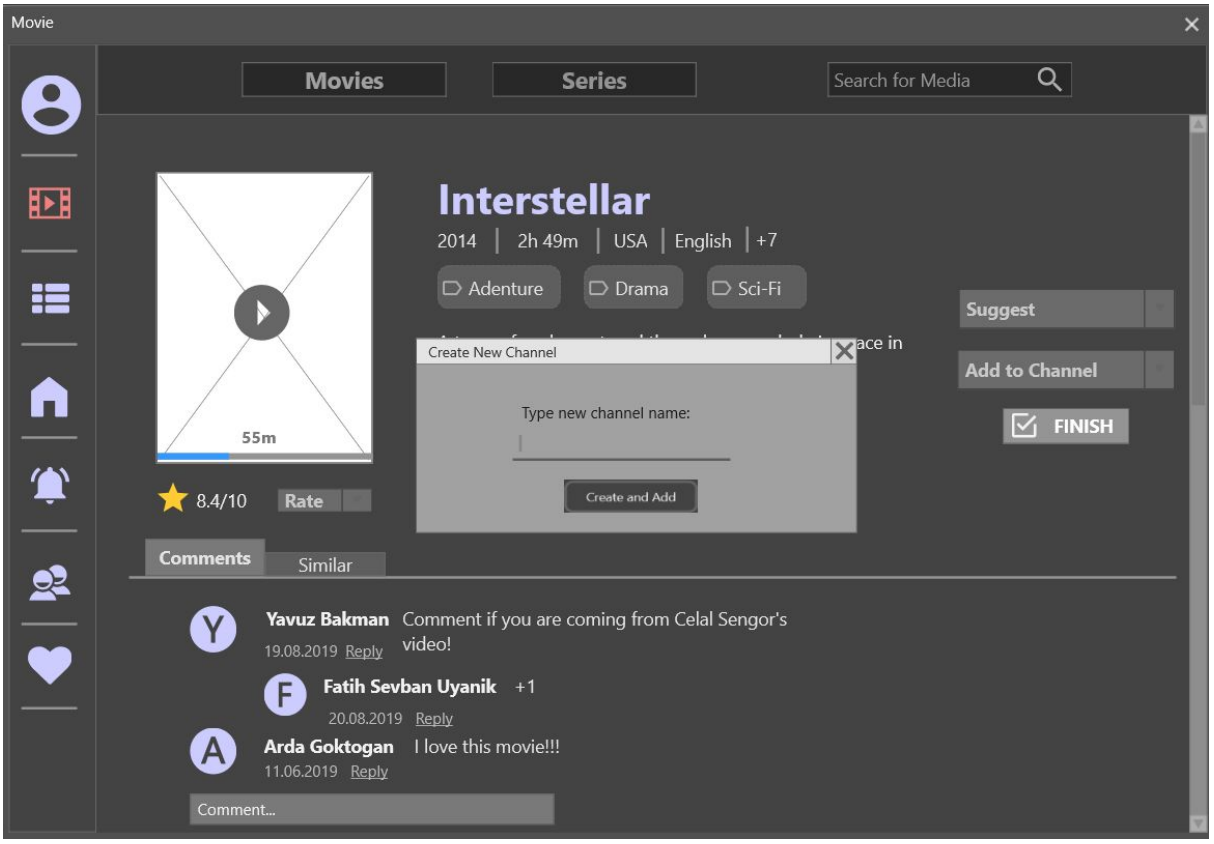
```
INSERT INTO suggestion
VALUES(@id, @media_id, @send_time)

INSERT INTO  friend_suggestion
VALUES(@sender_id, @receiver_id, @suggestion_id)
```

3.4.8. Add Media to Channel



<u>Select List of Channels Belonging to User</u>

```
SELECT      id,name
FROM        channel
WHERE       owner_id = @user_email
```



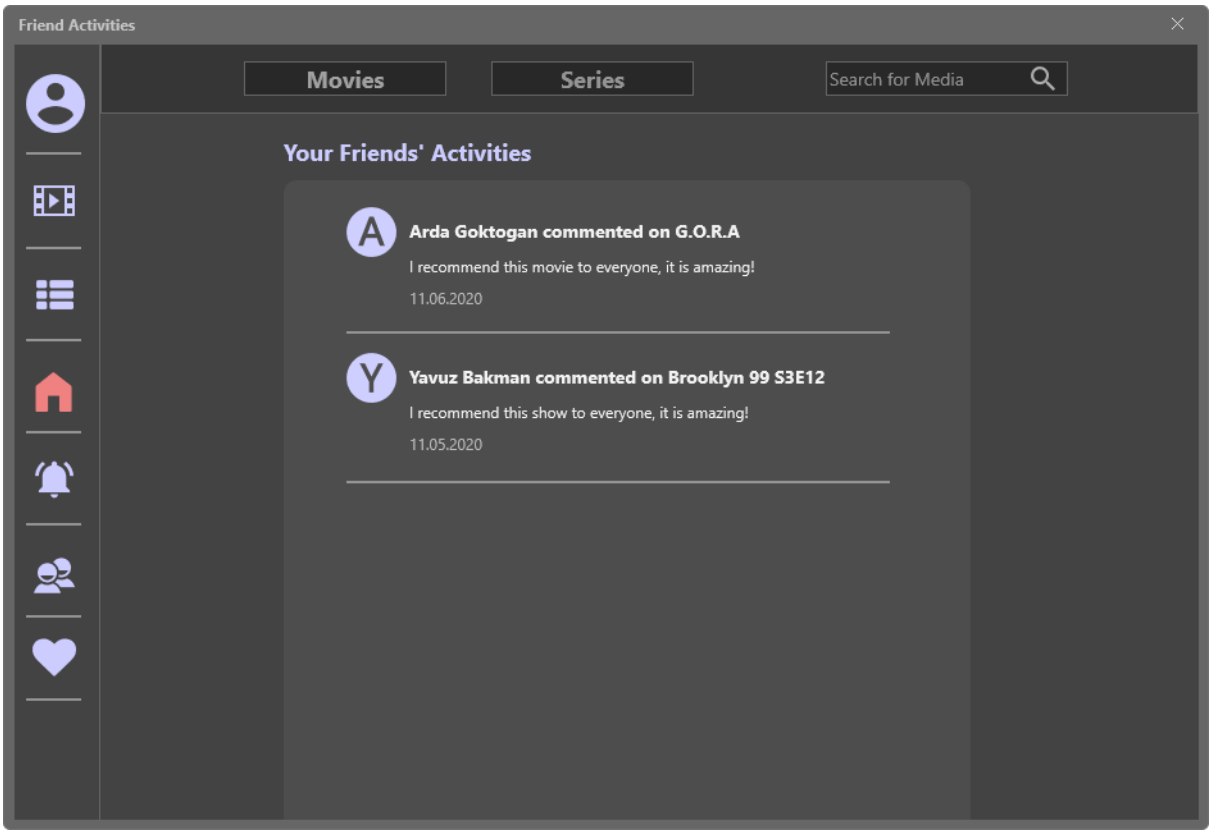
Create Channel

```
INSERT INTO channel
VALUES (@id , @owner_id, @name, @creation_time)
```

