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://alikemalozkan.github.io/MediaOnline/

Table of Contents

1. E-R Diagram

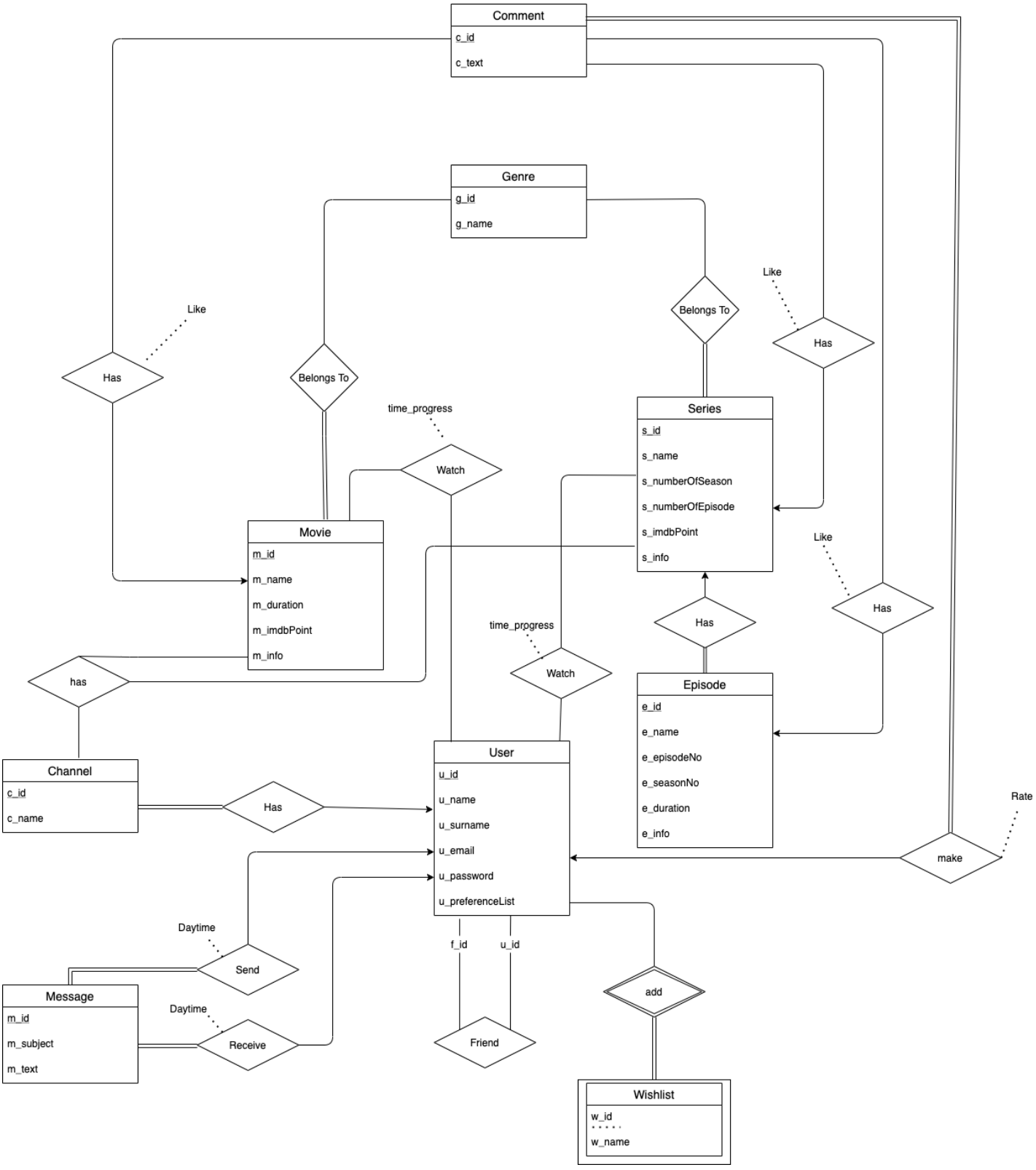


Figure 1: Revised E-R Diagram

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_belongsto_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_belongsto_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

