Assessment Documentation

Lojeen Shflwet

8th March 2018

Table of Contents

1. Background

This report documents the main stages and processes taken during the development of the attached solution. The requirements analysis, their design and finally their implementation will be specified.

2. Analysis

2.1. Requirements

The requirements were divided into two main categories: functional and non-functional requirements. The functional requirements define the main functionality of the developed prototype while the non-functional requirements describe the prototype's actual operations and define its qualities.

2.1.1. Functional Requirements

2.1.1.1. User Requirements

1. View latest movies
2. Search movies
3. View movie details
4. View cast details
5. Share movie link
6. Add or delete movie to/from favorites
7. View similar movies

2.1.2. Non-Functional Requirements

2.1.2.1. Interface Requirements

1. The layout and style will be defined by using CSS and HTML.

2. The interface should be consistent throughout the prototype.
3. The prototype should be simple and easy to navigate.

2.1.3. Risks and Constraints

2.1.3.1. Constraints

The following illustrates the main constraints for this assessment:

- Time constraints as the assessment components: prototype development and documentation are to be completed within 5 working days. Working within 7am to 6pm internship during the assessment significantly impacts the progress.

2.1.3.2. Risks

The following illustrates the main project risks:

- API calling with C# is not within the my current skill set. Due to the specified constraints, the acquisition of this skill is a risk in this assessment as it might take longer than anticipated to learn and implement affecting the overall submission.

In order to mitigate this risk, the assessment has been implemented via a local SQL Server database with attempting API calling in another solution.

2.1.4. Case Tools and Techniques

Computer-Aided Software Engineering (CASE) is a technique that uses tools to analyse, design and implement systems and software. The advantage of using CASE tools is that they ease the build of information systems, as they increase the productivity by modelling the systems throughout their stages and consequently increase their quality. CASE tools provide an overall system development framework that supports the different methodology categories: agile, structured analysis and object-oriented analysis. The chosen CASE tools for the analysis, process modelling and design for this project are as follows:

Analysis Techniques

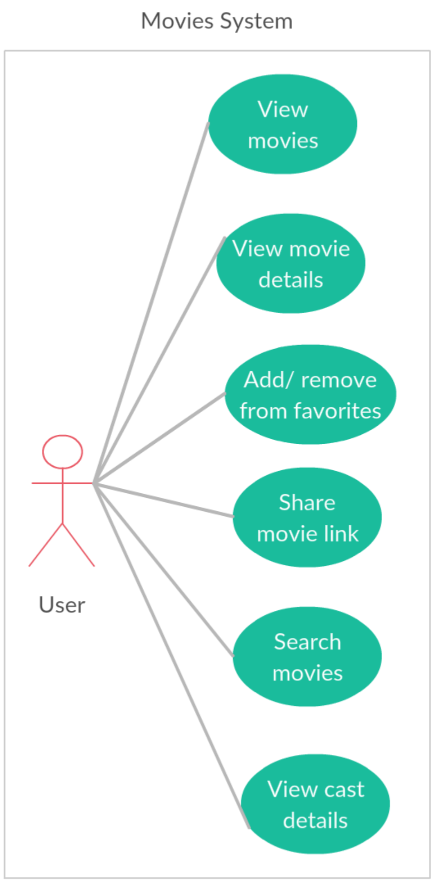
1. User Stories.
2. Use Case Diagram.

Design Techniques:

