Just another movie:

Project Report

# Introduction

Just Another Movie is a movie recommendation desktop application. Our purpose for creating this application is to improve the movie watching experience by eliminating the strife of finding a movie. In our application, users will be able to generate a list of recommended movies from our vast collections of movies. The problem that Just Another Movie solves is trying to make choosing what movie to watch an easier process, while also showing the user movies that they haven’t seen before.

Just Another Movie works by first showing the user with a startup screen that will have directions on how to use the application and how to use each screen. The user will then go to the search screen which will have a textbox for movie title year and director that the user will be able to enter information into in order to filter out some movies. There is also a dropdown box for the genre which the user has the option to choose from 32 different genres to choose from.

Once the user enters the information, they will be directed to the survey screen which will display 15 different movies for the user to choose from and select the movies they like or think they would like by clicking on the checkbox. This information is then saved to the csv file that the AI uses to base the recommendations it generates. After the survey is filled out and submitted, the user will have to wait for approximately a minute for the movie list to be generated because the AI must be trained, and all of the movie information is being pulled from the SQL server.

Once the survey information is processed, the user will be presented with the results screen that displays 15 of the movies that they would enjoy. There are two buttons on the screen that will load more movies or load the previous movies. The user can also click on the plus icon on the movies that will display more information about the movie like the director, actors, year of release, and the description of the movie. Once the user has found the movie, they want to watch they can exit out of application and go and enjoy the movie they selected.

# Technology

a.What technology was used and why was it chosen?

The technologies that were used in the development of this application was C#, and the programming language, SQL, being used to query our database and create our database. SQL was useful to filter out some of the results of the movie list to be able to deliver better results to the user. C# was chosen because all of the group members had at least some experience with the programming language, and C# allow for the use of WPF forms or windows presentation foundation. WPF forms make creating the UI of the application much easier due to the libraries that come with WPF.

Visual Studio was the IDE we chose for the development environment because the whole group has used it before, and it allows us to develop the user interface using WPF. ML.Net was used for the machine learning library in order to develop the AI. We used matrix factorization with collaborative filtering to be able to make predictions.

For the dataset we used a database we found on Kaggle that had 26 million different reviews in the large file. We had to use Python in order to extract the data out of the dataset. This was a complicated process because the owner of the dataset stored the data in a strange way and made it very difficult to extract the data and then store it in the SQL Server. Python was chosen because it includes specific libraries to process data easily. Most of the online tutorials on how to manipulate, clean, and store data appropriately were in Python, therefore we believe it would be best to include this programming language.

I. How easy or difficult was it to come up to speed on the technology?

Everyone had to use a technology that they were unfamiliar with. Therefore, our team struggled to catch up to speed with certain technology. For Robert, he was unfamiliar with implementing the user interface such as buttons, creating new pages, and developing classes. For Kaitlyn, she was unfamiliar with the data set we used. She had problems extracting the information from the data set and saving it into the SQL Server. For Emmett, he was unfamiliar with SQL, and he had to write SQL queries that the back end would use in order to pull the movie information from SQL Serve. In order to display it to the user, these queries had to be exact and clear which was difficult for an inexperienced person. For Hayden, he was unfamiliar with using ML.Net, the machine learning library. ML.Net is new and constantly being updated. Therefore, there is not a lot of online documentation to support the library. As a result, Hayden figuring out how to use the library and create a model to be able to make predictions took some time.

II. Did the technology you choose help solve the problem?

All of the technology we chose to use was helpful to us except for the Kaggle dataset. The dataset at first glance looked like it would provide us with sufficient data. One of the biggest struggles was first finding a dataset that contained all of the information that we needed: movies, genres for the movies, actors in movies, directors in the movie, and ratings for all the movies. Then we had to ensure that enough of that information was stored in the dataset to make sufficient predictions using the AI. Once we found our chosen dataset on Kaggle based on these two requirements, we accepted it. Another benefit of this dataset was that it contained additional useful information like languages, descriptions, and as well as the posters for the movies. However, figuring out how the data was stored ended up being a much bigger pain than what it was worth. The process of extracting data from the dataset took up too much time just trying to store the data in the SQL Server database. This took away the time that we could have used in development, and we could have implemented more features or fixed bugs in our application.

