

Project Design

PROJECT NAME: JUST ANOTHER MOVIE

GROUP MEMBERS: EMMETT STOREY, ROBERT JONES, HAYDEN RICHARDS, AND KAITLYN HARDIN

Abstract:

Just Another Movie is an application in which the user will be prompted to enter information about what type of movies they like. Users will be able to search by different genres or different actors or actresses that they like. The application will take that information and using different classification and recommendation techniques the application will search through our database and will return a list of movies that the user should enjoy. Users can also remove movies from the list if they have either seen it before or do not like the movie.

Tools and Technologies:

- Ø Windows OS

- Ø Visual Studio 2019

- Ø IMDB database

- Ø Discord communication

- Ø Desktop application

- Ø Kaggle dataset

- Ø C#, SQL

- Ø Github

Requirements:

Frontend

1. User can load the startup Window
 - 1.1. Design of the application will be seen immediately
 - 1.1.1. Includes app logo
 - 1.1.2. Includes the app's color scheme
2. New window in application loads for searching

- 2.2. User can search by inputting information about movie attributes
 - 2.2.1. User will put movie attribute information in input boxes for each attribute
 - 2.2.1.1. Input box for actors
 - 2.2.1.2. Drop-down menu for year
 - 2.2.1.3. Drop-down menu for genre
 - 2.2.1.4. Input box for director
 - 2.2.1.5. Input box for title
 - 2.3. Error messages will be displayed to the user for any invalid searches
 - 2.3.1. Invalid searches for actors, director, and title:
 - 2.3.1.1. Return message: "no results found"
- 3. Results Window

System outputs the recommended movie list in one page for the user using lazy loading.

The user will scroll through the movie list using left and right arrows.

 - 3.2. Search by combining movie attributes
 - 3.2.1. The system will output a movie list based on movie attributes inputted by the user.
 - 3.2.1.1. The user does not have to include information for all attributes before the system generates a movie list.
 - 3.2.2. The recommended movies are selected based on our selecting Model
 - 3.2.2.1. Model: ML.NET
 - 3.2.2.1.1. provided by our IDE: Visual Studio
- 4. Window for Movie List
 - 4.1. Shows list of movies
 - 4.1.1. The user can peruse through the list of movies
 - 4.2. The window includes a back button
 - 4.2.1. This returns the user the previous window
 - 4.3. The user can delete a movie from the list
 - 4.3.1. Each movie will have an "X" button that means delete
 - 4.3.2. The user will have a message window pop up to undo any recent deletes
 - 4.4. The user can click on the movie to get more information about that movie.
 - 4.4.1. New window will pop once the movie is clicked
 - 4.4.1.1. Movie window will include more information about the movie
 - 4.4.1.1.1. Year
 - 4.4.1.1.2. Title
 - 4.4.1.1.3. Director
 - 4.4.1.1.4. Date
 - 4.4.1.1.5. Genre
 - 4.4.1.1.6. Cast and Crew
 - 4.4.1.1.7. Average ratings
 - 4.4.1.2. The user will be able to create a new movie list based on the movie.
 - 4.4.1.2.1. There will be a "create list" button for the user
 - 4.4.1.2.2. Movie list format will be the same

Backend

5. Make connection to MS SQL Server
 - 5.1. Get connection string
 - 5.1.1. Create function to get the connection string
6. Get Movie information from the user
 - 6.1. Get input based by actor.
 - 6.2. Get input based on year.
 - 6.3. Get based on Genre.
 - 6.4. Get based on director
 - 6.5. Get input based on Title.
 - 6.5.1. If input consists solely of common words such as: the, and, or it will be considered invalid.
 - 6.6. Pass the criteria to the ML algorithm
7. Return list of movies
 - 7.1. Returns list of movies that match search criteria involving multiple inputs.
 - 7.2. Returns list of movies that match search criteria involving singular input.
 - 7.2.1. Returns list based by year
 - 7.2.2. Returns list based by director
 - 7.2.3. Returns list based by actor
 - 7.2.4. Returns list based by title
 - 7.3. Returns an error message if invalid input is given in search criteria.
 - 7.3.1. Returns an empty list if no movie matches search criteria.
 - 7.4. Receive list from ML algorithm.
8. Output list of movies to frontend component
 - 8.1. List ordered by user ratings
User can remove movies from list
 - 8.1.1. User can click a button to remove a movie from a list
 - 8.1.1.1. The movie the user selected will be removed from the list
 - 8.2. User can traverse the movie list
 - 8.2.1. Users can click on a right arrow to scroll right on the list.
 - 8.2.2. Users can click on a left arrow to scroll left on the list.
 - 8.3. User can click on movie
 - 8.3.1. User can look at movie details
 - 8.3.1.1. User can see the year it was made
 - 8.3.1.2. User can see director
 - 8.3.1.3. User can see title
 - 8.3.1.4. User can see genre
 - 8.3.1.5. User can see Movie poster if available
 - 8.3.2. User can create a new movie list from movie
 - 8.3.2.1. Create a list of movies they want to watch

