



CS 353 Database Systems

Project Design Report

Media Services Data Management System

Group 8

Ahmet Emre Zengin | 21400527

Ali Kemal Özkan | 21302087

Osman Can Yıldız | 21302616

Hasan Doğan | 21402109

23rd November, 2020

Project Website

<https://aliekemalozkan.github.io/MediaOnline/>

Table of Contents

E-R Diagram	3
Table Schemas	4
2.1 User	4
2.2 Episode	4
2.3 Series	4
2.4 Movie	5
2.5 Genre	5
2.6 Comment	5
2.7 Channel	6
2.8 Message	6
2.9 Wishlist	6
2.10 Watch series	6
2.11 Watch movie	7
2.12 User has channel	7
2.13 User send message	7
2.14 User receive message	8
2.15 Series has episode	8
2.16 Channel has series	8
2.17 Channel has movie	9
2.18 Movie has comment	9
2.19 Series has comment	9
2.20 Episode has comment	10
2.21 User make comment	10
2.22 Genre belongs to movie	10
2.23 Genre belongs to series	11
2.24 Friend	11
User Interface Design and Sql Statements	12
3.1 Login	12
3.2 Register	13
3.3 Admin Pages	13
3.3.1 Update/Delete Movie	13
3.3.2 Add Movie	14
3.3.3 Update/Delete Series	15
3.3.4 Add Series	16
3.3.5 Update/Delete Episodes	17
3.3.6 Add Episodes	18
3.3.7 Update/Delete Genre	19
3.3.8 Add Genre	20

3.3.9 Update/Delete Users	20
3.3.10 Wishlist	21
3.4 User Page	22
3.4.1 User Updatable Profile	22
3.4.2 User Inbox	23
3.4.3 User Sent Messages	23
3.4.4 User New Message	24
3.4.5 User New Wish	25
3.4.6 User Wall	25
3.5 Homepage	26
3.5.1 Movie Page	27
3.5.2 User Channels After Login	28

1. E-R Diagram

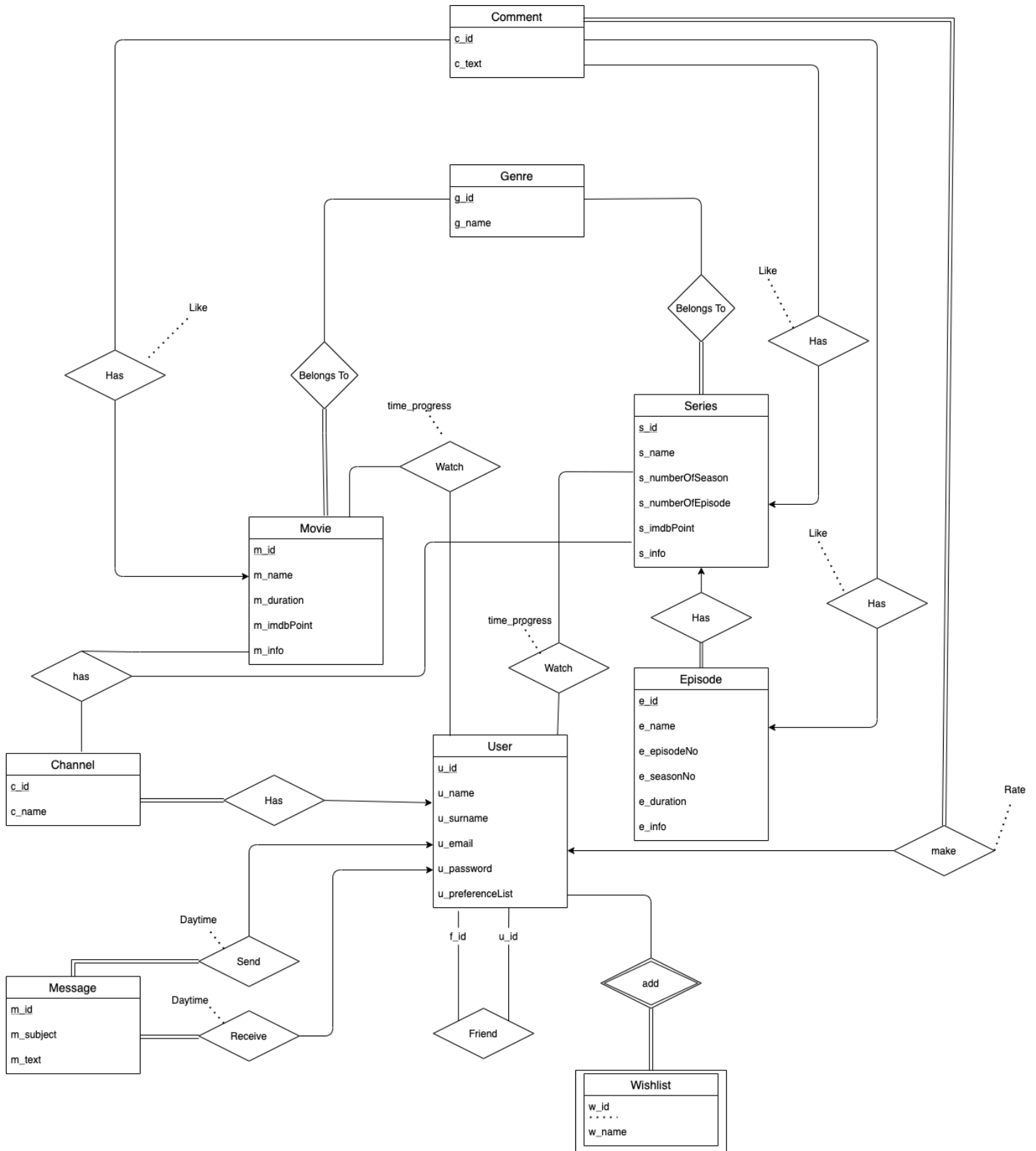


Figure 1: Revised E-R Diagram

2. Table Schemas

2.1 User

Relational Model: user(u_id, u_name, u_surname, u_email, u_password, u_preferencelist)

Candidate Keys: {(u_id), (u_surname), (u_email)}

Table definition:

```
create table user(  
    u_id            int NOT NULL,  
    u_name          varchar(50) NOT NULL,  
    u_surname       varchar(50) NOT NULL,  
    u_email         varchar(70) NOT NULL,  
    u_password      varchar(20) NOT NULL,  
    u_preferencelist varchar(1000),  
    primary key(u_id)  
);
```

2.2 Episode

Relational Model: episode(e_id, e_name, e_episodeNo, e_seasonNo, e_duration, e_info)

Candidate Keys: {(e_id)}

Table Definition:

```
create table episode (  
    e_id          int NOT NULL,  
    e_name        varchar(50) NOT NULL,  
    e_episodeNo   int NOT NULL,  
    e_seasonNo    int NOT NULL,  
    e_duration    int NOT NULL,  
    e_info        varchar(500),  
    primary key(e_id)  
);
```