1. Entity Relationship Diagram.
2. Data Dictionaries.

2.1.5. Use Case Diagram

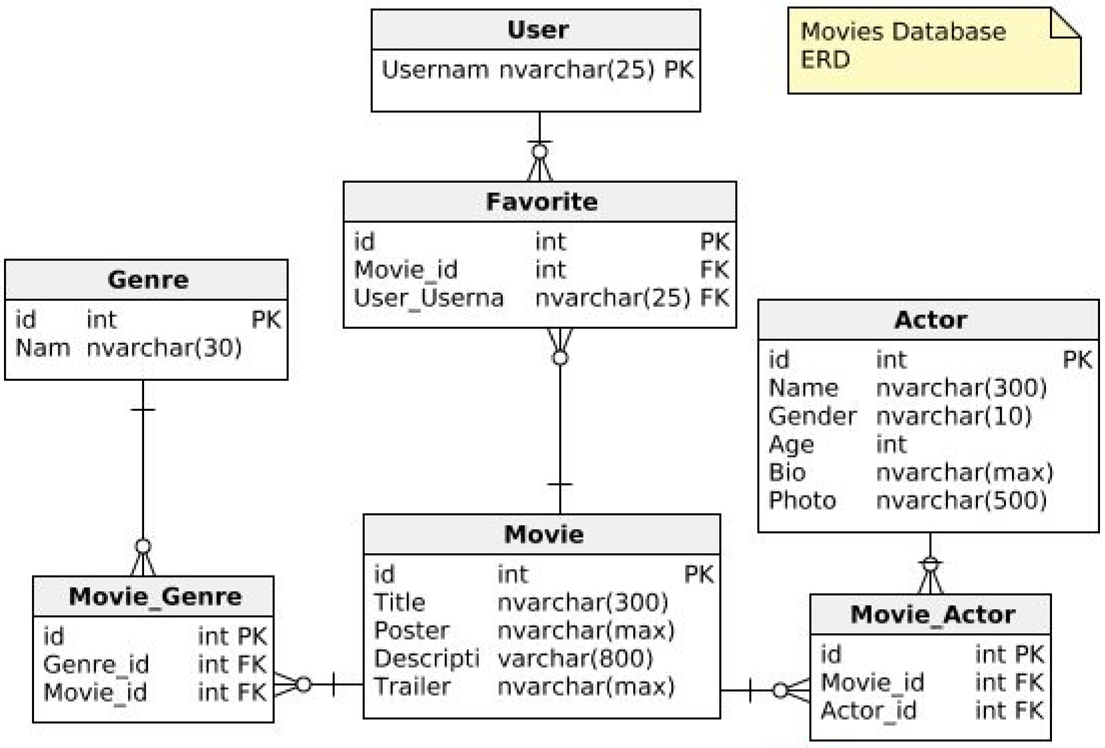
The Use Case diagram shown overleaf illustrates the main functionalities the user can perform. The diagram use cases and the relationships between them are derived from the functional requirements described in this report.



3. Design

3.1. Entity Relationship Diagram

An Entity Relationship Diagram (ERD) defines a system's entities and attributes and illustrates the relationships between them. It was used to design the developed prototype's database as shown overleaf. The ERD was designed to include entities that meet the requirements stated in this report.



3.2. Data Dictionary

Data dictionary is a structured design tool that is used as a storehouse of information about the data stored in the system. Data dictionaries describe all data elements and illustrate their relationships and the content of the entities. The tables below represent the data dictionary for the ERD designed and their references.

3.2.1. Tables

3.2.1.1. Table User

Column name	**Type**	**Properties**	**Description**
Username	nvarchar(25)	PK	

3.2.1.2. Table Movie

Column name	Type	Properties	Description
id	int	PK	
Title	nvarchar(300)		
Poster	nvarchar(max)		
Description	varchar(800)		
Trailer	nvarchar(max)		

3.2.1.3. Table Movie_Actor

Column name	Type	Properties	Description
id	int	PK	
Movie_id	int		
Actor_id	int		

3.2.1.4. Table Actor

Column name	Type	Properties	Description
id	int	PK	
Name	nvarchar(300)		
Gender	nvarchar(10)		
Age	int		
Bio	nvarchar(max)		
Photo	nvarchar(500)		

3.2.1.5. Table Genre

Column name	**Type**	**Properties**	**Description**
id	int	PK	
Name	nvarchar(30)		

3.2.1.6. Table Movie_Genre

Column name	**Type**	**Properties**	**Description**
id	int	PK	
Genre_id	int		
Movie_id	int		

3.2.1.7. Table Favorite

Column name	**Type**	**Properties**	**Description**
id	int	PK	
Movie_id	int		
User_Username	nvarchar(25)		