Database

9. Import .csv files from our movie dataset
 - 9.1. Online data source: Kaggle dataset
 - 9.2. Movie data files include:
 - 9.2.1. Credits
 - 9.2.2. Keywords
 - 9.2.3. Metadata
 - 9.2.4. User Rating
 - 9.2.5. User demographics
10. Store datasets in our chosen database server
 - 10.1. Database server: MS SQL Server
 - 10.2. Group data based on original files from the movie dataset
 - 10.2.1. Movies.csv -> movies table
11. Prepare the data
 - 11.1. Delete any unnecessary data or any data that will not be used for our project
 - 11.1.1. timestamps
 - 11.1.2.
 - 11.2. Combine data from other files based on relationship
 - 11.2.1. For example: User ratings being in the same table with movie information

Machine Learning (ML.NET)

- 1.1. Load data into model
 - 1.1.1. Create the connection string to the SQL server
 - 1.1.2. Load data into the model from the SQL server
- 1.2. Build and train model
 - 1.2.1. Add the appropriate Nuget packages
 - 1.2.1.1. Microsoft.ML
 - 1.2.1.2. System.Data.SqlClient
 - 1.2.1.3. Microsoft.Extensions.Configuration
 - 1.2.1.4. Microsoft.Extensions.Configuration.Json
 - 1.2.1.5. Microsoft.Extensions.Configuration.FileExtensions
 - 1.2.2. Add a movie class to store
 - 1.2.3. Use matrix factorization to train the model
 - 1.2.4. Use collaborative filtering to find recommendations
- 1.3. Evaluate and test model
 - 1.3.1. Train model using data from SQL server
 - 1.3.1.1. Input test data into model
 - 1.3.1.2. Run the model
 - 1.3.1.3. Make adjustments if necessary
 - 1.3.2. Evaluate model
 - 1.3.2.1. Check the output to see if the model works
 - 1.3.2.2. Make adjustments if necessary

- 1.4. Return list of recommendations to backend.
 - 1.4.1. Pass user info into model
 - 1.4.2. Generate recommendations
 - 1.4.3. Send recommendations to frontend

Training the ML algorithm

2. Create the model for which the ML will be trained
 - 2.1. The model will be trained from different user movie reviews
 - 2.1.1. Load the test data from the SQL server
 - 2.1.2. Use matrix factorization to create recommendations
 - 2.2. Run the model
 - 2.2.1. Run the model with the test data
 - 2.2.2. Check output to see if is giving good recommendations
 - 2.2.3. Make improvements
 - 2.3. Evaluate model
 - 2.3.1. Possible create multiple different generations if necessary
 - 2.4. Improve the model
 - 2.4.1. Make improvements so that the model can be more accurate with recommendations
 - 2.5. Save the final model
 - 2.5.1. Save the model to a zip file so that it can be used in the application
 - 2.5.2. The model will not be able to be updated if the user watches and likes a movie.

Timeline

Week Of	Hayden	Emmett	Kaitlyn	Robert
9/5	System design and SQL Server			
9/12	Database			

9/19	Training AI Backend	Backend Design	GUI Design	
9/26		Backend Design		
10/3		Backend Class implementation		
10/10		Backend Class implementation		
10/17		Backend Class implementation	Start Window	
10/24		Bug Fixing	Search Window	
10/31		Bug Fixing	List Window	
11/7		Bug Fixing	Movie Window	
11/14		Bug Fixing		
11/21	Test			
11/28				
12/5	Prepare for Demo and Presentation			

Design Description

Frontend

The Frontend consists of four classes: MovieList, MovieSearch, MovieInfo and Buttons. These classes pertain to the functionality of screens the user has to interact with the application.

The Button class contains functions for each button used in our application. The functions for the buttons are exit, create, delete, more, left_arrow, right_arrow, get_list, and close.

Exit button - exit the application

Create - creates another movie list

Delete - deletes a movie from the list

More - loads more information about a given movie

Left_arrow - gets the previous in the movie list

Right_arrow - gets the next movie in the movie list

Get_list - loads the Results Screen and displays the list receive from the Backend

Close - closes the current screen

After the user starts the application, the first screen called, the Search Screen, appears. On the screen, it displays five fields for the user to input and select information about a movie. These fields are title, genre, year, director, and actor. The fields for genre and year are drop-down menus. Drop-down menus were selected for these fields to combat user error with input and to enhance the user experience. Additionally, the fields for title, director, and actor are text boxes. The user can input information manually into each text box. The Error Handling class which is discussed on the backend would check for errors. The Search Screen also includes an exit button that exits the application and a button to get a list of recommended movies called 'Get List'. The MovieSearch class contains the functionality for the Search Screen. The class diagram for this class is _____. The class has two functions: getMovieListButton() and exitApp(). The getMovieListButton() calls the get_list function from the Button class. The get_list function loads the next screen that outputs the results of the movie list. The screen is called the Results Screen. The getMovieListButton() also sends a request to the backend to read and store the user's input. The exitApp() function calls the close function from the Button class to exit the application.