2.3 Series

Relational Model: series(s_id, s_name, s_numberofseason, s_numberofepisode, s_imdbpoint, s_info)

Candidate Keys: {(s_id), (s_name)}

Table Definition:

```
create table series (  
    s_id            int NOT NULL,  
    s_name          varchar(30) NOT NULL,
```

```

        s_numberOfSeason      int NOT NULL,
        s_numberOfEpisodes    int NOT NULL,
        s_imdbpoint           float,
        s_info                 varchar(500),
        primary key(s_id)
    );

```

2.4 Movie

Relational Model: movie(m_id, m_name, m_duration, m_imdbpoint, m_info)

Candidate Keys: m_id, m_name

Table Definition:

```

create table movie (
    m_id          int NOT NULL,
    m_name        varchar(50) NOT NULL,
    m_duration    int NOT NULL,
    m_imdbpoint   float,
    m_info        varchar(500),
    primary key(m_id)
);

```

2.5 Genre

Relational Model: genre(g_id, g_name)

Candidate Keys: {(g_id)}

Table Definition:

```

create table genre (
    g_id          int NOT NULL,
    g_name        varchar(50) NOT NULL,
    primary key( g_id)
);

```

2.6 Comment

Relational Model: comment(c_id, c_text)

Candidate Keys: {(c_id)}

Table Definition:

```

create table comment(
    c_id          int NOT NULL,
    c_text        varchar(1000),
    primary key( c_id)
);

```

2.7 Channel

Relational Model: channel(c_id, c_name)

Candidate Keys: {(c_id)}

Table Definition:

```
create table channel(  
    c_id          int NOT NULL,  
    c_name        varchar(30) NOT NULL,  
    primary key( c_id)  
);
```

2.8 Message

Relational Model: message(m_id, m_subject, m_text)

Candidate Keys: {(m_id)}

Table Definition:

```
create table message (  
    m_id          int NOT NULL,  
    m_subject      varchar(50) NOT NULL,  
    m_text         varchar(1000),  
    primary key(m_id),  
    foreign key(m_subject) references user(u_id)  
);
```

2.9 Wishlist

Relational Model: wishlist(u_id, w_id, w_name)

Candidate Keys: {(u_id, w_id)}

Foreign Keys: u_id to user

Table Definition:

```
create table wishlist(  
    u_id          int NOT NULL,  
    w_id          int NOT NULL,  
    w_name        varchar(50) NOT NULL,  
    primary key( u_id, w_id),  
    foreign key(u_id) references user(u_id)  
);
```

2.10 Watch series

Relational Model: watch_series(u_id, s_id, time_progress)

Candidate Keys: {(u_id, s_id)}

Foreign Keys: u_id to user, s_id to series

Table Definition:

```
create table watch_series (  

```

```

        u_id            int,
        s_id            int,
        time_progress   time,
        primary key( u_id, s_id),
        foreign key(u_id) references user(u_id),
        foreign key(s_id) references series(s_id)
    );

```

2.11 Watch movie

Relational Model: watch_movie(u_id, m_id, time_progress)

Candidate Keys: {(u_id, m_id)}

Foreign Keys: u_id to user, m_id to movie

Table Definition:

```

create table watch_movie (
    u_id            int,
    m_id            int,
    time_progress   time,
    primary key(u_id, m_id),
    foreign key(u_id) references user(u_id),
    foreign key(m_id) references movie(m_id)
);

```

2.12 User has channel

Relational Model: user_has_channel(c_id, u_id)

Candidate Keys: {(c_id)}

Foreign Keys: c_id to channel, u_id to user

Table Definition:

```

create table user_has_channel(
    u_id    int,
    c_id    int,
    primary key(u_id, c_id),
    foreign key(u_id) references user(u_id),
    foreign key(c_id) references channel(c_id)
);

```

2.13 User send message

Relational Model:

user_send_message(m_id, u_id, daytime)

Candidate Keys: {(m_id)}

Foreign Keys: m_id to message, u_id to user

Table Definition:

```

create table user_send_message(
    u_id            int,

```



```

        m_id          int,
        daytime       smalldatetime,
        primary key(u_id, m_id),
        foreign key(u_id) references user(u_id),
        foreign key(m_id) references message(m_id)
    );

```

2.14 User receive message

Relational Model: user_receive_message(m_id, u_id, daytime)

Candidate Keys: {(m_id)}

Foreign Keys: m_id to message, u_id to user

Table Definition:

```

create table user_receive_message(
    u_id          int,
    m_id          int,
    daytime       smalldatetime,
    primary key(u_id, m_id),
    foreign key(u_id) references user(u_id),
    foreign key(m_id) references message(m_id)
);

```

2.15 Series has episode

Relational Model: series_has_episode(e_id, s_id)

Candidate Keys: {(e_id)}

Foreign Keys: e_id to episode, s_id to series

Table Definition:

```

create table series_has_model(
    s_id  int,
    e_id  int,
    primary key(s_id, e_id),
    foreign key(s_id) references series(s_id),
    foreign key(e_id) references episode(e_id)
);

```

2.16 Channel has series

Relational Model: channel_has_series(c_id, s_id)

Candidate Keys: {(c_id, s_id)}

Foreign Keys: c_id to channel, s_id to series

Table Definition:

```

create table channel_has_series(
    c_id  int
    s_id  int,
    primary key(c_id, s_id),

```

```

foreign key(c_id) references channel(c_id),
foreign key(s_id) references series(s_id)
);

```

2.17 Channel has movie

Relational Model: channel_has_movie(c_id, m_id)

Candidate Keys: {(c_id, m_id)}

Foreign Keys: c_id to channel, m_id to movie

Table Definition:

```

create table channel_has_movie(
    c_id    int,
    m_id    int,
    primary key(c_id, m_id),
    foreign key(c_id) references channel(c_id),
    foreign key(m_id) references movie(m_id)
);

```

2.18 Movie has comment

Relational Model: movie_has_comment(c_id, m_id, like)

Candidate Keys: {(c_id, m_id)}

Foreign Keys: c_id to comment, m_id to movie

Table Definition:

```

create table movie_has_comment (
    m_id    int,
    c_id    int,
    like    boolean,
    primary key(c_id, m_id),
    foreign key(m_id) references movie(m_id),
    foreign key(c_id) references comment(c_id)
);

```

2.19 Series has comment

Relational Model: series_has_comment(c_id, s_id, like)

Candidate Keys: {(c_id, s_id)}

Foreign Keys: c_id to comment, s_id to series

Table Definition:

```

create table series_has_comment (
    s_id    int,
    c_id    int,
    like    boolean,
    primary key(s_id, c_id),
    foreign key(s_id) references series(s_id),

```

```
foreign key(c_id) references comment(c_id)
);
```

2.20 Episode has comment

Relational Model: episode_has_comment(c_id, e_id, like)