III. If you originally proposed using different technology but switched to a different technology, explain why you switched and if you believed it, was the correct decision.

Our system design changed throughout the development process. At first, we decided to include a server for the backend of us to communicate with to request information and receive information from the database. To create the server, we were going to use XAMPP and PHP as the server language. However, we could not use the design or technology in our application because we could not figure out how to host the server in the appropriate amount of time. For the purpose of timing, we chose the correct design and technology to implement our application. However, this design is flawed and has a major security risk. The design is not a reflection of a real-world deployable application.

# Design

## Frontend

The classes for the frontend component of our application include LoadScreen, InfoMovie, MovieInfo, NoResutsScreen, SearchScreen, ResultScreen, Survey, and SurveyMovies.

### LoadScreen

The LoadScreen Class implements the functionality for the Startup Screen and corresponds to the tools added to the Startup Screen window. The class contains five methods that display the tutorial information and implement the functionality of buttons.

#### Methods:

LoadScreen

This method is the constructor to load the startup screen. The constructor is public so other classes can create a LoadScreen object to display the window.

LoadTextBoxes

This method displays the information for each textbox on the screen. The text boxes include information about how each screen operates, and the information is assigned to the appropriate text box created. The text for the text boxes is manually assigned and are not dynamic.

Button\_Click/Button\_Click\_2

This method’s only function is to close the Startup Screen. The method corresponds to the “Close” button on the Startup Screen.

Button\_Click\_1

This method closes the Startup Screen and displays the Search Window. The method corresponds to the “Next” button on the Startup Screen.

## SearchScreen

The SearchScreen Class implements the functionality of the Search Screen. This class has five properties and nine methods.

The first four properties correspond to the text boxes on the Search Screen that take user input. The properties are private and are named SearchDirector, SearchGenre, SearchTitle, and SearchYear. These names are respective to the text boxes on the screen.

The last property is named GenreColletion which is a list of strings that contains all the genres available to choose from in our application.

#### Methods:

Genre\_SelectionChanged

This method executes when the user chooses a genre from the drop-down menu on the screen. The method stores the user input into the property, SearchGenre.

Get\_List\_Copy\_Cllick

This method executes when the user chooses the “Help” button. The method displays the Startup Screen with the tutorial on it.

Get\_List\_Copy

This method executes when the user chooses the “Search” button. The method displays the Survey Screen so the user can choose from a list of movies to choose from based on the genre selected.

getUserInput

This method assigns the values of the class properties to the UserInput properties. The method creates a UserInput object. Then it checks to see if the user input some text in the text boxes. If the properties are not empty, then the user input is stored using the userInput object created. Once all the properties have been checked, the method returns the userInput object.

## SearchScreen

This method is the constructor for the SearchScreen Class. The constructor initializes the component and the GenreCollection property. Then also sets the DataContext to the instance of this class.

TextBox\_TextChanged\_1

The function is executed when the text box for the title changes. The user input for the search box is assigned to the SearchTitle property.

TextBox\_TextChanged\_2

The function is executed when the text box for the year changes. The user input for the search box is assigned to the SearchYear property.

TextBox\_TextChanged\_4

The function is executed when the text box for the director changes. The user input for the search box is assigned to the SearchDirector property.

Button\_Click

This method executes when the user chooses the “Exit” button and it closes the Search Screen.

## Survey

The Survey Class implements the functionality of the Survey Screen. This class has five properties and ten methods.

The properties of the class are query which is a Query class object, search which is a Search object, genre which is a string to , and surveyResults is a list of Survey Movies objects.

The class also contains the Survey Movie class to store the name, ranking, and user id for each movie in the survey.

#### Methods:

The addSurveyResults method adds Survey Movie objects to the surveyResult list.

Button\_Click

This method loads the LoadScreen for Help Information.

Button\_Click\_1

This method loads the results of the movie list.