Moreover, on the Results Screen, there are two potential outputs.(see Appendix A3.3 and A3.4). If the search results are unsuccessful in the attempt to get the movie list, the screen displays “No Results Found” to the user. The user then has to option to exit the application using the exit button or to reattempt searching using the “try again” button. If the search results are successful, then the movie list is displayed in a linear format. The user can go through the list using a left and right arrow. The arrows will be between each movie, so the user can look at each movie individually. (See Sketch 2 for reference). The Results Screen includes the left and right arrows to view the list, a close button to close the screen, a more button to get more information about a movie, and a delete button to delete a movie from the list. The MovieList class contains the functionality for the Results Screen. The class has seven functions (see Appendix A2.1). These functions handle all functionality for the Results Screen. The closeMovieListScreen() calls the close button from the Button class to close the screen. The getPreviousMovieInList() calls the left_arrow function from the Button class. The getNextMovieInList() calls the right_arrow function from the Button class. The loadMovie() loads the movie poster of the currently displayed movie. The loadMovieSearchScreen() loads the Search Screen when the user closes the Results Screen. The getMoreMovieInfo() calls the more button from the Button class and loads the Movie Information Screen. (see Appendix A3.2). Lastly, the deleteMovieFromList() calls the delete button from the Button class.

If the user decides to get more information about a movie by clicking the more button on the Results Screen, then a new screen called the Movie Information Screen is displayed. This screen provides detailed information about each movie. The movie information includes: title, genre, year, director, actor(s), description, and an average rating of the movie. On the screen is an enlarged view of the movie’s poster or title if no poster is available, a create list button, and a close button. (see Sketch 4 for reference). The MovieInfo class handles the functionality for the Movie Information Screen. The MovieInfo class has four functions: closeMovieInfoScreen(), createList(), displayMovieInfo(), and loadMovieListScreen(). The closeMovieInfoScreen() calls the close function from the Button class to close the Movie Information Screen. The createList() function calls the Backend function to create a new movie list based off of the information of the current movie and calls the loadMovieListScreen(). The displayMovieInfo() receives the stored information about the given movie from the backend and displays it to the user. The loadMovieListScreen() loads the Results Screen.

Backend

The Backend will consist of several classes: A userInput class, an errorHandler class, a movie class, and a movieList class. The UserInput class takes the movie search criteria as input and passes it to an instance of the ErrorHandler class. The UserInput class has 5

fields: title, actor, director, genre, and year all are of type string. The value of all 5 fields are based on user input received from the frontend.

The ErrorHandler class checks whether the fields in the UserInput class is valid, ex: year= "Apple" and checks whether the UserInput fields hold valid data and checks if the movie list generated is empty. If the UserInput fields hold invalid data the inputValid field is set to false and a message is sent to the frontend. If the movieList object is found to be empty then listEmpty is set to true and a message is sent to the frontend.

The Movie class holds the details of a specific movie in the database. The class also holds the same fields as the UserInput class plus the addition of the description member. The description member is a type string which holds a short description of the movie's plot. Movie class is used to form a list to hold the data of the movies the A.I. picks for the list.

The AI class focuses on interacting with the A.I. and interpreting the data given by it and formatting it into a way usable by the other classes and the frontend. The startAI() method starts the A.I. algorithm. The movieSearch will pass the UserInput object to the A.I. for it to search the database for movies that match the criteria. The A.I. generates a list of integers corresponding to the MovieIDs of the movies that match the search with the generateID() method. This list is then passed to the convertID() method which generates a list of Movie objects that correspond to the integer list passed in. This list will then be passed to a constructor for the MovieList class.

The MovieList class holds a List of object type Movie that is generated by the A.I. The list is sent to the ErrorHandler method listEmpty() to see if any results were found. The list is sent to the frontend for it to be displayed if it is found to not be empty. The getMovie() method takes an integer representing the position of a movie in the list and returns that Movie object. It is used by the frontend to get the full details of a specific movie. The remFrmList() takes an integer representing the position of a Movie object in the list and removes it from the list.

AI

The AI will be built in a separate project separate from just another movie. The AI will use matrix factorization with collaborative filtering. This works by taking user reviews of movies and comparing them to other user reviews of the same movie or similar movies for example if user one has seen movies A, B, and C and liked them and user 2 has seen movies B and C then movie C would be recommended to user 2.