Candidate Keys: {(c_id)}

Foreign Keys: c_id to comment, e_id to episode

Table Definition:

```
create table episode_has_comment (
    e_id    int,
    c_id    int,
    like    boolean,
    primary key(e_id, c_id),
    foreign key(e_id) references episode(e_id),
    foreign key(c_id) references comment(c_id)
);
```

2.21 User make comment

Relational Model: user_make_comment(c_id, u_id, rate)

Candidate Keys: {(c_id)}

Foreign Keys: c_id to comment, u_id to user

Table Definition:

```
create table user_make_comment(
    u_id    int,
    c_id    int,
    rate    int,
    primary key(u_id, c_id),
    foreign key(u_id) references user(u_id),
    foreign key(c_id) references comment(c_id)
);
```

2.22 Genre belongs to movie

Relational Model: genre_belongs_to_movie(g_id, m_id)

Candidate Keys: {(g_id, m_id)}

Foreign Keys: g_id to genre, m_id to movie

Table Definition:

```
create table genre_belongs_to_movie(
    g_id    int,
    m_id    int,
    primary key(g_id, m_id),
    foreign key(g_id) references genre(g_id),
    foreign key(m_id) references movie(m_id)
);
```

2.23 Genre belongs to series

Relational Model: genre_belongsto_series(g_id, s_id)

Candidate Keys: {(g_id, s_id)}

Foreign Keys: g_id to genre, s_id to series

Table Definition:

```
create table genre_belongsto_series(  
    g_id    int,  
    s_id    int,  
    primary key(g_id, s_id),  
    foreign key(g_id) references genre(g_id),  
    foreign key(s_id) references series(s_id)  
);
```

2.24 Friend

Relational Model: friend(f_id, u_id)

Candidate Keys:

Foreign Keys: u_id to user

Table Definition:

```
create table friend(  
    f_id    int,  
    u_id    int,  
    foreign key(f_id) references user(u_id),  
    foreign key(u_id) references user(u_id)  
);
```

3. User Interface Design and Sql Statements

3.1 Login

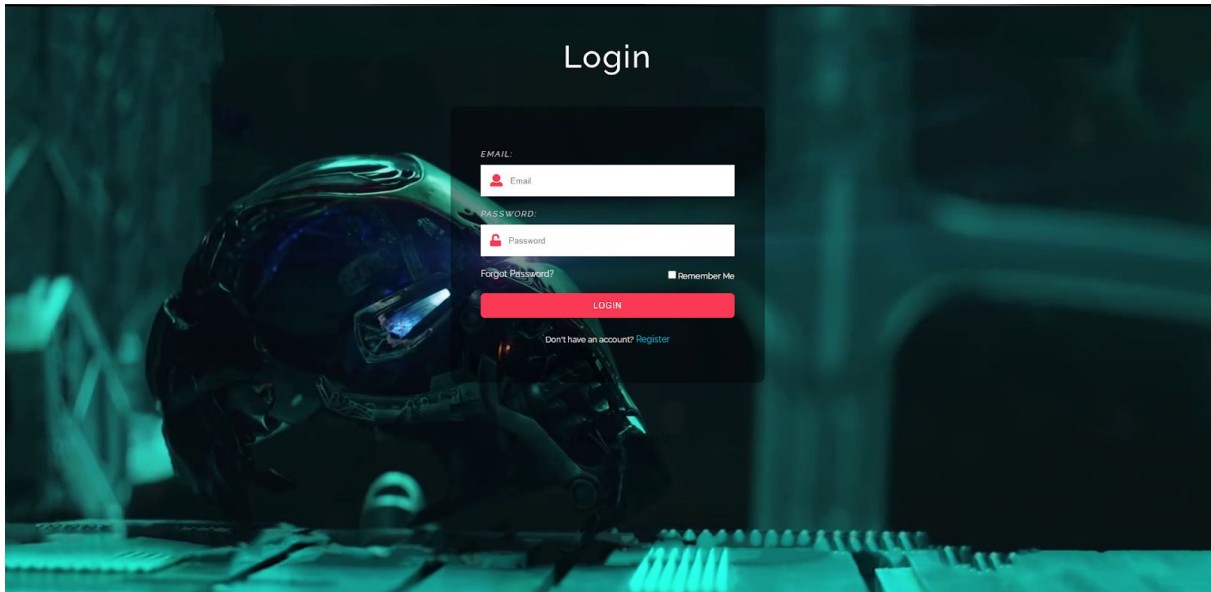


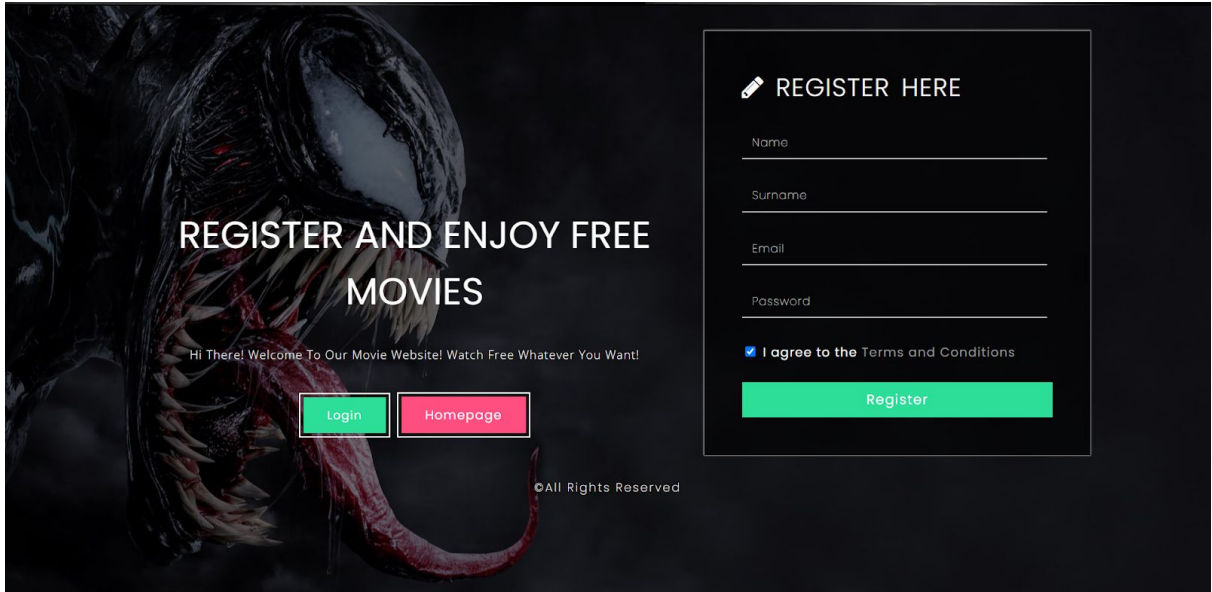
Figure 2: Login Page

Users can login with their e-mail address and password.