3.2.2. References

3.2.2.1. Reference Movie_Genre_Genre

Genre	0..*	Movie_Genre
id	<->	Genre_id

3.2.2.2. Reference Movie_Genre_Movie

Movie	0..*	Movie_Genre
id	<->	Movie_id

3.2.2.3. Reference Movie_Actor_Movie

Movie	0..*	Movie_Actor
id	<->	Movie_id

3.2.2.4. Reference Movie_Actor_Actor

Actor	0..*	Movie_Actor
id	<->	Actor_id

3.2.2.5. Reference Favorite_Movie

Movie	0..*	Favorite
id	<->	Movie_id

3.2.2.6. Reference Favorite_User

User	0..*	Favorite
Username	<->	User_Username

3.2.3. The Front End

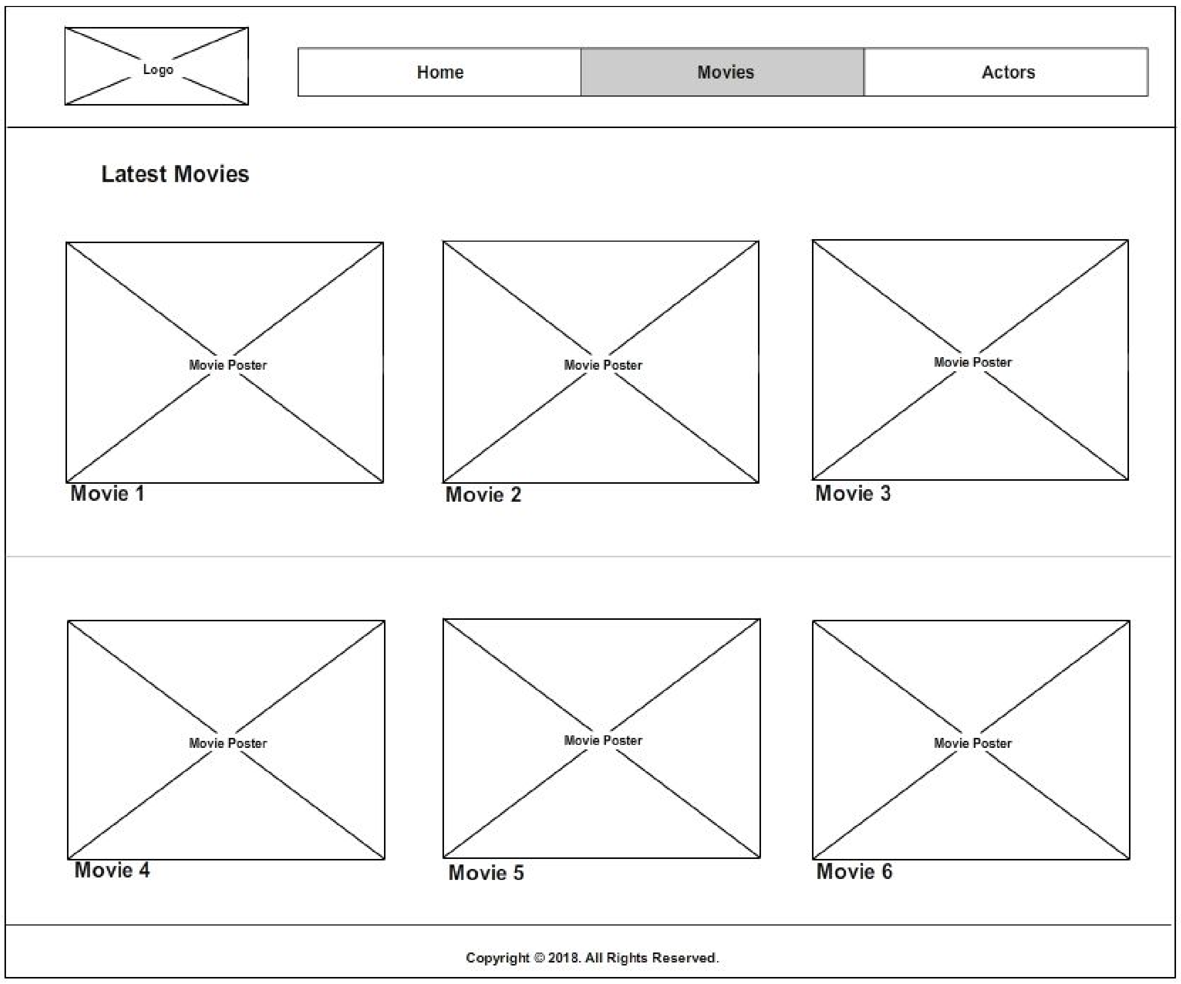
Keeping a user-friendly interface whilst providing the required functionality is important to encourage repeated visits. The prototype should be displayed as intended on the most commonly used Web browsers: Google Chrome, Mozilla Firefox and Microsoft Internet Explorer. The use of available fonts and colours on the users' computers are other aspects that need to be considered for the prototype design. The following illustrates the main user interface design elements for the developed prototype.