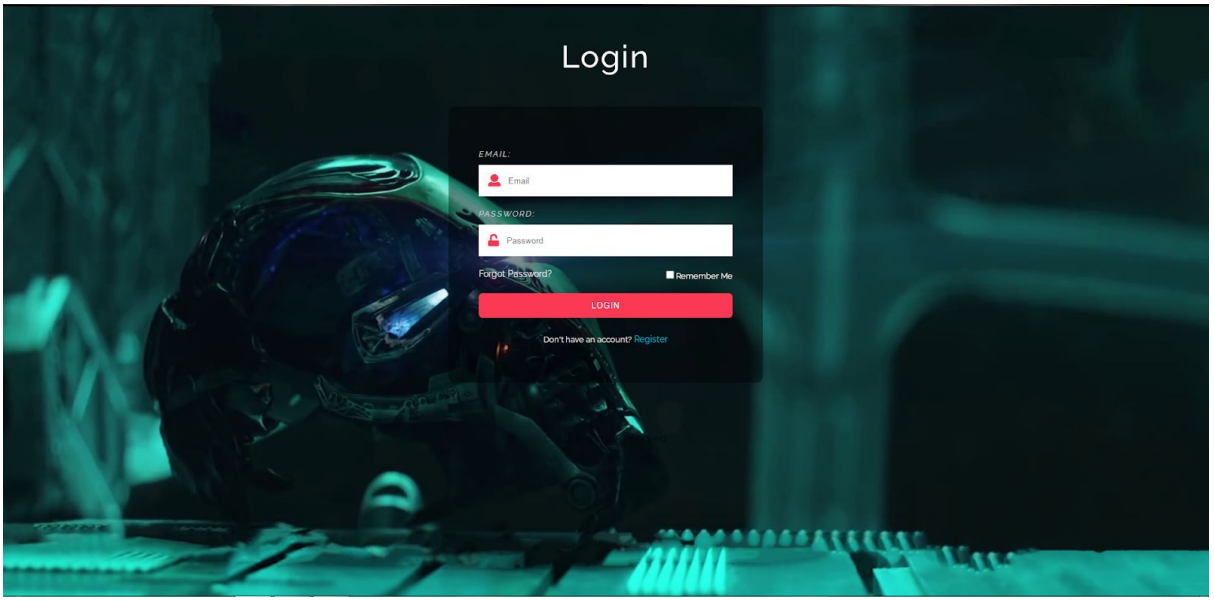


Figure 2: Login Page

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

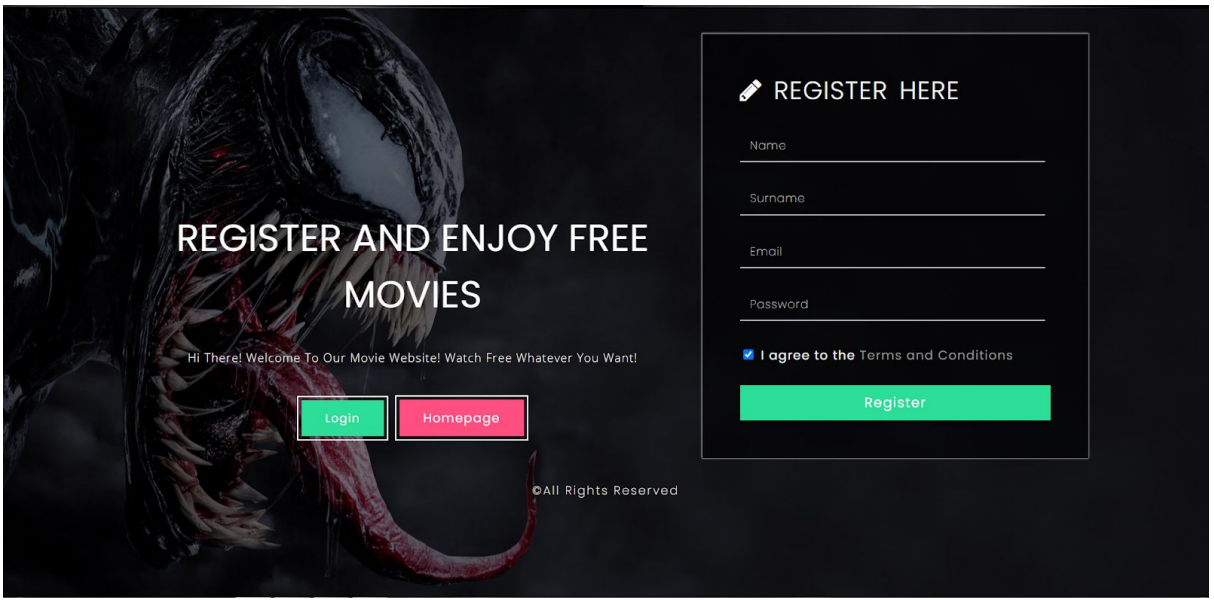


Figure 3: Register Page

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

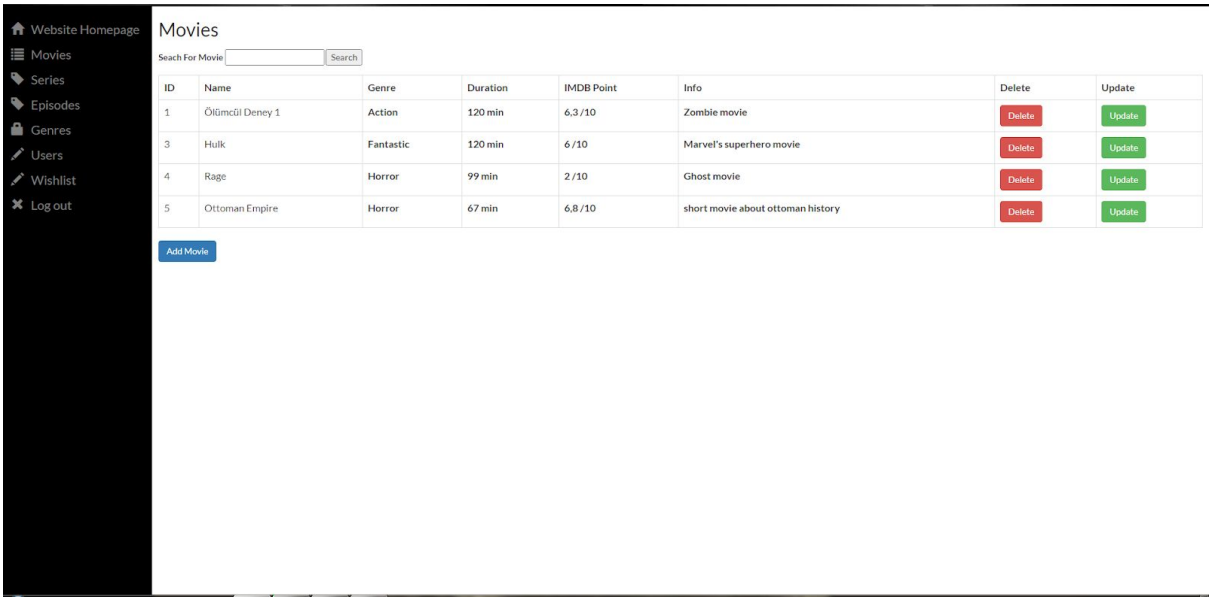


Figure 4: Update/Delete Movie Page

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

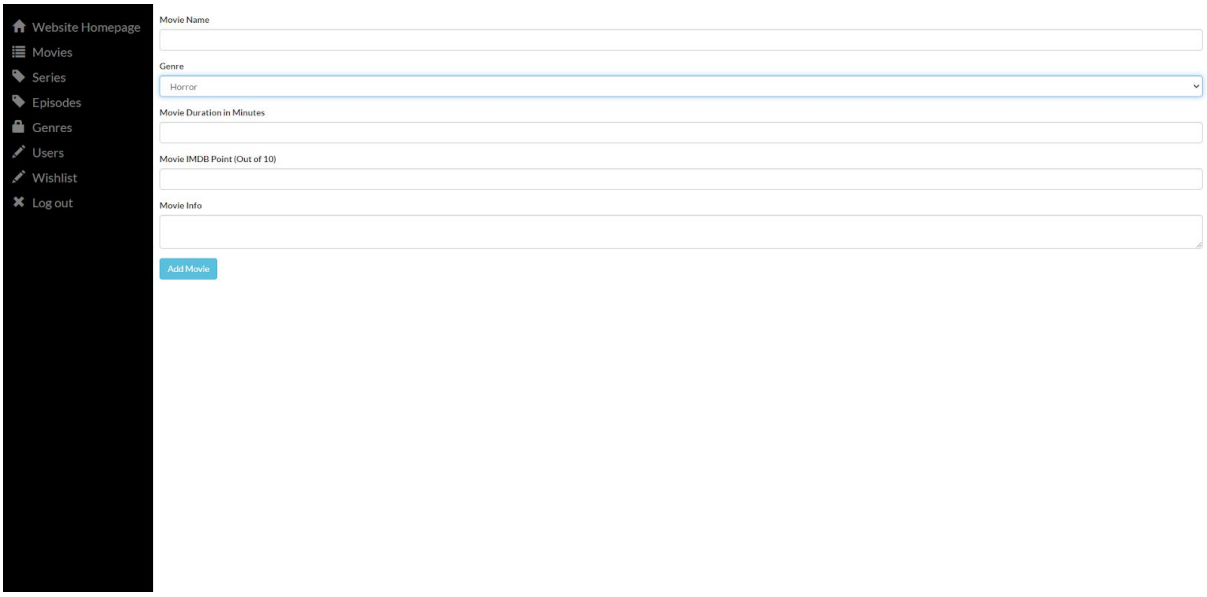


Figure 5: Add Movie Page

Admins can add new movies.

Sql Statements

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

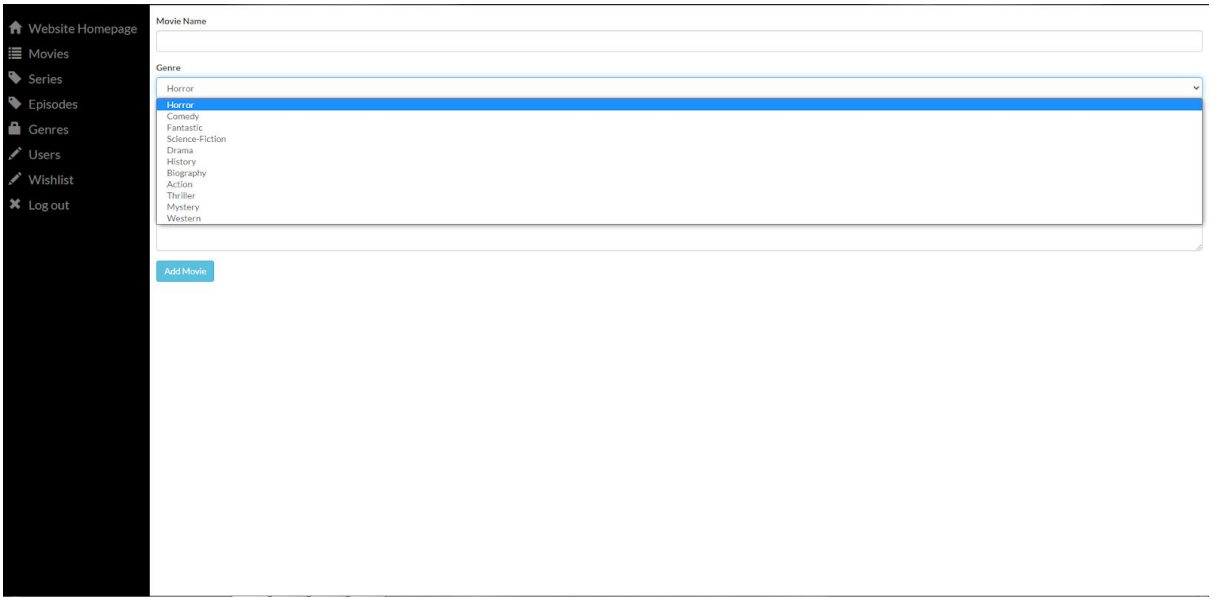


Figure 6: Choose Genre Page (While uploading movie)