Sql Statements

```
SELECT *  
FROM user  
WHERE user.u_email = @u_email AND user.u_password = @u_password;
```

3.2 Register



The screenshot shows a dark-themed registration page. On the left, there's a large, menacing image of a creature with sharp teeth. The text 'REGISTER AND ENJOY FREE MOVIES' is prominently displayed in white. Below it, a smaller line of text says 'Hi There! Welcome To Our Movie Website! Watch Free Whatever You Want!'. There are two buttons: a green 'Login' button and a pink 'Homepage' button. On the right, there's a white 'REGISTER HERE' form with fields for Name, Surname, Email, and Password. A checkbox for 'I agree to the Terms and Conditions' is checked. A green 'Register' button is at the bottom of the form. A small copyright notice '©All Rights Reserved' is at the bottom center.

Figure 3: Register Page

Users can register with a name, surname, email and password.

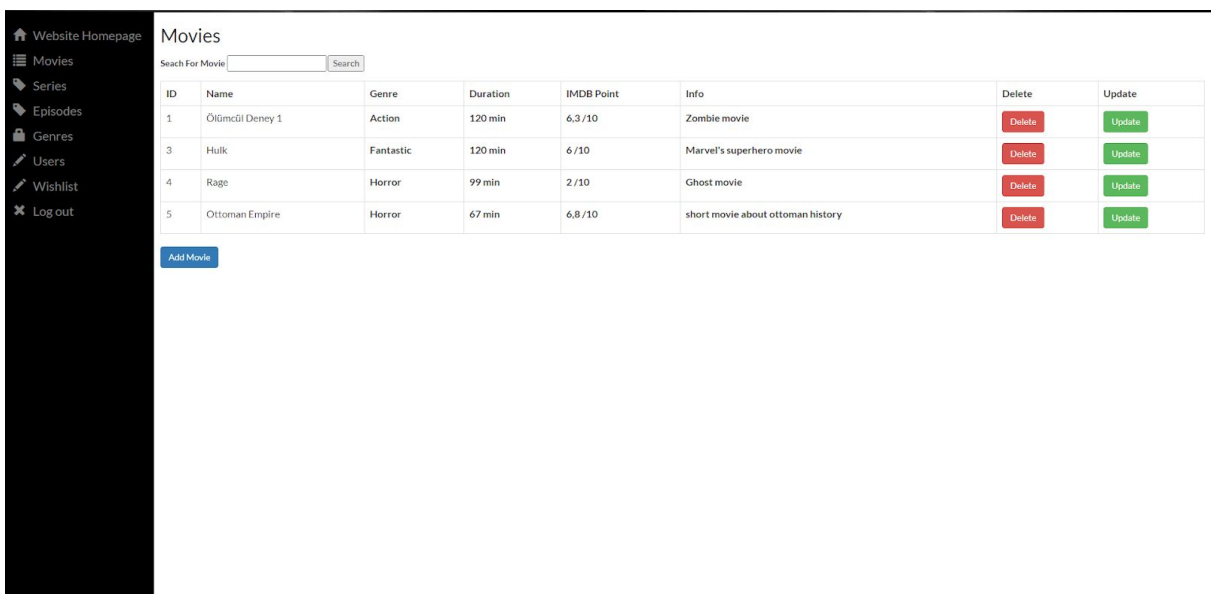
Sql Statements

INSERT INTO user

VALUES (@u_name, @u_surname, @u_email, @u_password, NULL);

3.3 Admin Pages

3.3.1 Update/Delete Movie



The screenshot shows an admin interface for managing movies. On the left is a sidebar with navigation links: Website Homepage, Movies, Series, Episodes, Genres, Users, Wishlist, and Log out. The main area is titled 'Movies' and contains a search bar and a table of movies. The table has columns for ID, Name, Genre, Duration, IMDB Point, Info, Delete, and Update. There are five movies listed. Below the table is an 'Add Movie' button.

ID	Name	Genre	Duration	IMDB Point	Info	Delete	Update
1	Ölümöl Deney 1	Action	120 min	6,3 /10	Zombie movie	Delete	Update
3	Hulk	Fantastic	120 min	6 /10	Marvel's superhero movie	Delete	Update
4	Rage	Horror	99 min	2 /10	Ghost movie	Delete	Update
5	Ottoman Empire	Horror	67 min	6,8 /10	short movie about ottoman history	Delete	Update

Figure 4: Update/Delete Movie Page

Admins can delete and update movies.

Sql Statements

```
DELETE *  
FROM movie  
WHERE m_id = @m_id
```

```
UPDATE movie  
SET name = "Edge of Tomorrow"  
WHERE m.id = @m_id
```

3.3.2 Add Movie

The screenshot shows the 'Add Movie' page. On the left is a dark sidebar with a home icon and the following menu items: Website Homepage, Movies, Series, Episodes, Genres, Users, Wishlist, and Log out. The main area is white and contains a form with the following fields: 'Movie Name' (text input), 'Genre' (dropdown menu showing 'Horror'), 'Movie Duration in Minutes' (text input), 'Movie IMDB Point (Out of 10)' (text input), and 'Movie Info' (text input). A blue 'Add Movie' button is positioned below the 'Movie Info' field.

Figure 5: Add Movie Page

Admins can add new movies.

Sql Statements

```
INSERT INTO movie  
VALUES (@m_name, @m_duration, @m_imdbPoint, @m_info);
```

Figure 6: Choose Genre Page (While uploading movie)

3.3.3 Update/Delete Series

ID	Name	Genre	Number Of Season	Number Of Episode	IMDB Point	Info	Delete	Update
1	Game Of Thrones	Fantastic	8	50	8.7 / 10	Khaalesi ruins the kingdom	Delete	Update
3	Walking Dead	Horror	11	88	7 / 10	Popular zombie series	Delete	Update
4	Modern Family	Comedy	7	150	8 / 10	Movie is about funny life of a family	Delete	Update
5	Mandalorian	Science-Fiction	3	21	9 / 10	Starwars' tv series	Delete	Update

Figure 7: Update/Delete Series Page

Admins can delete and update series.

Sql Statements

```
DELETE *
FROM series
WHERE s_id = @s_id
```


UPDATE series
SET name = "Game of Thrones"
WHERE s.id = @s_id

3.3.4 Add Series

The screenshot shows the 'Add Series' page. On the left is a dark sidebar with a home icon and links to 'Website Homepage', 'Movies', 'Series', 'Episodes', 'Genres', 'Users', 'Wishlist', and 'Log out'. The main area is light gray and contains a form with the following fields: 'Series Name' (text input), 'Genre' (dropdown menu showing 'Horror'), 'Number Of Season' (text input), 'Number Of Episode' (text input), 'Movie (IMDB Point (Out of 10))' (text input), and 'Movie Info' (text input). A blue 'Add Series' button is positioned below the 'Movie Info' field.

Figure 8: Add Series Page

Admins can add new series.

