|  |  |
| --- | --- |
|  | **DOKUZ EYLÜL UNIVERSITY**  **ENGINEERING FACULTY**  **DEPT. OF COMPUTER ENGINEERING** |

<RATE A MOVIE >

# CME 3201 Database Management Systems

# Term Project Report

Final Report

2019-2020 FALL

2016510086 Mustafa ÖZSARAÇ

2017510033 Ayşe Özge ERİŞ

## Introduction

This report is prepared to give brief information about the Rate a Movie Project. The document contains the problem domain, solution and purpose of the project.

**Problem Domain**

People love movies and actors/actresses however there is one thing that people like more: discussing about them. The problem is that there is not an environment that allows people to share their opinions and ratings.

That’s why our purpose in order to solve this problem is to create a web site for movie lovers to rate movies, share their comments, see movie information such as movie’s director, cast, release date or creating watch lists for users etc.

Anyone who is willing to bring people who are interested in movies together could be the stakeholder of this project. They can invest in this project to support us to do better.

**Intended Audience**

-Anyone who loves movies.

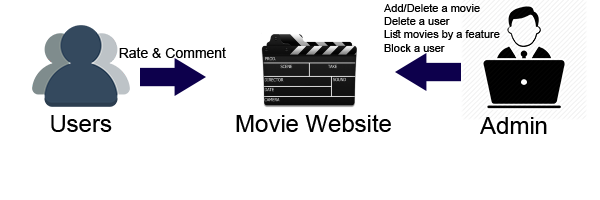
-Anyone who is curious about movies’ ratings.

-Anyone who is interested in discussing about movies.

Basically, this website will be for everyone who is keen on to movies.

## Purpose and Solution

The purpose of this project is to be solution to all those problems mentioned above. It will be a meeting point for all movie enthusiasts. Creating a practical and fun environment for people who are looking to movies.

**

The goal of this project is to create a website like “IMDB”. Users will be able to find all the information about movies or see watch lists of other users and their information.

## Overview

**2.1 Operations & Instructions**

**Create User**

New users can click “Create New” button on login screen to create their account with their selected username, name, surname, age and gender by an admin. All these will improve user’s experience in website.

**Login**

Users can login to the system.

**Admin Panel**

Administrators of the website can use this panel to control, change, delete and access most of the information and other things in their website.

**View Comment**

Allows users to view comments of other users for movies.

**View Movie**

Allows users to view the movies.

**Rate Movie**

Allows users to rate the movies from a scale from 1 to 10.

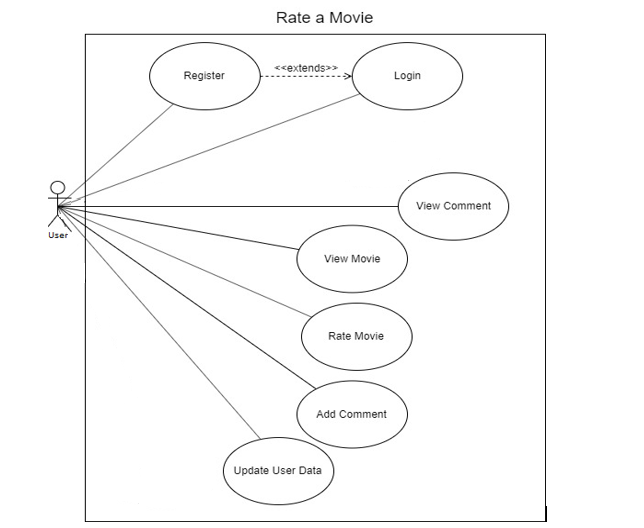
**Add Comment**

Allows users to share their opinions about the movies.

**Update User Data**

Users can edit their profiles by clicking “Edit” button in their profile page.

**2.2 Use Case Diagram**



**2.3 Stakeholders**

Anyone who wants to bring people who are into movies together could be the stakeholder of this project. They can invest in this project to support us to do better. The roles of the stakeholders are as follows:

* Voting and Decision Making
* Providing Expertise
* Managing Industrial Crises
* Corporate Social Responsibility

**2.4 Entity List**

**User Entity**

1. User ID(auto-increment - int)
2. Nickname(varchar(16))
3. Password(varchar(16))
4. E-Mail(varchar(16))
5. Phone Number(Optional) (int(11))
6. Birthdate(date)
7. About(string)
8. Is Admin(bool)

**Movie Entity**

1. Movie ID(auto-increment - int)

2. Movie Title(varchar(256))

3. Genre(array(varchar(50)))

4. Release Date(date)

5. Rating Average(float)

6. Rating Count(int)

7. Crew(crew-int)

8. Age Limit(int)

**Comment Entity**

1. Comment ID(auto-increment - int)
2. Movie ID(int)
3. User ID(int)
4. Title(varchar(50))
5. Comment(varchar(50))
6. Entry Date(date)

**Genres Entity**

1. Genre ID(auto-increment - int)
2. Name(varchar(50))

**Actor Entity**

1. Actor ID(auto-increment - int)
2. Name (varchar(50))
3. Surname(varchar(50))
4. Crew ID (int)

**Crew Entity**

1. Crew ID(auto-increment - int)
2. Name (varchar(50))
3. Surname(varchar(50))
4. Movie ID(int)

## Assumptions/Constraints/Risks

### Assumptions

With the help of the stakeholdersproviding expertise to this process, completing the project without any obstacles and on time is expected.