Add Media to Channel

```
INSERT INTO  channel_media
VALUES(channel_id , owner_id, media_id)
```

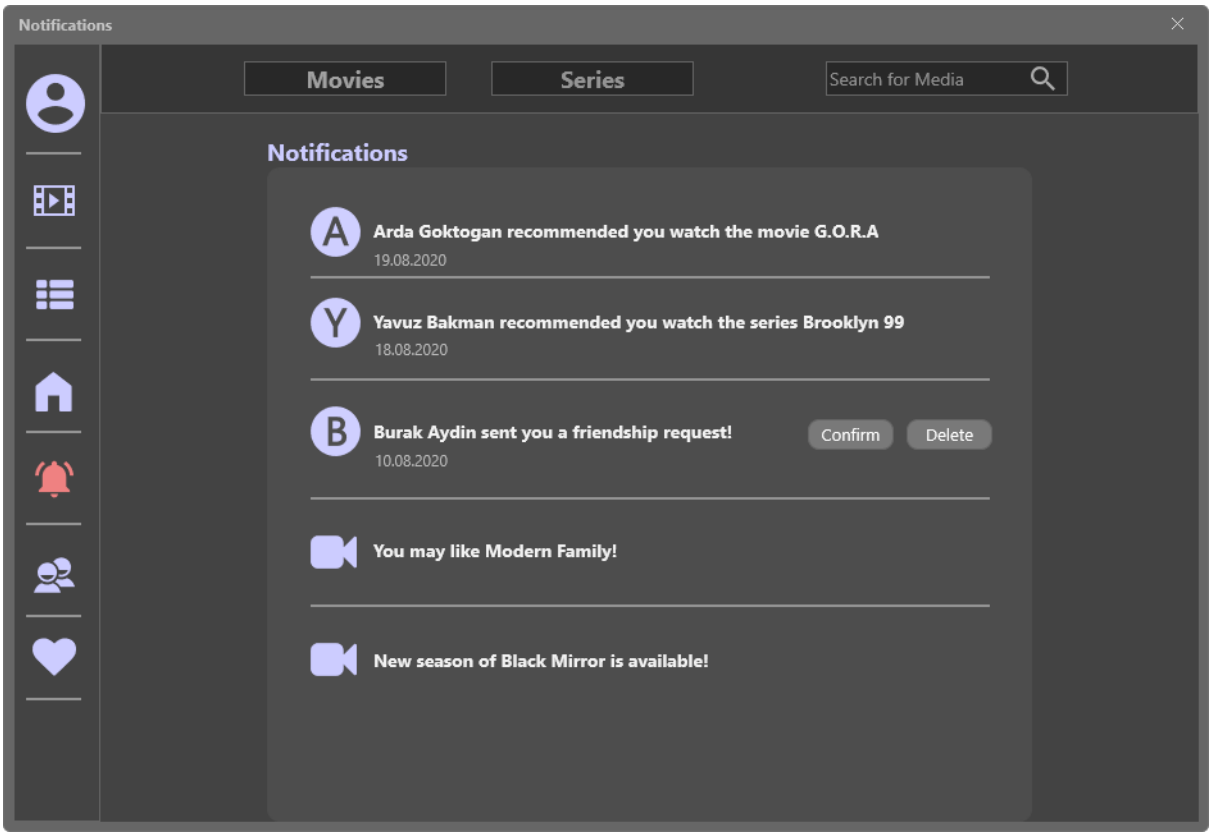
3.4.9. Friend Activities



Retrieve User Activities

```
SELECT *
FROM  comment JOIN audience ON (comment.owner_id = audience.email)
      LEFT OUTER JOIN comment_evaluate
ON comment.owner_id = comment_evaluate.audience_id
WHERE owner_id=@user_email
ORDER BY time
```

3.4.10. Notifications



Retrieve Notifications

```
SELECT *
FROM suggestion LEFT OUTER JOIN friend_suggestion
ON suggestion.id = friend_suggestion.suggestion_id
WHERE receiver_id=@receiver_id
ORDER BY time

SELECT *
FROM friend_request
WHERE receiver_id=@receiver_id
ORDER BY time

SELECT *
FROM suggestion
WHERE  id NOT IN (SELECT id as suggestion_id  FROM friend_suggestion)
```

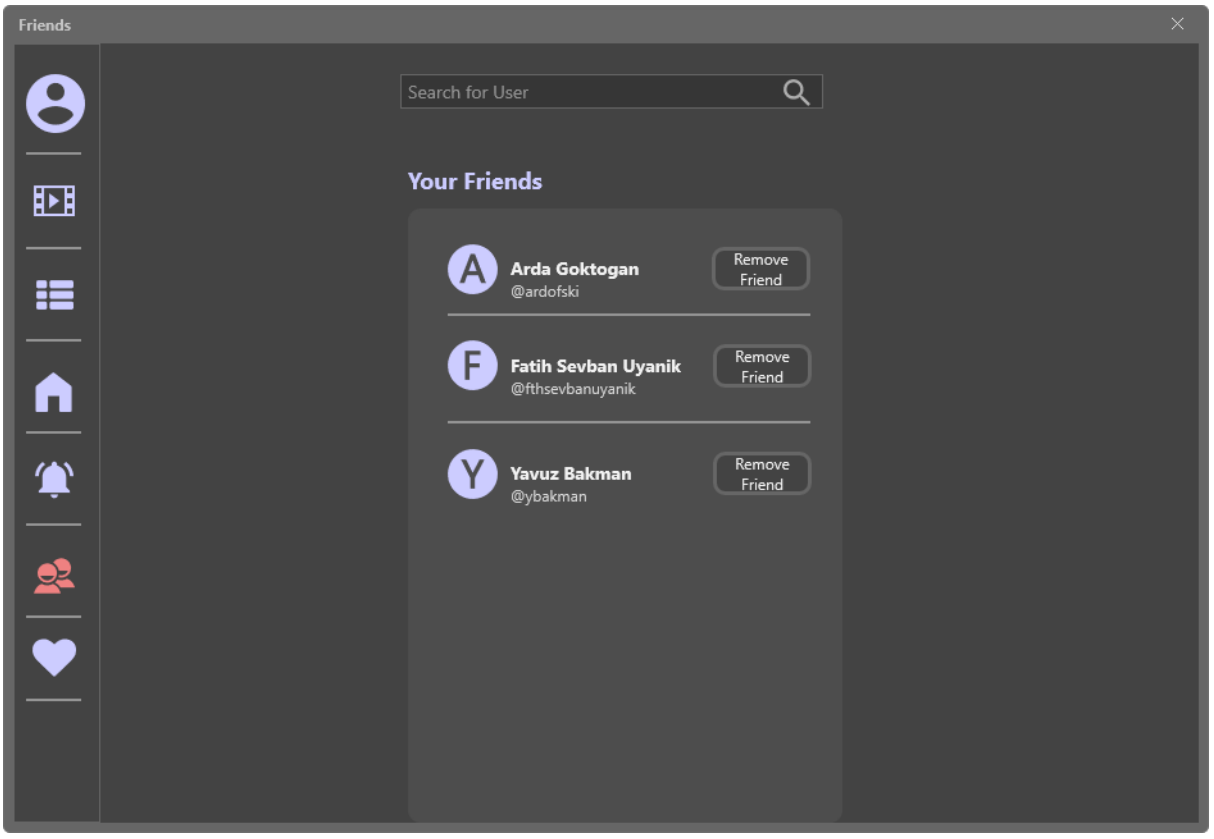
Accept Friend Request

```
INSERT INTO friendship
VALUES (@reciever_id, @requester_id )
```