Sql Statements

```
INSERT INTO series  
VALUES (@s_name, @s_numberOfSeason, @s_numberOfEpisode, @s_imdbPoint,  
@s_info);
```

```
INSERT INTO genre_belongsto_series  
VALUES (@s_name);
```

Series Name

Genre

Horror

Comedy

Fantastic

Science-Fiction

Drama

History

Biography

Action

Thriller

Mystery

Western

Movie Info

Add Series

Figure 9: Pick Genre for Series (While uploading series)

3.3.5 Update/Delete Episodes

Episodes

Search For Episode

ID	Name	Series	Season No	Episode No	Duration	Info	Delete	Update
4	Khaalesi	Game Of Thrones	1	1	52 min	Khaalesi introduced	Delete	Update
5	Khal	Game Of Thrones	1	2	42 min	Khal introduced	Delete	Update
6	Kingdoms	Game Of Thrones	1	3	55 min	North kingdoms starts the war	Delete	Update
7	Rage	Game Of Thrones	2	1	44 min	Khaalesi dead	Delete	Update
8	Baby Yoda	Mandalorian	1	1	66 min	long time ago in a galaxy far far away	Delete	Update
9	Mandoo	Mandalorian	1	2	45 min	The first mandalorian shows up	Delete	Update

Add Movie

Figure 10: Update/Delete Episodes

Admins can delete and update episodes.

Sql Statements

```
DELETE *
FROM episode
WHERE e_id = @e_id
```

```
UPDATE episode
SET name = "GoT 1"
WHERE e.id = @e_id
```

3.3.6 Add Episodes

Episode Name

Series

Game Of Thrones

Season Number

Episode Number

Episode Duration in Minutes

Movie Info

Add Episode

Figure 11: Add Episodes Page

Admins can add new episodes.

Sql Statements

INSERT INTO episode

VALUES (@e_name, @e_seasonNo, @e_duration, @s_imdbPoint, @e_info)

Episode Name

Series

Game Of Thrones

Game Of Thrones

Walking Dead

Modern Family

Mandalorian

Episode Number

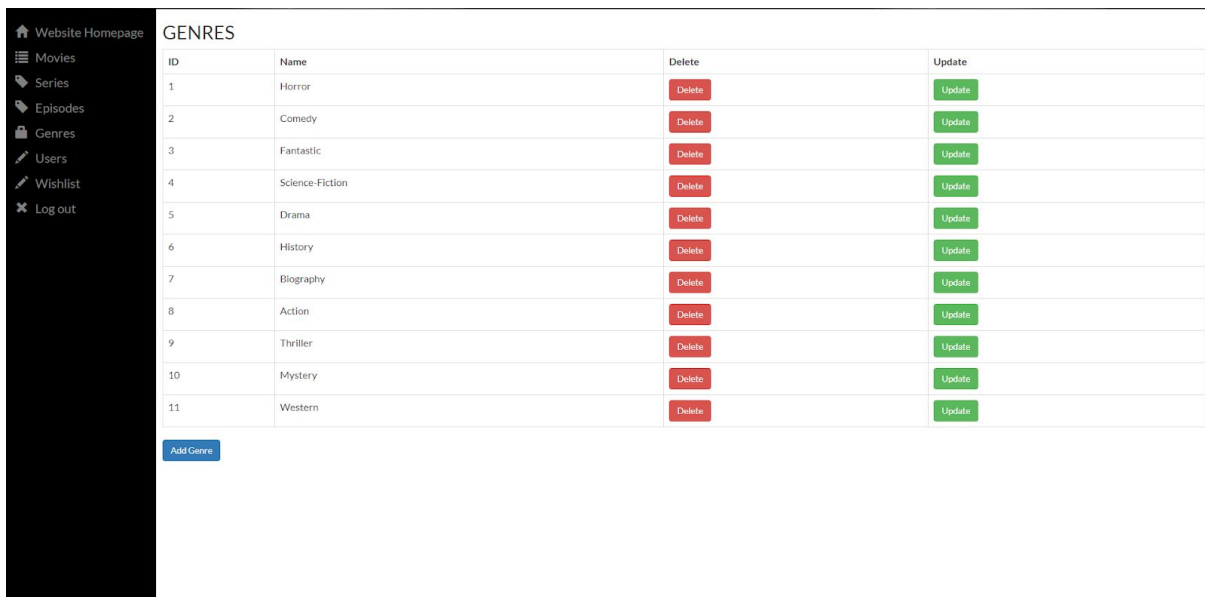
Episode Duration in Minutes

Movie Info

Add Episode

Figure 12: Choose Series Page (while uploading episodes)

3.3.7 Update/Delete Genre



ID	Name	Delete	Update
1	Horror	Delete	Update
2	Comedy	Delete	Update
3	Fantastic	Delete	Update
4	Science-Fiction	Delete	Update
5	Drama	Delete	Update
6	History	Delete	Update
7	Biography	Delete	Update
8	Action	Delete	Update
9	Thriller	Delete	Update
10	Mystery	Delete	Update
11	Western	Delete	Update

Add Genre

Figure 13: Update/Delete Genre Page

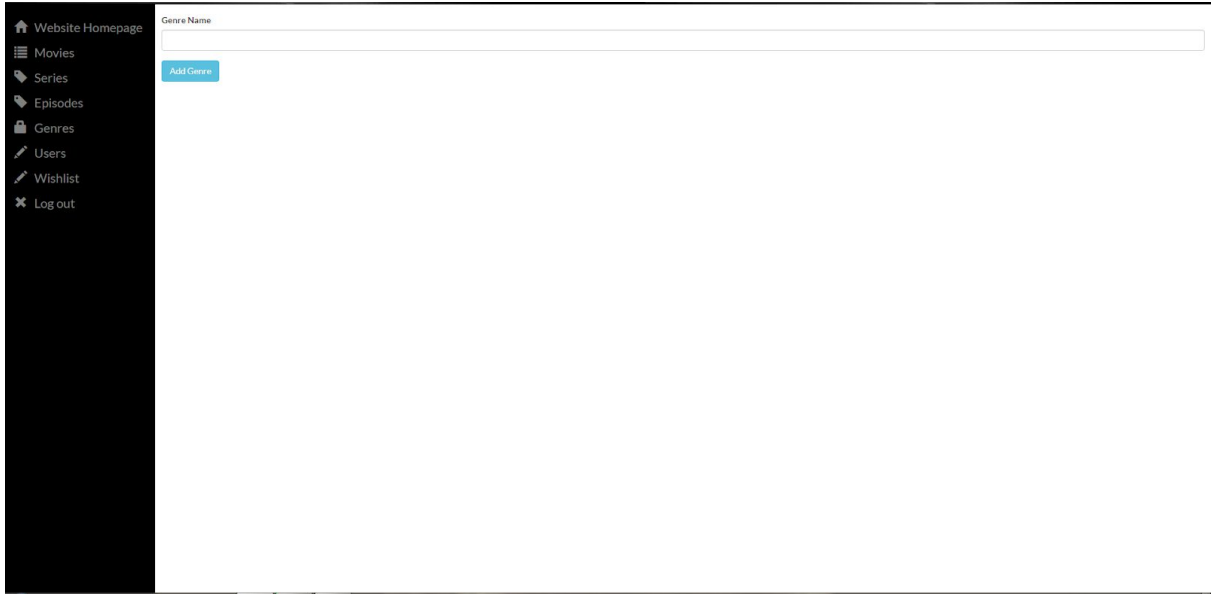
Admins can delete and update genres.