### Constraints

**Username**

While registering, users cannot have empty username. They must select a username that is not already in use.

**User Password**

All users must have a password for their account. The length must be 8 characters at least.

**User Birthdate**

Users must enter their birth date before creating their profile.

**Rate Scale**

Users must rate the movies from 1 to 10 and only as an integer number.

**User Email**

Users must enter proper emails. We check if there is ‘@’ character.

**IsAdmin**

There is a boolean typed entity to check if the current user is an admin or just a regular user.

### Risks

The expectable risks are that the possibility of data loss from the database, undetectable bug in the application, unexpected breakdown of the serves and potential delay of the project’s handing over to the customers. We expect to overcome these possible problems with the help of our stakeholders’ expertise.

## Software Architecture

In this project MVC architecture was used. Model View Controller popular as MVC is a widely-used model for the web development purpose in the current time. It is popular among various major programming languages today, such as Java, Python, Ruby, PHP, but MVC model is widely used for a web application development purpose for the ASP.net platform.

Today it is widely used as the powerful framework for building web applications using the MVC pattern. The Model represents the application core, whereas the View displays the data and the Controller handles the input. The MVC separation allows ease management of the complex applications because one can focus on one aspect at a time.

Rate A Movie project has a big scope that is why it works well for developing web applications which need to be supported by large teams of developers and for Web designers who want greater control over the application behavior.

The connection between database and front end was provided by the following code implementations below:

"ConnectionStrings": {

"DefaultPGSqlConnection": "User ID=postgres; Password=1234; Server=localhost; Port=5432; Database=rateamovie; Password=1234; Integrated Security=true;Pooling=true;"

},

The code part above was added to appsettings.json file. User ID represents the id of an user in database, password is our password to enter database, our server is on local host, our port number is 5432 and our database is called as rateamovie.

Then, we added connection string configuration to startup.cs file. The code related is down below:

public void ConfigureServices(IServiceCollection services)

{

services.AddControllersWithViews();

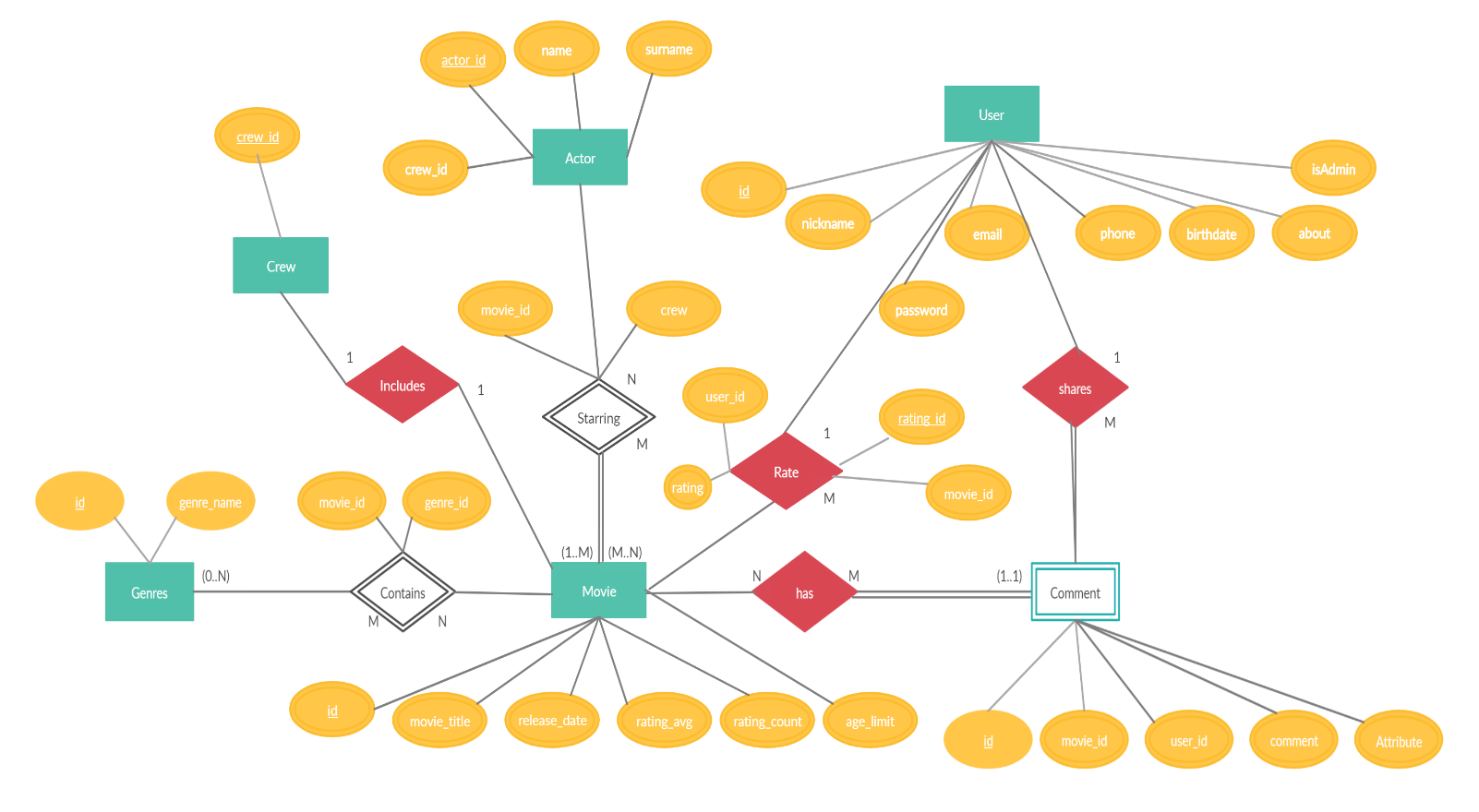
services.AddDbContext<rateamovieContext>(options=>options.UseNpgsql(Configuration.GetConnectionString("DefaultPGSqlConnection")));

