İhsan Doğramacı Bilkent University

Department of Computer Science

CS342 – DATABASE SYSTEMS

Online Movie Rental System

Design Report

Group 20:

Efe Ertürk 21902620

Oğuz Ata Çal 21903088

Yiğit Ekin 21901784

Arda Eren 21902505

1. Introduction

The topic of this project is designing an Online Movie Rental System. Our aim in this proposal is to clearly describe the project topic that we are designing and to discuss why and how we use a database for this system.

Then, we will specify functional and non-functional requirements for this system that detail the interaction between the different types of users and the system functionalities along with the system constraints. In the end of this section, we will list the programming language and frameworks we plan to use under the pseudo-requirements heading.

Finally, we will display the conceptual design of our database using an entity-relationship model that will include strong and weak entities, relationships between entities, cardinality constraints, keys and attributes and more.

2. Project Description

The main aim of this Online Movie Rental system is to provide customers a platform to easily rent movies online. Customers will be able to create a request for a movie to be added to the system if their desired movie is not available. The customer can also write reviews, rate films and like or dislike other reviews. While writing a review, one can warn users if the review includes a spoiler to the film or not. The reviews can be displayed according to their dates, from newest to oldest, or according to net likes, or one can choose to view the reviews that do not have any spoiler in it and so on. There is also a friends system that allows customers to add other customers as friends and recommend films to them. The recommendation can include a message for your friends too.

Customers will be able to search movies by title, genre, production year, director and actors. They will also be able to see the movies they are currently renting, their rental history showing the movies they rented previously, the movies that their friends recommended to them, the most rented movies, the best rated movies (which has the best average rating) and finally the list of movies that they put on their favorites. The system will also keep track of the twenty movies that are newly registered to the system.

Employees will be able to satisfy the requests of customer requests and register new movies to the system. They will also have the authority to delete customer accounts.

For the new customers, there will be a promotion code assigned to them, which enables them to rent films at a discounted price.

3. Requirements

a. Functional Requirements

i. Customer

- Customers can search movies based on title, genre, production year, director and actors
- Customers can see the movies they are currently renting, their rental history showing the movies they rented previously, the movies that their friends recommended to them, the most rented movies, the best rated movies and finally the list of movies that they put on their favorites
- Customers can rate movies
- Customers can request a movie to be registered to the system
- Customers can add other customers as friends
- Customers can recommend movies to their friends with messages
- Customers can write reviews and rate movies, indicating whether the review will include spoilers or not.
- Customers can view reviews that are sorted by their date, net like count. They can choose to not see the reviews that have spoilers in them
- Customers can like or dislike the reviews made by other customers
- Customers can favorite movies

ii. Employee

- Employees can register new films based on customer requests
- Employees can delete customer accounts

b. Non-Functional Requirements

i. Security

- Passwords and card information will be kept hashed in the database using salt encryption (this way, passwords will not be seen even by developers)
- Employees and customers will have different access rights to functionalities of the system which prevents employees from adding a new rentable film to the system or employees to modify the data of the customers.
- Non-users or unauthorized users should not be directed to sensitive pages by copying and pasting the url to that page (for

example, a customer shouldn't be able to access employee pages just by copying and pasting the url)

- At database crushes, there should be no data loss
- System needs to be secure against web attacks such as SQL injections and XSS attacks
- When an account is being deleted, all personal data related to that account should be deleted
- After logoff, even from the same ip address, when a certain url is tried to be accessed, until a next authentication, these url should not be accessible

ii. Performance

- Optimized queries will be used to retrieve data from the database
- Logins should take less than 5 seconds when the correct data is provided
- Directing to other endpoints within the website should take less than 5 seconds
- Logout operation should take less than 5 seconds