Sql Statements

```
DELETE *  
FROM genre  
WHERE g_id = @g_id
```

```
UPDATE genre  
SET name = "horror"  
WHERE g.id = @g_id
```

3.3.8 Add Genre



Genre Name

Add Genre

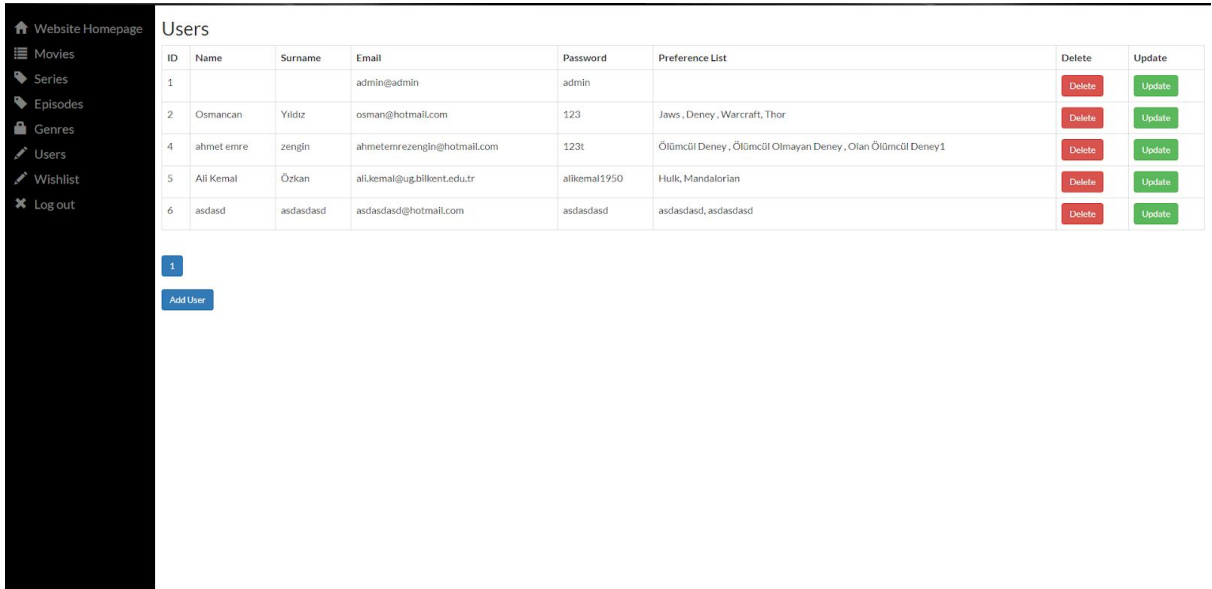
Figure 14: Add Genre Page

Admins can add genres.

Sql Statements

```
INSERT INTO genre  
VALUES (@g_name);
```

3.3.9 Update/Delete Users



ID	Name	Surname	Email	Password	Preference List	Delete	Update
1			admin@admin	admin		Delete	Update
2	Osman	Yildiz	osman@hotmail.com	123	Jaws , Deney , Warcraft, Thor	Delete	Update
4	ahmet emre	zengin	ahmetemrezengin@hotmail.com	123!	Ölümçül Deney , Ölümçül Olmayan Deney , Olan Ölümçül Deney1	Delete	Update
5	Ali Kemal	Özkan	ali.kemal@ug.bilkent.edu.tr	alikemal1950	Hulk, Mandalorian	Delete	Update
6	asdasd	asdasdasd	asdasdasd@hotmail.com	asdasdasd	asdasdasd, asdasdasd	Delete	Update

Add User

Figure 15: Update/Delete Users Page

Admins can delete and update users.

Sql Statements

```
DELETE *  
FROM user  
WHERE u_id = @u_id
```

```
UPDATE user  
SET name = "Abdurrezzak"  
WHERE u.id = @u_id
```

3.3.10 Wishlist

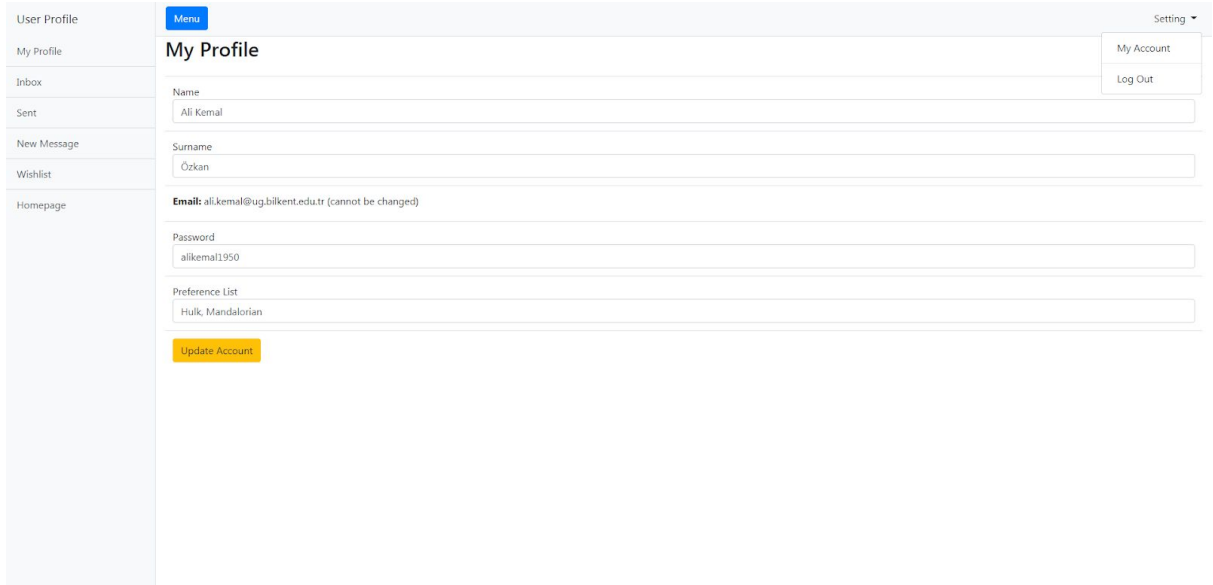


ID	User	Wish Name
1	osman@hotmail.com	Terminatör 3
2	osman@hotmail.com	Jaw
1002	ali.kemal@ug.bilkent.edu.tr	Terminatör 2
1003	ali.kemal@ug.bilkent.edu.tr	Deep Blue
1004	ali.kemal@ug.bilkent.edu.tr	World of Warcraft