}

This is how we connected our database to our MVC web application.

## Detailed System Design

### Entity-Relationship Diagram



### Relational Algebra Expressions

1. Write a query in SQL to list all the actors who have not acted in any movie between 1990 and 2000

SELECT \*

FROM Actor a, Crew b, Movie c

WHERE a.actor\_id=b.actor\_id

AND b.movie\_id=c.id

AND c.release\_date NOT BETWEEN 1990 and 2000;

πactors(σ1990<release\_date<2000(Actor) ⨝ Movie))

2)Write a query in SQL to list all the movies that has actor named “Ayşe Eriş”

SELECT title

FROM Actor a JOIN Movie b

WHERE a.actor\_name=”Ayşe Eriş”

πtitle(σname = “Ayşe Eriş”(Actor) ⨝ Movie))

1. Write a query in SQL to list all the movies with year and genres.

SELECT title, release\_date, name

FROM Movie JOIN Genre

Πtitle,release\_date,name (Genre ⨝ Movie)

4)Write a query in SQL to list all the movies’ rating that are higher than 7.

SELECT title

FROM Movie a

WHERE a.rating\_avg > 7

πtitle(σrating\_avg > 7 (Movie))

5)Write a query in SQL to list all the movie genres which are +18.

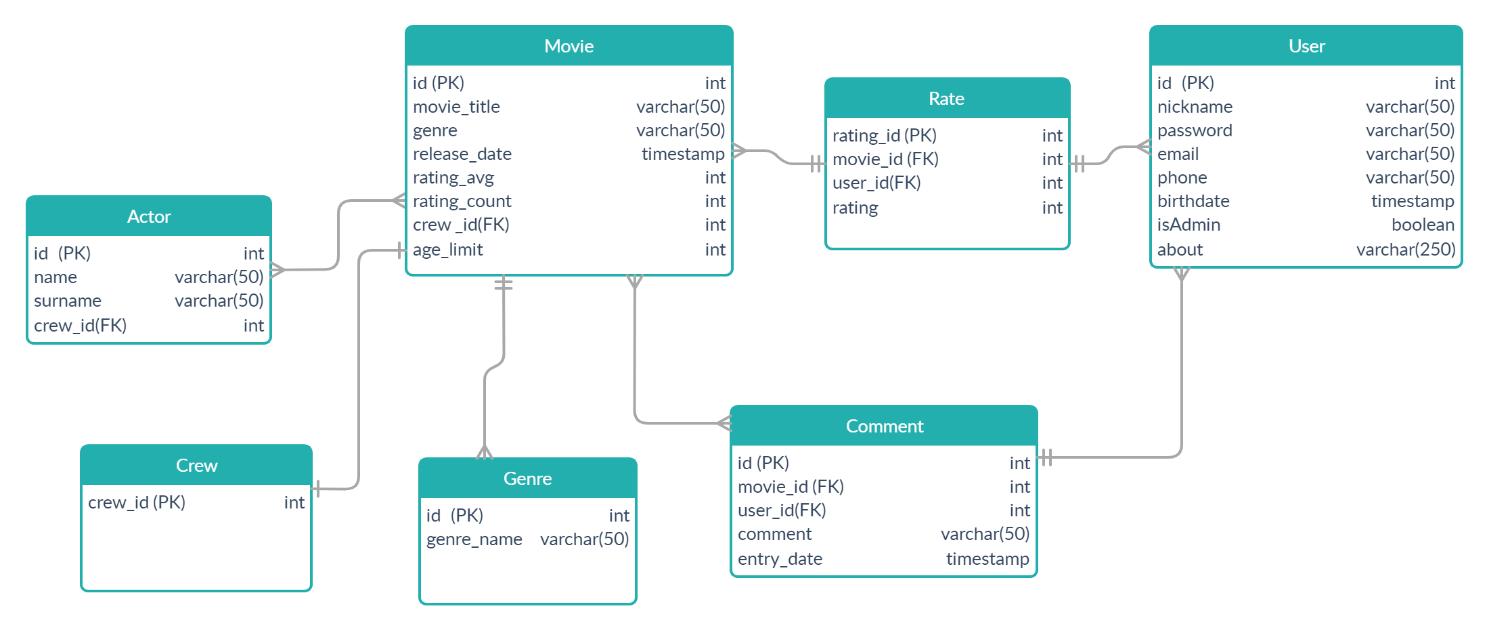
SELECT genres

FROM Movie a

WHERE a.age\_limit >= 18

πtitle(σage\_limit> 18 (Movie))

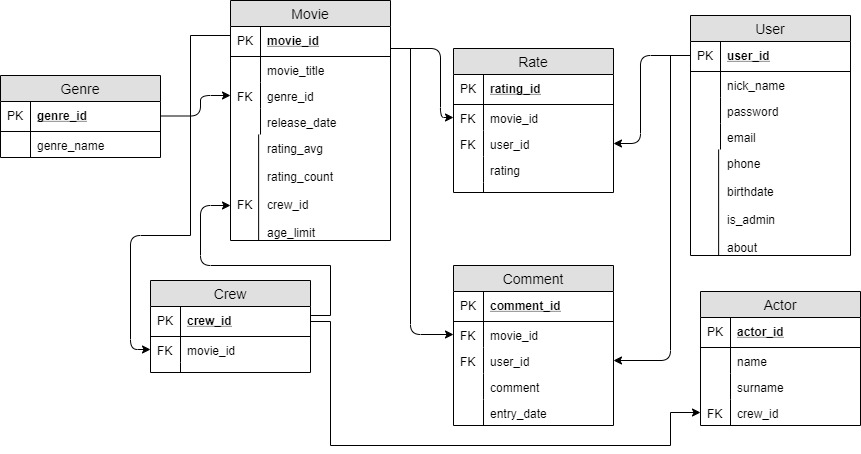
### Class Diagram



### CRUD Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Relation/Operations | Create | Read | Update | Delete |
| Contains |  | X | X | X |