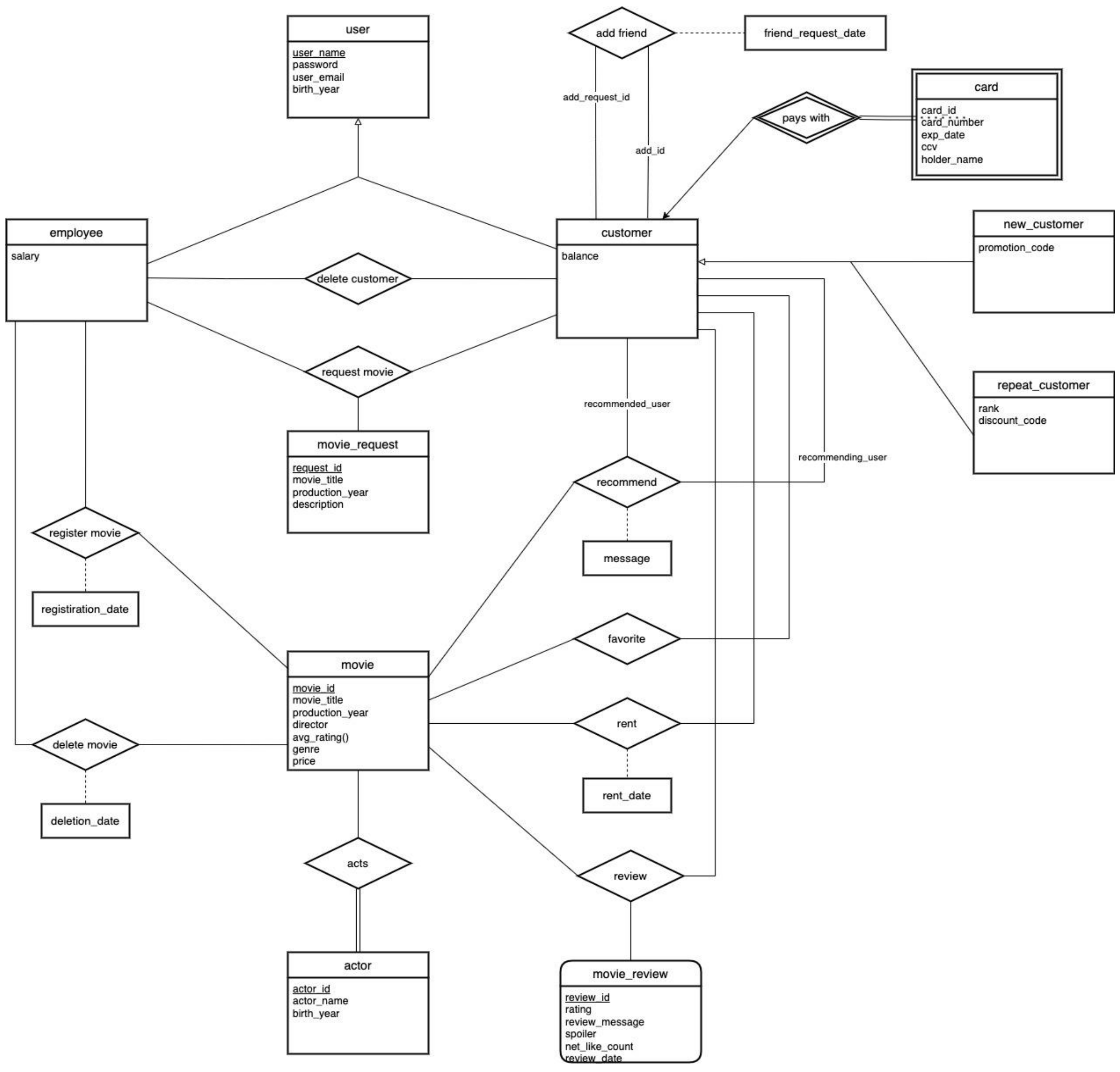
iii. Usability

- User friendly UI components will be used such as descriptionary labels, large buttons with hover effect, navbars, containers and in addition, responsive web design will be used.
- Use different font sizes to draw attention to headings (headings are larger in font size)
- Make the minimum font size (for bodies) 4 rem
- Use contrast color for background and texts of components (for example, beige color for background and dark gray for the texts)
- Give vivid colors to the component that needs extra attention ("Delete customer" button can be red)
- Use mobile first approach (users are likely to use the mobile version of this system), responsive design should be implemented

c. Pseudo Requirements

- MySQL will be used as the query language
- JavaScript and Typescript will be used interchangeably for the frontend with the help React framework, Bootstrap4 will be used as a CSS framework and Java will be used with SpringBoot for the backend

4. Entity-Relationship Model



Link to the project website:
https://cs353-group20-project-page.netlify.app/index.html

5. Relational Model

User(user_name, password, user_email, birth_year)

Employee(employee_name, salary)
    employee_name foreign key to user_name of User

Customer(customer_name, balance)
    customer_name foreign key to user_name of User

New_customer(customer_name, promotion_code)
    customer_name foreign key to customer_name of Customer

Repeat_customer(customer_name, rank, discount_code)
    customer_name foreign key to customer_name of Customer

Movie(movie_id, movie_title, production_year, director, avg_rating, genre, price)

Movie_request(request_id, movie_title, production_year, description, customer_name)
    customer_name foreign key to customer_name of Customer

Friend(add_request_name, add_name, friend_request_date)
    add_request_name foreign key to customer_name of Customer
    add_name foreign key to customer_name of Customer

Card(card_id,customer_name, card_number, exp_date, ccv, holder_name)

Actor(actor_id, actor_name, birth_year)

Acts(actor_id, movie_id)
    actor_id foreign key to actor_id of Actor
    movie_id foreign key to movie_id of Movie

Recommend(reccomended_user_name, recommender_user_name, movie_id, message)
    movie_id foreign key to movei_id of Movie
    recommended_user_name foreign key to customer_name of Customer
    recommender_user_name foreign key to customer_name of Customer

Favorite(<u>customer_name, movie_id</u>)
customer_name foreign key to customer_name of Customer
movie_id foreign key to movie_id of Movie

Rent(<u>customer_name, movie_id, rent_date</u>)
customer_name foreign key to customer_name of Customer
movie_id foreign key to movie_id of Movie

Movie_review(<u>review_id</u>, movie_id, customer_name, rating, review_message, spoiler, net_like_count, review_date)
customer_name foreign key to customer_name of Customer
movie_id foreign key to movie_id of Movie

Register_movie(<u>register_id,</u> employee_name, movie_id, registeration_date)

Delete_movie(<u>deletion_id</u>, employee_name, movie_id, deletion_date)

6. Checking 3NF

All of the tables are designed in 3NF form, in order to obtain minimum data duplication. To achieve this, tables are decomposed into other tables, like the case in user, employee and customer. These can be represented in one big table (user), but in this case, the salary of the customers should be empty and the balance of the employees should be null, which is not good for a design and creates redundancy. Thus, we have separated these tables. Same thing applies for the new customer and repeat customer case as well, we used a vertical mapping approach in representing the inherited structures, like in this case. This allowed us to enforce 3NF structures to our design. We tried to keep only the relevant attributes in the same table and when we are referencing another entity, like a movie or customer, we just use the primary key of it for reference. In this way, duplicating tuples are prevented and unrelated data is not stored in the same table. Other than the primary keys, no attribute depends on other attributes in a table. Further explanations of the tables and their dependencies are below.