The AI has several classes: Movie, Movie Predictions, and Training. The movie class will be responsible for getting the movie data from the SQL server and storing that information as a class object. We will accomplish this with a function that gets the ID of the movie, User

rating, and User Id and stores the values to the respected member variable. The next class is movie predictions which is responsible for the training of the AI. It will represent the results that the AI returns. The class will have two members: MovieID and MovieScore. The MovieID is the movie that is suggested represented in a numerical number that we will have to convert to the actual movie by looking it up in the SQL server. The MovieScore is how likely the user is to like the suggested movie. The next class is the TrainModel class which is responsible for training and building the AI. This class will have one member that is MLContext which is part of ML.NET which is the machine learning framework that we are using in this project. MLContext allows for multiple different AI specific tasks but for us the forecasting and model methods will be used in this project. IT also consists of several methods including: getData(), buildModel(), evaluateModel(), testModel, and saveModel(). The getData() method will be responsible for getting the test data and the training data. We will accomplish that by using the MLContext variable because it has access to IDataView which will allow for information that we need to be loaded into the AI for testing. Next is the buildModel() method which will be responsible for the building and training of the AI. We will use ITransformer and IEstimator which will allow us to achieve this, both of the interfaces produce the schema and create the tables and rows for the matrix factorization.

We will also take advantage of the MatrixFactorizationTrainer.Options we can edit the number of iterations that the algorithm goes through by default it is set to 10, we can also change the number of recommendations that it generates. Next we will need to use the fit() method which will train the model with the given data. After the model has run and gone through all the iterations it will return back the trained model which is a transformer. Next is the testModel() method which will be responsible for checking the model and verifying that it is producing good recommendations. We will do this with the Transform() method which will make the predictions for the different rows from the data we get from the SQL server. Next once we have the recommendations, we will compare them to the test data using the root of mean squared error which measures the differences between the test values that were entered and what the model predicted the lower the number it produces the more accurate the model it is. Also, we will use the r square value which is a number between 0 and 1 where 1 being how similar the items are, we are looking for any number over .7 for the movie recommendation. The final method saveModel() is responsible for saving the model once it is finished. We will do this using the MLContext variable which has access to the path method which will allow us to save the model and the data schema. The method saves the model to a .zip file which will then be loaded into just another movie program and be used to make the predictions for movies.

Database

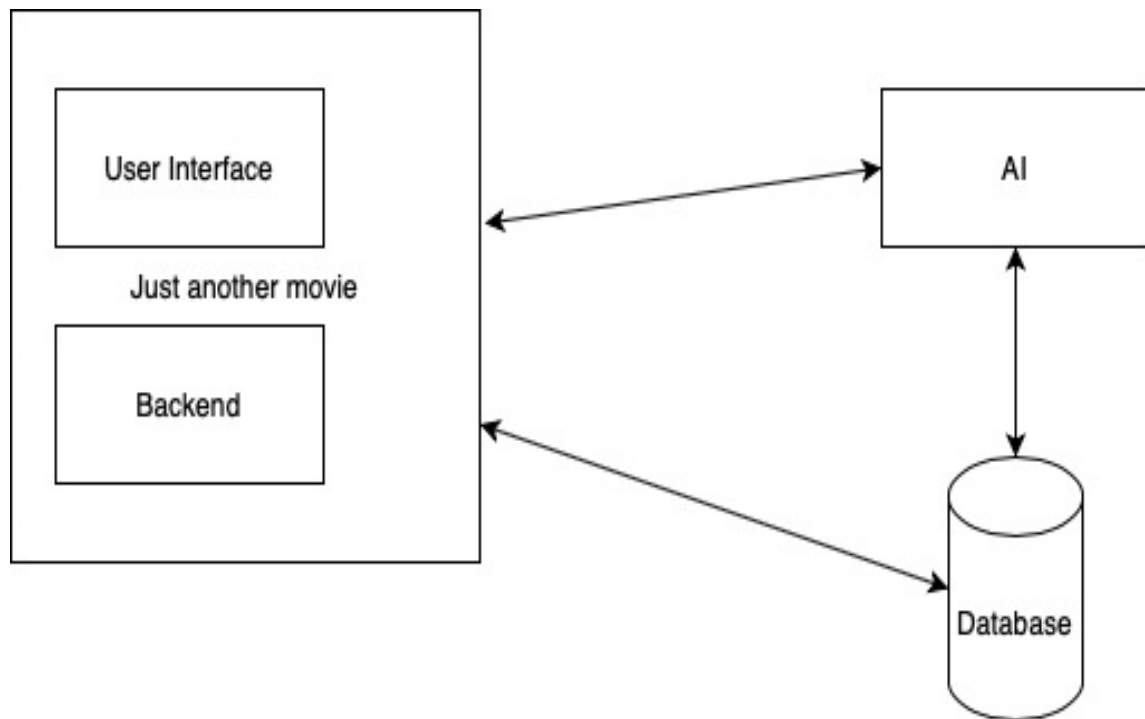
Our database is hosted on the Microsoft SQL Server Management Studio (SSMS). This server was chosen because the software is provided by the Computer Science Dept. of Austin Peay State University. The database has 2 tables, Movie and User. The tables store information from an online dataset from Kaggle.com. The Kaggle online dataset stores the information as a .csv file. This data is transformed in a table to more efficient storing and accessibility.