Button\_Click\_2

This method loads the Search Screen.

getSurveyResults

This returns a list of SurveyMovies objects that hold a movieTitle and ranking for that movie.

HandleCheck

This method is executed when the checkbox is checked and the boolean value is true. This function checks which checkbox is checked, adds the correct movie title and ranking to the SurveyMovies object, and then adds the SurveyMovies object to the surveyResults list.

HandleThirdState

This method is executed when the checkbox is unchecked and the boolean value is false. This function checks which checkbox is checked, adds the correct movie title and ranking to the SurveyMovies object, and then adds the SurveyMovies object to the surveyResults list.

HandleUnchecked

This method is executed when the checkbox is considered null. This function checks which checkbox is checked, adds the correct movie title and ranking to the SurveyMovies object, and then adds the SurveyMovies object to the surveyResults list.

setMovieTextBoxes

This function sets the text for each TextBox to the appropriate movie title. It expects a string list of movie titles

Survey

The constructor initializes the component, the genre property, the movies property, and calls setMovieTextBoxes.

## ResultScreen

The ResultScreen Class implements the functionality of the Result Screen. This class has five properties and eleven methods.

The properties of the class are mi which is a Movie list class object, movieCount which is an integer that represents the constant value of movies, movieIndex which is an integer that represents the current index of the movie in the movie list, and movieTitle is a list of Movie objects.

#### Methods:

ResultScreen

The constructor initializes the component, assigns the movie list to the movieTitles property and calls LoadMovieTitles.

Button\_Click

This method closes the screen.

Button\_Click\_1

This method gets more information about the chosen movie and loads the Movie Info Screen.

Button\_Click\_2

The get loads the Startup Screen.

LoadMovieTitles

This method loads the new set of fifteen movies to the screen.

LoadNewMovieList

This method loads the new set of fifteen movies from the movie list to the screen.

LoadNewMoviesButton\_Click

This method is executed when the user clicks the “Load More Movies” button and calls the LoadNewMovieList method.

LoadPreviousMovieList

This method loads the previous set of fifteen movies from the movie list to the screen.

LoadPreviousMoviesButton\_Click

This method is executed when the user clicks the “Load Previous Movies” button and calls the LoadPreviousMovieList method.

LoadSearchScreenButton\_Click

This method executes when the user clicks the “Search” button and displays the Search screen.

## NoResultsScreen

The NoResultsScreen Class implements the functionality of the NoResults Screen when no movie list is returned from the AI.

#### Methods:

NoResultsScreen

This method is the constructor and initializes the component.

TrySearchAgainButton\_Click

This method is executed when the user clicks the “Try Again” and displays the Search Screen.

ExitAppButton\_Click

This method is executed when user clicks the “Exit” and exits the application.

## UI Design

Startup Screen

The classes that were used for User Interface were startup screen, search screen and results screen. These screens were created using Microsoft Visual in C#. The user is welcomed to a colorful and vibrant startup screen. It includes the app's logo and color scheme. It includes help tutorials for each particular screen. It provides information on how to use each screen.It displays each screen and explains in detail the features of how to use this application. It can easily be located at the bottom of the startup screen. It is very noticeable because of its design.It includes survey, search, results and movie information. The survey toolbar informs the user to select the checkbox for each movie you like. Search help button gives a tutorial and explains that the user must click the x button, explains that it displays the movie attributes and to click x button to exit. The help tutorial for the results screen. Last but not least; movie information tutorial gives a tutorial on how to use this screen. The user should choose one genre from the menu and select the get list button. It can be missed because it is surrounded with a yellow box. It was well designed for any user to navigate this application. I can truly say that there is not any other application like this one.

