



FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

**CS 353**

**Database Systems Design Report**

**Media Services Data Management System**

**Watchflix**

Group 6

Anar Huseynov 21603023

Javid Haji-zada 21701336

Leyla Hashimli 21701341

Leyla Ismayilova 21701299

Date 23.11.2020

## Table of Contents

<b>1. Revised E-R Diagram</b>	<b>7</b>
<b>2. Relational and Table Schemas</b>	<b>8</b>
2.1 User	8
Relational Model	8
Functional Dependencies	8
Candidate Key	9
Primary Key	9
Table Definition	9
2.2 CompanyUser	9
Relational Model	9
Functional Dependencies	9
Candidate Key	9
Primary Key	9
Table Definition	9
2.3 ChannelCreated	9
Relational Model	9
Functional Dependencies	10
Candidate Key	10
Primary Key	10
Table Definition	10
2.4 ChatMessageSent	10
Relational Model	10
Functional Dependencies	10
Candidate Key	10
Primary Key	10
Table Definition	10
2.5 ChatGroup	11

Relational Model	11
Functional Dependencies	11
Candidate Key	11
Primary Key	11
Table Definition	11
2.6 CommentPosted	11
Relational Model	11
Functional Dependencies	11
Candidate Key	11
Primary Key	11
Table Definition	12
2.7 MediaProduct	12
Relational Model	12
Functional Dependencies	12
Candidate Key	12
Primary Key	12
Table Definition	12
2.8 Series	12
Relational Model	12
Functional Dependencies	13
Candidate Key	13
Primary Key	13
Table Definition	13
2.9 Episode	13
Relational Model	13
Functional Dependencies	13
Candidate Key	13
Primary Key	13
Table Definition	13

2.10 Movie	14
Relational Model	14
Functional Dependencies	14
Candidate Key	14
Primary Key	14
Table Definition	14
2.11 Genre	14
Relational Model	14
Functional Dependencies	14
Candidate Key	14
Primary Key	14
Table Definition	14
2.12 watch	15
Relational Model	15
Functional Dependencies	15
Candidate Key	15
Primary Key	15
Table Definition	15
2.13 prefers	15
Relational Model	15
Functional Dependencies	15
Candidate Key	16
Primary Key	16
Table Definition	16
2.14 like-dislike	16
Relational Model	16
Functional Dependencies	16
Candidate Key	16
Primary Key	16

Table Definition	16
2.15 friend	17
Relational Model	17
Functional Dependencies	17
Candidate Key	17
Primary Key	17
Table Definition	17
2.16 memberOf	17
Relational Model	17
Functional Dependencies	17
Candidate Key	17
Primary Key	17
Table Definition	18
2.17 belongsTo	18
Relational Model	18
Functional Dependencies	18
No functional dependencies	18
Candidate Key	18
Primary Key	18
Table Definition	18
2.18 contain	18
Relational Model	18
Functional Dependencies	18
No functional dependencies	19
Candidate Key	19
Primary Key	19
Table Definition	19
2.19 rate	19
Relational Model	19

Functional Dependencies	19
Candidate Key	19
Primary Key	19
Table Definition	19
<b>3. Functional Dependency and Normalization of Tables</b>	<b>20</b>
<b>4. Functional Components</b>	<b>20</b>
4.1 Use Cases / Scenarios	20
<b>5 User Interface Design and Corresponding SQL Statements</b>	<b>23</b>

## 1. Revised E-R Diagram

After getting feedback on our reviewed Project Proposal from the Teaching Assistant, we revised our ER Diagram. We made necessary changes such that it suits better to our Media Management Data Management System. Here are some changes we have made:

- We renamed Media Products to MediaProduct and Movies to Movie.
- We renamed the nick attribute of User to username.
- We combined “like” and “dislike” relationships into a single “like-dislike” relationship between User and MediaProduct.
- We introduced “ChatGroup” and “ChatMessage” entities. We made a ternary relationship called chat between entities of User, ChatGroup and ChatMessage.
- We introduced Genre entity and created a relationship with MediaProduct called belongsTo.
- We removed the “{preferences}” attribute of User and instead we created the relationship called “prefers” between entities of User and Genre.
- Recursive “friend” relationship on our User entity are made many-to-many.
- We removed the “comment” relationship and created the “Comment” Entity and ternary relationship of “commented” between entities of User, Comment and MediaProduct.
- We removed the “type” attribute from Channel.
- We replaced “Episode” entity with “Series” entity and created a weak entity of “Episode” and the identifying relationship of “has” between entities of “Series” and “Episode”.
- We removed duration and film-duration of Episode and Movie entities correspondingly, as we can get this data from the front-end component.
- We added a Company User entity and created a relationship published between Company user and Media Product.
- We added the rate relationship between Media product and User. Rate relationship has an attribute of rated-score which has a range of 1-10.
- We also added a replied-comment-id attribute to the Comment entity to add reply functionality to our application.

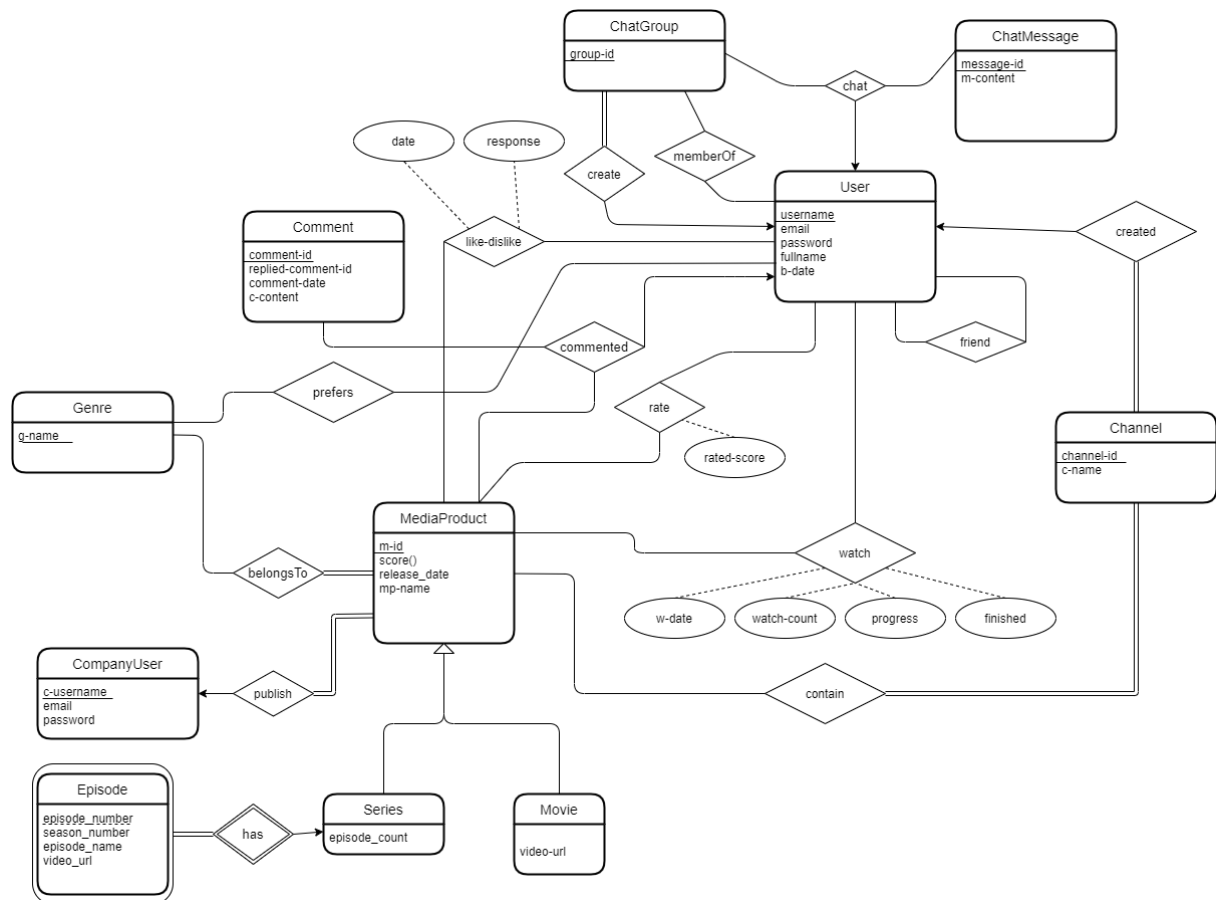


Figure 1. Revised ER Diagram

## 2. Relational and Table Schemas

### 2.1 User

#### Relational Model

User(username, email, password, fullname, b-date)

#### Functional Dependencies

username  $\rightarrow$  email, password, fullname, b-date



### **Candidate Key**

{{username}}

### **Primary Key**

(username)

### **Table Definition**

```
CREATE TABLE User(  
    username    VARCHAR(16) UNIQUE NOT NULL,  
    email       VARCHAR(60) NOT NULL,  
    password    VARCHAR(12) NOT NULL,  
    fullname    VARCHAR(40) NOT NULL,  
    b-date     DATE NOT NULL,  
    PRIMARY KEY(username));
```

## **2.2 CompanyUser**

### **Relational Model**

CompanyUser(c-username, email, password)

### **Functional Dependencies**

c-username → email, password

### **Candidate Key**

{{(c-username)}}

### **Primary Key**

(c-username)

### **Table Definition**

```
CREATE TABLE CompanyUser(  
    c-username  VARCHAR(16) UNIQUE NOT NULL,  
    email       VARCHAR(60) NOT NULL,  
    password    VARCHAR(12) NOT NULL,  
    PRIMARY KEY(c-username));
```

## **2.3 ChannelCreated**

### **Relational Model**

ChannelCreated(username, channel-id, c-name)

username: Foreign key to User (username)

### Functional Dependencies

channel-id  $\rightarrow$  c-name, username

### Candidate Key

{(channel-id)}

### Primary Key

(channel-id)

### Table Definition

```
CREATE TABLE Channel(  
    username    VARCHAR(16) NOT NULL,  
    channel-id  CHAR(12) UNIQUE NOT NULL,  
    c-name      VARCHAR(24) NOT NULL,  
    PRIMARY KEY(channel-id)  
    FOREIGN KEY (username) references User (username));  
ENGINE = InnoDB;
```

## 2.4 ChatMessageSent

### Relational Model

ChatMessageSent(group-id, username, message-id, m-content)

username: Foreign key to User (username)

group-id: Foreign key to ChatGroup (group-id)

### Functional Dependencies

message-id  $\rightarrow$  m-content

### Candidate Key

{(message-id)}

### Primary Key

(message-id)

### Table Definition

```
CREATE TABLE ChatMessage(  
    group-id    CHAR(12) NOT NULL,  
    username    VARCHAR(16) NOT NULL,  
    message-id  CHAR(12) UNIQUE NOT NULL,  
    m-content    VARCHAR(300) NOT NULL,  
    PRIMARY KEY(message-id)  
    FOREIGN KEY (group-id) REFERENCES GroupChat (group-id));
```

ENGINE = InnoDB;

## 2.5 ChatGroup

### Relational Model

ChatGroup( group-id, creatorName)

creatorName: Foreign key to User (username)

### Functional Dependencies

group-id  $\rightarrow$  creatorName

### Candidate Key

{{group-id}}

### Primary Key

(group-id)

### Table Definition

```
CREATE TABLE ChatGroup(  
    group-id      CHAR(12) UNIQUE NOT NULL,  
    creatorName   VARCHAR(16) NOT NULL,  
    PRIMARY KEY(group-id)  
    FOREIGN KEY (creatorName) REFERENCES User (username));
```

## 2.6 CommentPosted

### Relational Model

CommentPosted(comment-id, replied-comment-id, m-id, username, comment-date, c-content)

username: Foreign key to User (username)

m-id: Foreign key to MediaProduct (m-id)