Delete Friend Request

```
DELETE FROM friend_request
WHERE sender_id = @requester_id AND reviever_id = @reciever_id
```

3.4.11. Friends



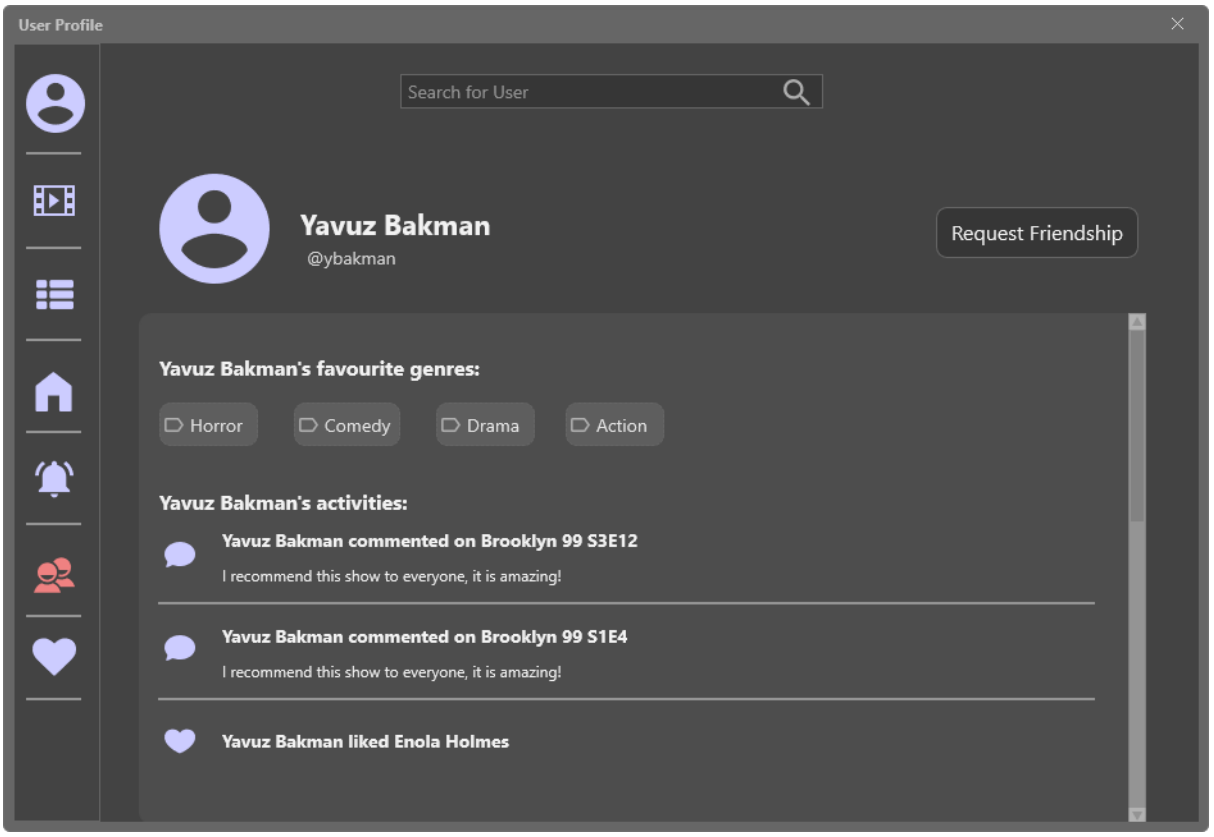
Retrieve Friends

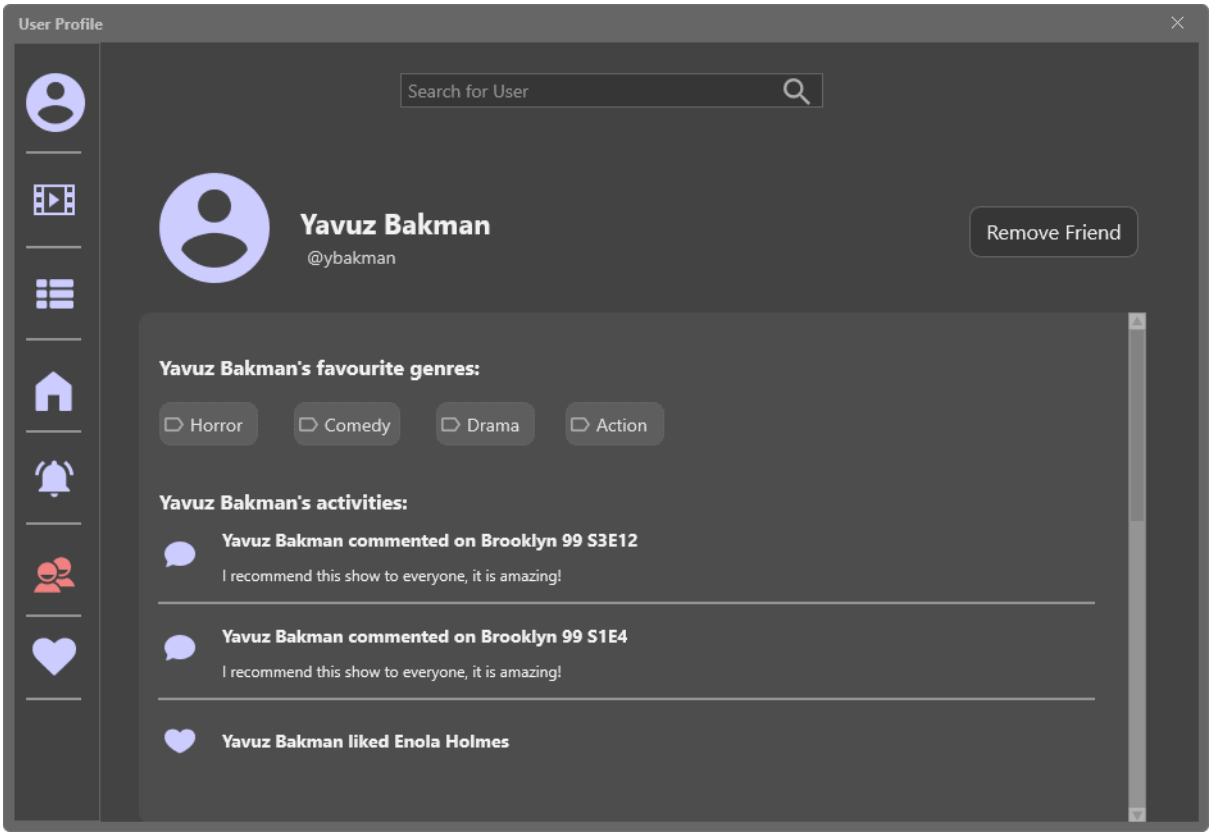
```
WITH related_friends(email) AS (SELECT email
                                FROM friendship JOIN audience
                                ON friendship.friend_id =
                audience.email
                                WHERE user_id = @user_id)
SELECT * FROM user NATURAL JOIN related_friends
```