3.3.3 Update/Delete Series

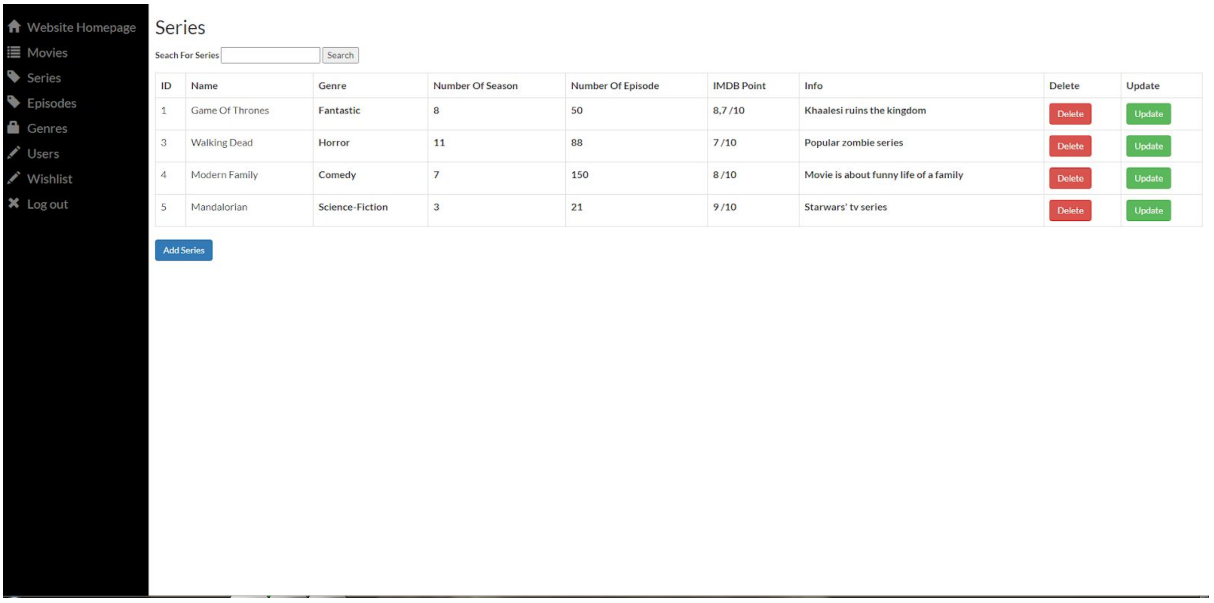


Figure 7: Update/Delete Series Page

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

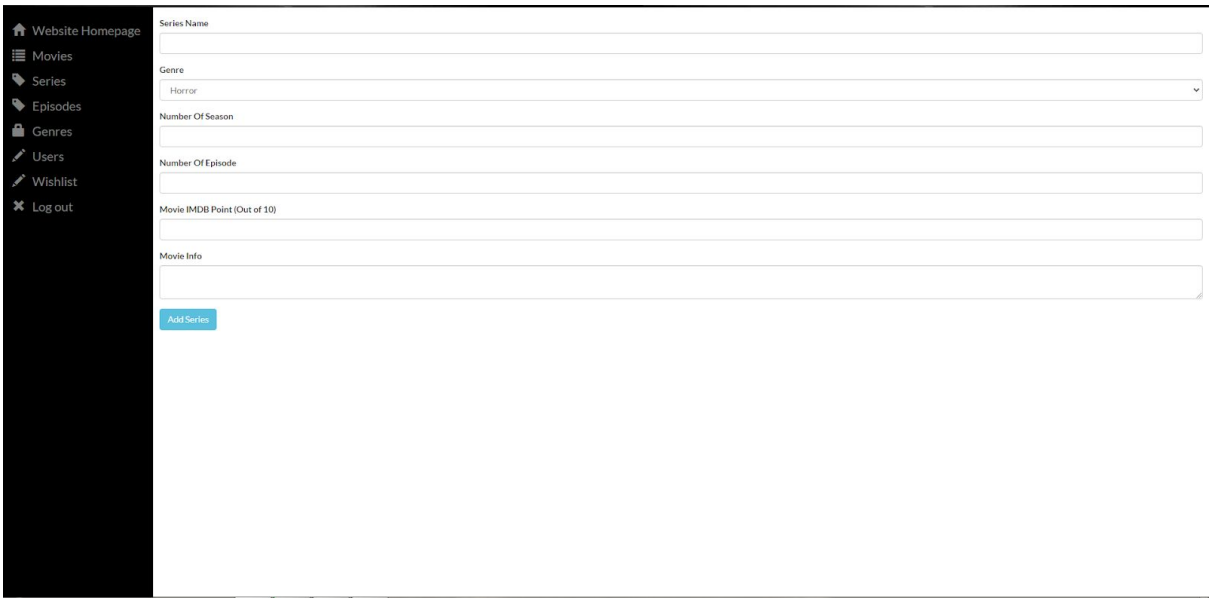


Figure 8: Add Series Page

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);

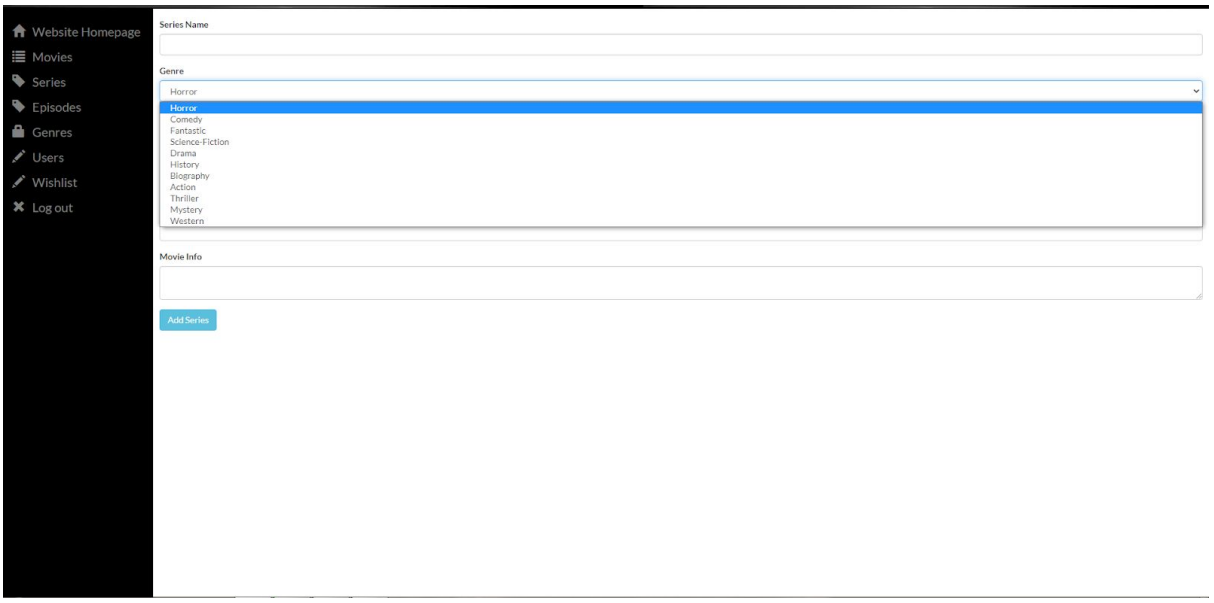


Figure 9: Pick Genre for Series (While uploading series)