### Functional Dependencies

comment-id  $\rightarrow$  comment-date, c-content

### Candidate Key

{{comment-id}}

### Primary Key

(comment-id)

### Table Definition

```
CREATE TABLE Comment(  
    m-id          CHAR(12) NOT NULL,  
    username      VARCHAR(16) NOT NULL,  
    comment-id    CHAR(12) UNIQUE NOT NULL,  
    replied-comment-id CHAR(12),  
    comment-date  DATE NOT NULL,  
    c-content     VARCHAR(500) NOT NULL,  
    PRIMARY KEY(comment-id),  
    FOREIGN KEY (username) references User (username),  
    FOREIGN KEY (m-id) references MediaProduct(m-id));  
ENGINE = InnoDB;
```

## 2.7 MediaProduct

### Relational Model

MediaProduct(m-id, publisher, release\_date, mp-name)

publisher: Foreign key to CompanyUser(c-username)

### Functional Dependencies

m-id → publisher, release\_date, mp-name

### Candidate Key

{(m-id)}

### Primary Key

(m-id)

### Table Definition

```
CREATE TABLE MediaProduct(  
    m-id          CHAR(12) UNIQUE NOT NULL,  
    publisher     VARCHAR(16),  
    release_date  DATE NOT NULL,  
    mp-name      VARCHAR(168)  
    PRIMARY KEY(m-id)  
  
    FOREIGN KEY(publisher) REFERENCES CompanyUser(c-username));  
  
ENGINE = InnoDB;
```

## 2.8 Series

### Relational Model

Series( m-id, episode\_count)

m-id: Foreign key to MediaProduct (m-id)

### Functional Dependencies

m-id  $\rightarrow$  episode\_count

### Candidate Key

{(m-id)}

### Primary Key

(m-id)

### Table Definition

```
CREATE TABLE Movie(  
    m-id          CHAR(12) UNIQUE NOT NULL,  
    episode_count INT NOT NULL,  
    PRIMARY KEY(m-id)  
    FOREIGN KEY (m-id) references MediaProducts (m-id));  
ENGINE = InnoDB;
```

## 2.9 Episode

### Relational Model

Episode(m-id, episode\_number, season\_number, episode\_name, video\_url)

m-id: Foreign key to MediaProduct (m-id)

### Functional Dependencies

m-id, episode\_number  $\rightarrow$  season\_number, episode\_name, video\_url

### Candidate Key

{(m-id, episode\_number)}

### Primary Key

(m-id, episode\_number)

### Table Definition

```
CREATE TABLE Episode(  
    m-id          CHAR(12) NOT NULL,  
    season_number NUMERIC(4,0) NOT NULL,  
    episode_number INT NOT NULL,  
    episode_name  VARCHAR(40) NOT NULL,  
    video_url     VARCHAR(256) NOT NULL,  
    PRIMARY KEY(m-id, season_number, episode_number)
```

FOREIGN KEY (m-id) references MediaProducts (m-id));  
ENGINE = InnoDB;

## 2.10 Movie

### Relational Model

Movie(m-id, video-url)

m-id: Foreign key to MediaProduct (m-id)

### Functional Dependencies

m-id → video-url

### Candidate Key

{(m-id)}

### Primary Key

(m-id)

### Table Definition

```
CREATE TABLE Movie(  
    m-id          CHAR(12) UNIQUE NOT NULL,  
    video-url     VARCHAR(256) NOT NULL,  
    PRIMARY KEY(m-id)  
    FOREIGN KEY (m-id) references MediaProducts (m-id));  
ENGINE = InnoDB;
```

## 2.11 Genre

### Relational Model

Genre(g-name)

### Functional Dependencies

No functional dependencies

### Candidate Key

{(g-name)}

### Primary Key

(g-name)

### Table Definition

```
CREATE TABLE Genre(  
    g-name        VARCHAR(20) UNIQUE NOT NULL,
```

PRIMARY KEY(g-name));

## 2.12 watch

### Relational Model

watch(m-id, username, w-date, watch-count, progress, finished)

m-id: Foreign key to MediaProducts (m-id)

username: Foreign key to User (username)

### Functional Dependencies

m-id, username → progress, w-date, watch-count, finished

### Candidate Key

{(m-id, username)}

### Primary Key

(m-id, username)

### Table Definition

```
CREATE TABLE chat(  
    m-id          CHAR(12) NOT NULL,  
    username      VARCHAR(16) NOT NULL,  
    w-date        DATE NOT NULL,  
    progress      TIME NOT NULL,  
    watch-count   INT NOT NULL,  
    finished      BOOLEAN NOT NULL,  
    PRIMARY KEY( m-id, username),  
    FOREIGN KEY(m-id) REFERENCES MediaProduct(m-id),  
    FOREIGN KEY(username) REFERENCES User(username));  
ENGINE = InnoDB;
```

## 2.13 prefers

### Relational Model

prefers(g-name, username)

g-name: Foreign key to Genre (g-name)

username: Foreign key to User (username)

### Functional Dependencies

No functional dependencies

### **Candidate Key**

{(g-name, username)}

### **Primary Key**

(g-name, username)

### **Table Definition**

```
CREATE TABLE chat(  
    g-name      VARCHAR(20) NOT NULL,  
    username    VARCHAR(16) NOT NULL,  
    PRIMARY KEY( g-name, username),  
    FOREIGN KEY(g-name) REFERENCES Genre(g-name),  
    FOREIGN KEY(username) REFERENCES User(username));  
ENGINE = InnoDB;
```

## **2.14 like-dislike**

### **Relational Model**

like-dislike(m-id, username, date, response)

m-id: Foreign key to MediaProduct (m-id)

username: Foreign key to User (username)

### **Functional Dependencies**

### **Candidate Key**

{(m-id, username)}

### **Primary Key**

(m-id, username)

### **Table Definition**

```
CREATE TABLE chat(  
    m-id        CHAR(12)      NOT NULL,  
    username    VARCHAR(16) NOT NULL,  
    date        DATE,  
    response    BIT,  
    PRIMARY KEY( m-id, username),  
    FOREIGN KEY(m-id) REFERENCES MediaProduct(m-id),  
    FOREIGN KEY(username) REFERENCES User(username));  
ENGINE = InnoDB;
```



## 2.15 friend

### Relational Model

friend(username1, username2)

username1: Foreign key to User (username1)

username2: Foreign key to User (username2)

### Functional Dependencies

No functional dependencies

### Candidate Key

{{username1, username2}}

### Primary Key

(username1, username2)

### Table Definition

```
CREATE TABLE chat(  
    username1    VARCHAR(16) NOT NULL,  
    username2    VARCHAR(16) NOT NULL,  
    PRIMARY KEY( username1, username2),  
    FOREIGN KEY(username1) REFERENCES User(username)  
    FOREIGN KEY(username2) REFERENCES User(username));  
ENGINE = InnoDB;
```

## 2.16 memberOf

### Relational Model

memberOf(group-id, username)

group-id: Foreign key to ChatGroup (group-id)

username: Foreign key to User (username)

### Functional Dependencies

No functional dependencies

### Candidate Key

{{group-id, username}}

### Primary Key

(group-id, username)

### Table Definition

```
CREATE TABLE chat(  
    group-id    CHAR(12)    NOT NULL,  
    username    VARCHAR(16) NOT NULL,  
    PRIMARY KEY( group-id, username),  
    FOREIGN KEY(group-id) REFERENCES Group(group-id),  
    FOREIGN KEY(username) REFERENCES User(username));  
ENGINE = InnoDB;
```

## 2.17 belongsTo

### Relational Model

belongsTo(g-name, m-id)

g-name: Foreign key to Genre (g-name)

m-id: Foreign key to MediaProduct (m-id)

### Functional Dependencies

No functional dependencies

### Candidate Key

{{g-name, m-id}}

### Primary Key

(g-name, m-id)

### Table Definition

```
CREATE TABLE chat(  
    g-name      VARCHAR(20) NOT NULL,  
    m-id        CHAR(12) NOT NULL,  
    PRIMARY KEY( g-name, m-id),  
    FOREIGN KEY(g-name) REFERENCES Genre(g-name),  
    FOREIGN KEY(m-id) REFERENCES MediaProduct(m-id));  
ENGINE = InnoDB;
```

## 2.18 contain

### Relational Model

contain(channel-id, m-id)

channel-id: Foreign key to Channel (channel-id)

m-id: Foreign key to MediaProduct (m-id)

### Functional Dependencies

No functional dependencies

### Candidate Key

{(channel-id, m-id)}

### Primary Key

(channel-id, m-id)

### Table Definition

```
CREATE TABLE chat(  
    channel-id    CHAR(12)    NOT NULL,  
    m-id          CHAR(12)    NOT NULL,  
    PRIMARY KEY( channel-id, m-id),  
    FOREIGN KEY(channel-id) REFERENCES Genre(channel-id),  
    FOREIGN KEY(m-id) REFERENCES MediaProduct(m-id));  
ENGINE = InnoDB;
```

## 2.19 rate

### Relational Model

rate(username, m-id, rated-score)

username: Foreign key to User(username)

m-id: Foreign key to MediaProduct (m-id)

### Functional Dependencies

username, m-id  $\rightarrow$  rated-score

### Candidate Key

{(m-id,username)}

### Primary Key

(m-id,username)

### Table Definition

```
CREATE TABLE chat(  
    username      VARCHAR(16) NOT NULL,  
    m-id          CHAR(12)    NOT NULL,  
    rated-score    Numeric(3,1) NOT NULL,  
    PRIMARY KEY(m-id,username),  
    FOREIGN KEY(username) REFERENCES User(username),  
    FOREIGN KEY(m-id) REFERENCES MediaProduct(m-id));
```

ENGINE = InnoDB;

### 3. Functional Dependency and Normalization of Tables

After getting Relational Schemas from ER Diagram, all the functional dependencies are in Boyce-Codd Normal Form (BCNF). So, no later normalization or decomposition has been done.

## 4. Functional Components

### 4.1 Use Cases / Scenarios

**Register:** Users can register into the system by specifying a username (unique), email, password, their full names and birthdates. During registration they should also specify their genre preference.

**Prefer Genre:** Users can specify their preferred genres to get related suggestions.

**Login:** Users can login to their previously created accounts with their username and password.

**Watch Movie:** Users can watch movies they choose from the list.

**Watch Episode:** Users can watch episodes of series that are available in the application.

**Create Channel:** Users can create channels and name them to categorize Media Products based on their choice.

**Add Media product to channel:** Users can add Media Products to channels they created before.

**Like/Dislike Media Product:** Users can like or dislike Media Products to give feedback.

**Create Group Chat:** Users can create group chats to watch movies together and their friends can join these chats.

**Join Group Chat:** Users can join group chats created by their friends to discuss while watching.

**Send Chat Message:** Users can send messages to members of the group in chats.

**Add Friend:** Users can add friends by accepting their friend request or by sending a request to them.

**Comment on Media product:** Users can comment on MediaProducts.

**Filter Media:** Standard users can filter media products by their genres, scores and release-dates. Company users can also filter Media Products they uploaded.

**Search Media:** Users can search media products and series by their names. Company users can also search Media Products they uploaded.