Remove Friend

```
DELETE FROM friendship
WHERE user_id = @user_id AND friend_id = @friend_id
```

3.4.12. User Page





Retrieve User Profile

```
SELECT username, surname, email
FROM user WHERE email=@user_email
```

Retrieve User Activities

```
SELECT *
FROM comment LEFT OUTER JOIN comment_evaluate
ON comment.owner_id = comment_evaluate.audience_id
WHERE owner_id=@user_email
ORDER BY time
```

Retrieve Favourite Genres

```
SELECT genre_name
FROM favor_genre
WHERE audience_id=@user_email
LIMIT 5
```

Check If User is Friend

```
SELECT *
FROM friendship
WHERE user_id = @account_id AND friend_id = @user_email
```

Request Friendship

```
INSERT INTO friend_request
VALUES (sender_id=@sender_id, receiver_id=@receiver_id,
send_time=@send_time)
```

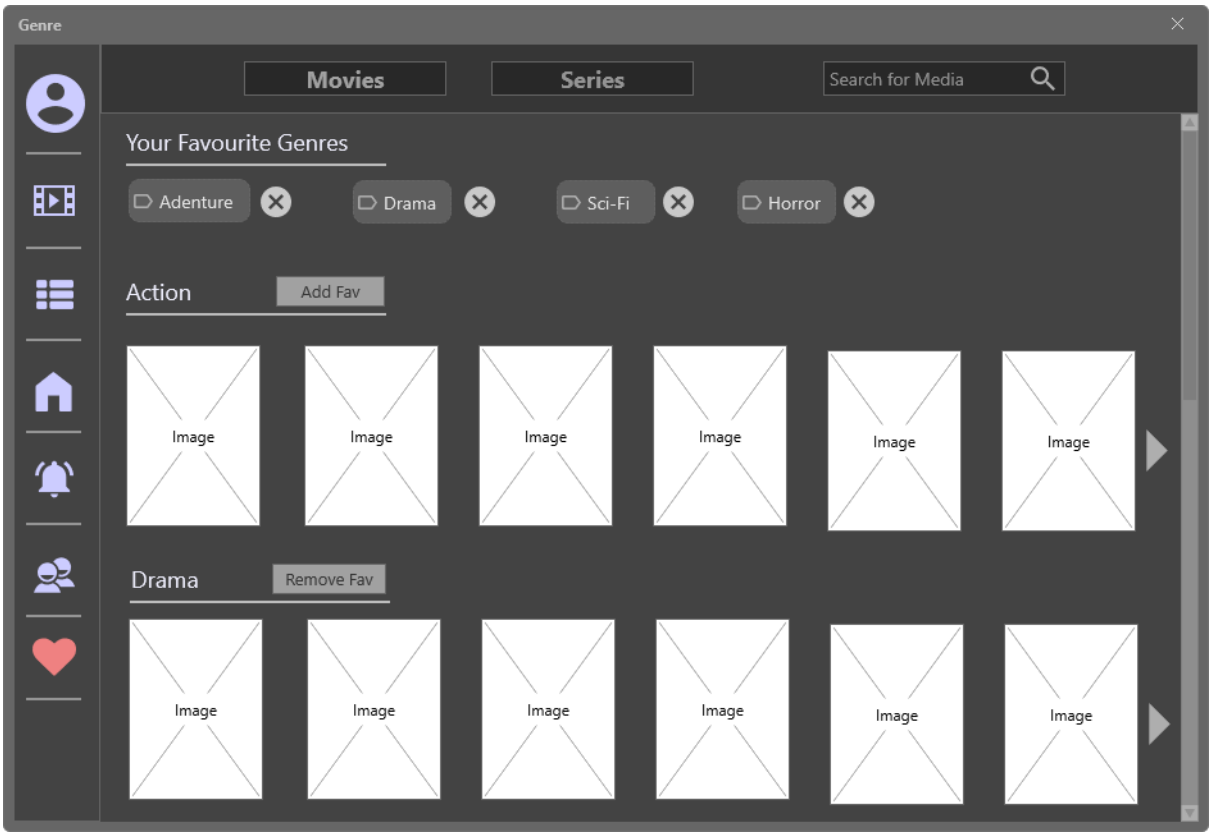
Friendship AND Request Trigger

```
create trigger friendship_request
after insert on Friendship
referencing new row as new_row

for each row
DELETE FROM friend_request
WHERE new_row.user_id = sender_id OR new_row.user_id = receiver_id or
      new_row.friend_id = sender_id OR
      new_row.friend_id = receiver_id
```

Remove Friendship

```
DELETE FROM friendship
WHERE (user_id=@user_id AND friend_id=@friend_id) OR
         (user_id=@friend_id AND friend_id=@user_id)
```

3.4.13. Genres



Retrieve Favourite Genres

```
SELECT genre_name
FROM favor_genre
WHERE audience_id=@user_email
LIMIT 5
```

Delete Favourite Genre

```
DELETE FROM favor_genre
WHERE audience_id=@user_email , genre_name=@genre_name
```

Retrieve Genre Names

```
SELECT name
FROM genre
```

Favour Genre

```
INSERT INTO favor_genre
VALUES (@user_email,@genre_name)
```

Unfavour Genre

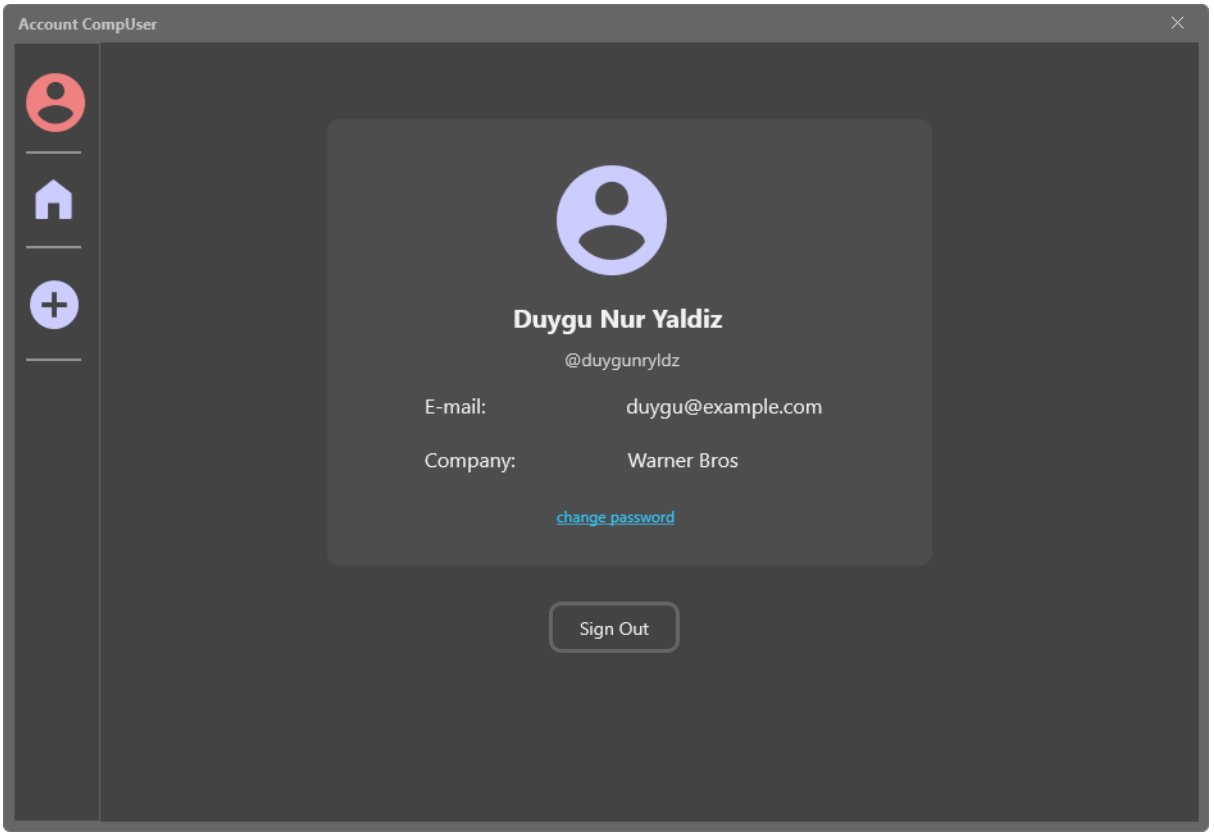
```
DELETE FROM favor_genre
WHERE audience_id = @user_email AND genre_name = @genre_name
```