Search Screen

The textboxes are created to store information for these classes; Author, Director, Genre and Year. The Genre class was designed using a combo box. This particular class fits this category because it gives the user better capabilities to select a category from a list of 32 items in a Genre list. As a result, the user is able to use a mouse on the cursor and scroll to their particular category. The user can search by combining movie attributes. They will put movie attributes information in input boxes for each attribute. There were also functions called next, previous, help, exit and getlist. Next button was created to continue to the next screen. The previous button allows the user to return to the last page that they have made progress on without losing the data that has been recorded. The help button, which was designed by Kaitlyn, our design specialist; informs the user on how to use the application. The help information is located throughout the entire application for each access. After the user inputs his or her criteria they are able to proceed and click getlist. Getlist was created to store the criteria and send to the backend for further processing.The user should then click next to proceed to the result screen. After the stored has been sent to the backend for processing it is then passed back to the result screen.

Result Screen

The result screen displays movie attribute information about the particular movie and recommendations. The results will be displayed at which it includes the title of each movie. The user can select the title which will display movie information about that particular movie. The user can also select and load previous movies. This will load a selection of movies that the user has made previously. The load more movies button will load additional movies. The search button populates a list of movies and the help button loads the help screen. The system will output a movie list based on the criteria entered. After the user selects his or her choices from the survey window. It will display count the movie recommendations based upon previous users. The user will be able to see the year the movie was made, director information, title and genre. It is stored in a class in the AI that is recorded each time a user inputs their criteria.

## Back end classes

The classes that make up the backend are the Backend, UserInput, Movie, MovieList, and Query. The Backend interacts with all the other classes in the backend except the UserInput class and is located in a separate file from them. only consists of one function GetMovieList. GetMovieList takes in a UserInput class, creates a Query class instance with the UserInput, runs the AI.runAi method and passes the resulting moviesForList list into the Query’s class’s SearchQuery method and returns the resulting MovieList object.

The UserInput has 11 methods including the constructor, getters, and setters and 5 variables. The setters and getters are for the string Title, Director, Actor, Genre, and int Year variables, and are made with built in C# getter and setter creator. The constructor accepts no input and is just a default constructor.

The Movie class has 20 methods including 9 getters and 9 setters, 2 constructors, and 9 variables and the getters and setters are both made with built in C# getter and setter creator. The methods are the constructor for the Movie class instances used in the survey class which accepts a string title, float rating, and int movieId and each are assigned to the variable of the same name. The 2nd constructor accepts a moviesForList object and then searches the database for a title, director, actors, genre, year, and posterlink associated with the movieId in the database. The string variables Title, Director, Genre, PosterLink, string list Actors, and int Year should be self explanatory by their names.

The MovieList class holds a list of Movie objects and what is ultimately returned to the frontend. The MovieList has a list of Movie objects. The MovieList has two constructors, a default constructor and another one that accepts a list of moviesForList that is passed into the Movie constructor and added to the movieList via the addMovie method.

The addMovie method takes a single moviesForList object and then uses it to create a new Movie object and then pushes the new Movie object to the end of movieList.

The getMovieList method returns the movieList object, this is used to get the Movie objects and help get the data to output in the frontend.The movieList object is a list of Movie objects that fit the criteria of the UserInput.  
The Query class interacts with the database and makes queries to retrieve data and search for movies that fit the UserInput criteria and retrieves movies for the survey class. Query has two constructors, a default and one that accepts a UserInput object and assigns it to the input object. The class has two variables input holding the user input and hasResults which keep track of if any of the queries fail to find results. The Query class also has the SearchQuery, FormMovieIdQuery, getMovieId, getMoreMovieInfoQuery, TitleQuery, SurveyQuery, DirectorQuery, ActorQuery, GenreQuery, YearQuery. Query has an UserInput object called input which holds the search criteria inputted by the user.

The SearchQuery method accepts a list of moviesForList objects that the AI came up with for recommendations for the user. The SearchQuery filters these movies by checking if the movieIds of the list of moviesForClass fit the search criteria of the input object.Depending on if a variable in the input object is left blank or empty the method will call the corresponding query method for the variable that have input by the user in them whether it by title, director, actor, genre, or year. The method then returns the newly filtered list of moviesForList back.

FormMovieIdQuery method formats the query string so it only looks for movies corresponding to those in the list of moviesForList generated by the AI. It accepts a list of moviesForList and a string representing the query. The method returns the newly formatted query.