**Sort Uploaded Media:** Company users can sort Media products they uploaded

**Public Media Product:** Company users can publish Media Products.

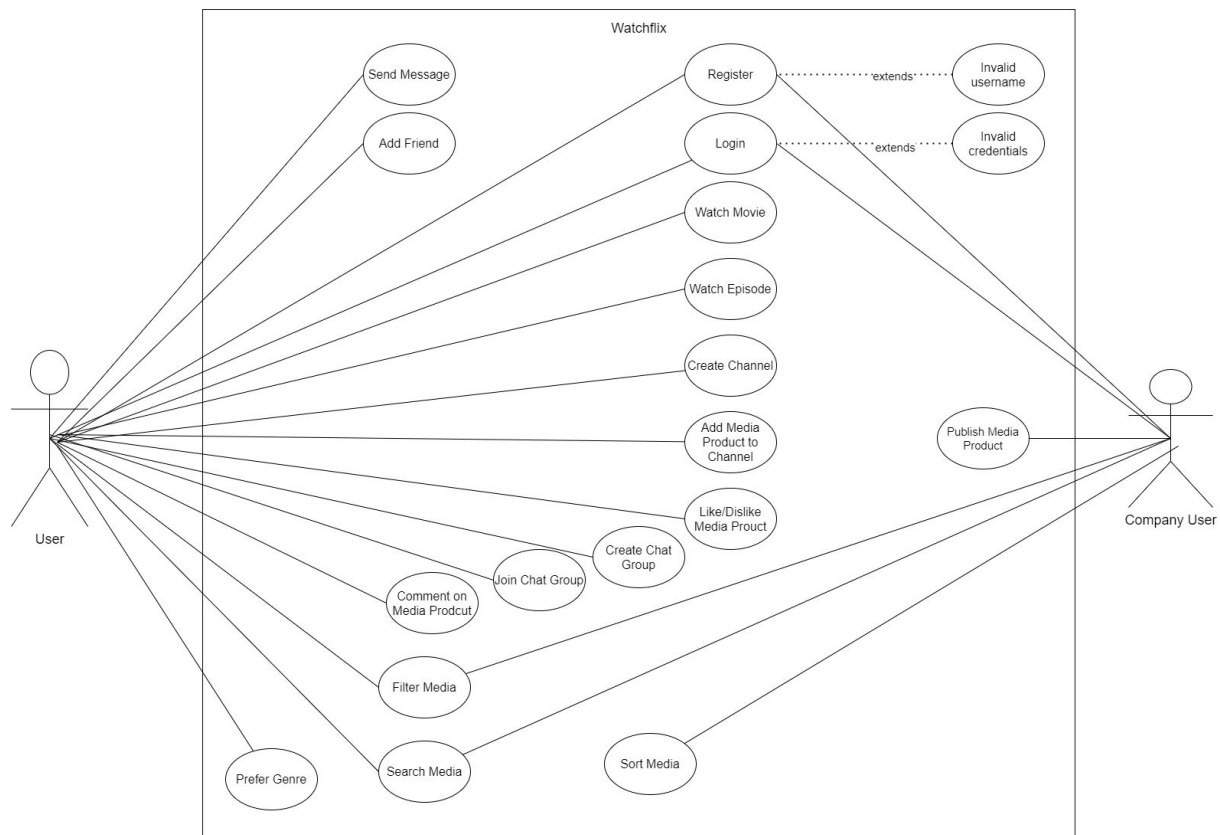


Figure 2: Use Case Diagram

## 4.2 Algorithms

### 4.2.1 Media Product Suggestion Algorithm

The application will suggest media products to users based on their genre preferences. A user can prefer more than one genre. Suggested media products are going to be found by list search algorithm. Products with highest scores will be suggested.

Firstly products that have at least one of the genres preferred are going to be found by list search algorithm. Then they are going to be sorted by quicksort algorithm. Then, the top 21 products are going to be suggested to the user.

#### **4.2.2 Filtering and Search Algorithm**

Our application will offer a filtering function to its users and by using this functionality users will be able to see Media Products filtered according to specified features. Users can filter media products based on genre, release date and score criteria. If the genre criterion is not specified all genres will be valid, if the release date is not specified all products released after 1895th year will be filtered since it is the date for the first Media Product and if the score criterion is not specified all products with the score more than 0 will be chosen. Search Algorithm will filter all products whose name contains the searched string.

## 5 User Interface Design and Corresponding SQL Statements



Figure 1: Register page



Figure 2: Choose User Type

The screenshot shows a web browser window titled "Watchflix" with the URL "https://watchflix.com.tr". The page features the "Watchflix" logo and the tagline "Just a few steps to enjoy movie night". A red-bordered box contains the registration form. The form is divided into two columns. The left column has three sections: "Enter your name:" with a "Name" input field, "Enter your Surname:" with a "Surname" input field, and "Birth Date:" with a "Date" input field and a calendar icon. The right column has three sections: "Create your username:" with a "Username" input field and a checkmark icon, "Create your password:" with a "Password" input field, and "Email address:" with an "E-mail" input field. Below the input fields is a red "Next" button.

Watchflix

https://watchflix.com.tr

# Watchflix

Just a few steps to enjoy movie night

Enter your name:  
Name

Enter your Surname:  
Surname

Birth Date:  
Date

Create your username:  
Username ✓  
Your username should be unique.

Create your password:  
Password  
Your password should be at least 8 characters.

Email address:  
E-mail

Next

Figure 3: Standard User Registration Form

The screenshot shows the same web browser window as Figure 3, but the registration form is at Step 2. The red-bordered box contains the "Choose your preferences:" section, which is a list of movie genres and types with checkboxes. The genres are arranged in three columns: Sci-fi, Musicals, Fantasy, Classics, Fantasy; Comedy, Horror, Dramas, Children, Romance; and Documentaries, Award winning, Biography. Below the list is a red "Register me" button.

Watchflix

https://watchflix.com.tr

# Watchflix

Just a few steps to enjoy movie night

Choose your preferences:

<input type="checkbox"/> Sci-fi	<input type="checkbox"/> Comedy	<input type="checkbox"/> Documentaries
<input type="checkbox"/> Musicals	<input type="checkbox"/> Horror	<input type="checkbox"/> Award winning
<input type="checkbox"/> Fantasy	<input type="checkbox"/> Dramas	<input type="checkbox"/> Biography
<input type="checkbox"/> Classics	<input type="checkbox"/> Children	
<input type="checkbox"/> Fantasy	<input type="checkbox"/> Romance	

Register me



Figure 4: Choose Genre

The screenshot shows a web browser window titled "Watchflix" with the URL "https://watchflix.com.tr". The page features the "Watchflix" logo in red and the tagline "Just a few steps to enjoy movie night". A registration form is highlighted with a red border. The form contains the following fields and instructions:

- Company name:** A text input field labeled "Name".
- Email address:** A text input field labeled "E-mail".
- Create your username:** A text input field labeled "Username" with a checkmark icon to its right. Below it, the text "Your username should be unique." is displayed.
- Create your password:** A text input field labeled "Password". Below it, the text "Your password should be at least 8 characters." is displayed.

A red "Register me" button is located at the bottom of the form.

Figure 5: Company User Registration Form

**FOR STANDARD USER:**

**Inputs:** @Username, @Name, @Surname, @Email, @Password, @BirthDate, @Genres

**Process:** The register page will be shown for the first time standard users. Personal information of full-name, email, password, birth-date and a unique username will be used to create an account.

**SQL Statements:**

```
INSERT INTO User(username, email, password, fullname, b-date)
VALUES(@Username, @Email, @Password, @Name+@Surname, @BirthDate);

DECLARE

    a VARCHAR(20);

BEGIN

    FOR a in @Genre LOOP

        INSERT INTO prefers(g-name, username)
```

```
VALUES(a, @UserName);

END LOOP;

END;
```

#### FOR COMPANY USER:

**Inputs:** @Username, @Email, @Password

**Process:** The register page will be shown for the first time company users. Personal information of email, password and a unique username will be used to create an account.

#### SQL Statements:

```
INSERT INTO CompanyUser(username, email, password)

VALUES(@Username, @Email, @Password);
```



Figure 6: Sign in Page

#### FOR STANDARD USER:

**Inputs:** @Username, @Password

**Process:** Username and password will be used to sign in to registered accounts.

#### SQL Statements:

```
SELECT *
```

```
FROM User
```

```
WHERE(@Username = User.username AND @Password = User.password);
```

#### FOR COMPANY USER:

**Inputs:** @username, @password

**Process:** Username and password will be used to sign in to registered company accounts.

#### SQL Statements:

```
SELECT *
```

```
FROM CompanyUser
```

```
WHERE(@username = CompanyUser.username AND @Password =  
CompanyUser.password);
```