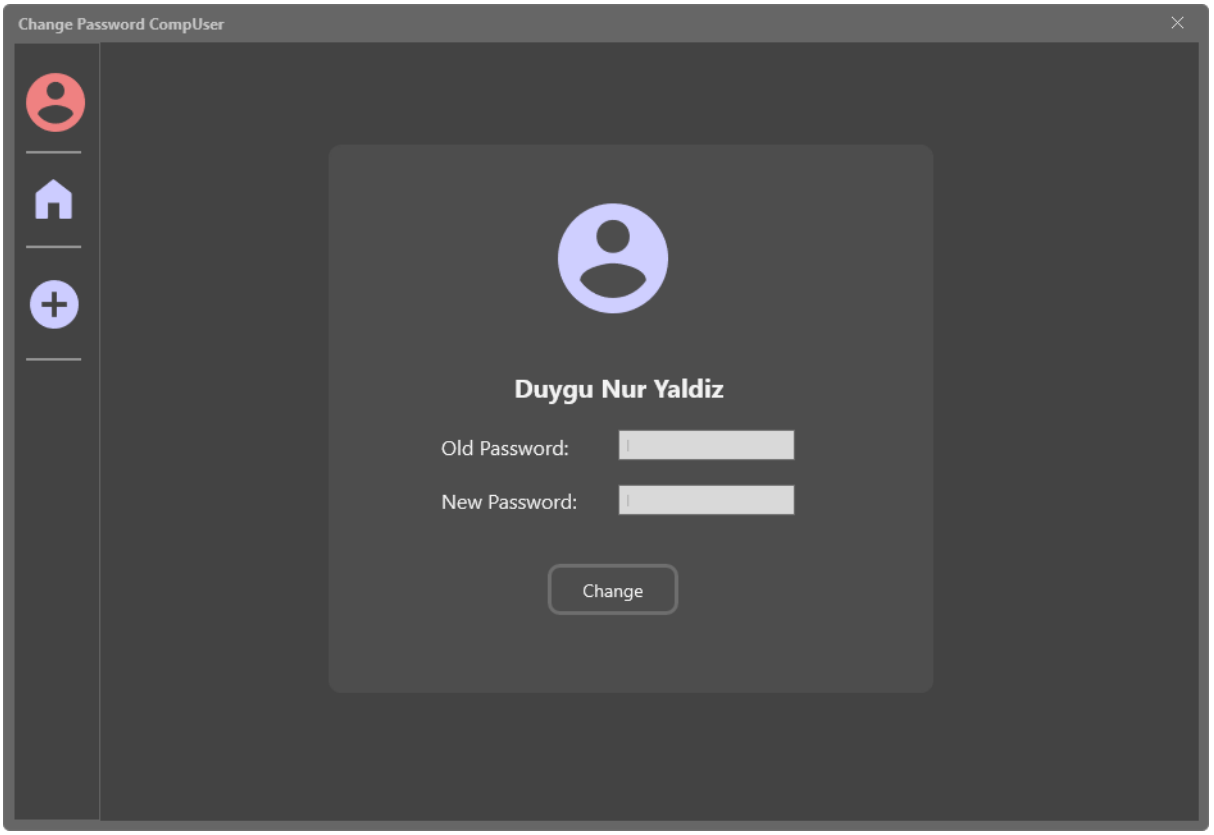
Retrieve Media with Specified Genre

```
SELECT *
FROM media JOIN media_genre ON media.id = media_genre.media_id
WHERE genre_name=@genre_name
```

3.5. Company Member

3.5.1. Account





Retrieve Company User Profile

```
SELECT username, surname, email, company
FROM user NATURAL JOIN producer
WHERE email=@user_email
```

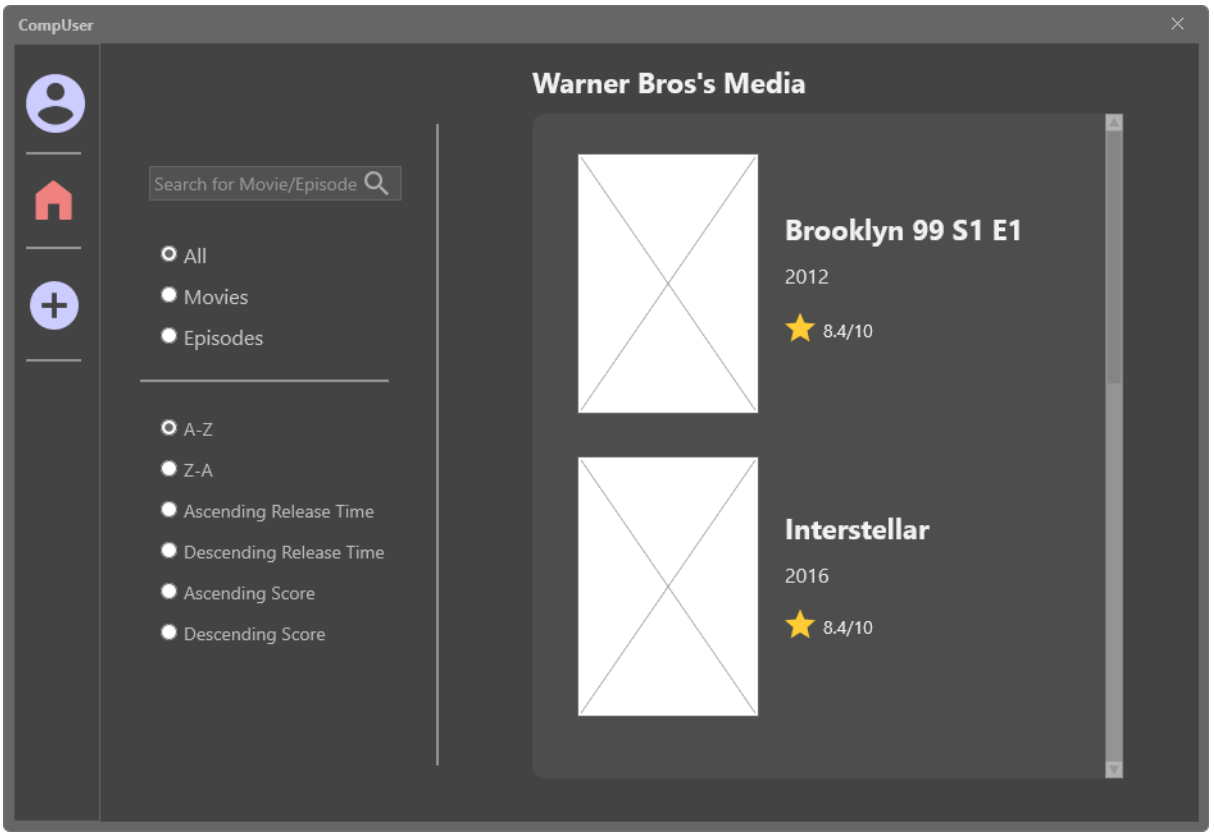
Check Whether Old Password Matches

```
SELECT *
FROM user
WHERE password=@password AND email=@email
```