The chosen fields for Movie table and User table are found in the ERD Diagram. The two tables have a many-to-many relationship since movies are rated by many users, and users can rate many movies. The two tables have a primary key of their respective ids. The User table has the movie id as a foreign key to establish the relationship between the two tables.

A connection is made between the web server and database server to maintain data integrity and to reduce issues with creating a connection to the database. The backend connects with the web server to extract the information it is requesting from the database.

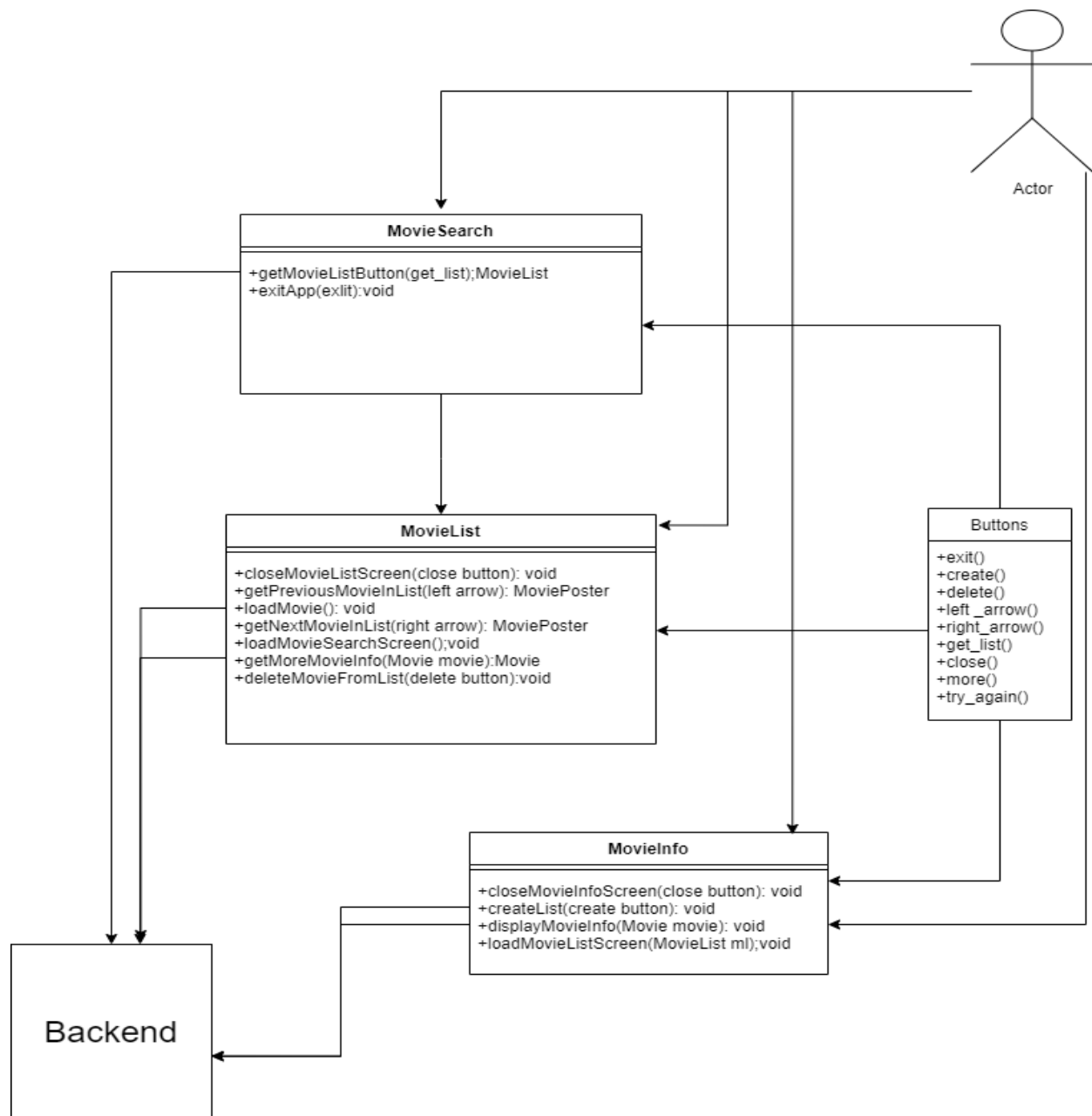
Appendix

A1. Block Diagram