| Rate | X | X | X | X |
| Share | X | X | X | X |
| Has |  | X |  |  |
| Starring | X | X | X | X |
| Includes | X | X | X | X |

### Database Schema



### Appendix A: SQL Statements

CREATE TABLE user(

user\_id SERIAL PRIMARY KEY,

nick\_name TEXT NOT NULL,

email TEXT NOT NULL UNIQUE,

phone TEXT NOT NULL UNIQUE,

password TEXT NOT NULL,

birth\_date DATE NOT NULL,

is\_admin BOOLEAN NOT NULL,

about VARCHAR(250),

CHECK(nick\_name !~ '\s'),

CHECK (email ~\* '^\w+@\w+[.]\w+$'),

CHECK (char\_length(password)>=8)

);

CREATE TABLE rate(

rating\_id SERIAL PRIMARY KEY,

movie\_id INT NOT NULL REFERENCES movie(movie\_id),

user\_id INT NOT NULL REFERENCES user(user\_id),

rating INT NOT NULL,

CHECK (rating IN (1,2,3,4,5,6,7,8,9,10))

);

CREATE TABLE movie(

movie\_id SERIAL PRIMARY KEY,

movie\_title TEXT NOT NULL,

genre\_id INT NOT NULL REFERENCES genre(genre\_id),

release\_date DATE NOT NULL,

rating\_avg INT NOT NULL,

rating\_count INT NOT NULL,

crew\_id INT NOT NULL REFERENCES crew(crew\_id),

age\_limit INT NOT NULL

);

CREATE TABLE genre(

genre\_id SERIAL PRIMARY KEY,

genre\_name TEXT NOT NULL

);

CREATE TABLE crew(

crew\_id SERIAL PRIMARY KEY

);

CREATE TABLE comment(

comment\_id SERIAL PRIMARY KEY,

movie\_id INT NOT NULL REFERENCES movie(movie\_id),

user\_id INT NOT NULL REFERENCES user(user\_id),

comment VARCHAR(50) NOT NULL,

entry\_date DATE NOT NULL

);