The getMovieId method gets the movie id that is based on a title. The method accepts a string that is holding the title of the movie and returns an int representing the movie Id of that movie.

The getMovieInfoQuery method returns a InfoMovie object based on the information gathered from queries. The method has a string holding the title of the movie as the only parameter passed in. The method has four queries, one for the director, one for the year, another for the actors, and finally one for the description. The method then assigns the results to the corresponding variable in a InfoMovie object before returning the object.

The TitleQuery checks all movies titles corresponding to the movie ids in the moviesForList list and adds back those which match the user input criteria. The method returns a list of moviesForList objects containing those that matched the criteria.

The SurveyQuery method retrieves movie titles belonging to a certain genre and stores them in a list before returning said list. The method accepts a string containing the genre the movies need to be in. Then it returns a list of movie titles found in that genre

DirectorQuery The DirectorQuery checks the directors of all movies corresponding to the movie ids in the moviesForList list and adds back those which match the user input criteria. The method returns a list of moviesForList objects containing those that matched the criteria.

The ActorQuery checks all actors in the movies corresponding to the movie ids in the moviesForList list and adds back those which match the user input criteria. The method returns a list of moviesForList objects containing those that matched the criteria.

The GenreQuery checks all the genres of the movies corresponding to the movie ids in the moviesForList list and adds back those which match the user input criteria. The method returns a list of moviesForList objects containing those that matched the criteria.

The YearQuery checks the year of release for all movies corresponding to the movie ids in the moviesForList list and adds back those which match the user input criteria. The method returns a list of moviesForList objects containing those that matched the criteria.

## AI classes

The classes that were used were the AI class, movie ratings, rating predictions, moviesForList. The AI class is the class that is responsible for creating and training the AI. It consists of 5 functions that are runAi, getData, buildAndTrainModel, createRecomendationList, and orderList.

The runAi function calls all of the necessary functions to be able to train and create the recommendation list. First it creates an MLContext object which gives access to the required functions to be called from the ML.net library. It then calls the getData function and saves the return form the getData function to the training and test data. The runAI function then calls the buildAndTrainModel and passes the MLContext variable and the training data into the function which will return the trainedModel. The next function that is called by runAI is createRecomendationList which is passed the MLContext variable and the trained model inorder to make the predictions and save them to a list that contains the movieId’s and the prediction score. The list is then passed to the orderList Function which orders the list into descending order.

The getData function works by taking the MLContext variable as a parameter and returns two different IDataView Objects of the testing and training data. The function works by getting the current working directory of the and loading the ratings.csv file or the ratings\_small.csv file the smaller file was used for testing and demonstration purposes because it trained the AI faster, do to the smaller amount of data. Once the csv file is located it is converted to a IDataView object because that is what the AI wants the data to be formatted. The data is then split into testing and training data using the trainTestSplit and 20% of the data is stored in the testing data with the rest in the train data. The function then returns both of the test and train data.

The buildAndTrainModel is responsible for building and training the model and takes the MLContext and the training data as parameters and returns a ITransFormer of the trained model. The first thing the function does is convert the data in the train file and map the data to key values so that, that way the duplicate items won’t be used. The next thing is that an IEstimator is created which is set to matrix factorization and adds the different options for the training of the model. The different options include the column and row index which is mapped to the userId and movieId, along with the number of iterations that the AI goes through the data, and approximate rank which is the accuracy of the recommendation. The function then adds the options onto the untrained model, and then the last thing the function then fits the model with the matrix factorization options to the training data to train the model. It then returns the trained model so that the predictions can be made.

The createRecomendation function takes the MLContext and trained model as parameters. A prediction engine is created and set equal to createPredictionEngine which takes the two classes movieRatings and ratingPrediction and the trains model as input. It then creates a list that will hold the predicted recommended movies with the predicted score the user would give the movie. A for loop is used to loop through the movie ID’s, a movieRatings object is created that holds the userId that is set to 672 which is the new user and the movieId is set to i so that all of the movies will be looped through. A predictionEngine is created with the movieRatings object passed in as an parameter. An if statement is used to see if the predicted score from the prediction engines above a 3.5 which would be 3 and half stars. If the predicted score is above 3.5 an object is created that holds the movieId and the predicted score and added to the list that will be returned to the back end to be able to display them to the user. The last function is the orderList function which takes the list that is returned from the createRecomendationList function and sorts the list so that the movies with the highest recommendation are at the start of the list.