User(<u>user_name</u>, password, user_email, birth_year)
user_name → password, user_email, birth_year

Employee(<u>employee_name</u>, salary)
employee_name → salary

Customer(<u>customer_name</u>, balance)
customer_name → balance

New_customer(customer_name, promotion_code)
customer_name → promotion_code

Repeat_customer(customer_name, rank, discount_code)
customer_name → rank, discount_code

Movie(movie_id, movie_title, production_year, director, avg_rating, genre, price)
movie_id → movie_title, production_year, director, avg_raiting, genre, price

Movie_request(request_id, movie_title, production_year, description, customer_name)
request_id → movie_title, production_year, description, customer_name

Friend(add_request_name, add_name, friend_request_date)
add_request_name, add_name → firend_request_date

Card(card_id,customer_name, card_number, exp_date, ccv, holder_name)
card_id → customer_name, card_number, exp_date, ccv, holder_name

Actor(actor_id, actor_name, birth_year)
actor_id → actor_name, birth_year

Acts(actor_id, movie_id)
There is no non-trivial functional dependency

Recommend(reccomended_user_name, recommender_user_name, movie_id, message)
reccomended_user_name, recommender_user_name, movie_id → message

Favorite(customer_name, movie_id)
There is no non-trivial functional dependency

Rent(customer_name, movie_id, rent_date)
There is no non-trivial functional dependency

Movie_review(review_id, movie_id, customer_name, rating, review_message, spoiler, net_like_count, review_date)
review_id → movie_id, customer_name, rating, review_message, spoiler, net_like_count, review_date

Register_movie(register_id, employee_name, movie_id, registeration_date)
register_id → employee_name, movie_id, registeration_date

Delete_movie(<u>deletion_id</u>, employee_name, movie_id, deletion_date)
    deletion_id → employee_name, movie_id, deletion_date

7. SQL Queries for Topic Specific Functionalities

Note: Assume that the "input_attributes" are the inputs that will be taken from the user for a search operation

1a:
** Search by title, director, production year, genre or actor **

SELECT * FROM movie WHERE movie_title = input_title;

SELECT * FROM movie WHERE director = input_director;

SELECT * FROM movie WHERE production_year = input_year;

SELECT * FROM movie WHERE production_year = 2022;

SELECT * FROM movie WHERE genre = input_genre;

SELECT * FROM movie NATURAL JOIN acts NATURAL JOIN actor WHERE actor = input_actor;

1b:
** Filter the results by price or rating, and sort them **

SELECT * FROM movie ORDER BY production_year DESC;

SELECT * FROM movie ORDER BY avg_rating DESC;

SELECT * FROM movie ORDER BY price ASC;

SELECT * FROM movie WHERE avg_rating BETWEEN input_lower_bound AND input_upper_bound ;

SELECT * FROM movie WHERE avg_rating BETWEEN 8 AND 9;

SELECT * FROM movie WHERE price BETWEEN 2 AND 4;

1c:
SELECT * FROM movie WHERE movie_id = selected_movie_id;

1d:
Return date of the rented movie is automatically set to 1 month later.

1e:
Note: this_customer_name is the username of the customer that is willing to rent the movie, selected_movie_id is the selected movie's id and today is the date that the renting takes place.

```
INSERT INTO Rent VALUES (this_customer_name, selected_movie_id, today);

UPDATE Customer SET balance = balance - movie_price
WHERE customer_name = this_customer_name;
```

8. More SQL Queries

Assuming default engine is InnoDB
if not, for all tables, the following query will be run.
ALTER TABLE <table_name> ENGINE=InnoDB;