CREATE TABLE actor (

actor\_id SERIAL PRIMARY KEY,

actor\_name VARCHAR(50) NOT NULL,

actor\_surname VARCHAR(50) NOT NULL,

crew\_id integer NOT NULL REFERENCES crew(crew\_id)

);

CREATE user\_view AS

SELECT user\_table.nick\_name,

user\_table.email,

user\_table.birth\_date,

user\_table.about

FROM user\_table;

CREATE top10\_view AS

SELECT m.rating\_avg,

m.movie\_title

FROM movie m

ORDER BY m.rating\_avg DESC

LIMIT 10;

CREATE genre\_view AS

SELECT m.movie\_title,

g.genre\_name

FROM movie m,

genre g

WHERE m.genre\_id = g.genre\_id

ORDER BY g.genre\_name;

CREATE PROCEDURE insert\_actor(id integer, actor\_name varchar , actor\_surname varchar, crew\_id integer)

LANGUAGE SQL

AS $$

INSERT INTO actor VALUES (id,actor\_name,actor\_surname, crew\_id);

$$;

CREATE OR REPLACE FUNCTION ratelog\_trg() RETURNS

TRIGGER AS

$$

BEGIN

IF TG\_OP = 'INSERT' THEN

INSERT INTO ratelog SELECT NEW.\*, 'I',

NOW();

UPDATE movie SET rating\_avg = ((new.rating + (rating\_avg\* rating\_count)) / ((rating\_count+1))) where movie\_id = new.movie\_id;

UPDATE movie SET rating\_count = (rating\_count + 1) where movie\_id = new.movie\_id;

ELSIF TG\_OP = 'UPDATE' THEN

INSERT INTO ratelog SELECT NEW.\*, 'U',

NOW();

UPDATE movie SET rating\_avg = (((rating\_avg \* rating\_count) - old.rating + new.rating))/rating\_count where movie\_id = new.movie\_id;

ELSIF TG\_OP = 'DELETE' THEN

INSERT INTO ratelog SELECT OLD.\*, 'D',

NOW();

UPDATE movie SET rating\_avg = (((rating\_avg \* rating\_count) - old.rating))/((rating\_count-1)) where movie\_id = new.movie\_id;

UPDATE movie SET rating\_count = (rating\_count -1) where movie\_id = old.movie\_id;

END IF;

RETURN NULL; --ignored since this is after trigger

END;

CREATE TRIGGER ratelog AFTER

INSERT OR UPDATE OR DELETE ON rate

FOR EACH ROW

EXECUTE PROCEDURE ratelog\_trg ();

CREATE OR REPLACE FUNCTION movielog\_trg() RETURNS

TRIGGER AS

$$

BEGIN

IF TG\_OP = 'INSERT' THEN

INSERT INTO movielog SELECT NEW.\*, 'I',

NOW();

ELSIF TG\_OP = 'UPDATE' THEN

INSERT INTO movielog SELECT NEW.\*, 'U',

NOW();

ELSIF TG\_OP = 'DELETE' THEN

INSERT INTO movielog SELECT OLD.\*, 'D',

NOW();

END IF;

RETURN NULL; --ignored since this is after trigger

END;

CREATE TRIGGER movielog AFTER

INSERT OR UPDATE OR DELETE ON movie

FOR EACH ROW

EXECUTE PROCEDURE movielog\_trg ();

CREATE OR REPLACE FUNCTION commentlog\_trg() RETURNS

TRIGGER AS

$$

BEGIN

IF TG\_OP = 'INSERT' THEN

INSERT INTO commentlog SELECT NEW.\*, 'I',

NOW();

ELSIF TG\_OP = 'UPDATE' THEN

INSERT INTO commentlog SELECT NEW.\*, 'U',

NOW();

ELSIF TG\_OP = 'DELETE' THEN

INSERT INTO commentlog SELECT OLD.\*, 'D',

NOW();

END IF;

RETURN NULL; --ignored since this is after trigger

END;