3.2.3.1. Wireframes

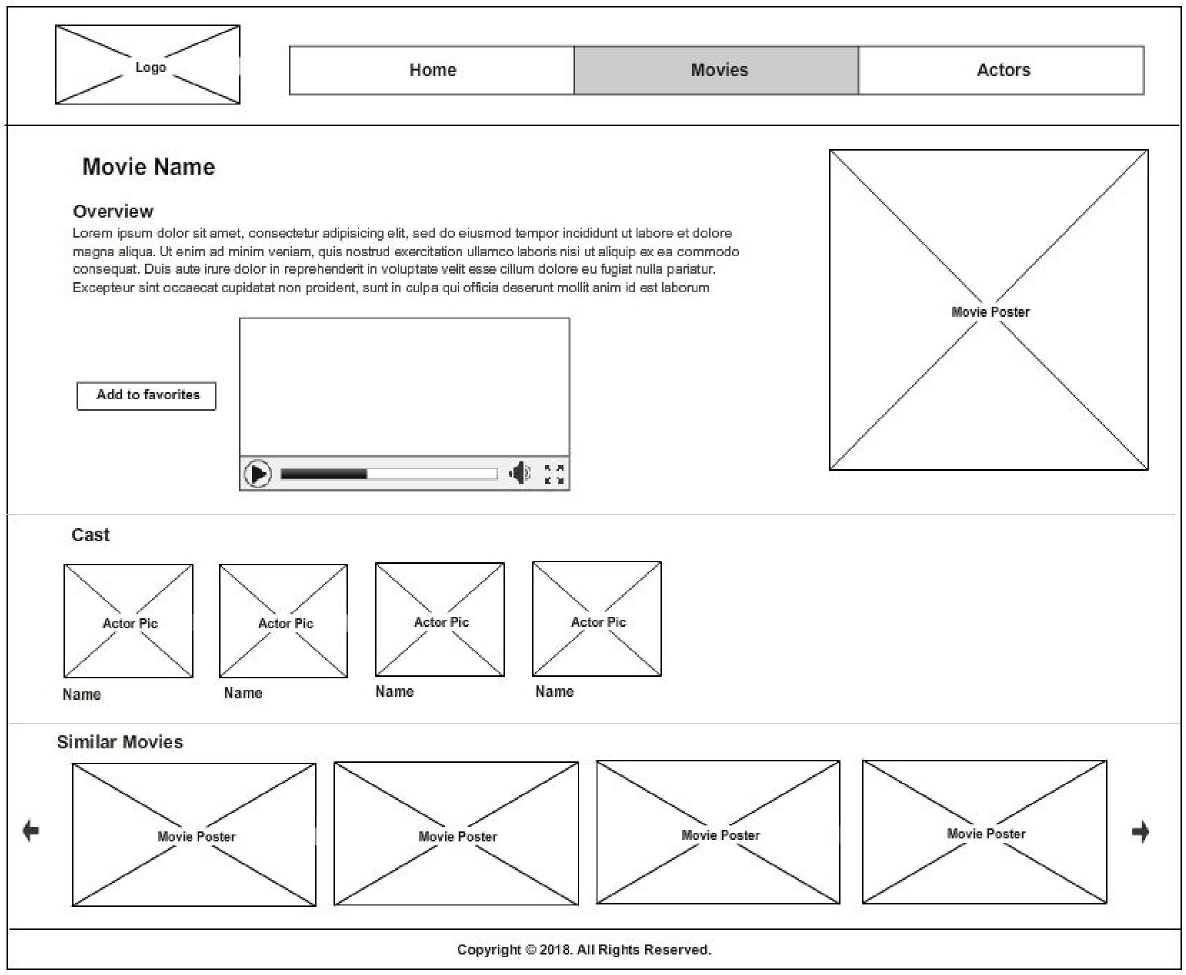
Wireframes are page design representations that are used to plan the navigational architecture, structure and design of website's pages and their elements. For the developed prototype, a header and a footer with one main column page layout was used for the Master page. These act as page frames for all the other pages, which unifies their designs. This page structure was achieved by using HTML div tags and CSS.

Separating the layout into three main areas: header, content area and footer allows the header and footer functionalities to be clearly identified. The header navigation mechanisms consist of the main navigation bar, which links to the main pages in the system, in addition to hyperlinks that navigate to the access pages on the prototype, such as the Login, Logout and Register hyperlinks.