Figure 16: Wishlist

3.4 User Page

3.4.1 User Updatable Profile



The screenshot shows a web application interface for a user profile. On the left is a vertical sidebar with a light blue background, containing a list of navigation items: 'User Profile', 'My Profile', 'Inbox', 'Sent', 'New Message', 'Wishlist', and 'Homepage'. The 'My Profile' item is currently selected. To the right of the sidebar is the main content area. At the top of this area is a header bar with a blue 'Menu' button on the left and a 'Setting' dropdown menu on the right. Below the header, the title 'My Profile' is displayed. The profile information is organized into several sections: 'Name' with the value 'Ali Kemal', 'Surname' with the value 'Özkan', 'Email' with the value 'ali.kemal@ug.bilkent.edu.tr' (marked as 'cannot be changed'), 'Password' with the value 'alikemal1950', and 'Preference List' with the value 'Hulk, Mandalorian'. At the bottom of the profile section is a yellow 'Update Account' button. In the top right corner of the main content area, there are two buttons: 'My Account' and 'Log Out'.

Figure 17: User Profile

Users can update their profiles.

Sql Statements

```
UPDATE user
SET surname = "Curcunacıgillerdenoğlu"
WHERE g.id = @g_id
```

3.4.2 User Inbox

User Profile	Menu	Setting
My Profile	Inbox	
Inbox		
Sent		
New Message		
Wishlist		
Homepage		

From	Subject	Message	Date
asdasdasd@hotmail.com	film önerisi	bence ghost'u izleyebilirsin güzel film	23.11.2020 00:00:00
ahmetemrezengin@hotmail.com	film hakkında	hulk filmi güzeldi beğendim ama iki daha güzeldi	23.11.2020 00:00:00

Figure 18: Inbox Page

Users can check their inboxes.

Sql Statements

```
SELECT *  
FROM user_receive_message  
WHERE u_id = @u_id
```

3.4.3 User Sent Messages

User Profile	Menu	Setting
My Profile	Sent Messages	
Inbox		
Sent		
New Message		
Wishlist		
Homepage		

To	Subject	Message	Date
ahmetemrezengin@hotmail.com	film hakkında	mandalorian hakkındaki gelecek bölümdeki doğru mu?	23.11.2020 00:00:00
ahmetemrezengin@hotmail.com	selam	Hulk filmi beğendin mi	23.11.2020 00:00:00
asdasdasd@hotmail.com	korku filmi önerisi	başka korku filmi önerir var mı	23.11.2020 00:00:00

Figure 19: Outbox Page

Users can send messages to other users.

Sql Statements

```
SELECT *  
FROM user_send_message  
WHERE u_id = @u_id
```

3.4.4 User New Message

The screenshot shows a web application interface for creating a new message. On the left is a sidebar with navigation links: 'User Profile', 'My Profile', 'Inbox', 'Sent', 'New Message' (which is highlighted), 'Wishlist', and 'Homepage'. The main area is titled 'New Message' and contains three text input fields labeled 'To:', 'Subject', and 'Message'. Below the 'Message' field is a blue 'Send' button. In the top right corner of the main area, there is a 'Menu' button and a 'Setting' dropdown menu.

Figure 20: Create New Message Page

Users can create new messages to send other users.

Sql Statements

```
INSERT INTO message  
VALUES (@m_subjects, @m_text);
```

3.4.5 User New Wish

User Profile

My Profile

Inbox

Sent

New Message

Wishlist

Homepage

Menu

Wishlist

Wish Name

Send

Setting ▾

Figure 21: New Wish Page

Users can add new wishes to wishlist.

Sql Statements

```
INSERT INTO wishlist
VALUES (@w_name);
```

3.4.6 User Wall

User Profile

My Profile

Inbox

Sent

New Message

Wishlist

Wall

Homepage

Menu

My Wall

My Actions

You liked a movie (Underworld: Evolution)	23.11.2020 00:00:00
You commented on a movie (Underworld: Evolution)	23.11.2020 00:00:00
You disliked a movie (Hulk)	23.11.2020 00:00:00
You commented on a movie (Hulk)	23.11.2020 00:00:00

Setting ▾

Figure 22: Wall Page

Users can follow their actions by displaying their walls.

3.5 Homepage

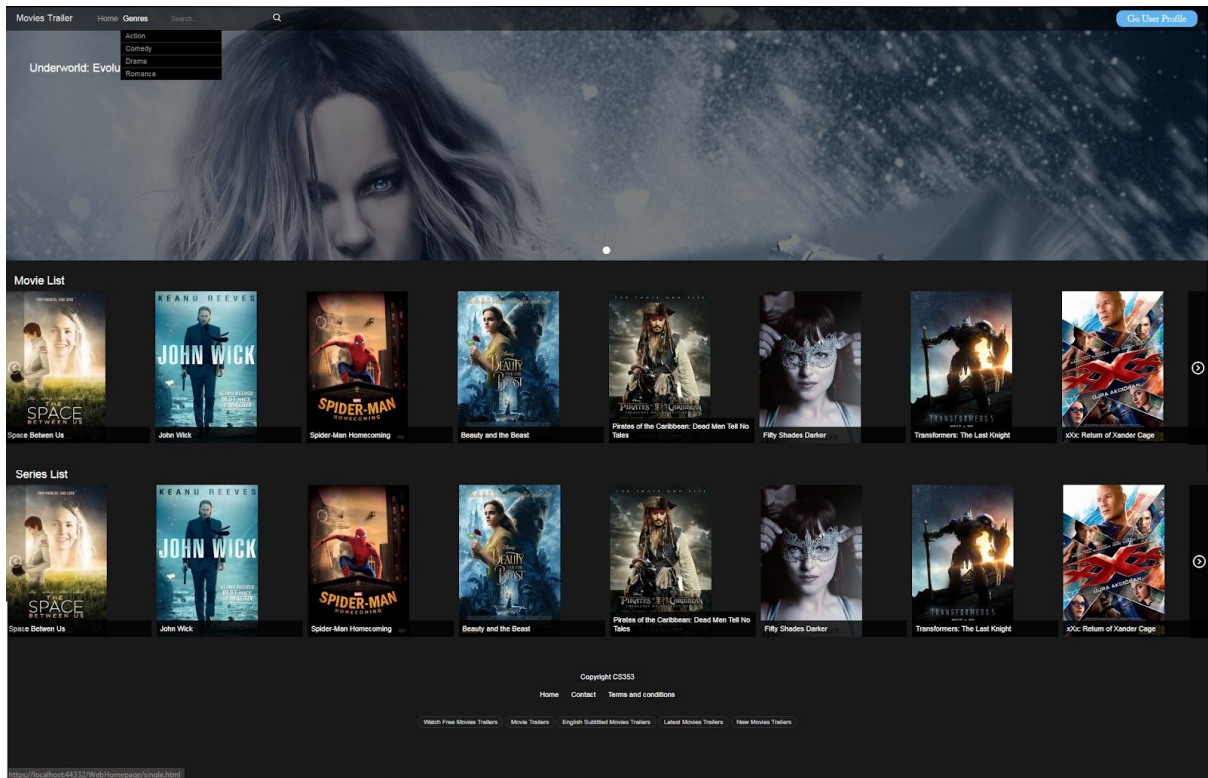


Figure 23: Home Page

Films are listed in the homepage. In the top bar, genres can be found and users can display their profiles by clicking the "Go user profile" button on the top right.