CREATE TRIGGER commentlog AFTER

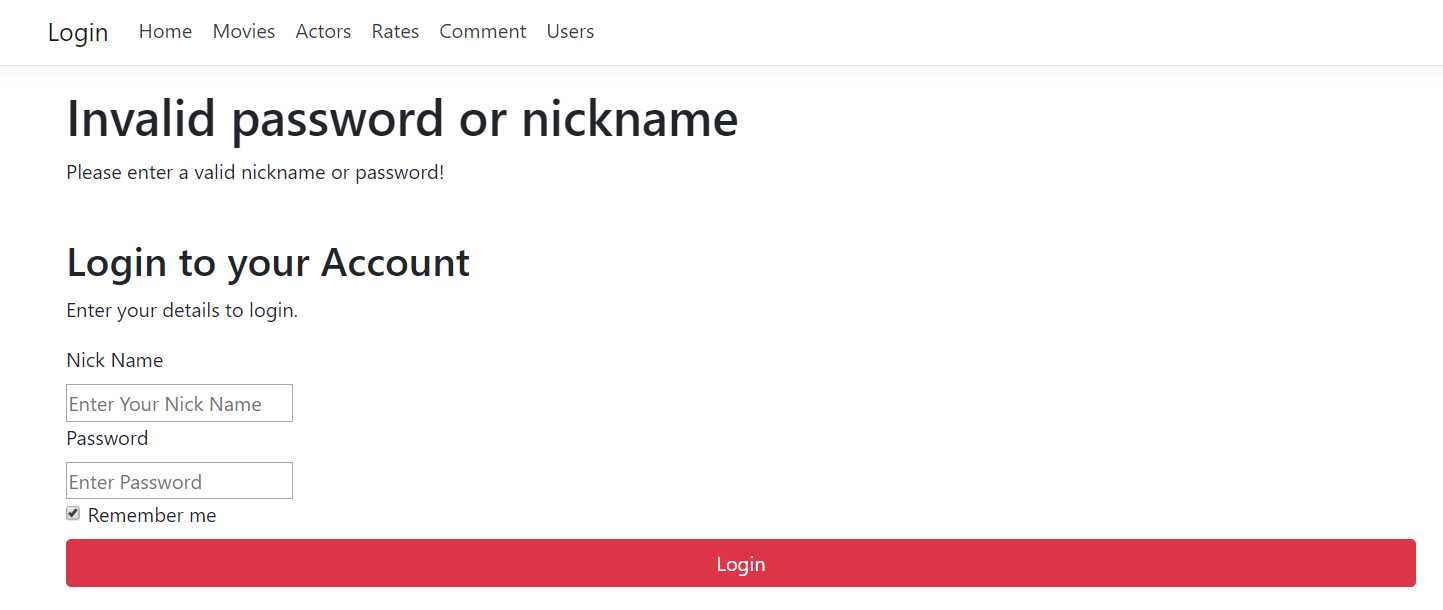
INSERT OR UPDATE OR DELETE ON comment

FOR EACH ROW

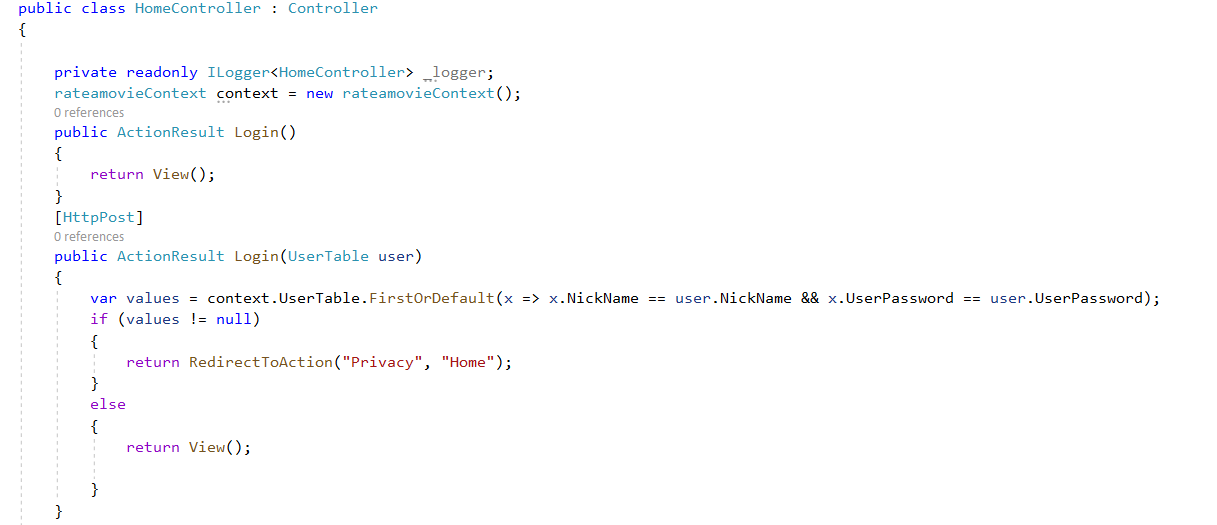
EXECUTE PROCEDURE commentlog\_trg ();