3.3.5 Update/Delete Episodes

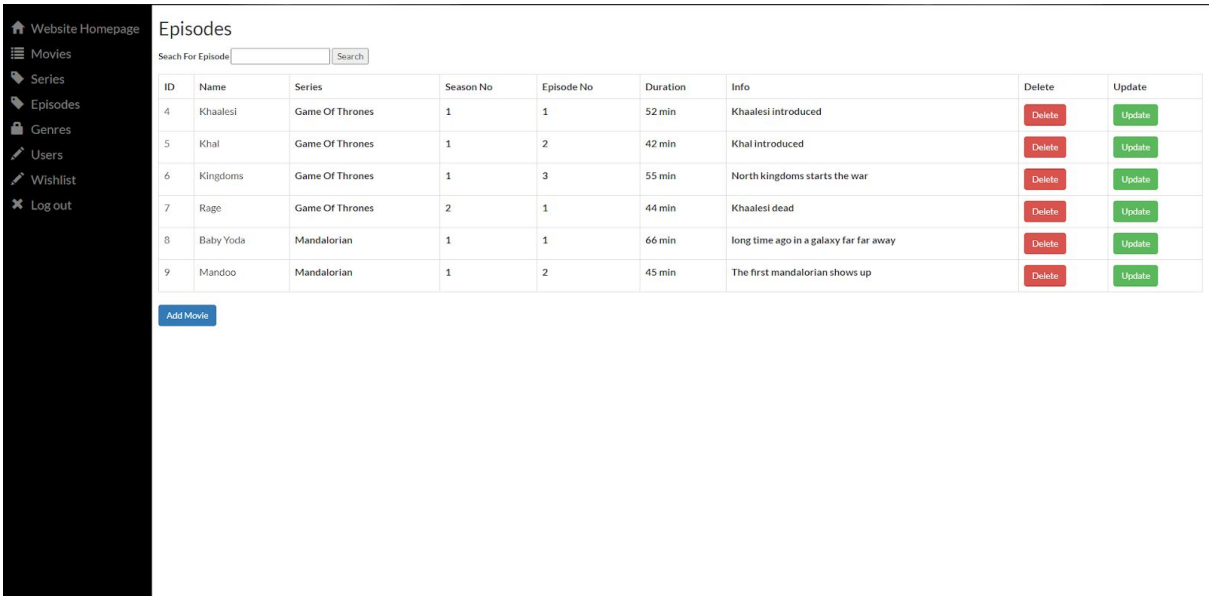


Figure 10: Update/Delete Episodes

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

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)

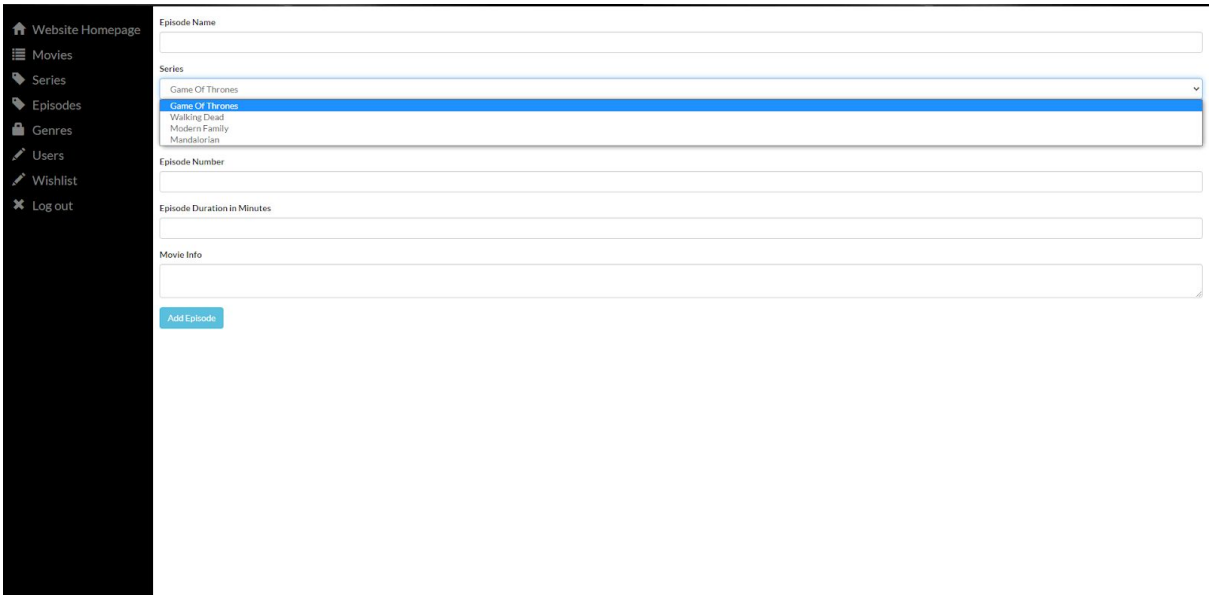


Figure 12: Choose Series Page (while uploading episodes)