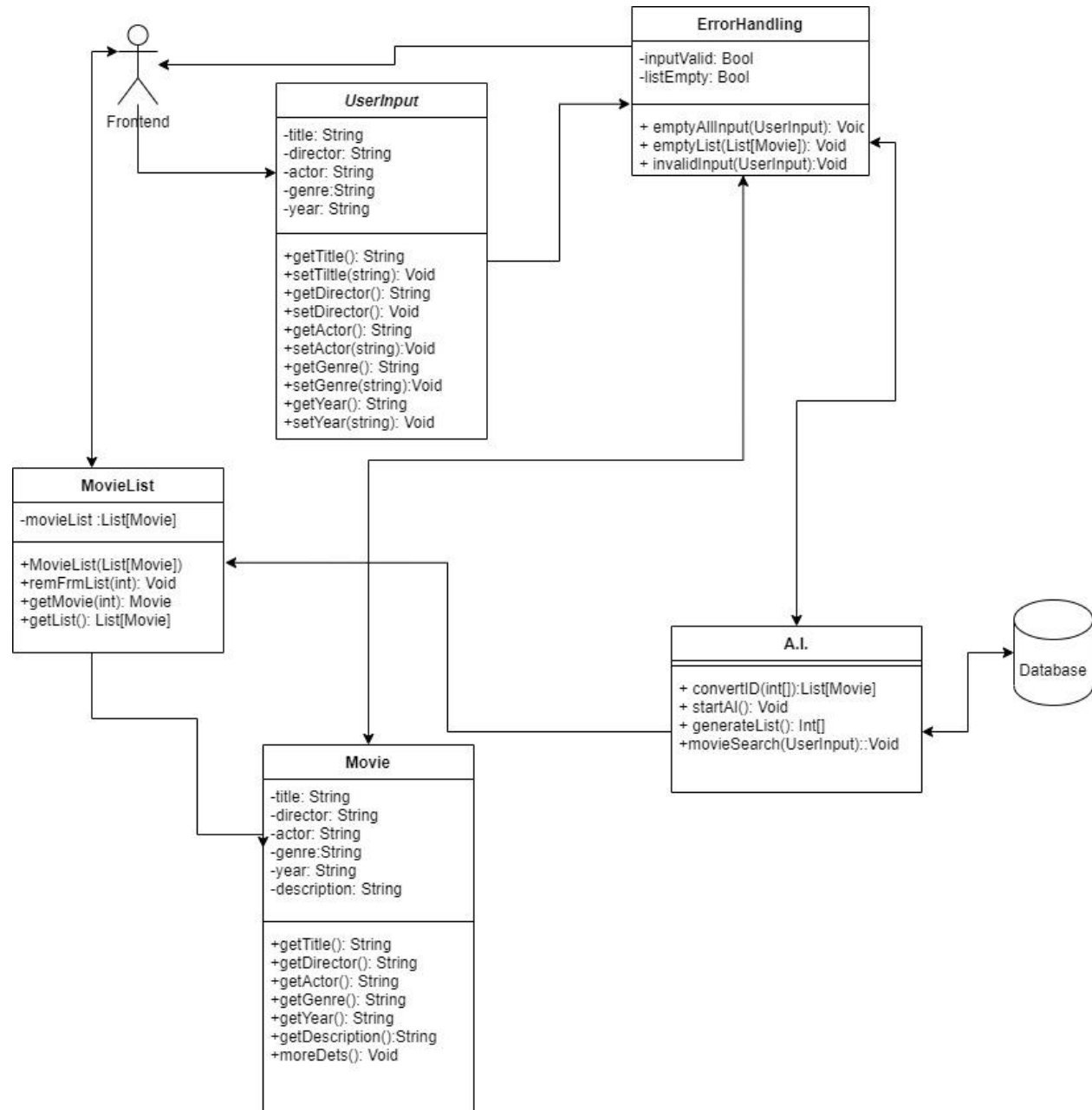


A2. Class Diagram

A2.1 Frontend Class Diagram



A2.2 Backend Class Diagram



A3. User Interface Sketches

A3.1 Sketch1



Search Screen

Exit button

Title

Director

Actor

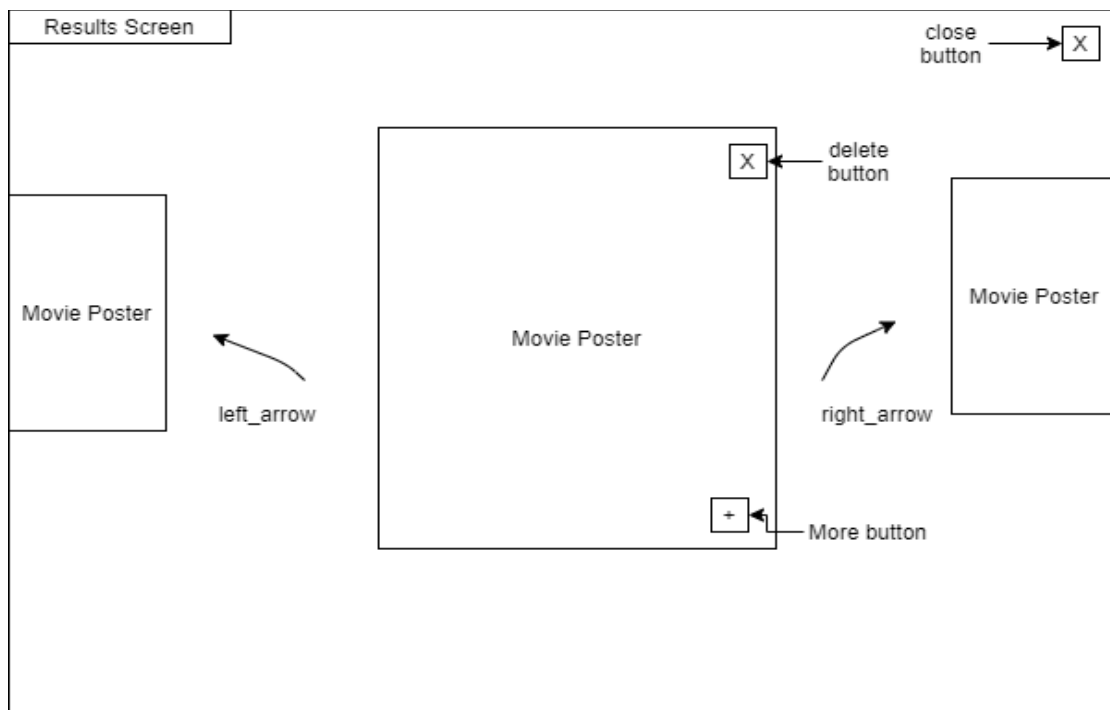
Genre

Year

GET LIST

This sketch shows a search interface. At the top left is a title bar labeled "Search Screen". In the top right corner is a small square button with an "X" inside, labeled "Exit button" with an arrow pointing to it. Below the title bar, there are five labels: "Title", "Director", "Actor", "Genre", and "Year", each followed by a rectangular input field. At the bottom center of the screen is a rounded rectangular button labeled "GET LIST".