3.5.1 Movie Page

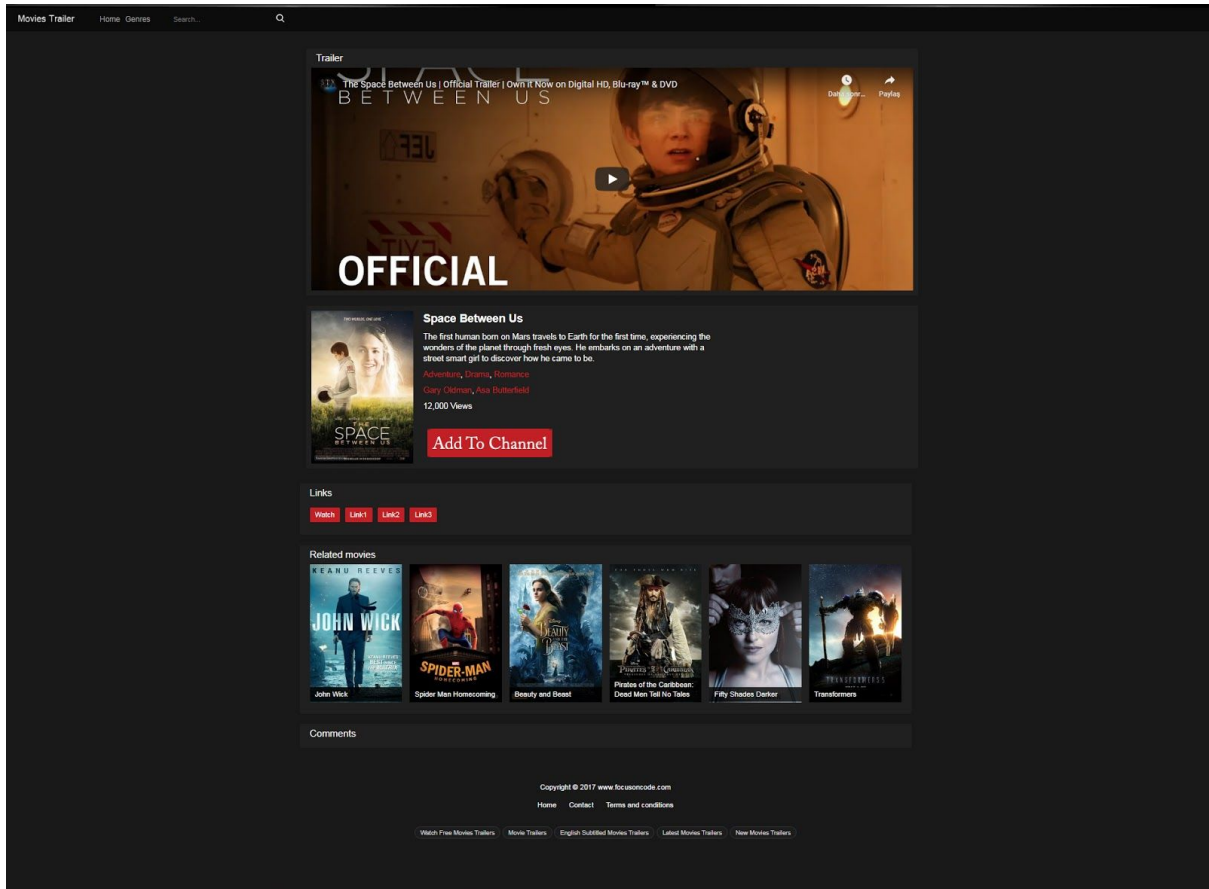


Figure 24: Movie Page

In the movie page, the movie and its trailer as well as its name and short info are shown. Users can add the movie to their channels by clicking the “Add To Channel” button.

3.5.2 User Channels After Login

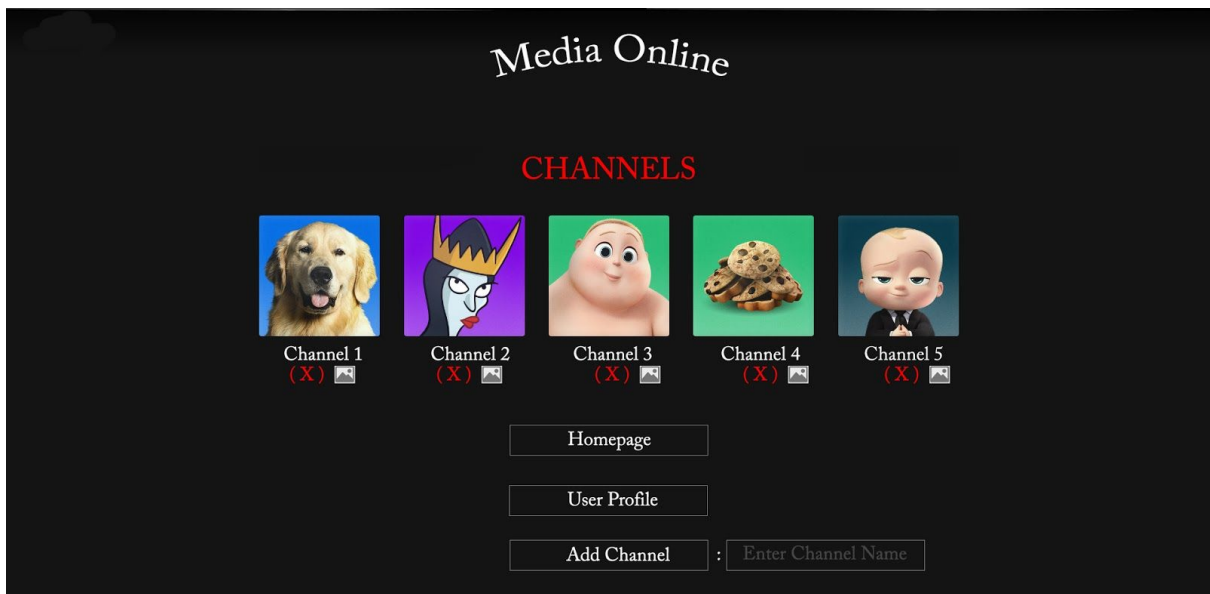


Figure 25: Channels Page

After login, users encounter different channels, can select any of them and enjoy the films that are added in the relative channel.

Sql Statements

```
SELECT *  
FROM channel  
WHERE c_id = @c_id
```

```
INSERT INTO channel  
VALUES (@c_name)
```