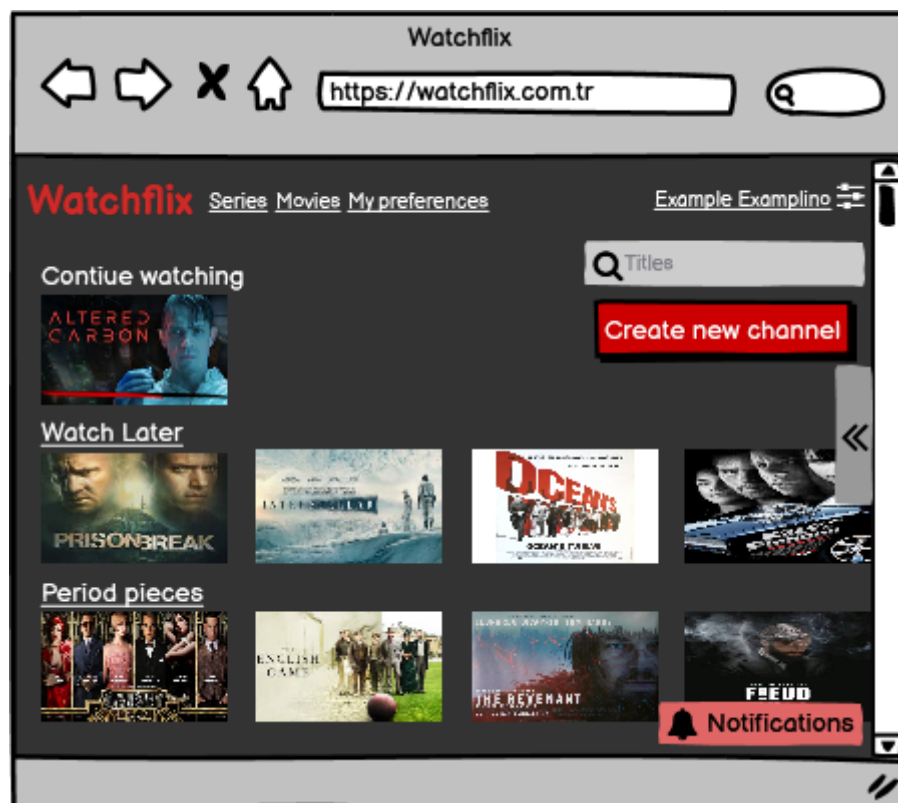


Figure 7: Standard User Main Page

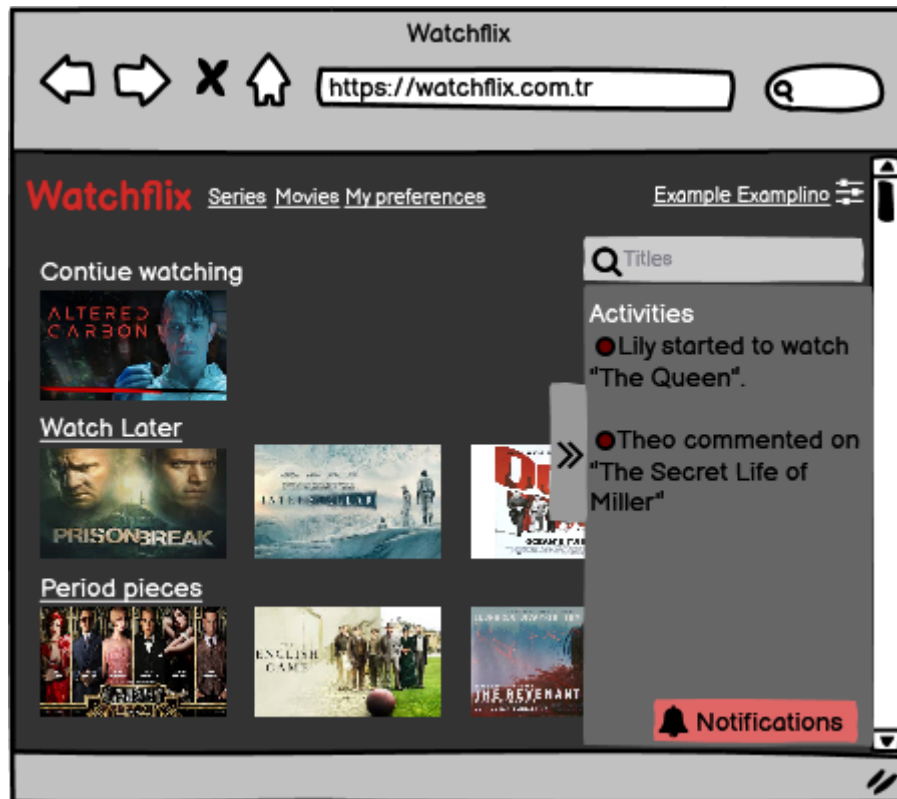


Figure 8: Standard User Activities Bar

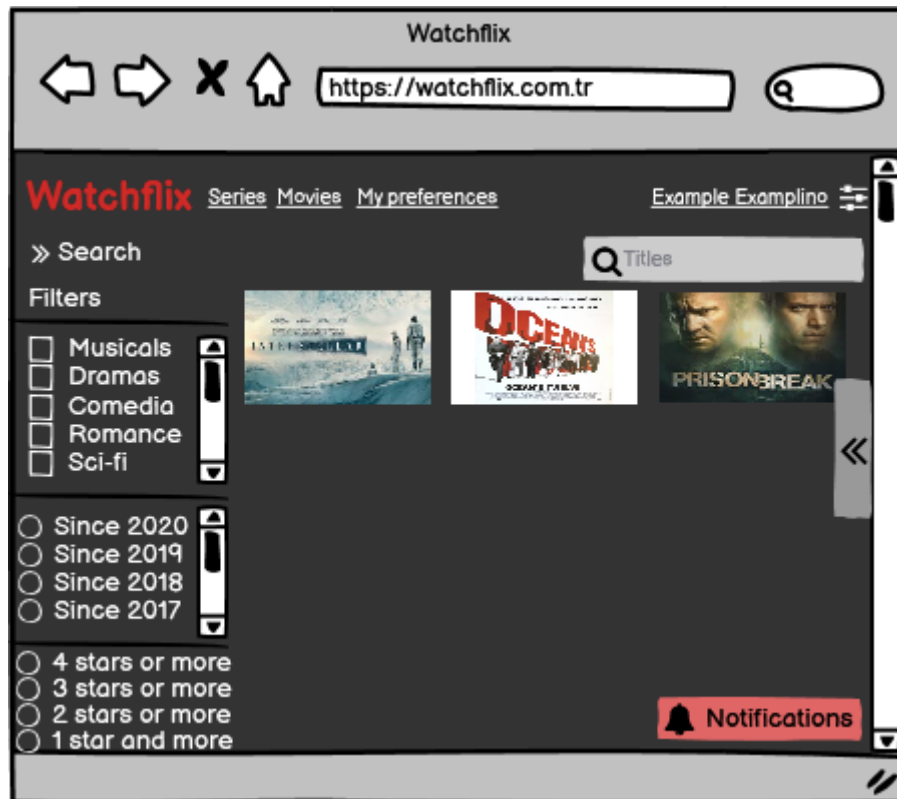


Figure 9: Filtering and Search for Standard User

#### SEARCHING:

**Inputs:** @mediaProductName

**Process:** media products with specified name will be found. Also, media products whose name contain the specified string will be displayed.

#### SQL Statements:

```
SELECT *
FROM MediaProduct
WHERE MediaProduct.mp-name CONTAINS (mp-name,
'*@mediaProductsName*')
```

#### FILTERING:

**Inputs:** @genre, @score, @release-date

**Process:** will filter media products based on specified criteria

#### SQL Statements:

```
SELECT *
FROM MediaProduct
```

```

WHERE score >= @score AND release-date >= @release-date AND
EXISTS (SELECT *
FROM belongsTo
WHERE belongsTo.g-name = @genre AND belongsTo.m-id =
MediaProduct.m-id);

```

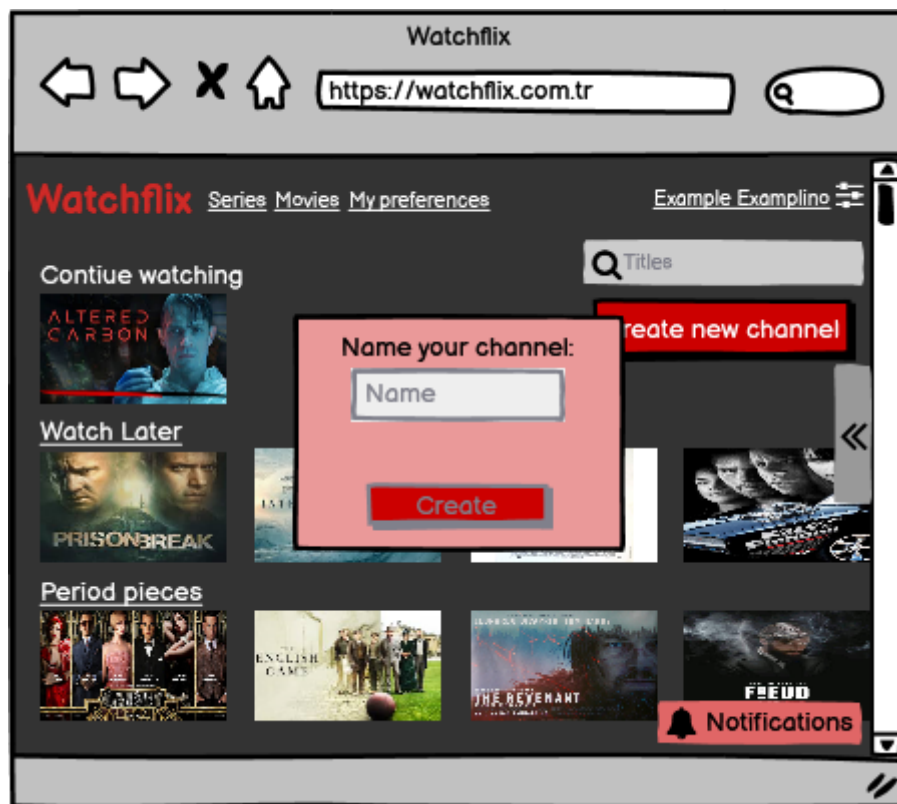


Figure 10: New Channel Pop-Up

**Inputs:** @username, @channelID, @channelname

**Process:** creating new channel

**SQL Statements:**

```

INSERT INTO ChannelCreated(username, channel-id, c-name)
VALUES (@username, @channelID, @channelname);

```

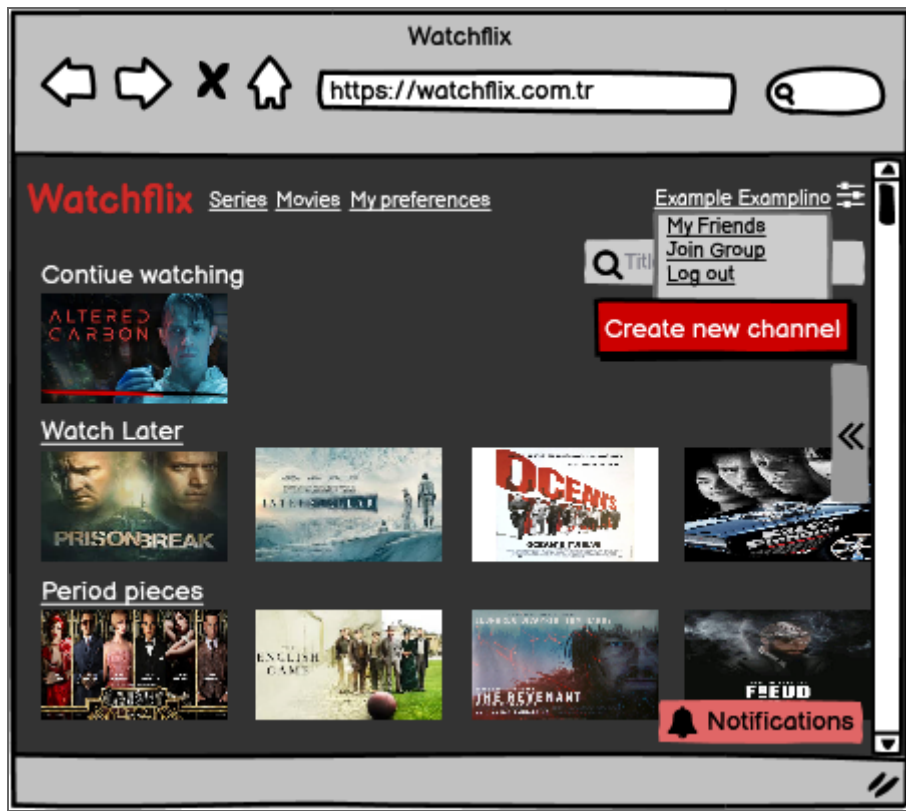


Figure 11: Hover on username

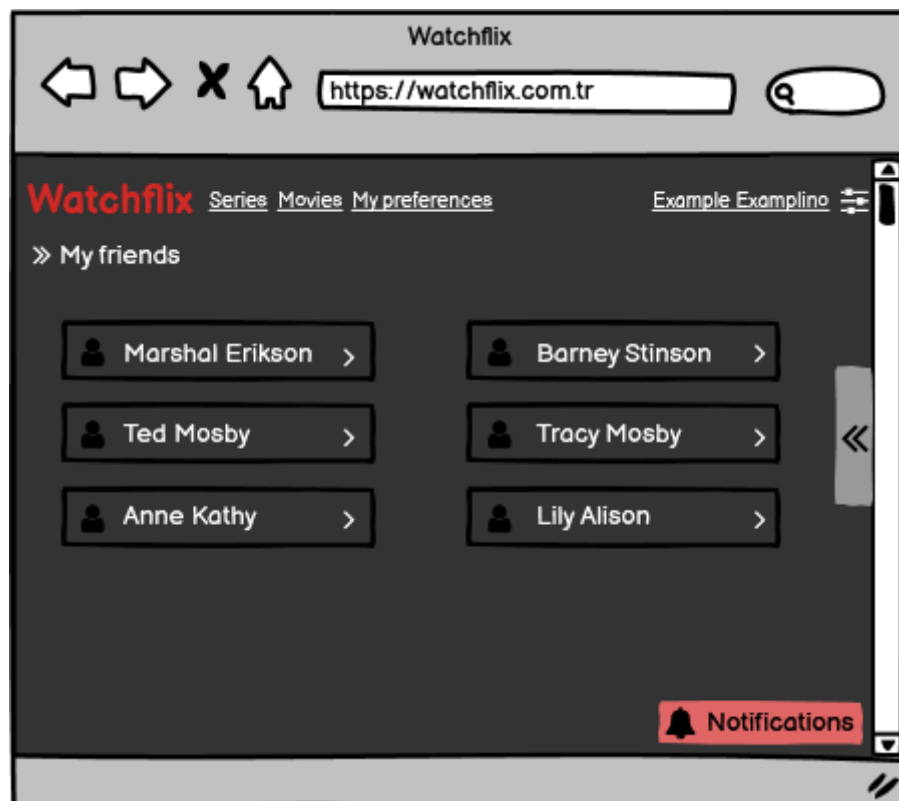


Figure 12: Friends List

**Inputs:** @username

**Process:** see the friend list

**SQL Statements:**

```
SELECT username2 AS friends(username)
FROM friend
WHERE friend.username1 = @username
SELECT *
FROM friends NATURAL JOIN User
```