The wireframe below illustrates the design for the ‘movies list’ page.

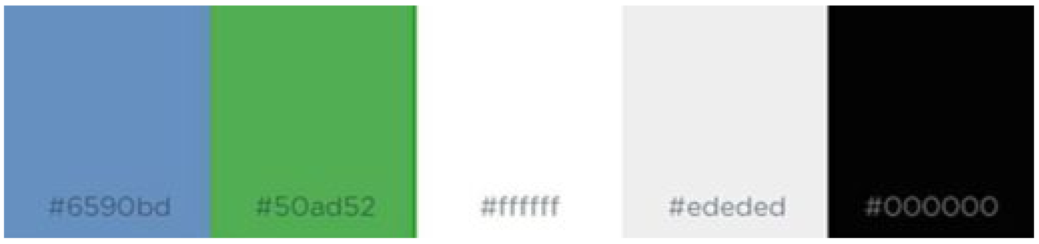


The wireframe below illustrates the layout for the movie details page.


Logo
Home
Movies
Actors
Movie Name
Overview
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum
Add to favorites
Movie Poster
Cast
Actor Pic
Name
Actor Pic
Name
Actor Pic
Name
Actor Pic
Name
Similar Movies
Movie Poster
Movie Poster
Movie Poster
Movie Poster
Copyright © 2018. All Rights Reserved.

3.2.3.2. Colour Scheme

The colour scheme was used as follows:



Element	**Colour**	**Description**
Header	#6590BD	This colour was used for the header to clearly separate it from the rest of the prototype's sections.
Background	#FFFFFF	White was used for the background in order to have a clean design.
Footer	#EDEDED	This colour was used for the footer to clearly separate it from the rest of the prototype's sections.
Content	#000000	This colour was used for the font of both the headings and paragraphs to have a sufficient contrast with the background.

4. Implementation

4.1. The ASP.NET - SQL Database Prototype

The following user stories were implemented in this prototype as follows:

- ***As a user, I can see the latest movies.***

This feature has been implemented through the use of ASP.NET Data Lists, where a list control was linked to the database to retrieve the stored movies. These then were ordered by date in a descending order.

- ***As a user, I can search movies.***

The search functionality was implemented by creating two lists: one that displays the movies list before the search is performed and the second list displays the search results; each is hidden if the other list is displayed.

- ***As a user, I can add a movie or remove it from my favorites.***

This functionality has been implemented by creating a side navigation bar that links to the favorites list. This bar is hidden until a login with the user role ‘Member’ is performed. Once the ‘add to favorites’ link is clicked on, it automatically adds a new record for the logged in member in the ‘favorites’ database table. The triggered onclick method from the code behind file passes the movie id from the url, the username from ASP.NET security controls and the favorite id, which is an auto increment field, as a new entry.

- ***As a user, I can see movie details (Movie Name, Poster, Movie Description, Genres)***

The required data for this feature was fetched from the database with a join SQL query.

- ***As a user, I can share movie link to IMDB site.***

As the IMDB site allows sharing movies through its lists feature and not through direct sharing, this website was replaced with Twitter that provides the same functionality.

- ***As a user, I can see similar movies for each movie.***

Grouping movies by their genres was used when retrieving them from the database for this feature, allowing for only movies from the same genre to be displayed beneath a film.

- ***As a user, I can see as more movie details (Cast names and posters).***

The required data for this feature was fetched from the database with a join SQL query.

- ***As a user, I can see star details with the list of movies casted in.***

The required data for this feature was fetched from the database with a join SQL query. The same was performed for retrieving the movies a star was casted in.

4.2. The API Calling Attempt

API calling was attempted and raw data was returned. However, due to the lack of time, passing beyond this point was not possible. Below are examples from the returned data.

API Key: eb13fbf4cf582aa9479b2b2723c54f22

C# code for searching with the query ‘Movie’:

```
var client = new
RestClient("https://api.themoviedb.org/3/search/company?page=1&query=movie&api_key=eb13f
bf4cf582aa9479b2b2723c54f22");
var request = new RestRequest(Method.GET);
request.AddParameter("undefined", "{}", ParameterType.RequestBody);
IRestResponse response = client.Execute(request);
```

Sample returned data:

```
{
   "page": 1,
   "results": [
      {
         "id": 3845,
         "logo_path": null,
         "name": "Gigantic Movie"
      },
      {
         "id": 3500,
         "logo_path": null,
         "name": "Move Movie"
      },
      {
         "id": 4291,
         "logo_path": null,
         "name": "Movie%26Art"
      },...etc
```