3.3.7 Update/Delete Genre

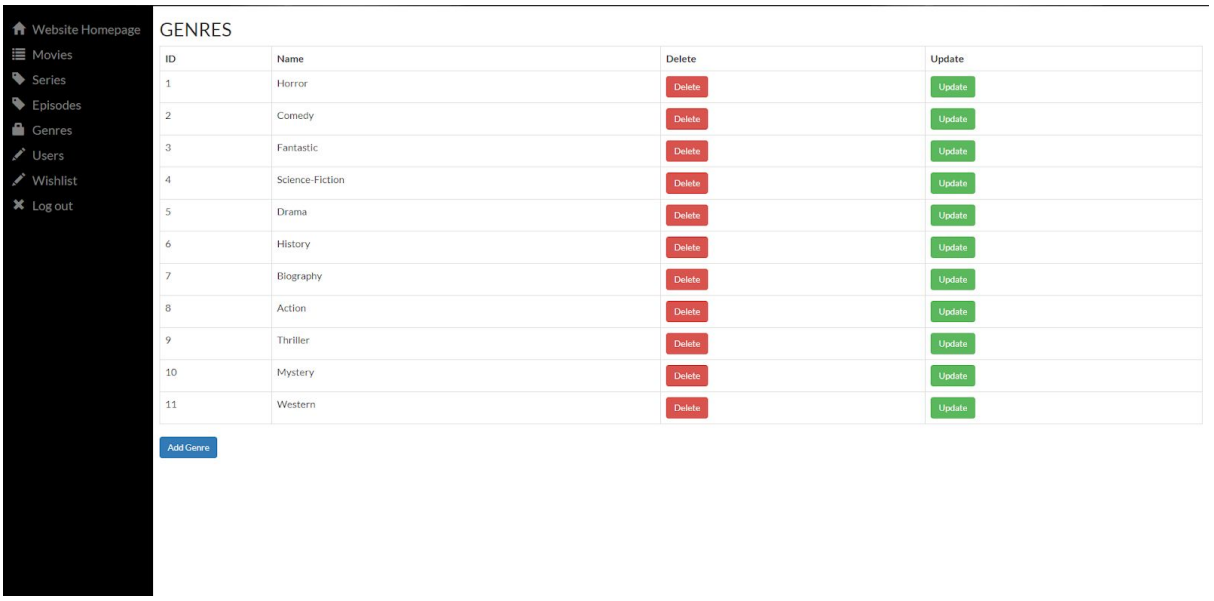


Figure 13: Update/Delete Genre Page

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

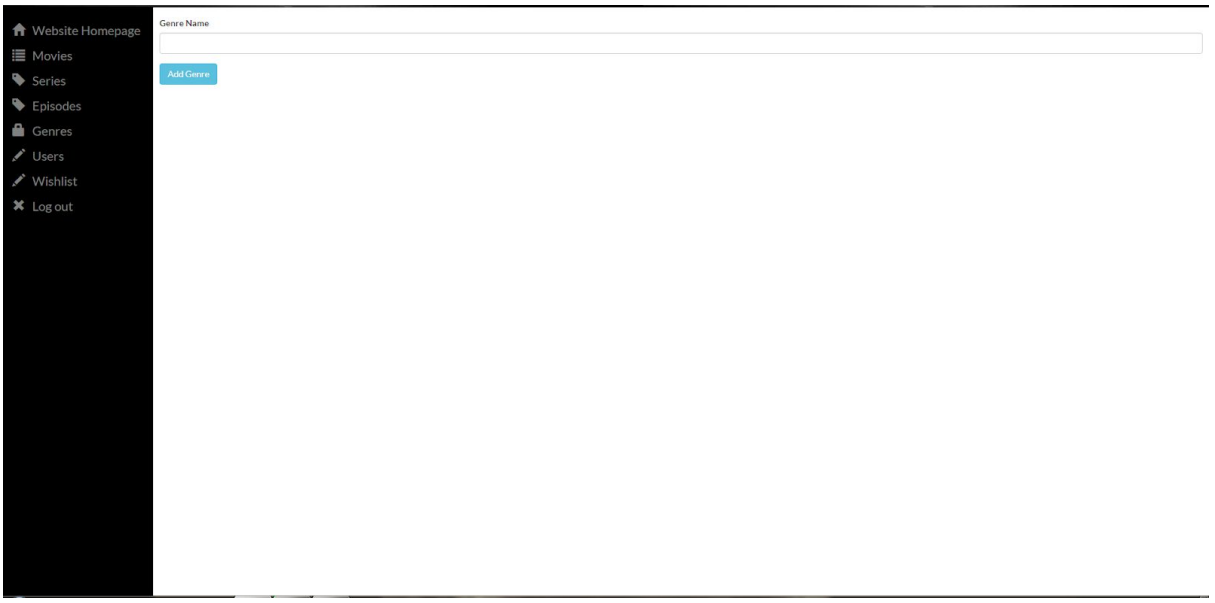


Figure 14: Add Genre Page

Admins can add genres.

Sql Statements

INSERT INTO genre
VALUES (@g_name);

3.3.9 Update/Delete Users

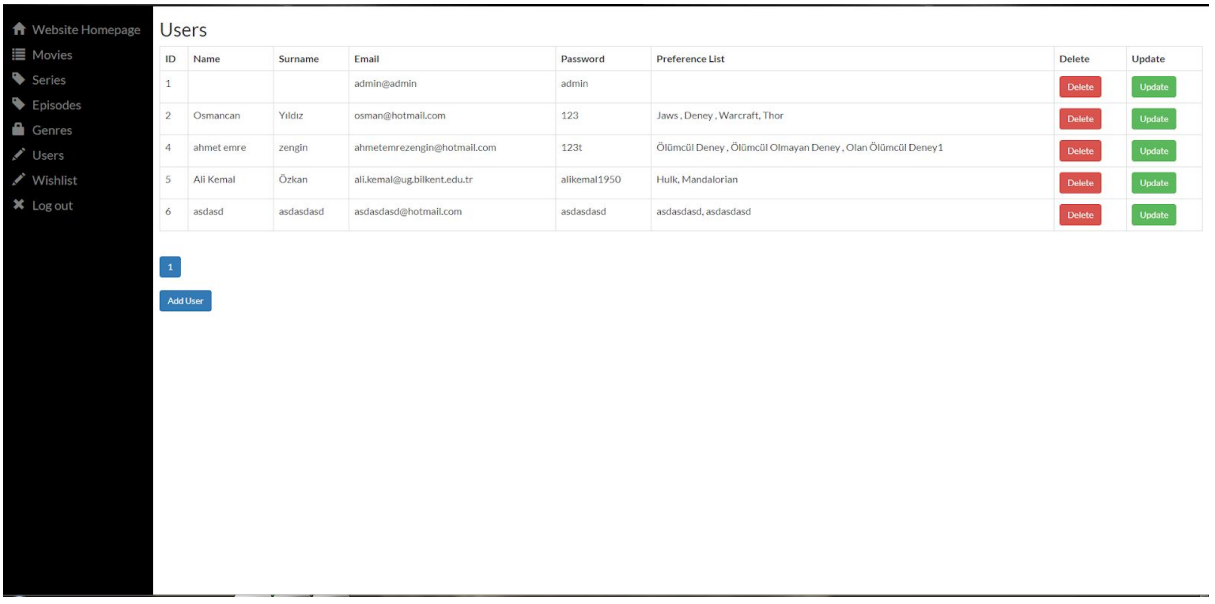


Figure 15: Update/Delete Users Page

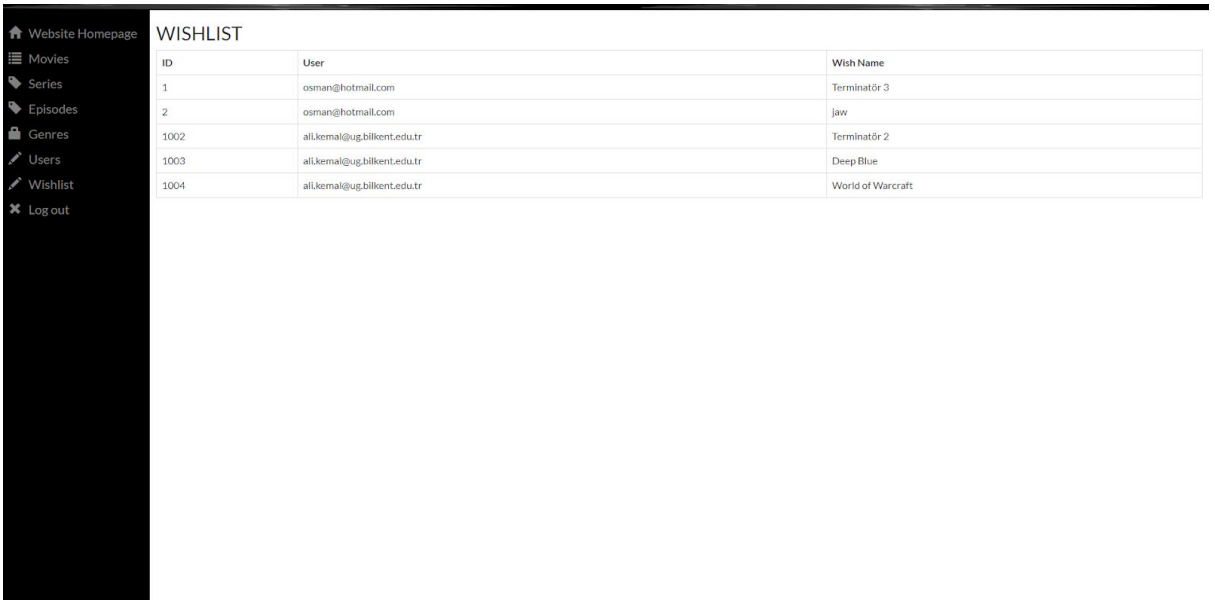
Admins can delete and update users.