a. Table and Trigger Creation Queries

```
CREATE TABLE User(
   user_name varchar(30),
   password varchar(30),
   user_email varchar(30),
   birth_year date,
   PRIMARY KEY(user_name));


CREATE TABLE Employee(
   employee_name varchar(30),
   salary int,
        PRIMARY KEY (employee_name),
   FOREIGN KEY (employee_name) REFERENCES User(user_name) ON UPDATE
CASCADE ON DELETE CASCADE);
```

```
CREATE TABLE Customer(
   customer_name varchar(30),
   balance float,
        PRIMARY KEY (customer_name),
   FOREIGN KEY (customer_name) REFERENCES User(user_name) ON UPDATE
CASCADE ON DELETE CASCADE);


CREATE TABLE New_customer(
   customer_name varchar(30),
   promotion_code char(10),
        PRIMARY KEY (customer_name),
   FOREIGN KEY (customer_name) REFERENCES Customer(customer_name) ON
UPDATE CASCADE ON DELETE CASCADE);


CREATE TABLE Repeat_customer(
   customer_name varchar(30),
   rank varchar(20),
   discount_code char(10),
        PRIMARY KEY (customer_name),
   FOREIGN KEY (customer_name) REFERENCES Customer(customer_name) ON
UPDATE CASCADE ON DELETE CASCADE);


CREATE TABLE Movie(
   movie_id int AUTO_INCREMENT,
   movie_title varchar(50),
   production_year int,
   director varchar(30),
   avg_rating float,
   genre varchar(20),
   price int,
        PRIMARY KEY (movie_id),
   CHECK (price >= 0));
```

```
CREATE TABLE Movie_request(
   request_id int AUTO_INCREMENT,
   movie_title varchar(50),
   production_year int,
   description varchar(300),
   customer_name varchar(30) NOT NULL,
   PRIMARY KEY (request_id),
   FOREIGN KEY (customer_name) REFERENCES Customer(customer_name) ON
UPDATE CASCADE ON DELETE CASCADE);


CREATE TABLE Friend(
   add_request_name varchar(30) NOT NULL,
   add_name varchar(30) NOT NULL,
   friend_request_date date,
   PRIMARY KEY (add_request_name, add_name),
   FOREIGN KEY (add_request_name) REFERENCES Customer(customer_name)
ON UPDATE CASCADE ON DELETE CASCADE,
   FOREIGN KEY (add_name) REFERENCES Customer(customer_name) ON
UPDATE CASCADE ON DELETE CASCADE);



CREATE TABLE Card(
   card_id int AUTO_INCREMENT,
   customer_name varchar(30) NOT NULL,
   card_number char(16) NOT NULL,
   exp_date char(5) NOT NULL,
   ccv char(3) NOT NULL,
   holder_name varchar(30),
   PRIMARY KEY (card_id),
   FOREIGN KEY (customer_name) REFERENCES Customer(customer_name) ON
UPDATE CASCADE ON DELETE CASCADE);


CREATE TABLE Actor(
   actor_id int AUTO_INCREMENT,
   actor_name varchar(30) NOT NULL,
   birth_year int,
   PRIMARY KEY (actor_id));
```

```
CREATE TABLE Acts(
   actor_id int NOT NULL,
   movie_id int NOT NULL,
   PRIMARY KEY (actor_id, movie_id),
   FOREIGN KEY (actor_id) REFERENCES Actor(actor_id) ON UPDATE CASCADE
ON DELETE CASCADE,
   FOREIGN KEY (movie_id) REFERENCES Movie(movie_id) ON UPDATE
CASCADE ON DELETE CASCADE);


CREATE TABLE Recommend(
   recommended_user_name varchar(30) NOT NULL,
   recommender_user_name varchar(30) NOT NULL,
   movie_id int NOT NULL,
   message varchar(300),
   PRIMARY KEY (recommended_user_name, recommender_user_name,
movie_id),
   FOREIGN KEY (recommended_user_name) REFERENCES
Customer(customer_name) ON UPDATE CASCADE ON DELETE CASCADE,
   FOREIGN KEY (recommender_user_name) REFERENCES
Customer(customer_name) ON UPDATE CASCADE ON DELETE CASCADE,
   FOREIGN KEY (movie_id) REFERENCES Movie(movie_id) ON UPDATE
CASCADE ON DELETE CASCADE);


CREATE TABLE Favorite(
   customer_name varchar(30) NOT NULL,
   movie_id int NOT NULL,
   PRIMARY KEY (customer_name, movie_id),
   FOREIGN KEY (customer_name) REFERENCES Customer(customer_name) ON
UPDATE CASCADE ON DELETE CASCADE,
   FOREIGN KEY (movie_id) REFERENCES Movie(movie_id) ON UPDATE
CASCADE ON DELETE CASCADE);


CREATE TABLE Register_movie(
   register_id int AUTO_INCREMENT,
   employee_name varchar(30) NOT NULL,
   movie_id int NOT NULL,
   registration_date date,
   PRIMARY KEY (register_id));
```

```
CREATE TABLE Delete_movie(
   deletion_id int AUTO_INCREMENT,
   employee_name varchar(30) NOT NULL,
   movie_id int NOT NULL,
   deletion_date date,
   PRIMARY KEY (deletion_id));

CREATE TABLE Rent(
   customer_name varchar(30) NOT NULL,
   movie_id int NOT NULL,
   rent_date date NOT NULL,
   PRIMARY KEY (customer_name, movie_id, rent_date),
   FOREIGN KEY (customer_name) REFERENCES Customer(customer_name) ON
UPDATE CASCADE ON DELETE CASCADE,
   FOREIGN KEY (movie_id) REFERENCES Movie(movie_id) ON UPDATE
CASCADE ON DELETE CASCADE);


CREATE TABLE Movie_review(
   review_id int AUTO_INCREMENT,
   movie_id int NOT NULL,
   customer_name varchar(30) NOT NULL,
   rating float,
   review_message varchar(300),
   spoiler tinyint,
   net_like_count int,
   review_date date,
   PRIMARY KEY (review_id),
   FOREIGN KEY (movie_id) REFERENCES Movie(movie_id) ON UPDATE
CASCADE ON DELETE CASCADE,
   FOREIGN KEY (customer_name) REFERENCES Customer(customer_name) ON
UPDATE CASCADE ON DELETE CASCADE,
   CHECK (rating >= 0 AND rating <= 10),
   CHECK (spoiler = 0 OR spoiler = 1));
```

```
DELIMITER $$
CREATE TRIGGER update_average_rating AFTER INSERT ON Movie_review
        FOR EACH ROW BEGIN
                IF NEW.rating IS NOT NULL THEN
                        UPDATE Movie
                        SET avg_rating = (SELECT  AVG(rating) FROM Movie_review
        WHERE movie_id = NEW.movie_id AND rating IS NOT NULL)
                        WHERE movie_id = NEW.movie_id;
                END IF;
        END;
$$
DELIMITER ;
```

b. Login Queries

```
SELECT  * FROM user WHERE user.user_name = input_user_name and
user.password = input_password;
```

c. Signup Queries

** Sign up for employee **

```
INSERT INTO user VALUES (user_name, password, user_email, birth_year);
INSERT INTO employee VALUES (user_name, salary);
```