Update Password

```
UPDATE user
SET password=@password
WHERE email=@email
```

3.5.2. List Media



Get Company

```
SELECT company
FROM   producer
WHERE  email = @producer
```

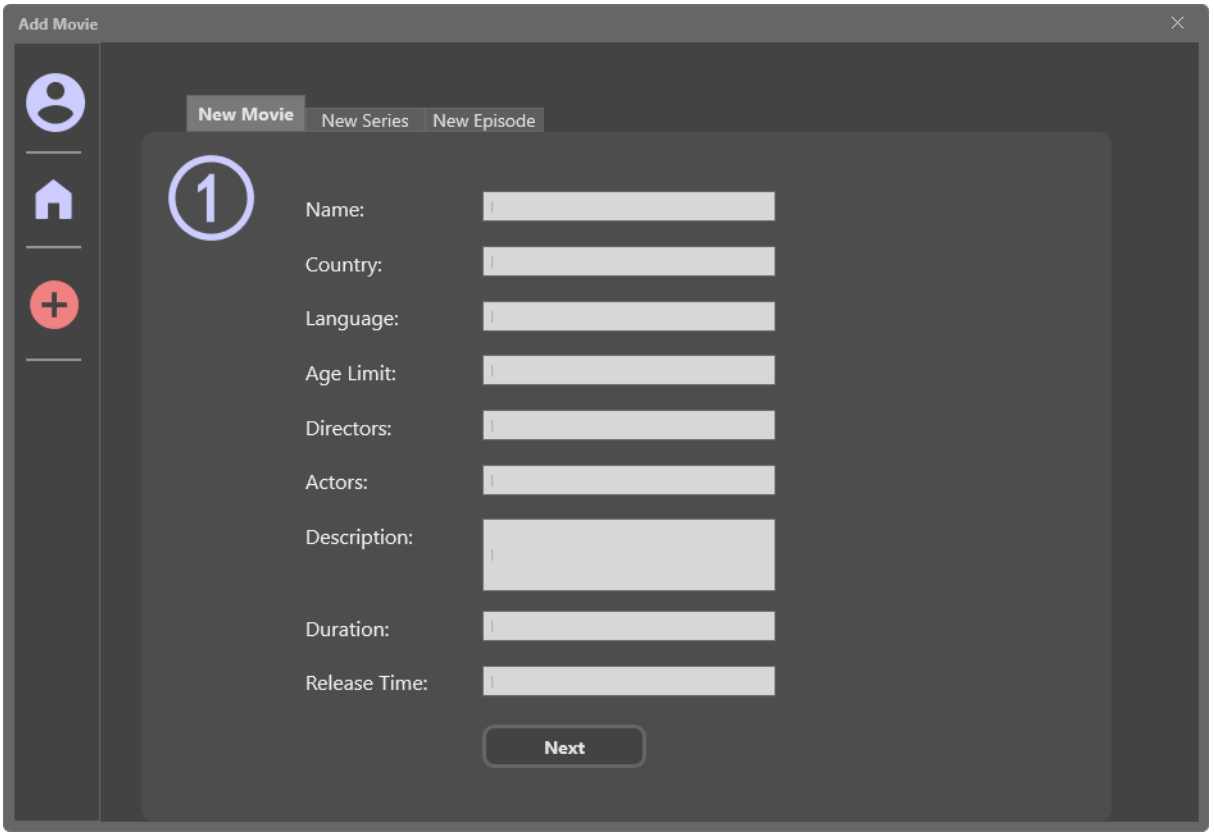
Retrieve Movies by <order_specification>

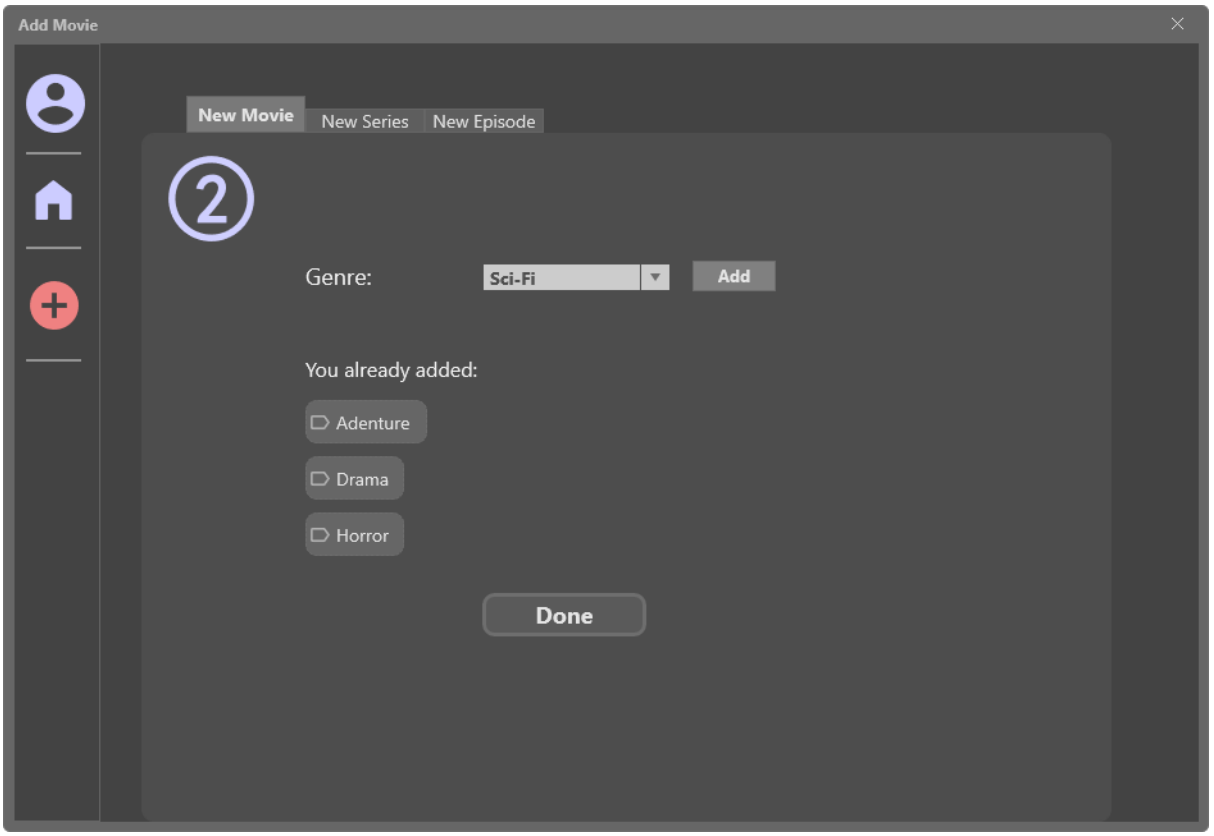
```
SELECT      M1.name, M1.release_time, temp.avg_score
FROM        media M1,
      (SELECT     M2.id, avg(score) as avg_score
      FROM        movie M2, watch_movie W1
      WHERE       M2.id = W1.movie_id
      GROUP BY    M2.id) as temp(movie_id, avg_score)
WHERE       M1.id = temp.movie_id AND
            M1.id in (  SELECT O.media_id
                        FROM media_owner O, Producer P
                        WHERE O.producer_id = P.email AND
                        P.company = @Producer.company )
ORDER BY <order_specification>
```