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

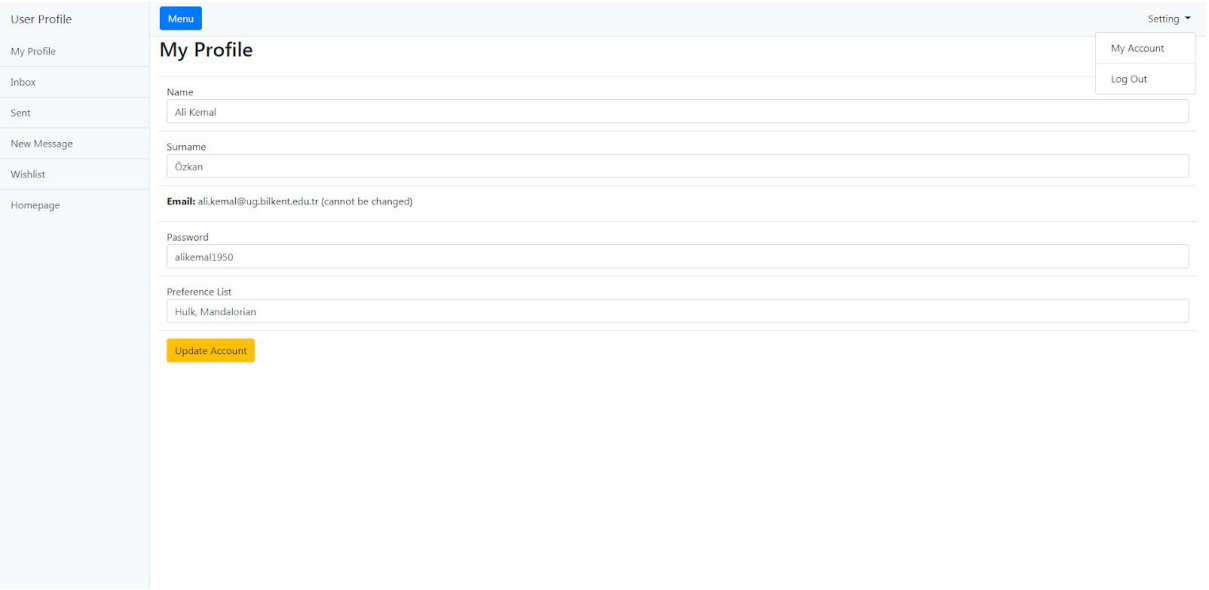


Figure 17: User Profile

Users can update their profiles.

Sql Statements

UPDATE user
SET surname = “Curcunacıgillerdenoğlu”
WHERE g.id = @g_id

3.4.2 User Inbox

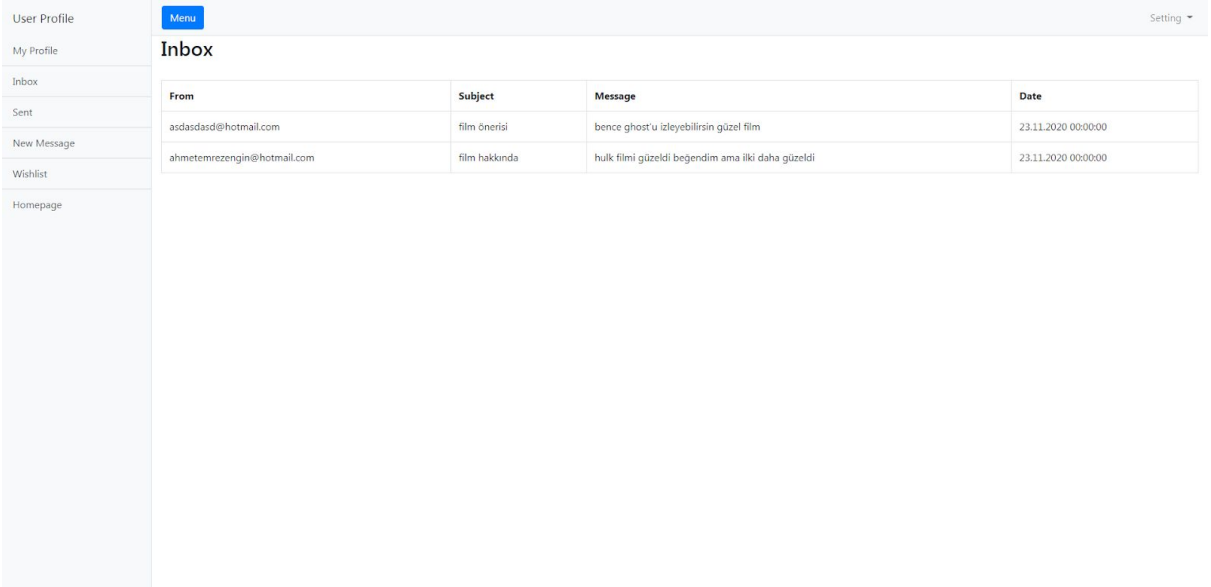


Figure 18: Inbox Page

Users can check their inboxes.