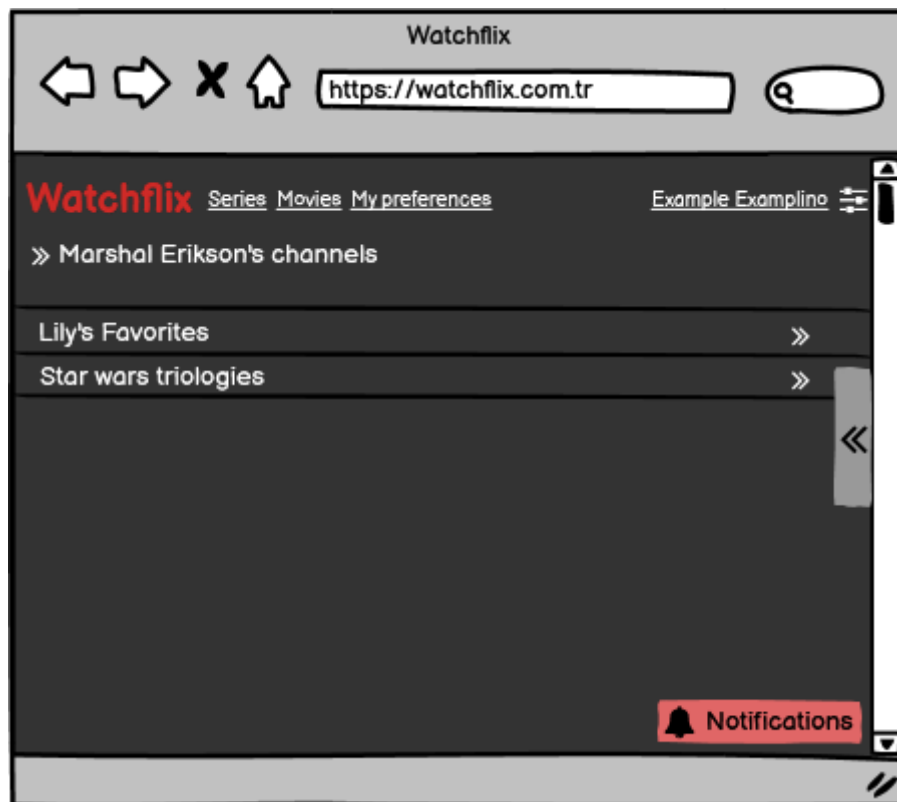


Figure 13: Friend's channels

**Inputs:** @friend\_username

**Process:** see the channels of your friend

**SQL Statements:**

```
SELECT channel-id, c-name
FROM ChannelCreated
WHERE ChannelCreated.username = @friend_username
```



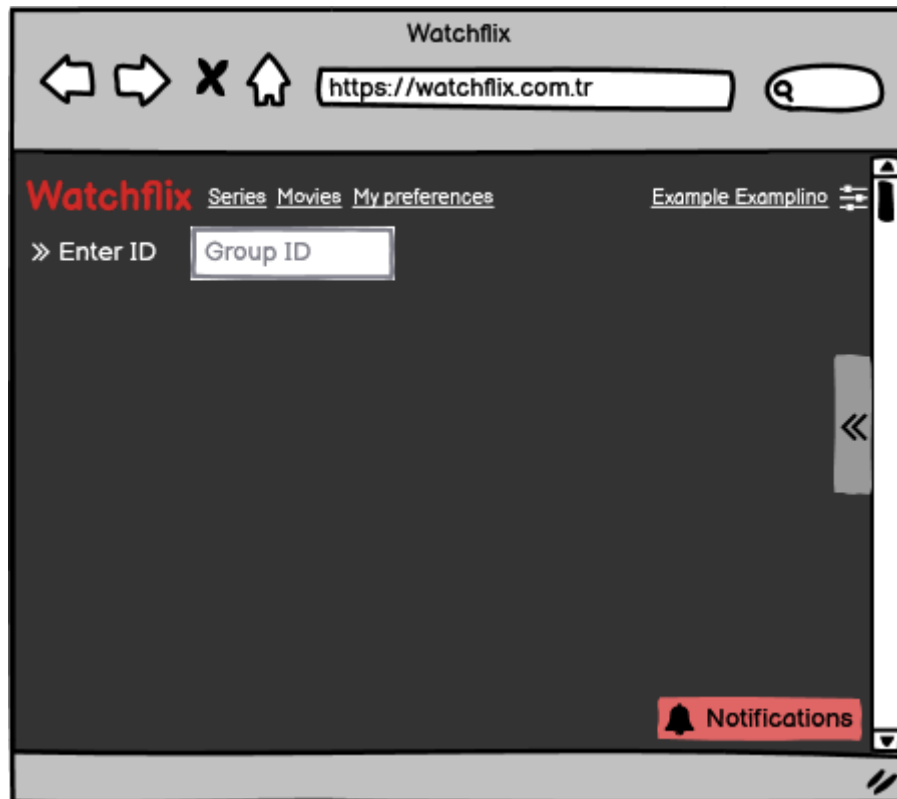


Figure 14: Find Group

**Inputs:** @group-id

**Process:** find group that your friend created

**SQL Statements:**

```
SELECT *
```

```
FROM ChatGroup
```

```
WHERE (ChatGroup.group-id = @GroupID);
```

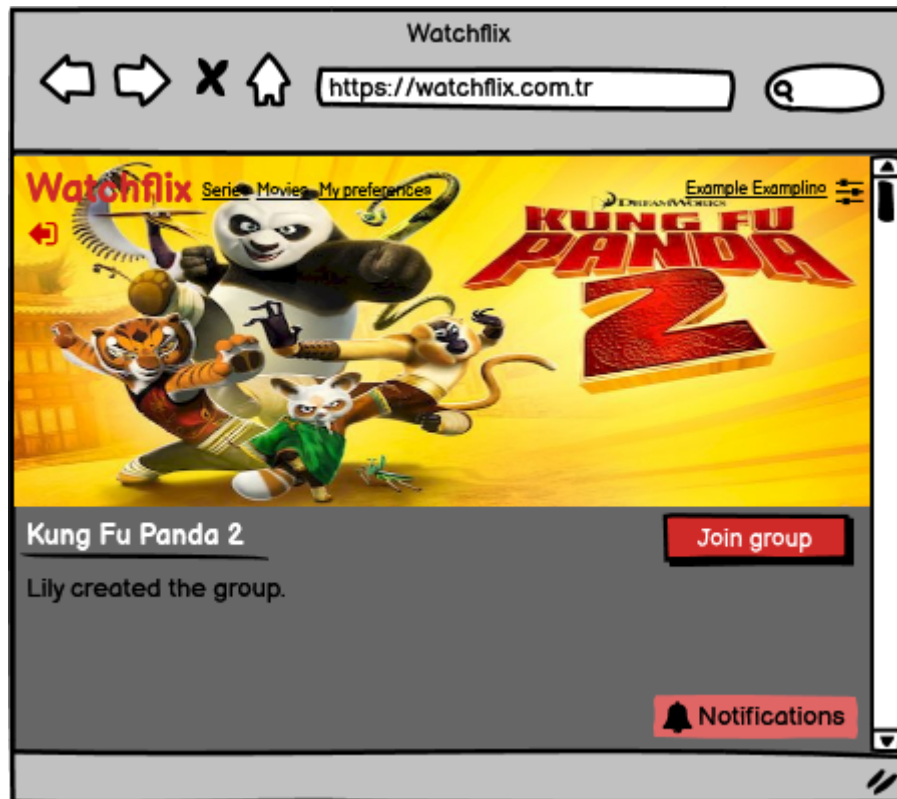


Figure 15: Join Group

**Inputs:** @groupID, @username,

**Process:** join an already created group by its id.

**SQL Statements:**

```
SELECT username2 AS friends(creatorName)
```

```
FROM friend
```

```
WHERE friend.username1 = @username;
```

```
INSERT INTO memberOf(group-id, username)
```

```
VALUES(@groupID, @username)
```

```
WHERE @groupID in
```

```
(SELECT group-id
```

```
FROM friends NATURAL JOIN ChatGroup);
```

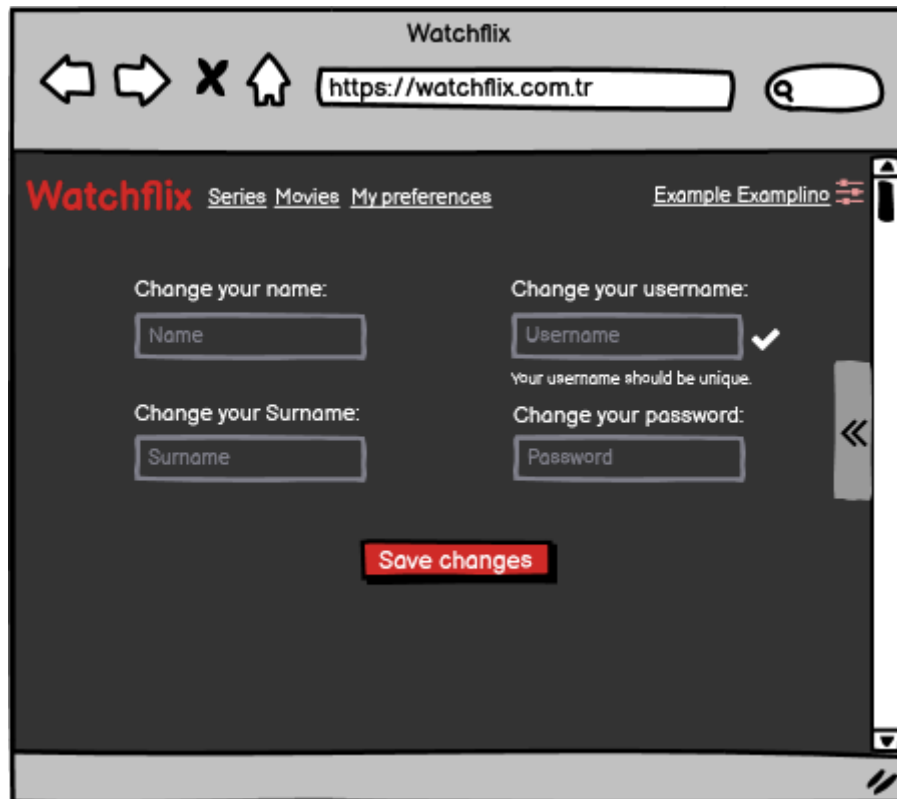


Figure 16: Update credentials

**Inputs:** @Username, @newname, @newsurname, @newusern, @newpassword

**Process:** List suggestions according to user's preference

**SQL Statements:**

```
UPDATE User
```

```
SET User.username = @newusern, User.fullname = @newname +  
@newsurname, User.password = @newpassword
```

```
WHERE (User.username = @Username);
```