### Appendix B: Screenshots

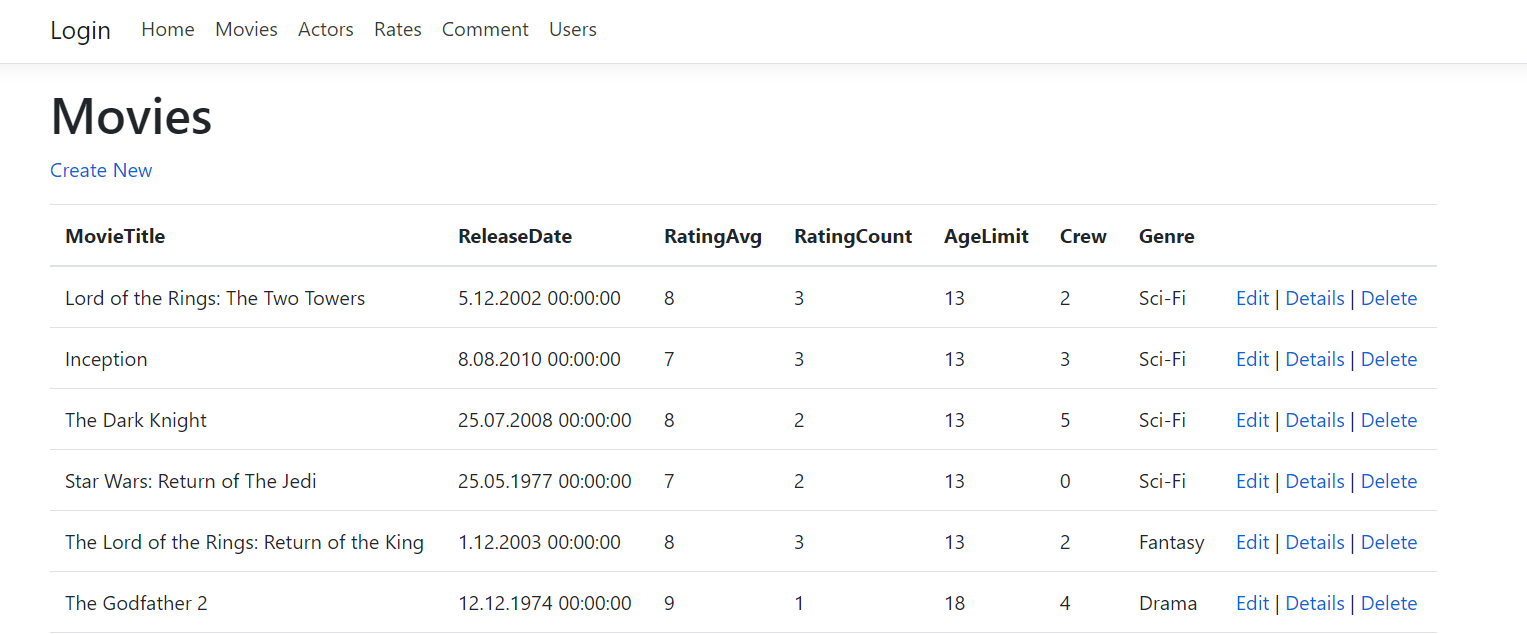
When you open our web application this page comes up first. This is our login page. You need to enter your nick name and your password in order to login. You can also click to the “remember me” button to enter quickly.



If you enter your password or nick name wrong, this is the page that is shown to you. So that you can enter your information once again.



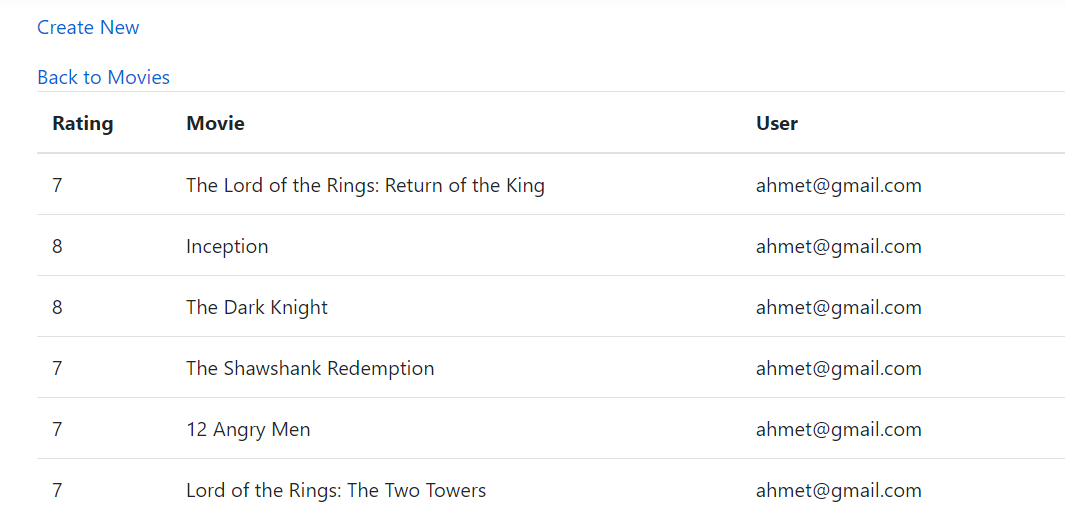
The code part above shows our HomeController. We started with the login page. If nick name and password are correct, we provide a safe entrance to the site. Users that could manage to login are redirected to home page, users who failed to login are redirected to login page once again in order to make them write their information correctly.