A3.2 Sketch2



Results Screen

close button

delete button

left_arrow

right_arrow

More button

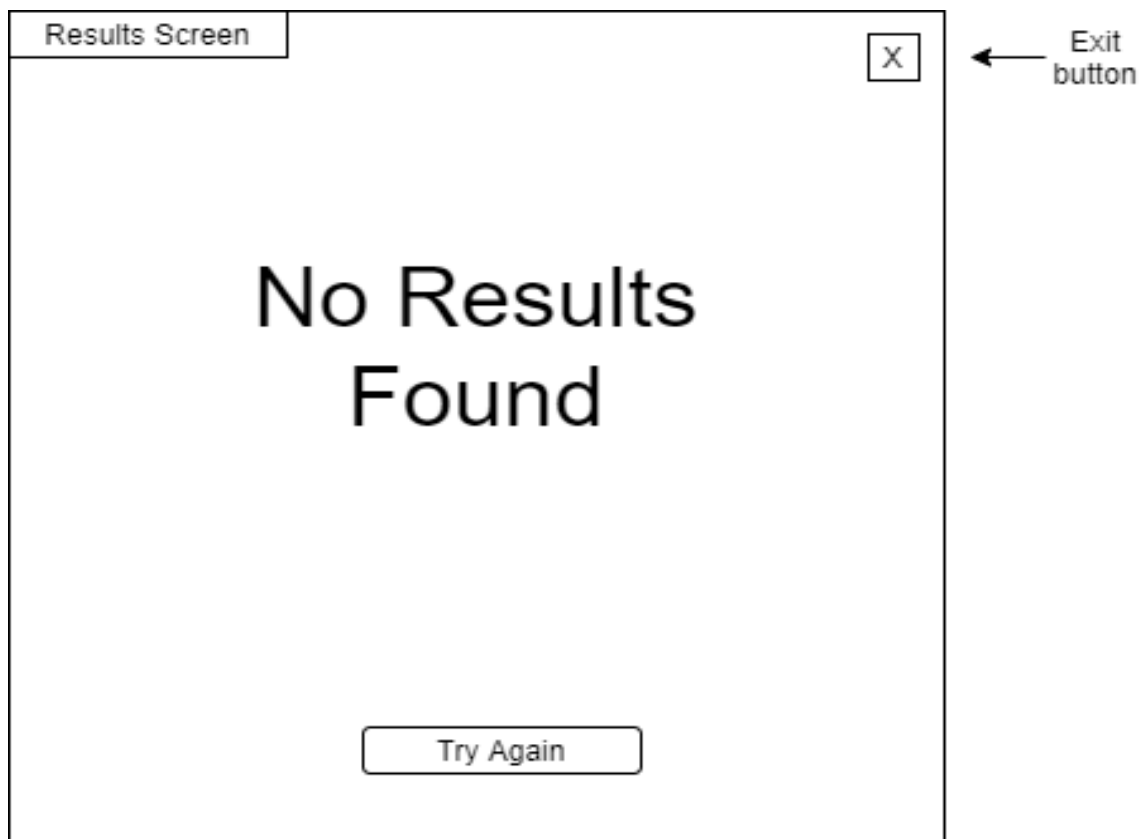
Movie Poster

Movie Poster

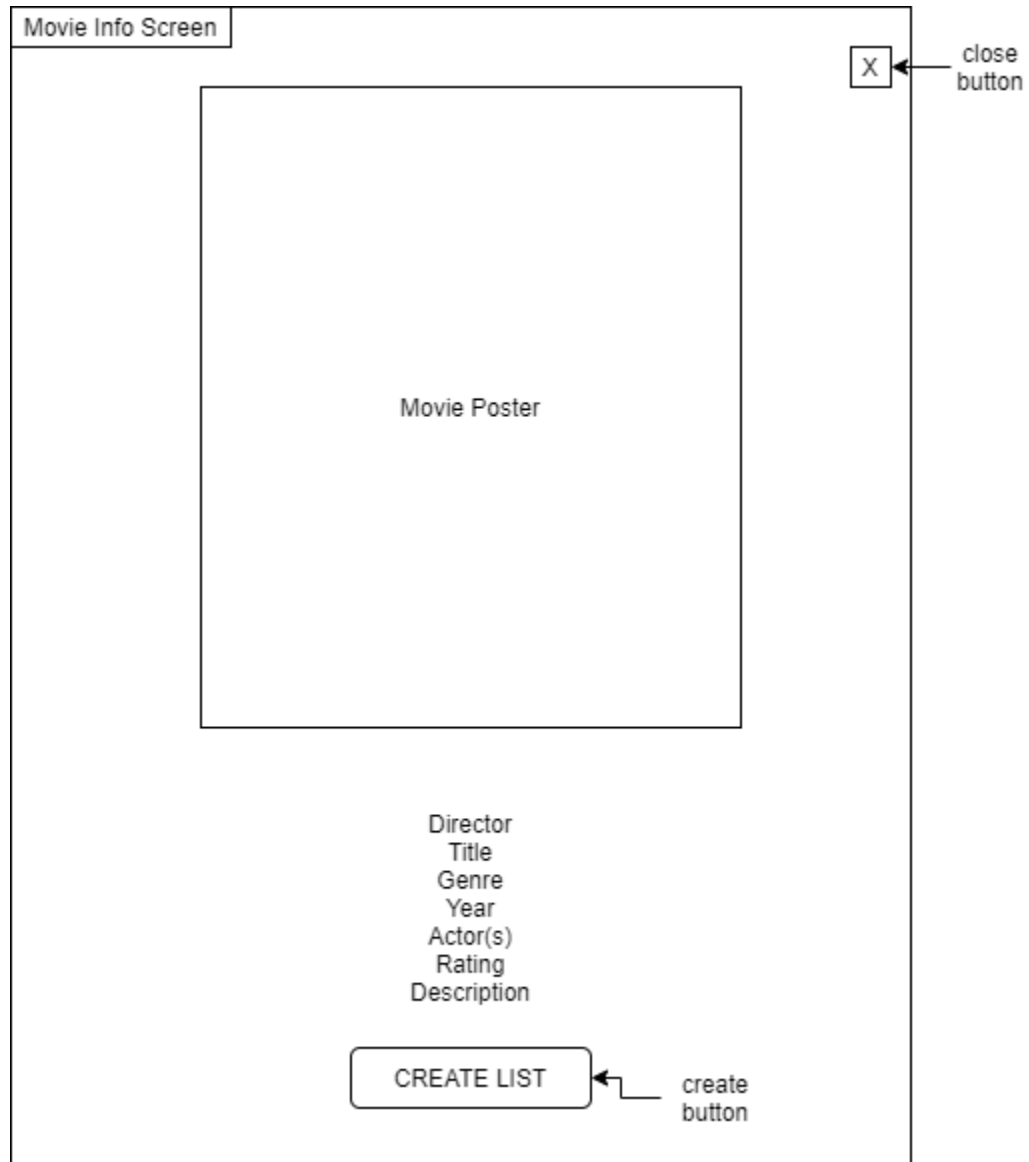
Movie Poster

This sketch shows a results interface. At the top left is a title bar labeled "Results Screen". In the top right corner is a small square button with an "X" inside, labeled "close button" with an arrow pointing to it. The main area contains three "Movie Poster" placeholders. The central placeholder is larger than the two flanking it. Each placeholder has a small square button with an "X" in the top right corner; the one on the central poster is labeled "delete button" with an arrow. Below the central poster is a small square button with a "+" sign, labeled "More button" with an arrow. On the left side, between the two side posters, is a curved arrow pointing left, labeled "left_arrow". On the right side, between the two side posters, is a curved arrow pointing right, labeled "right_arrow".

A3.3 Sketch3



A3.4 Sketch 4



A4. Message Documentation

A5. Storage Documentation

A5.1 ERD Diagram