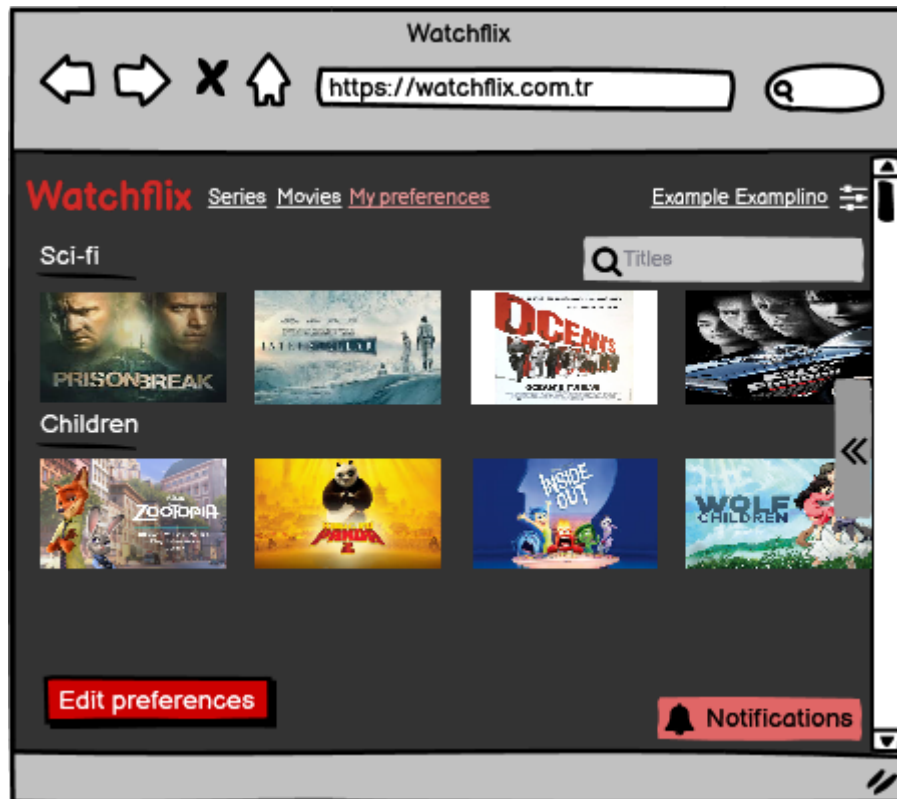


Figure 17: Preference based suggestions

**Inputs:** @Username

**Process:** List suggestions according to user's preference

**SQL Statements:**

```
SELECT g-name AS preferences(g-name)
FROM prefers
WHERE(preferes.username = @Username)
SELECT m-id AS preferredfilms(m-id)
FROM preferences NATURAL JOIN belongsTo
SELECT *
FROM preferredfilms NATURAL JOIN MediaProduct
```

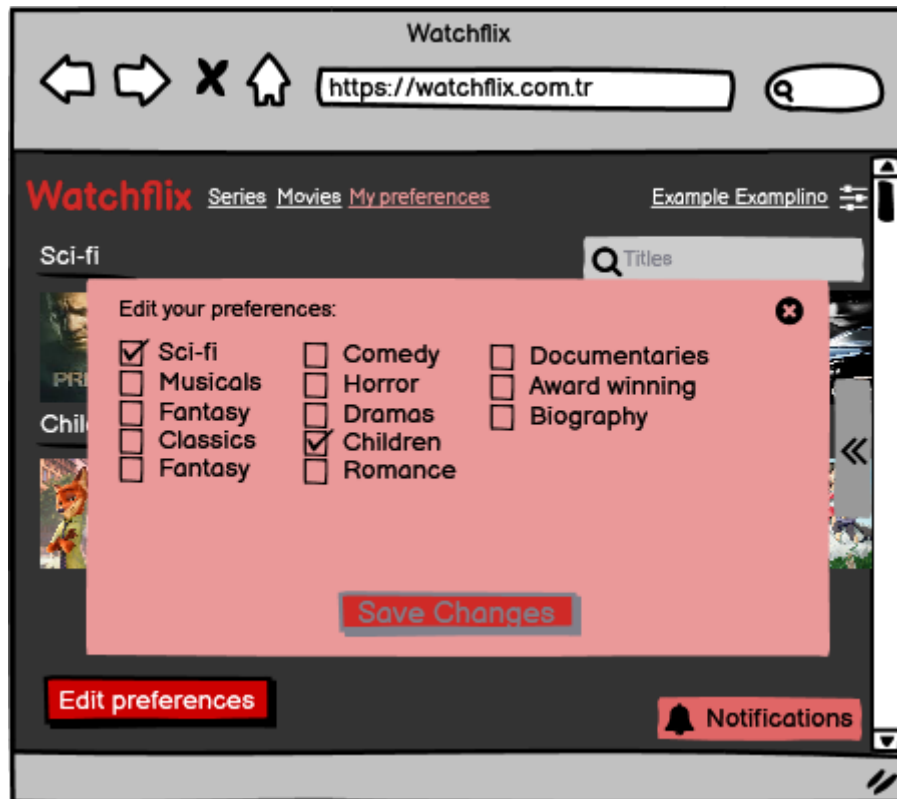


Figure 18: Change Preferences

**Inputs:** @username, @newgenres

**Process:** change preferences of user

**SQL Statements:**

```
DELETE FROM prefers
```

```
WHERE prefers.username = @username;
```

```
DECLARE
```

```
    genre VARCHAR(20);
```

```
BEGIN
```

```
    FOR genre in @newgenres LOOP
```

```
        INSERT INTO prefers(g-name, username)
```

```
        VALUES(genre, @username);
```

```
    END LOOP;
```

```
END;
```

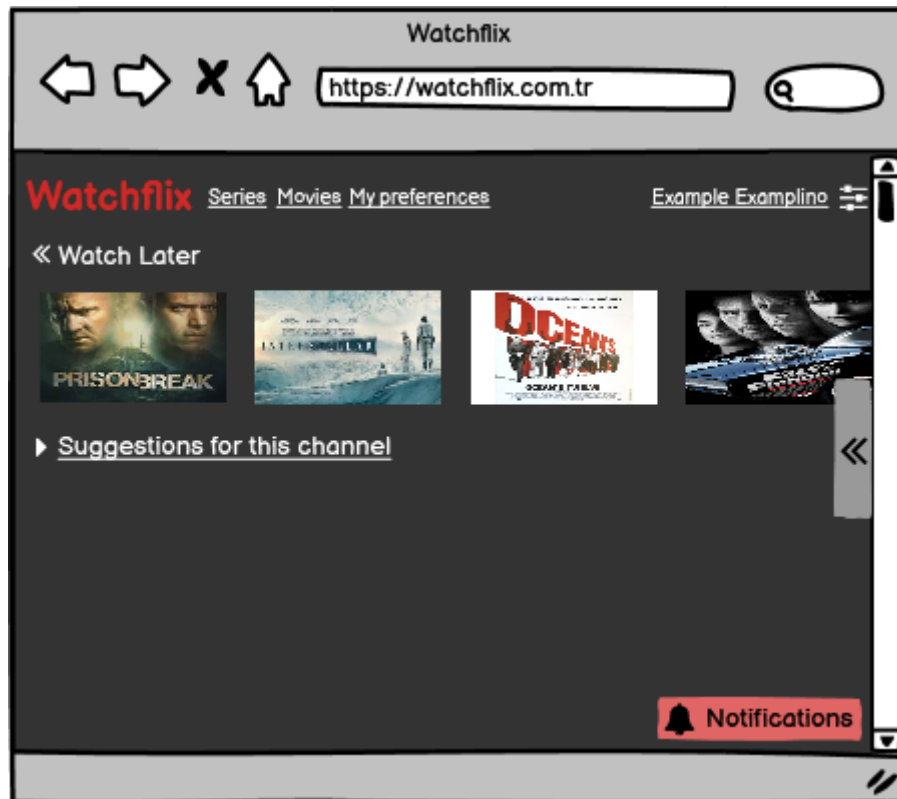


Figure 19: Channel

**Inputs:** @ChannelID

**Process:** Media products in the channel will be shown

**SQL Statements:**

```
SELECT *  
FROM contain NATURAL JOIN MediaProduct  
WHERE (contain.channel-id = @ChannelID);
```

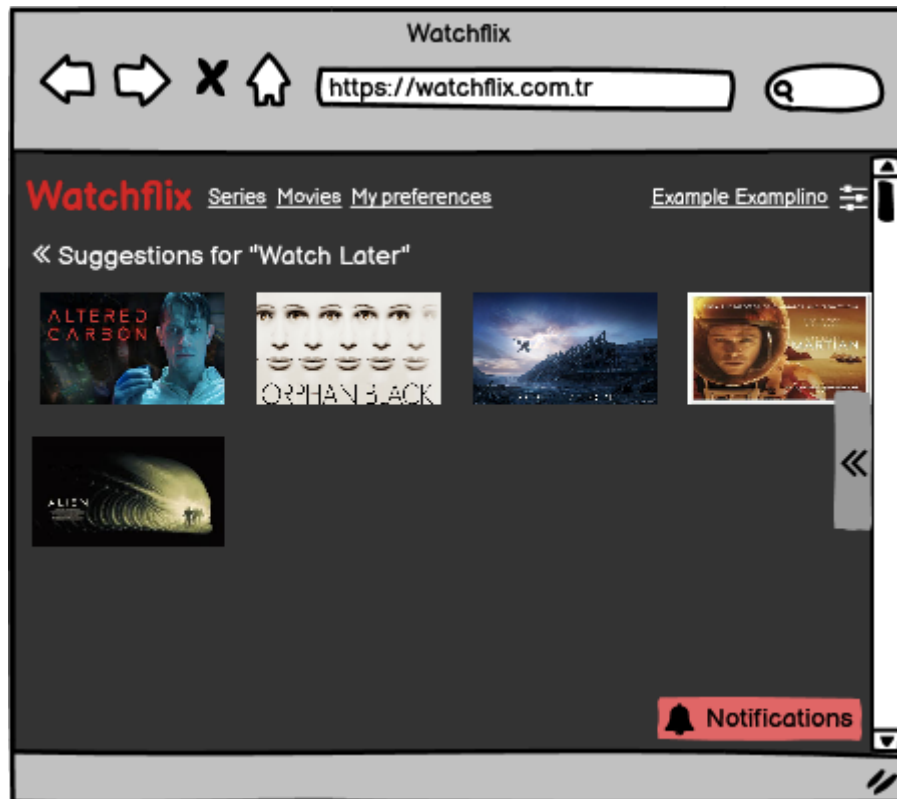


Figure 20: Watch Later Suggestions

**Inputs:** @genres, @username

**Process:**

**SQL Statements:**

```
SELECT g-name AS preferences(g-name)
```

```
FROM prefers
```

```
WHERE(preferes.username = @username)AND prefers.g-name in  
(@genres)
```

```
SELECT m-id AS preferredfilms(m-id)
```

```
FROM preferences NATURAL JOIN belongsTo
```

```
SELECT *
```

```
FROM preferredfilms NATURAL JOIN MediaProduct
```