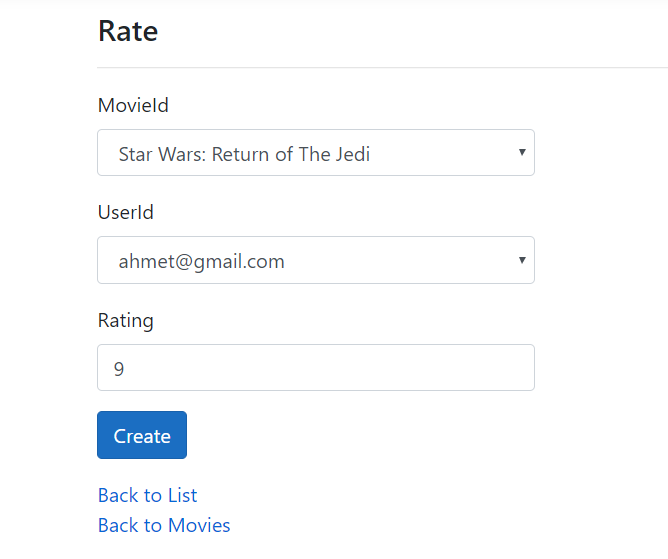
This is the page where we show our movies. Users are able to display all the movies that are in our database. Admins can use the ‘Create New’ button to add a new movie, ‘Edit’ button to update the information of a movie, ‘Delete’ button to remove the movie from the site. Admins are able to perform all operations that were mentioned, however a non-admin user can only display the details by using ‘Details’ button.



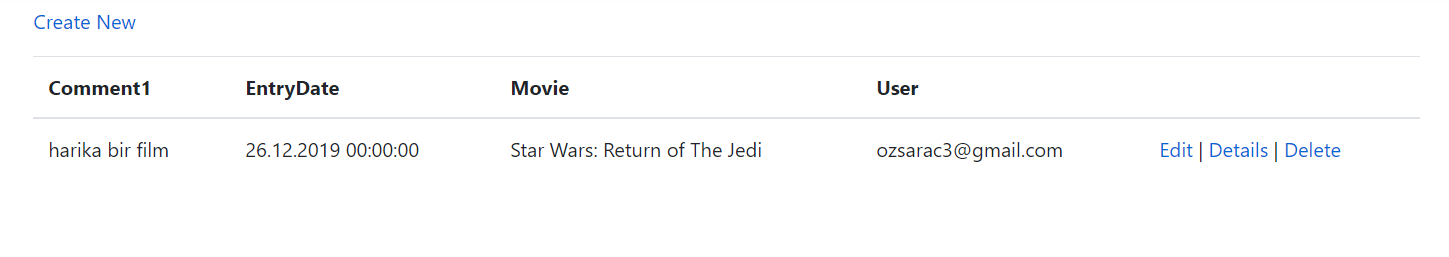
This part represents the operations done in order to perform the creating a new movie operation.



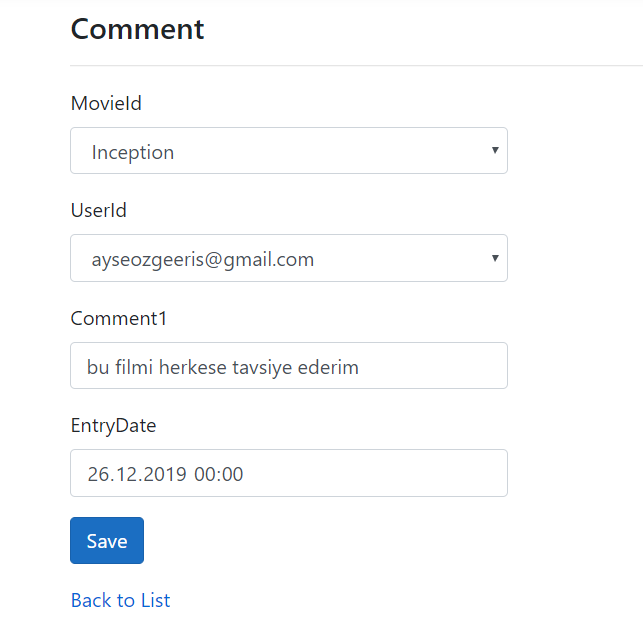
Here is the page that shows the given rates to several movies by a non-admin user. With ‘Create New’ button, the user is able to rate any movies easily.



When ‘Create New’ is clicked, the page above shows up. The name of the movie, rating and the user who is rating that movie is shown clearly. After rating operation is finished, the user can easily go back to the list of the previous rates and the list of movies.



The user can add a comment to movies and see other comments by other users. The user can also edit, delete or show details of his/her comments.



When a user would like to add a comment, this page shows up. The user chooses the movie that he/she wants to comment, types his/her comments about the specified movie, then by clicking save button, the comment has been shared.

Here is the create part of our CommentController. This code shows the operations performed in order to create a new comment.