C# code for retrieving a list of genres:

```
var client = new
RestClient("https://api.themoviedb.org/3/genre/movie/list?language=en-US&api_key=eb13fbf4cf5
82aa9479b2b2723c54f22");
var request = new RestRequest(Method.GET);
request.AddParameter("undefined", "{}", ParameterType.RequestBody);
IRestResponse response = client.Execute(request);
```

Sample returned data:

```
{
   "genres": [
      {
         "id": 28,
         "name": "Action"
      },
      {
         "id": 12,
         "name": "Adventure"
      },
      {
         "id": 16,
         "name": "Animation"
      },...etc
```

5. Appendix A

The SQL script used to create the database:

```
-- tables

-- Table: Actor

CREATE TABLE Actor (

        id int  NOT NULL IDENTITY,

        Name nvarchar(300)  NOT NULL,

        Gender nvarchar(10)  NOT NULL,

        Age int  NOT NULL,

        Bio nvarchar(max)  NOT NULL,

        Photo nvarchar(500)  NOT NULL,

        CONSTRAINT Actor_pk PRIMARY KEY  (id)

);


-- Table: Favorite

CREATE TABLE Favorite (

        id int  NOT NULL IDENTITY,

        Movie_id int  NOT NULL,

        User_Username nvarchar(25)  NOT NULL,

        CONSTRAINT Favorite_pk PRIMARY KEY  (id)

);


-- Table: Genre

CREATE TABLE Genre (

        id int  NOT NULL IDENTITY,

        Name nvarchar(30)  NOT NULL,
	CONSTRAINT Genre_pk PRIMARY KEY  (id)
);

-- Table: Movie
CREATE TABLE Movie (
	id int  NOT NULL IDENTITY,
	Title nvarchar(300)  NOT NULL,
	Poster nvarchar(max)  NOT NULL,
	Description varchar(800)  NOT NULL,
	Trailer nvarchar(max)  NOT NULL,
	CONSTRAINT Movie_pk PRIMARY KEY  (id)
);

-- Table: Movie_Actor
CREATE TABLE Movie_Actor (
	id int  NOT NULL IDENTITY,
	Movie_id int  NOT NULL,
	Actor_id int  NOT NULL,
	CONSTRAINT Movie_Actor_pk PRIMARY KEY  (id)
);

-- Table: Movie_Genre
CREATE TABLE Movie_Genre (
	id int  NOT NULL IDENTITY,
	Genre_id int  NOT NULL,
	Movie_id int  NOT NULL,
	CONSTRAINT Movie_Genre_pk PRIMARY KEY  (id)
```

```
);


-- Table: User
CREATE TABLE "User" (
        Username nvarchar(25)  NOT NULL,
        CONSTRAINT User_pk PRIMARY KEY  (Username)
);


-- foreign keys
-- Reference: Favorite_Movie (table: Favorite)
ALTER TABLE Favorite ADD CONSTRAINT Favorite_Movie
        FOREIGN KEY (Movie_id)
        REFERENCES Movie (id);


-- Reference: Favorite_User (table: Favorite)
ALTER TABLE Favorite ADD CONSTRAINT Favorite_User
        FOREIGN KEY (User_Username)
        REFERENCES "User" (Username);


-- Reference: Movie_Actor_Actor (table: Movie_Actor)
ALTER TABLE Movie_Actor ADD CONSTRAINT Movie_Actor_Actor
        FOREIGN KEY (Actor_id)
        REFERENCES Actor (id);


-- Reference: Movie_Actor_Movie (table: Movie_Actor)
ALTER TABLE Movie_Actor ADD CONSTRAINT Movie_Actor_Movie
        FOREIGN KEY (Movie_id)
        REFERENCES Movie (id);

-- Reference: Movie_Genre_Genre (table: Movie_Genre)
ALTER TABLE Movie_Genre ADD CONSTRAINT Movie_Genre_Genre
        FOREIGN KEY (Genre_id)
        REFERENCES Genre (id);

-- Reference: Movie_Genre_Movie (table: Movie_Genre)
ALTER TABLE Movie_Genre ADD CONSTRAINT Movie_Genre_Movie
        FOREIGN KEY (Movie_id)
        REFERENCES Movie (id);

-- End of file.
```