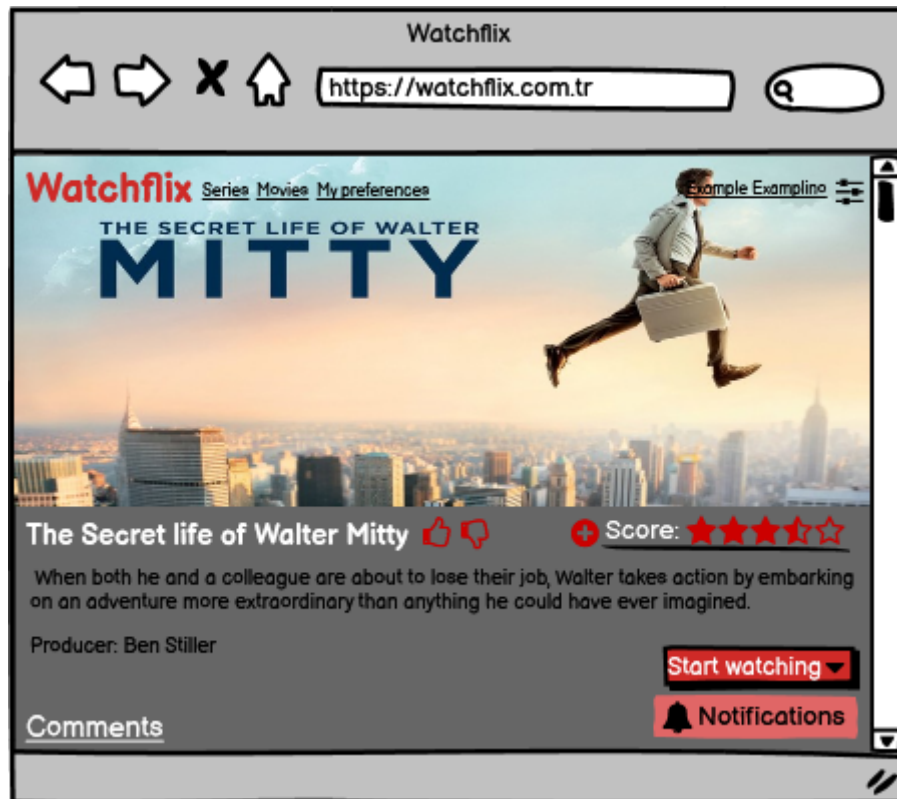


Figure 21: Media Product Page

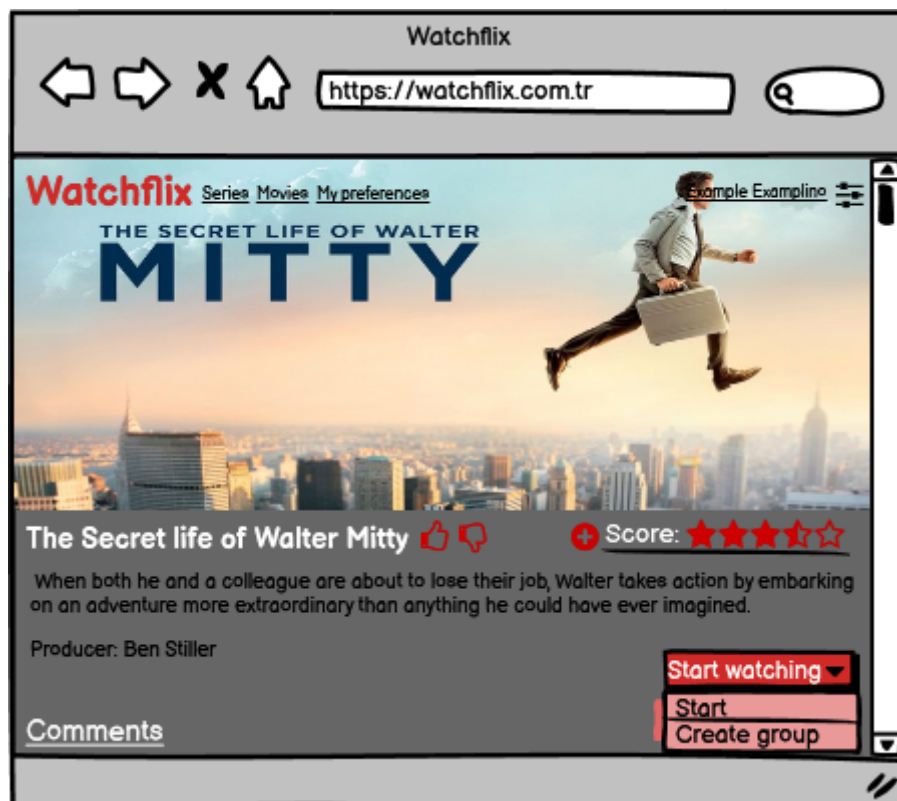


Figure 22: Group creation





Figure 23: Addition of media to channel

**Inputs:** @channelID, @mediaID

**Process:** Created channels of user will be shown

**SQL Statements:**

```
INSERT INTO contain(channel-id, m-id)
```

```
VALUES (@channelID, @mediaID)
```



Figure 24: Comment on media products

**Inputs:** @username, @mediaID, @commentID, @c-content,  
@replied-commentID, @comment-date

**Process:** add comment to media product or other comments. If the comment is done to product @replied-commentID will be NULL

**SQL Statements:**

```
INSERT INTO CommentPosted(comment-id, replied-comment-id, m-id,  
username, comment-date, c-content)  
VALUES (@commentID, @replied-commentID, @mediaID, @username,  
@comment-date, @c-content)
```



Figure 25: Add somebody as friend

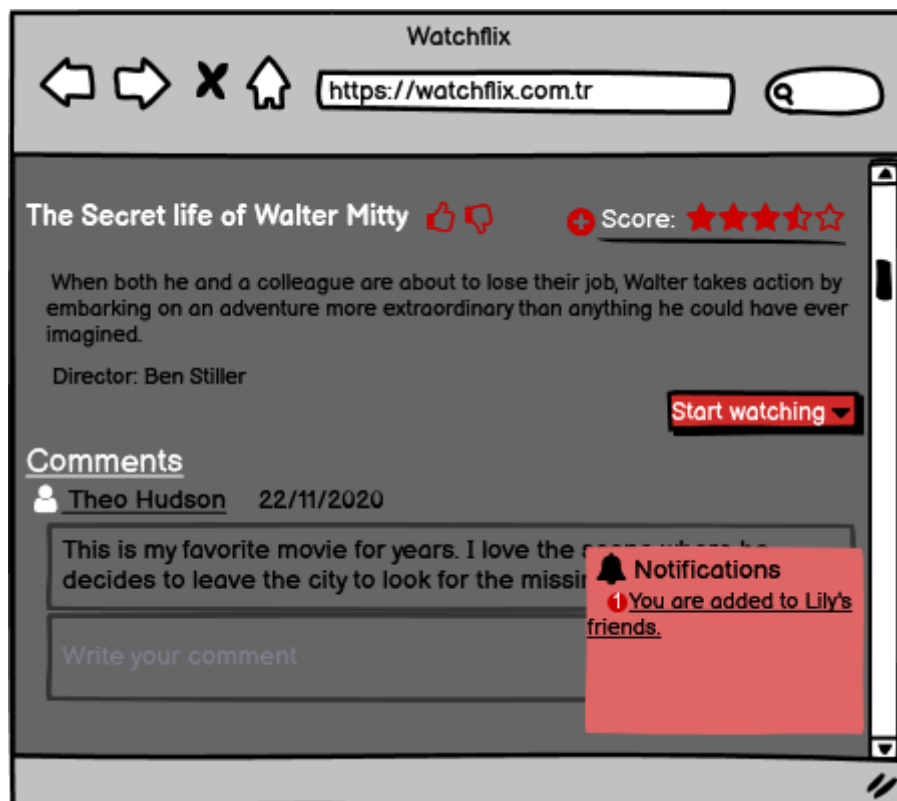


Figure 26: To be added to friend list of another user

**Inputs:** @username, @friend\_username

**Process:** add friends

**SQL Statements:**

```
INSERT INTO friend
VALUES (@username, @friend_username),
      (@friend_username, @username);
```

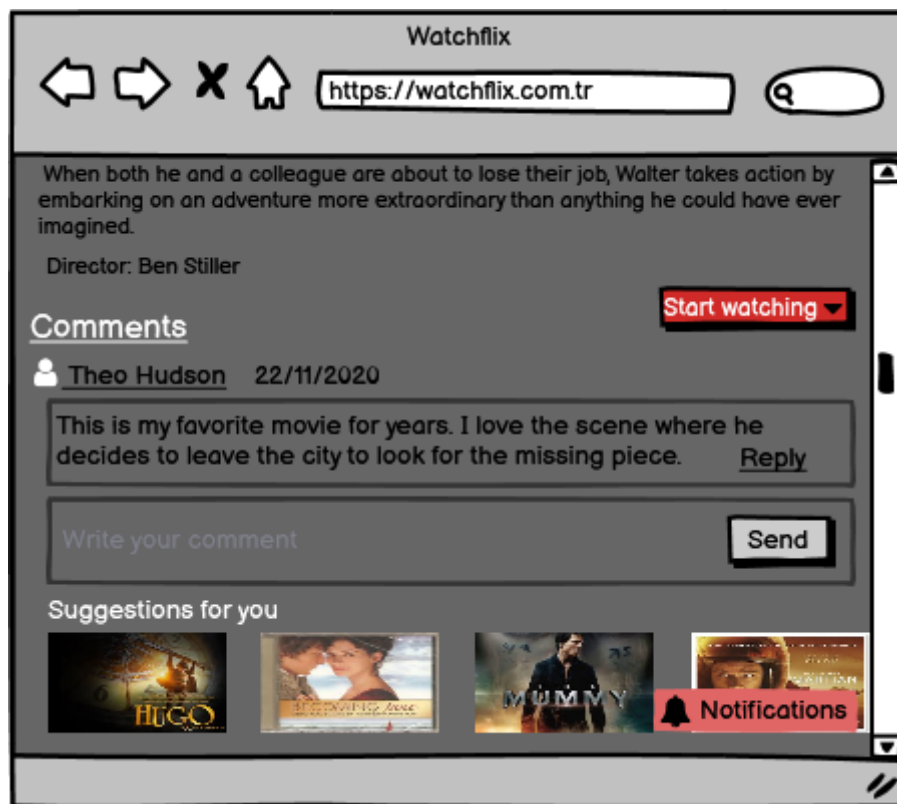


Figure 27: Suggestions according to genre of media product

**Input:** @m-id

**Process:** Application will offer suggestions based on the genre of the Media Product in product profile.

**SQL Statements:**

```
SELECT g-name AS genres(g-name)
FROM belongsTo
WHERE(belongTo.m-id = @m-id)
SELECT m-id AS preferredfilms(m-id)
FROM genres NATURAL JOIN belongsTo
```

SELECT \*

FROM preferredfilms NATURAL JOIN MediaProduct



Figure 28: Continue Watching Option



Figure 29: Continue and Finish Options

**Inputs:** @m-id, @username, @today

**Process:** continue to watch film or finish watching. "finish" button will be clickable after the watch choice has been selected at least three times. progress will be changed each time user stops to watch

**START WATCHING:**

```
INSERT INTO watch
VALUES(@m-id, @username, w-date = @today, watch-count = 0,
progress = 0, finished = FALSE)
```

**CONTINUE WATCHING:**

**SQL Statements:**

```
SELECT * AS watchedfilm
FROM watch
WHERE watch.username = @username AND watch.m-id = @m-id
UPDATE watchedfilm
```

```
SET watch_count = watch_count + 1
```

#### **FINISH WATCHING:**

##### **SQL Statements:**

```
SELECT * AS watchedfilm
```

```
FROM watch
```

```
WHERE watch.username = @username AND watch.m-id = @m-id
```

```
UPDATE watchedfilm
```

```
SET finished = TRUE
```



Figure 30: Finished and Rate

**Input:** @m-id, @score, @username

**Process:** After finishing the media product, users can rate it on the scale of 1 to 10.

##### **SQL statement:**

```
SELECT * AS watchstatus
```

```
FROM watch
```

```

WHERE watch.username = @username AND watch.m-id = @m-id

INSERT INTO rate(username, m-id, rated-score)

VALUES (username, m-id, score)

WHERE TRUE in (SELECT finished

                FROM watchstatus);