** Sign up for customer **

```
INSERT INTO user VALUES (user_name, password, user_email, birth_year);
INSERT INTO customer VALUES (user_name, 0);
INSERT INTO new_customer VALUES(user_name, promotion_code);
```

d. Queries related to the Card Functionality (Additional Functionality)

** Adding a new card to the system **

```
INSERT INTO Card VALUES (customer_name, card_number, exp_date, ccv,
holder_name);
```

** Update a card **

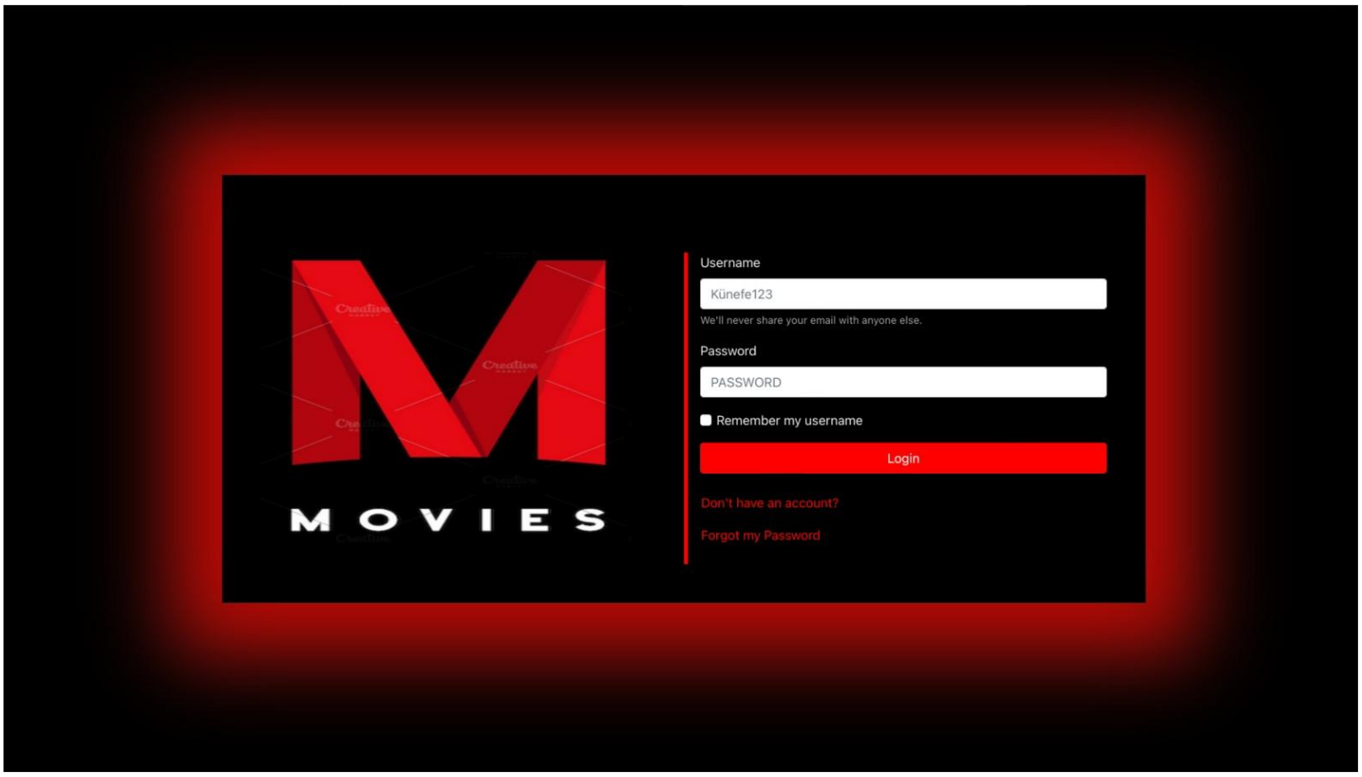
```
UPDATE Card
SET customer_name = input_name,  card_number = input_number, exp_date =
input_date, ccv = input_ccv , holder_name = input
WHERE card_id = input_card_id;
```