Sql Statements

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

3.4.3 User Sent Messages

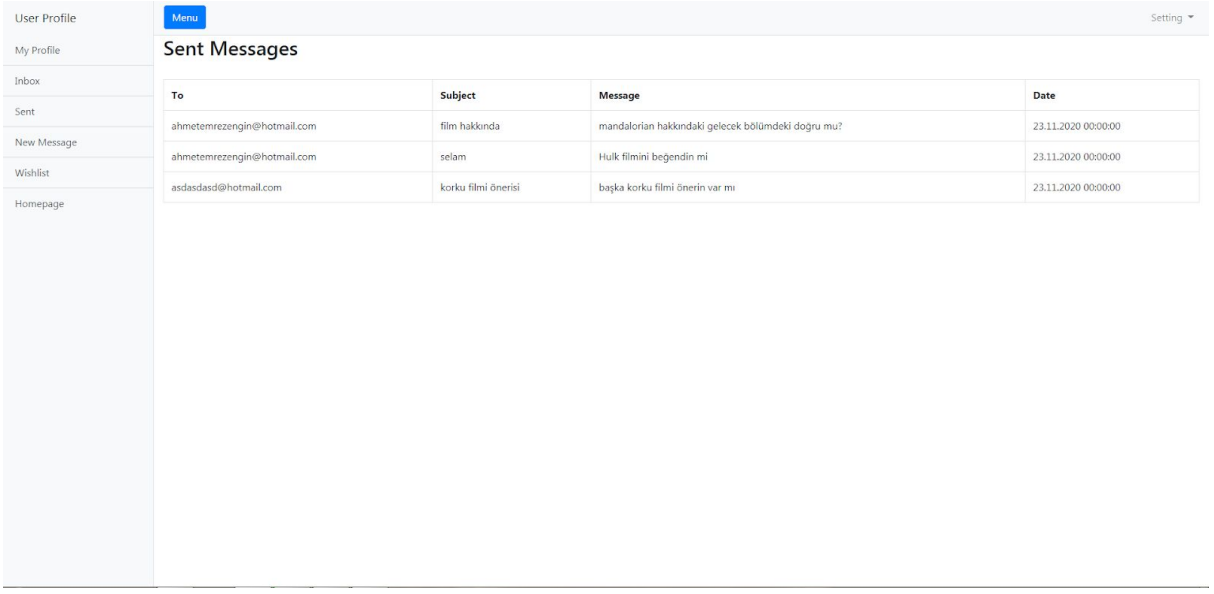


Figure 19: Outbox Page

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

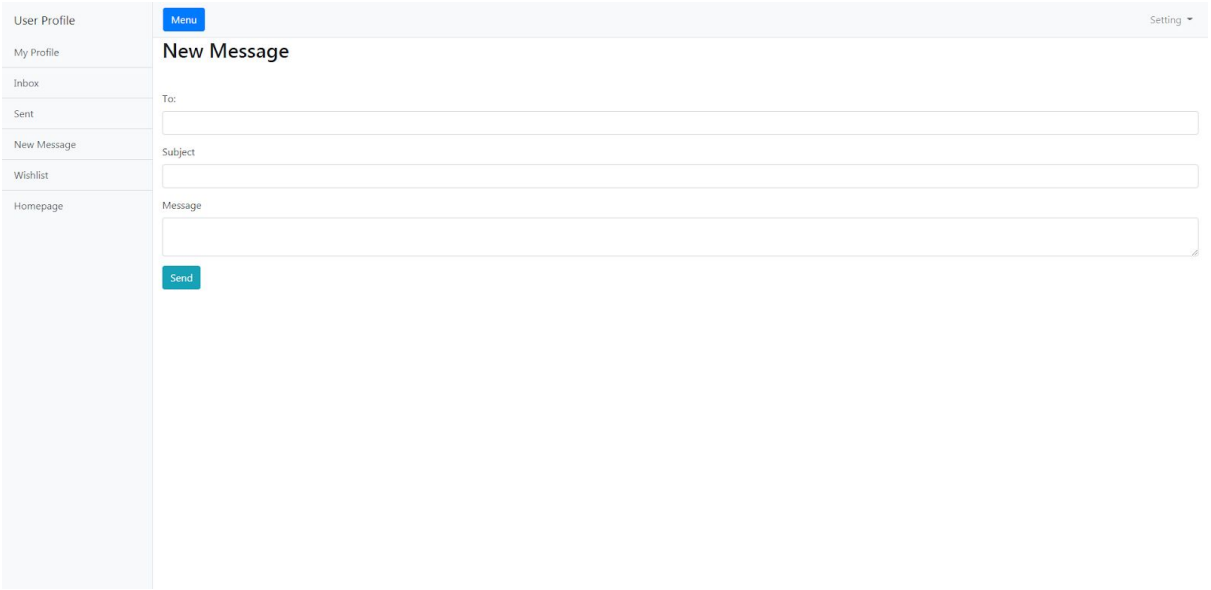


Figure 20: Create New Message Page

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

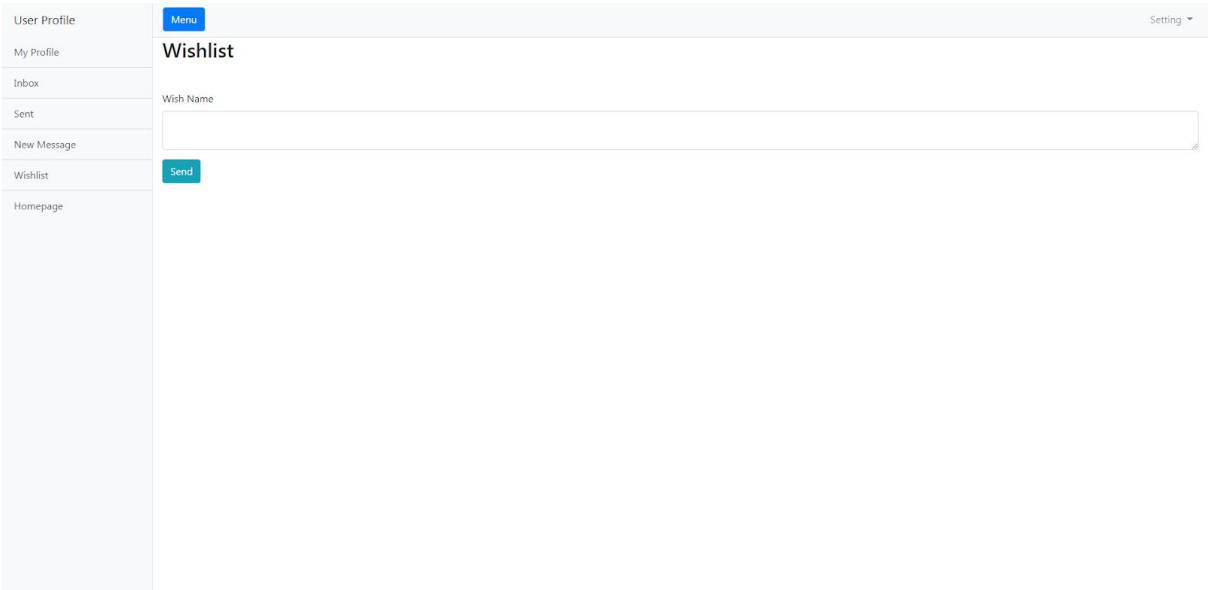


Figure 21: New Wish Page

Users can add new wishes to wishlist.

Sql Statements

INSERT INTO wishlist
VALUES (@w_name);

3.4.6 User Wall

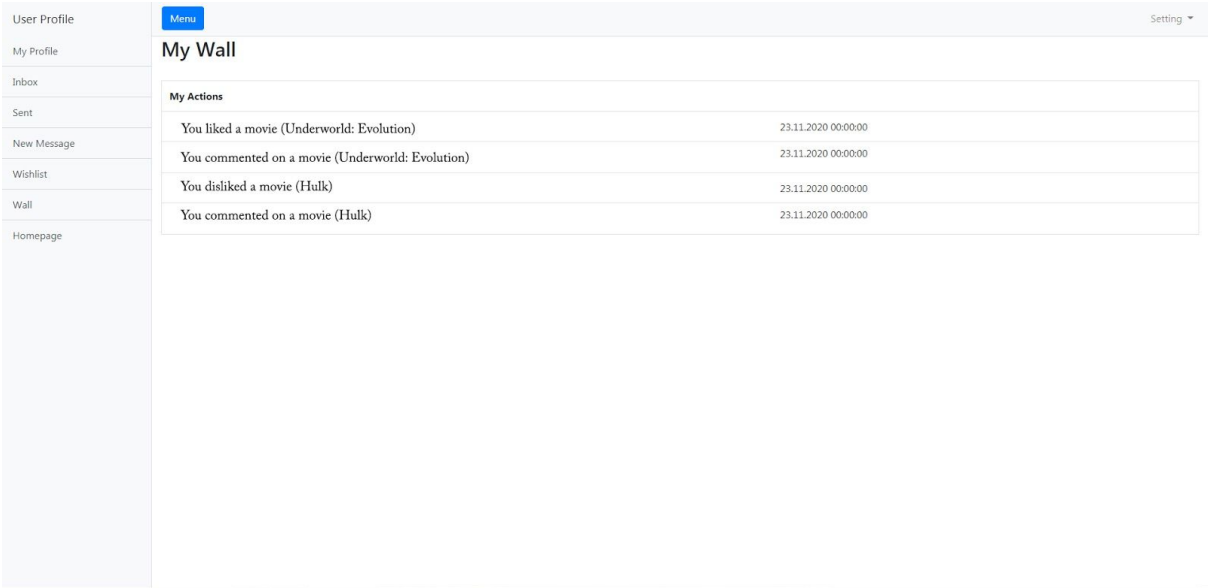


Figure 22: Wall Page

Users can follow their actions by displaying their walls.