```

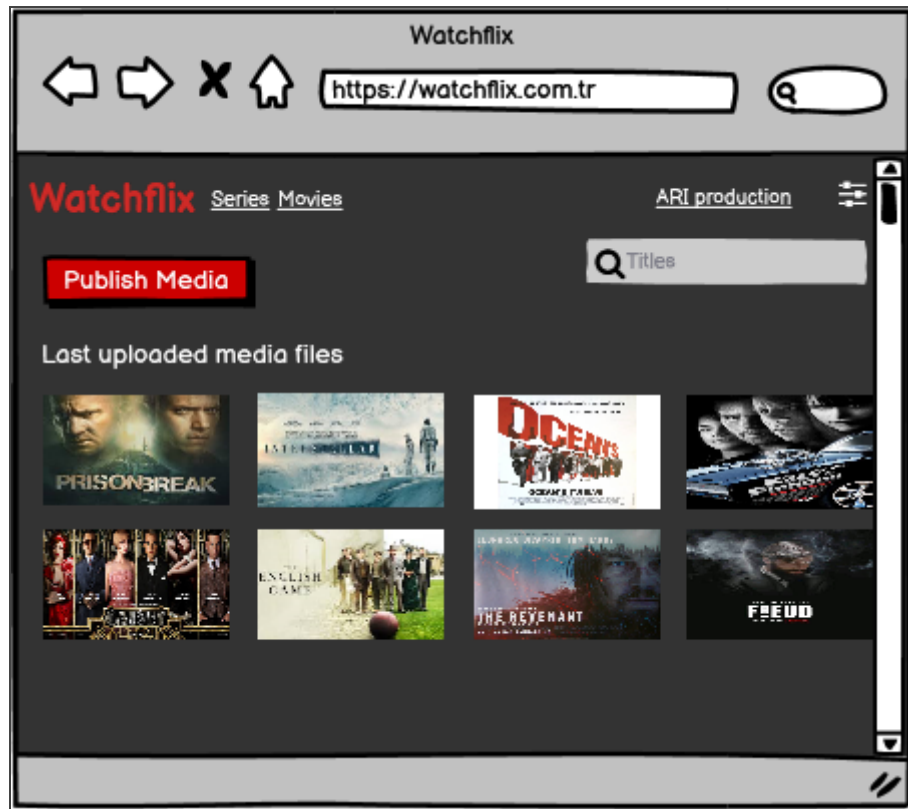


Figure 31: Company User Main Page

**Input:** @companyUsername

**Process:** Lists all media products uploaded by user.

**SQL statement:**

```

SELECT *

FROM MediaProduct(m-id, publisher, release_date, mp-name)

WHERE (MediaProduct.publisher = @companyUsername)

```



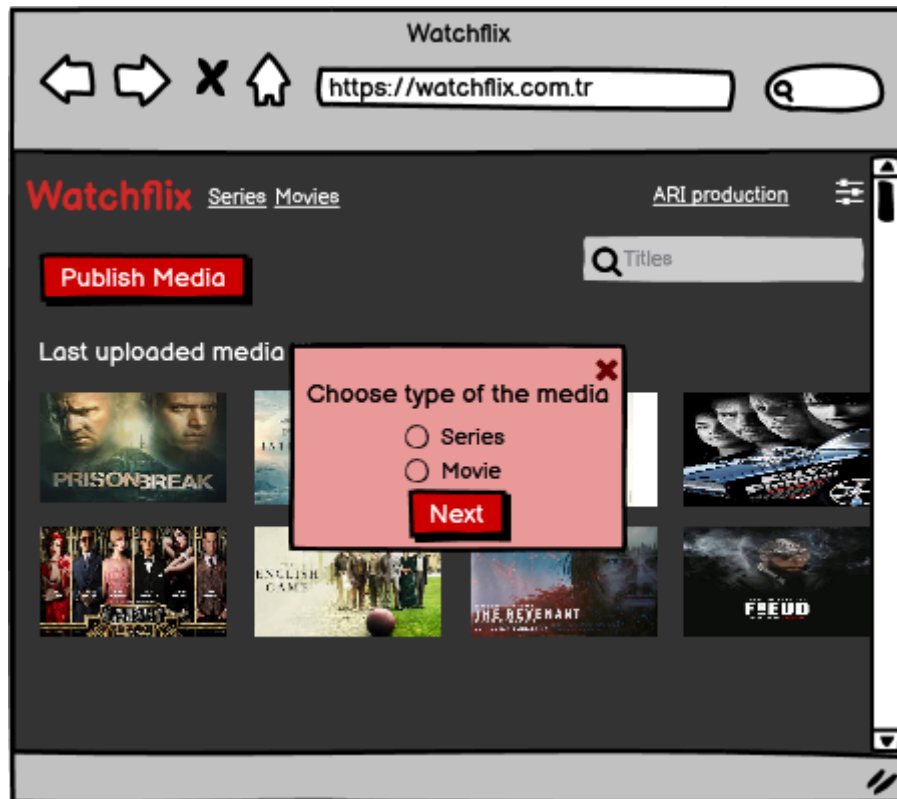


Figure 32: Choose Media Type



Figure 33: Enter series details

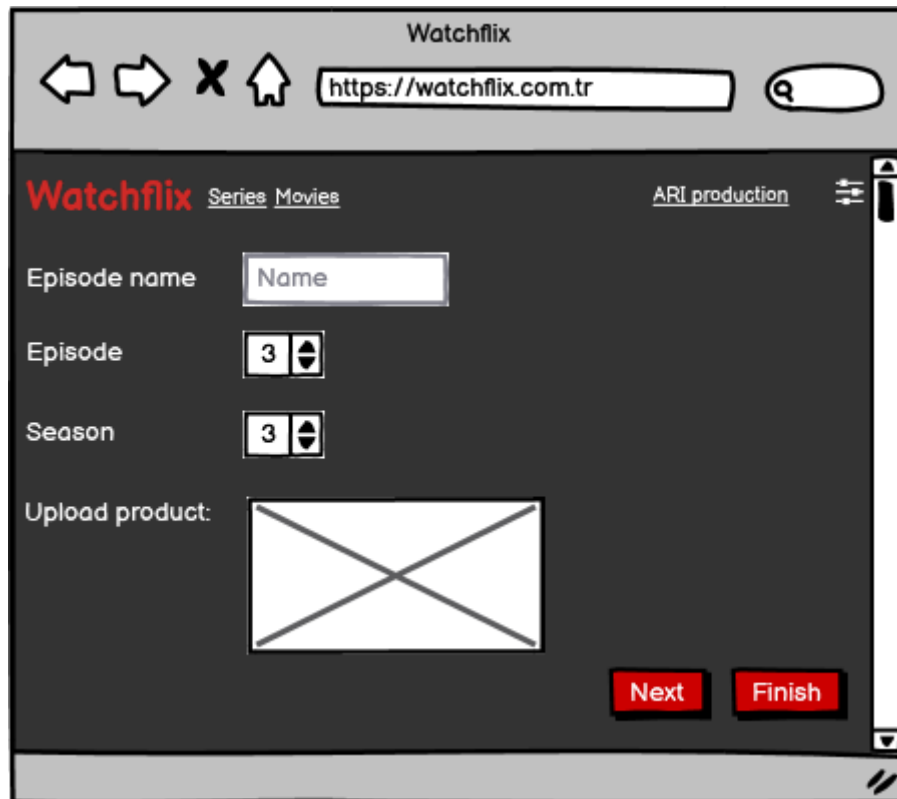


Figure 34: Enter episode details

**Input:** @companyUsername, @m-id, @episodeName, @eNumber, @sNumber, @release-date, @genres, @mp-name, @episode\_count, @videoURL

**Process:** Company user adds Episode by choosing relevant information.

**SERIES ADD:**

**SQL statement:**

```
INSERT INTO MediaProduct
VALUES(@m-id, @companyUsername, @release-date, @mp-name);
INSERT INTO Series
VALUES(@m-id, @episode_count)
```

**EPISODE ADD:**

**SQL statements:**

```
INSERT INTO Episode
VALUES(@m-id, @eNumber, @sNumber, @eName, @videoURL);
DECLARE
```

```
    genre VARCHAR(20);
```

```
BEGIN
```

```
    FOR genre in @genres LOOP
```

```
        INSERT INTO belongsTo(g-name, m-id)
```

```
        VALUES(genre, @m-id);
```

END LOOP;

END;

The screenshot shows a web browser window titled "Watchflix" with the URL "https://watchflix.com.tr". The page has a dark background and a light-colored header. The header contains the "Watchflix" logo, links for "Series" and "Movies", and a link for "ARI production". The main content area is a form for entering movie details. It includes a "Media name" input field, a "Release date" input field with a calendar icon, and a section for "Add genres for the product". This section contains a list of checkboxes for various genres: Sci-fi, Musicals, Fantasy, Classics, Fantasy, Comedy, Horror, Dramas, Children, Romance, Biography, and Children. A note states "You cannot choose more than three genres." A red "Upload movie" button is located at the bottom right of the form.

Watchflix

Series Movies

ARI production

Media name

Release date

Add genres for the product:

☐ Sci-fi ☐ Comedy ☐ Biography

☐ Musicals ☐ Horror ☐ Children

☐ Fantasy ☐ Dramas ☐ Children

☐ Classics ☐ Romance

☐ Fantasy

You cannot choose more than three genres.

Upload movie

Figure 35: Enter movie details

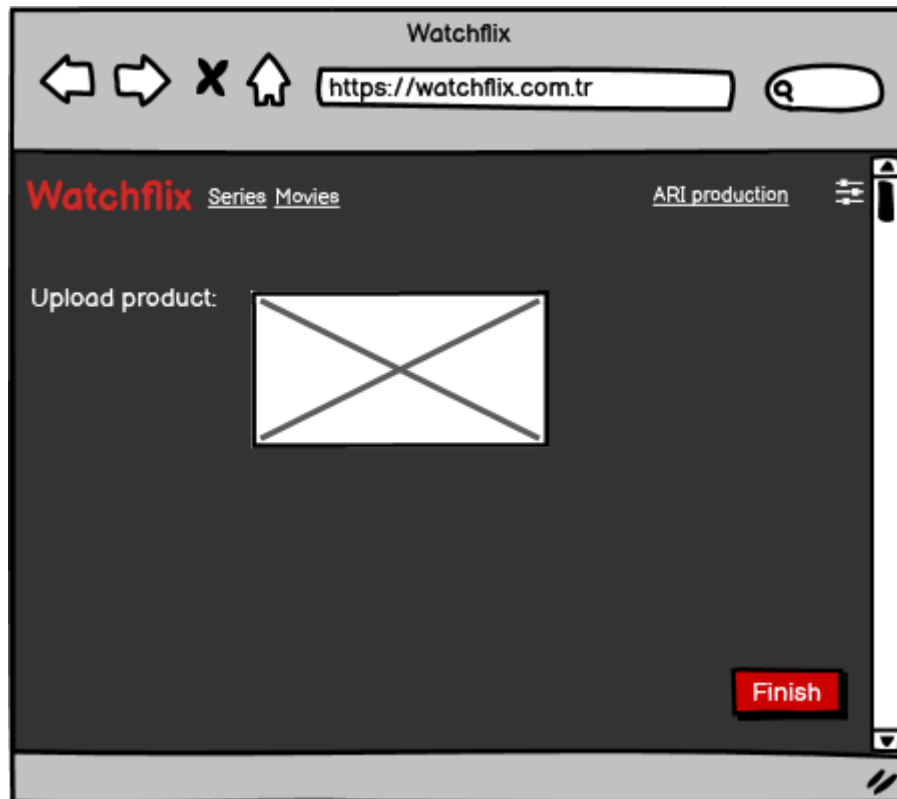


Figure 36: Upload movie

**Input:** @companyUsername, @m-id, @mp-name, @release-date, @genres, @videoURL

**Process:** Company user adds Episode by choosing relevant information.

**MOVIE ADD:**

**SQL statement:**

```
INSERT INTO MediaProduct
VALUES(@m-id, @companyUsername, @release-date, @mp-name);
INSERT INTO MOVIE
VALUES(@m-id, @videoURL)
DECLARE
    genre VARCHAR(20);
BEGIN
    FOR genre in @genres LOOP
```

```
        INSERT INTO belongsTo (g-name, m-id)
        VALUES (genre, @m-id);
    END LOOP;
END;
```