** Deleting a card from the system **

```
DELETE FROM Card WHERE card_id = input_card_id;
```

9. UI Mockups

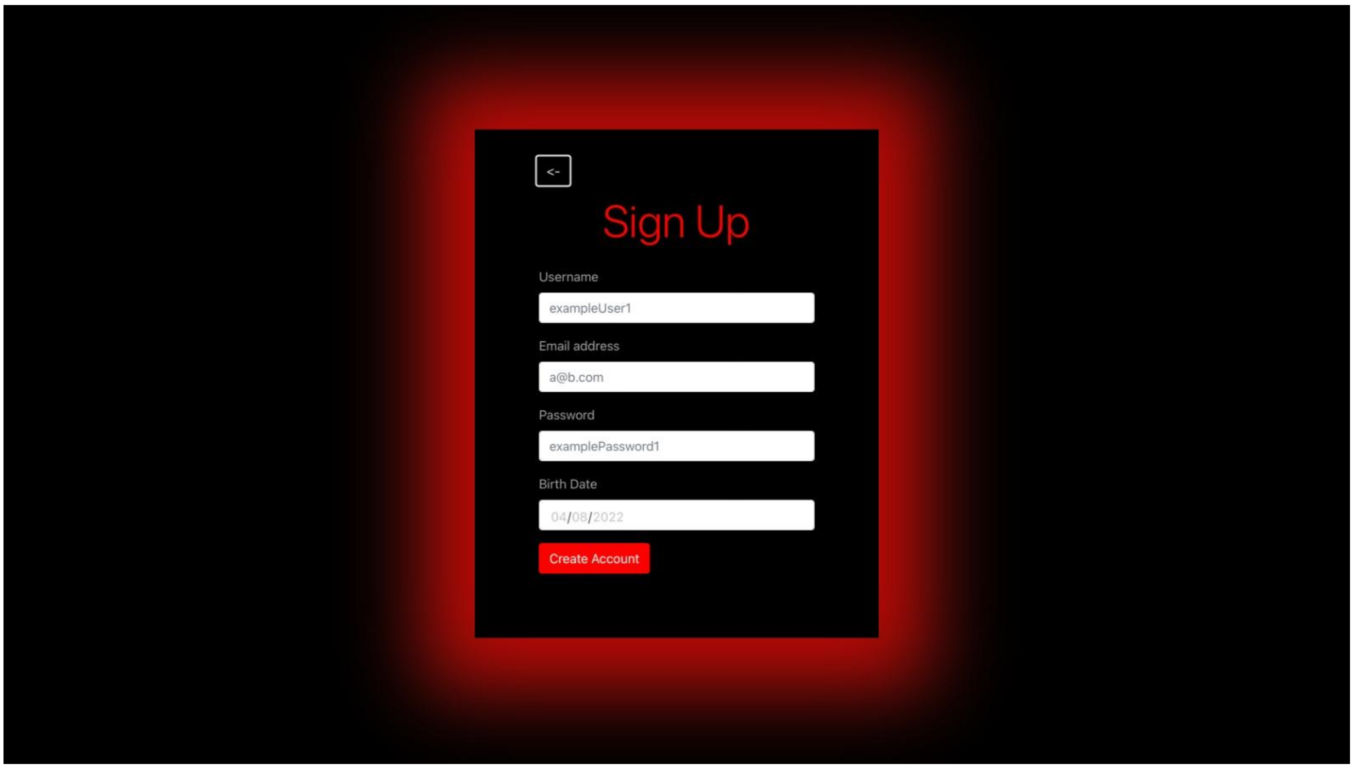
a. Login Page



Username
Künefe123
We'll never share your email with anyone else.
Password
PASSWORD
Remember my username

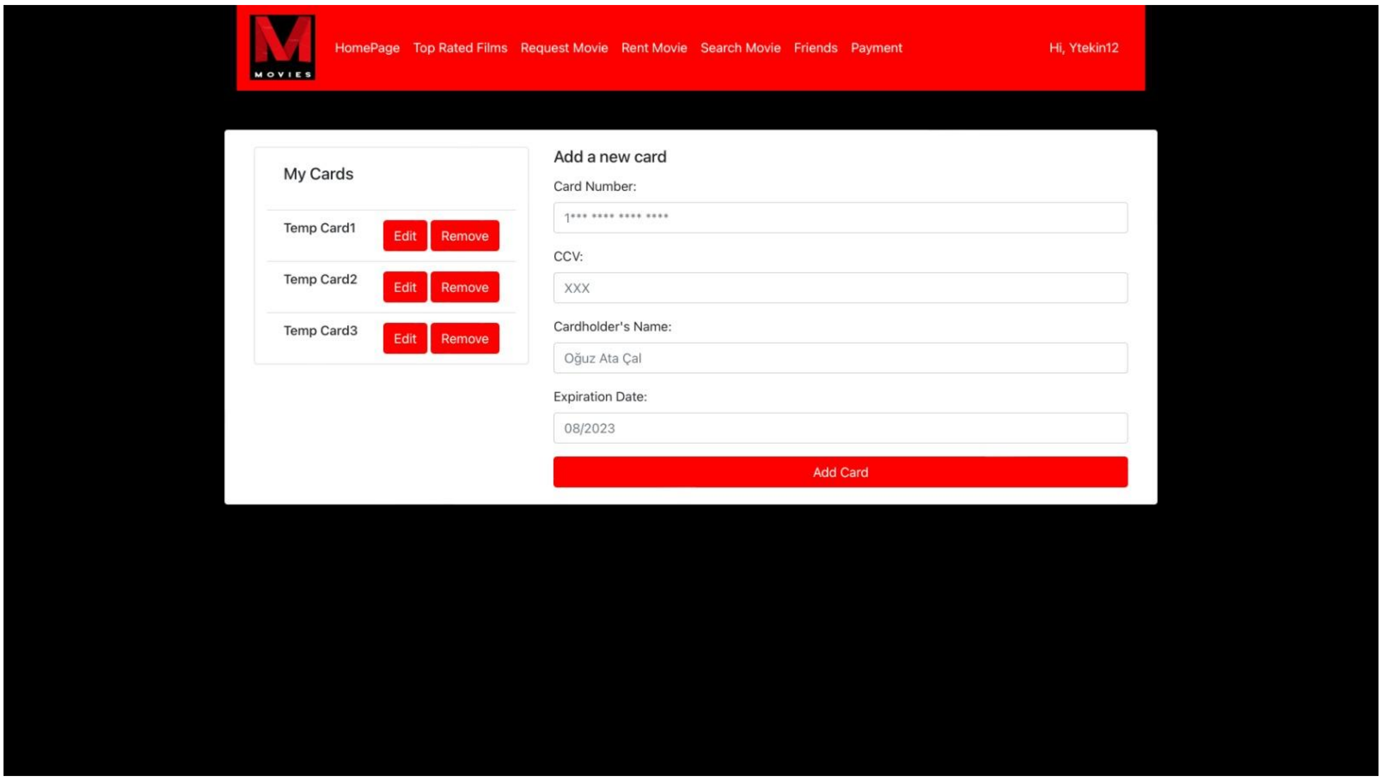
Login Page of the application. The user can enter their username and password and click on the login button to try their login process. If the user does not have an account, the “Don’t have an account?” the link below the login button must be clicked and the user will be redirected to the sign up page. Also, the user can click the remember my username checkbox to store their username in local storage which then later on can be obtained for their usage. Finally, if the user does not remember their password, they can change the password by “forgot my password” link below the sign up link.