Retrieve Episodes by <order_specification>

```
SELECT S.name, E1.season_no,
E1.episode_no, E1.release_time, temp.avg_score
FROM  episode E1, series S,
(SELECT E2.series_id, E2.season_no, E2.episode_no, avg(score) as avg_score
FROM Episode E2, watch_episode W1
WHERE E2.series_id = W1.series_id AND E2.season_no = W1.season_no AND
E2.episode_no = W1.episode_no
GROUP BY E2.series_id, E2.season_no, E2.episode_no) as temp(series_id,
season_no, episode_no)
WHERE E1.series_id = S.series_id AND E1.series_id = temp.series_id AND
E1.season_no = temp.season_no AND E1.episode_no = temp.episode_no AND
E1.series_id in ( SELECT O.media_id
                        FROM media_owner O, Producer P
                        WHERE O.producer_id = P.email AND
                        P.company = @Producer.company )
ORDER BY <order_specification>
```

3.5.3. Add Movie





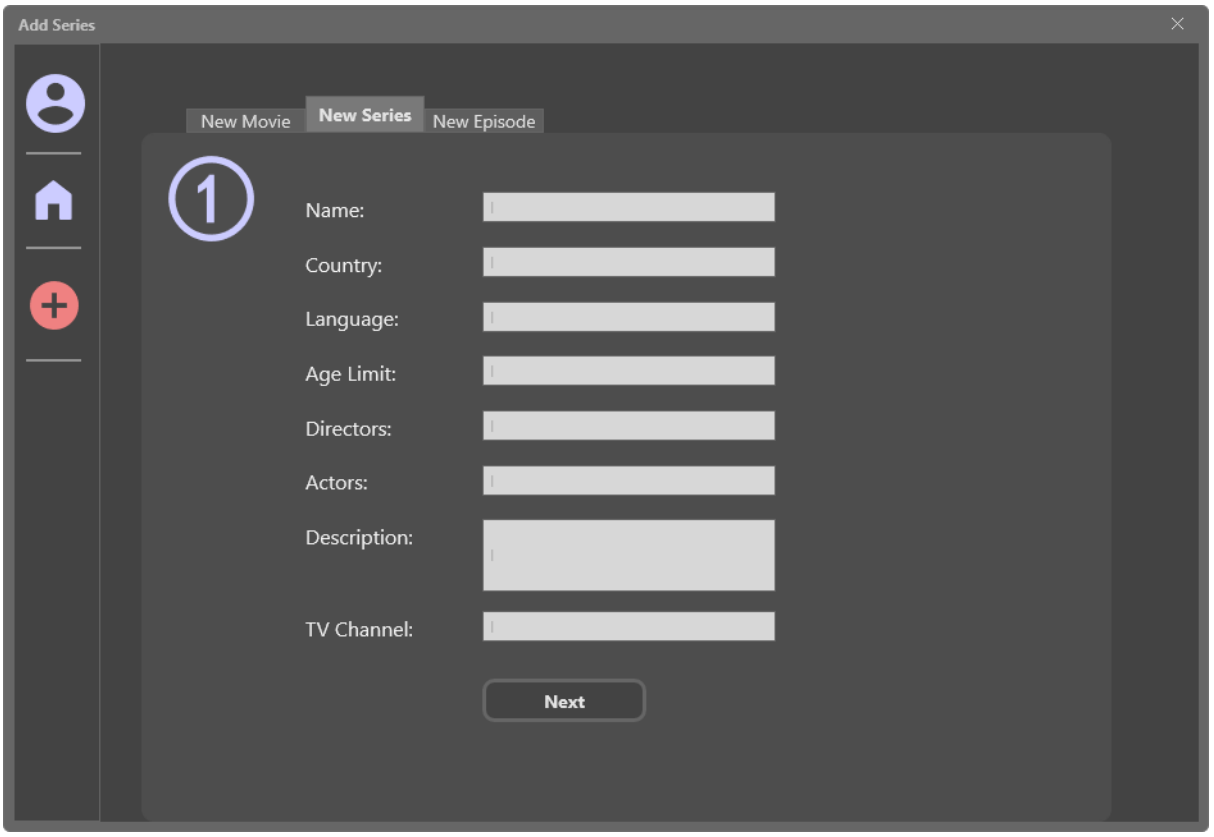
Select Genre Options

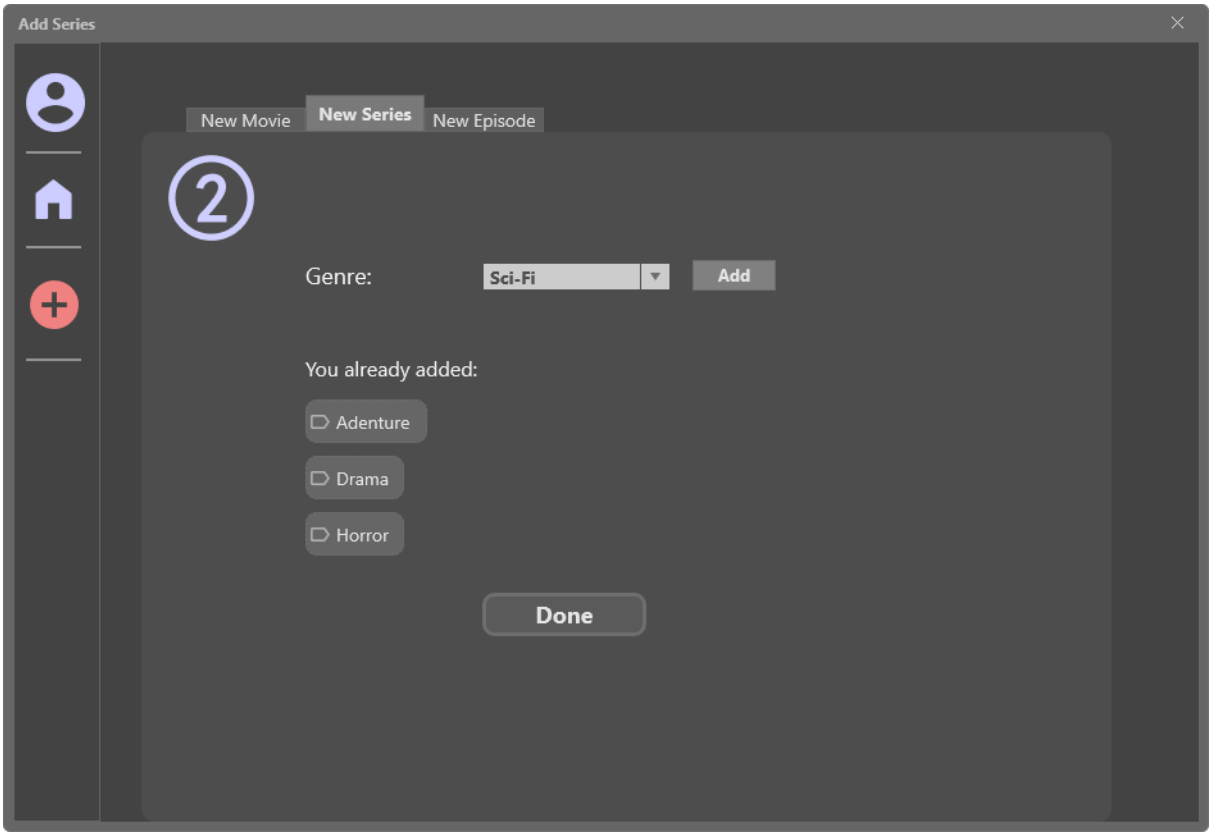
```
SELECT name
FROM genre
```

Create Movie

```
INSERT INTO media
VALUES (@id, @name, @age_limit, @country, @original_language,
@description,  @actors, @director)

INSERT INTO movie VALUES (@id, @duration, @release_time)

INSERT INTO media_genre VALUES (media_id , genre_name)

INSERT INTO media_owner VALUES (media_id ,@producer.email)
```

3.5.4. Add Series





Select Genre Options

```
SELECT name
FROM genre
```

Create Series

INSERT INTO media