3.5 Homepage

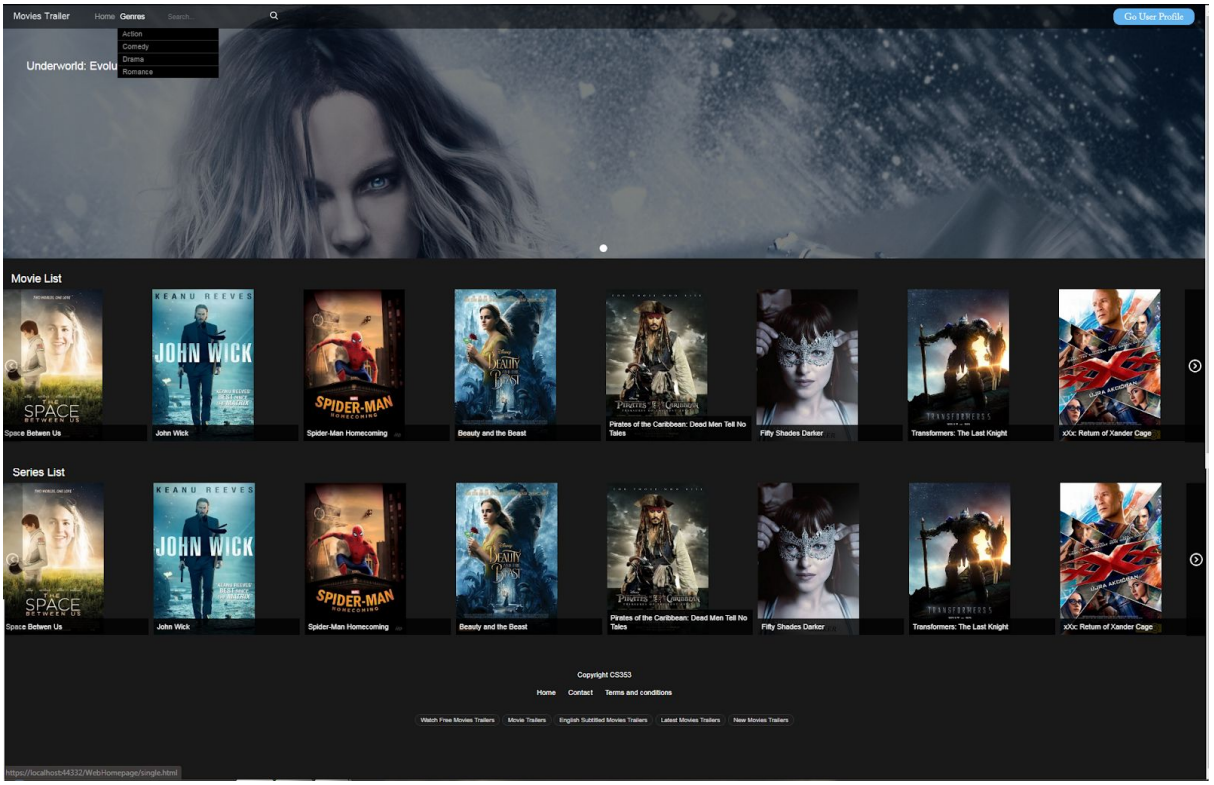


Figure 23: Home Page

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

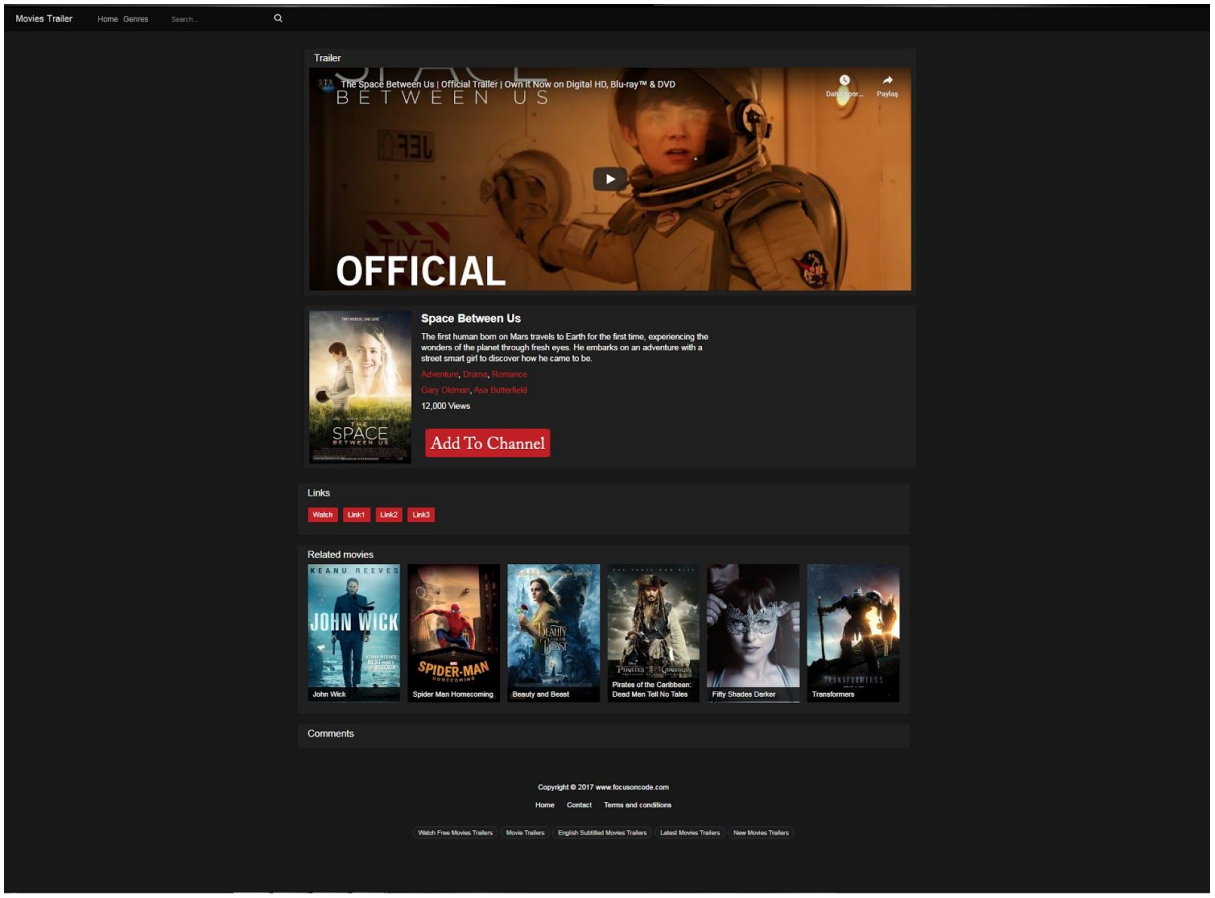


Figure 24: Movie Page

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

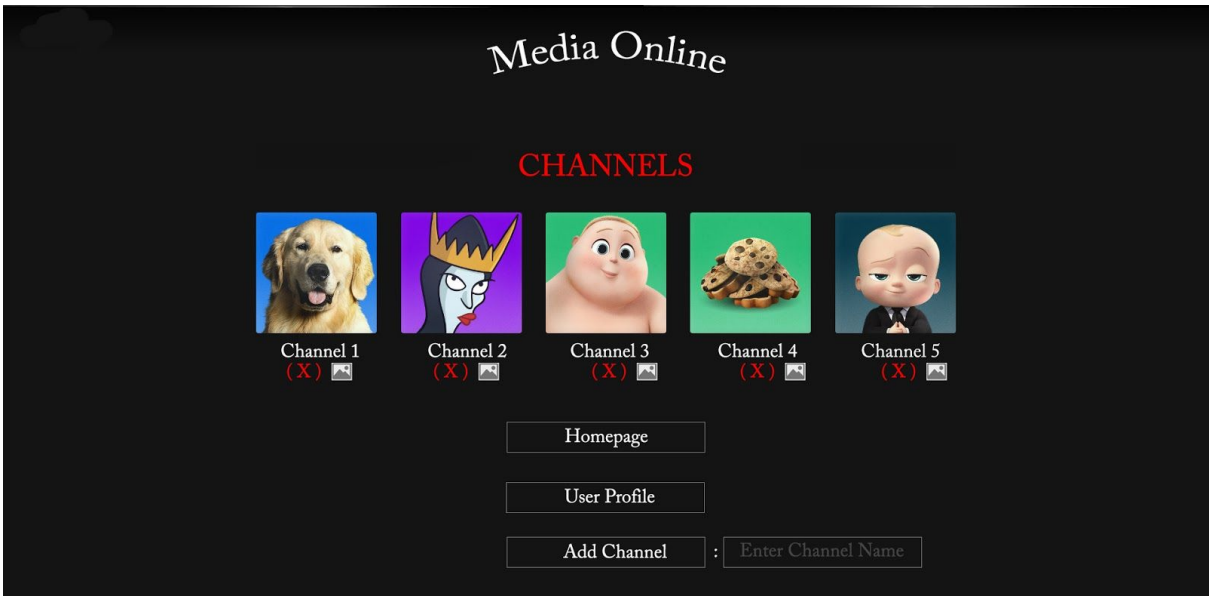


Figure 25: Channels Page

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)
```