b. Sign up page



Sign Up page of the application. In order to create an account, the user will enter the username, email address, password and birth date inputs and after giving appropriate data, the account can be created by clicking the create account button below.

c. Add Credit Card Page (additional feature)



My Cards

Temp Card1	Edit	Remove
Temp Card2	Edit	Remove
Temp Card3	Edit	Remove

Add a new card

Card Number: 1*** **** **** ****

CCV: XXX

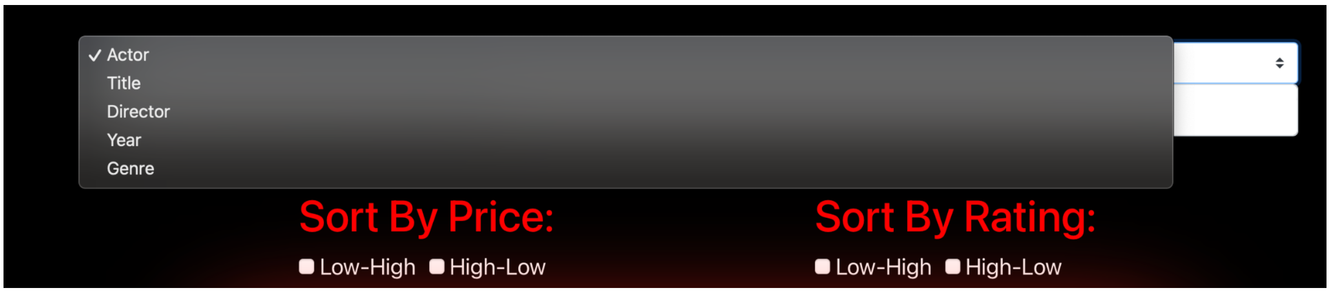
Cardholder's Name: Oğuz Ata Çal

Expiration Date: 08/2023

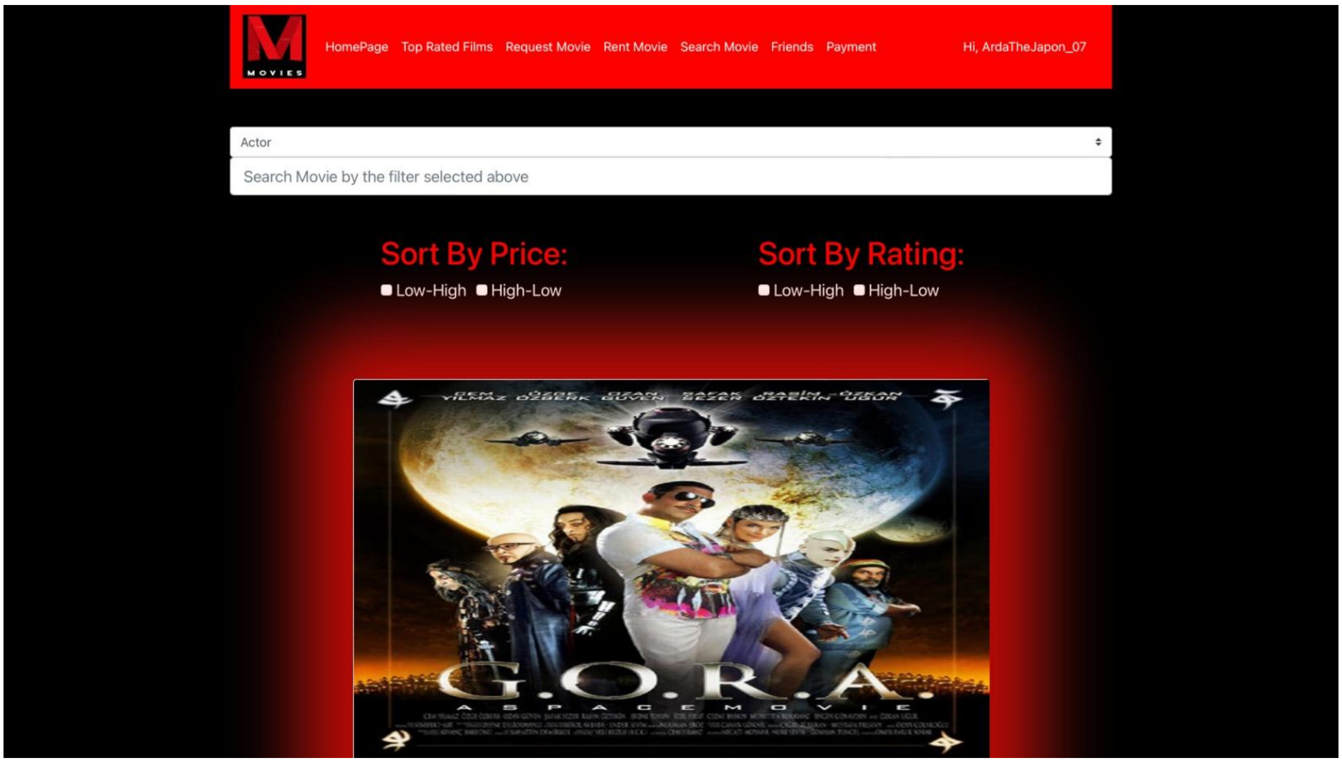
Add Card

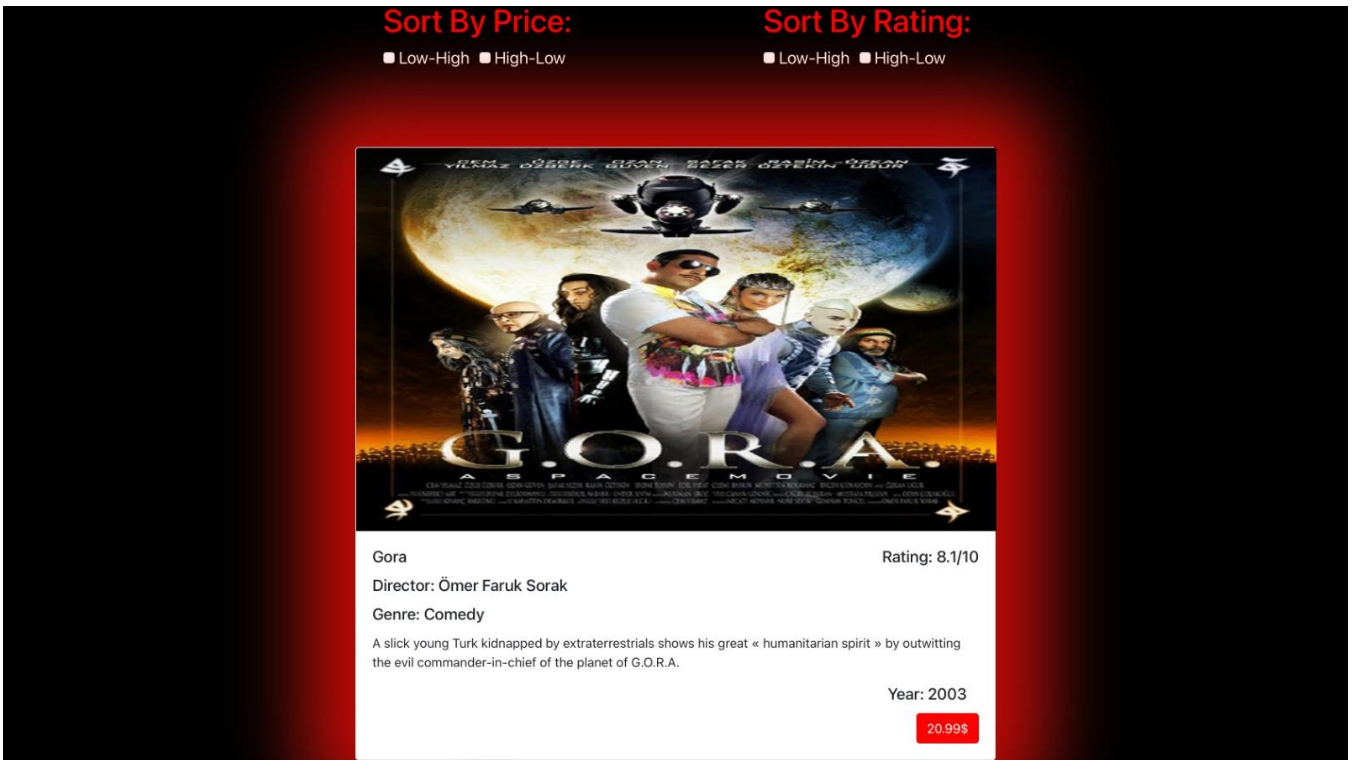
In this page, the user can see their cards in the system from my cards section. Also, the user can edit and remove them. In addition, in the right side of the page, there is an add new card form which allows the user to enter the correct credentials and insert their new card into the system. In terms of the navigation bar, most of the links are self-explanatory; however, the friends and payment parts can be further explained. Payment link will redirect the user to this page where he/she can edit and see their payment information. Friends link will redirect the user to a friendly activity page where users can review movies and recommend movies to other users.

d. Search Movie Page



In this page, the user can search for movies. Filtering is done by the select component right below the navbar and the search is done by the search bar below the select component. Also, the user can sort the search query by price or rating by the checkboxes below the Labels "Sort by Price" and "Sort by Rating".




Sort By Price:

Low-High High-Low

Sort By Rating:

Low-High High-Low

Gora

Rating: 8.1/10

Director: Ömer Faruk Sorak

Genre: Comedy

A slick young Turk kidnapped by extraterrestrials shows his great « humanitarian spirit » by outwitting the evil commander-in-chief of the planet of G.O.R.A.

Year: 2003

This is an example of a returned card component for a movie. The cover image, title, director, rating, genre, description, year and price of the movie can be seen. The user can rent the movie by clicking the red button with the price on it.