VALUES (@id, @name, @age_limit, @country, @original_language, @description, @actors, @director)

INSERT INTO series VALUES (@tv_channel)

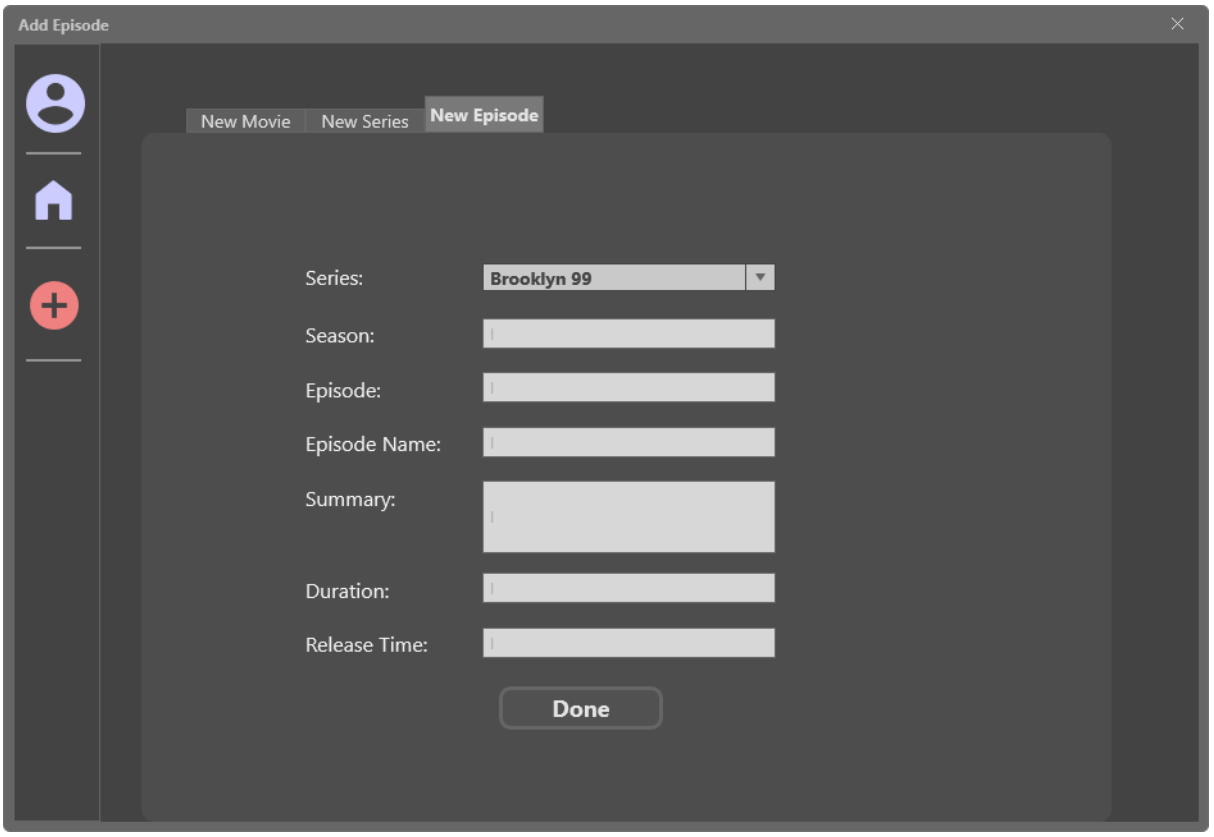
INSERT INTO media_genre VALUES (media_id , genre_name)

```
INSERT INTO media_owner VALUES (media_id ,@producer.email)
```

Series Episode Time Assertion

```
CREATE ASSERTION episode_order
check ( not exist (SELECT *
            from episode E, medio_owner M1, episode E2, media_owner M2,
            WHERE E1.series_id = M1.media_id and E2.series_id =
            M2.series_id and E1.series_id = E2.series_id and M1.time <
            M2.time and ((E1.season_no > E2.season_no) or ((E1.season_no =
            E2.season_no) and (E1.episode_no > E2.episode_no)) ) ))
```

3.5.5. Add Episode



Select Series Belonging To Producer's Company

```
SELECT      M.id, M.name
FROM        series S, producer P1, producer P2,media_owner O, media M
WHERE       S.id = M.id                AND
            P1.company = P2.company    AND
            O.media_id = S.id          AND
            O.producer_id = P2.email   AND
            P1.email = @producer_email
```

Create Episode

```
INSERT INTO episode
VALUES (@series_id, @season_no, @episode_no, @episode_name, @summary,
@duration, @release_time)
```

4. Implementation Plan

We will be use Node.js for the backend and MySQL as the database. For the user interface, we will use Vue JS and Javascript.

5. Website

Reports can be found at : https://ybakman.github.io/Lava