The next class the AI uses is the movieRatings class which acts as the input for the AI to use. It has 3 members being userId, movieId, and label with the load column function being used from the ML.net library that specifies which column to load from the csv file and store from the column specified.

The next class that is used by the AI is the ratingPrediction class that acts as the output class for the AI which has two members the label and score which is responsible for holding the movieId and the predicted score.

The last class that is used by the AI is the moviesForList class which has two members movieId and movieScore which hold the movie id and score of the predicted liked movie. The class has a constructor which takes movieId and movieScore as parameters and sets them equal to the members. The class also has getters for movieId and movieScore. The UML diagram for the AI is in the appendix section and is labeled A.1 for reference.

## Database tables

The database tables were created from an online Kaggle dataset that stored movie information in csv files. All the files from the dataset were not pertinent for the application, so every file was not used. Also, not all of the information in the files that we chose were included in our database tables.

The credits.csv file contains cast and crew information for all movies in the dataset and are stored as Python objects. This file was used to extract information about the actors and directors of a movie. The ratings.csv and ratings\_small.csv contained ratings from multiple users on all the movies in the dataset. The information contained in these files included a movie id , user id , rating, and timestamp. The timestamp was not used in our application. These files were used to train and test the AI. The final csv file that we used from the Kaggle dataset is the movies\_metadata.csv file. This file contained all other movie related information for our application such as: genre, title, release date, poster link, movie id, description.

There are seven database tables that we created from the Kaggle dataset. These tables were implemented to store significant movie information for the application and to increase the efficiency of the backend component of our application to query the database for appropriate information.

Movies

The Movies table holds the movie id, title, description, poster link, and release year for each movie.

Actor

This table contains the actor id and actor’s name for every distinct actor in a movie from the dataset.

Director

This table contains the director id and director’s name for every distinct director in a

movie from the dataset.

Genre

This table contains the genre id and genre name for every genre listed in the dataset.

MovieGenres

This table was created to create a relationship between the Movies and Genres tables. Its purpose is to link the genre to the appropriate movie and store redundant information without compromising data integrity. It stores a unique id called movieGenreId to be the primary key in the table, then movie id and genre id are added to create the relationship. Therefore, multiple genres are listed for one movie and multiple movies can have the same genre without the data being convoluted.

MovieActors

This table was created to create a relationship between the Movies and Actors tables. Its purpose is to link the genre to the appropriate movie and store redundant information without compromising data integrity. It stores a unique id called movieActorId to be the primary key in the table, then movie id and actor id are added to create the relationship. Therefore, multiple actors are listed for one movie and multiple movies can have the same actor without the data being convoluted.

MovieDirectors

This table was created to create a relationship between the Movies and Directors tables. Its purpose is to link the genre to the appropriate movie and store redundant information without compromising data integrity. It stores a unique id called movieDirectorId to be the primary key in the table, then movie id and director id are added to create the relationship. Therefore, multiple directors are listed for one movie and multiple movies can have the same director without the data being convoluted.

An example of the MovieGenres tables is shown in the Appendix.

How to deploy or build the application

To deploy our application, you would have to download the executable file and create a shortcut to run it as a program on your desktop.

# Known bugs

In the Results screen, the window cannot display the movie posters, the Load More Movies and Load Previous Movies buttons do not work, and it cannot load all movie titles. We do not know what component is causing the issue. It can be an issue with the backend, frontend, or the database.

In the MovieInfo screen, the screen cannot display all movie information. We do not know what component is causing the issue. It can be an issue with the backend, frontend, or the database.

Some Information cannot be retrieved by the frontend though what cannot be retrieved varies by movie.

The AI pulls the data from the csv file that is located in the debug folder instead of the data folder that was created to hold the data.

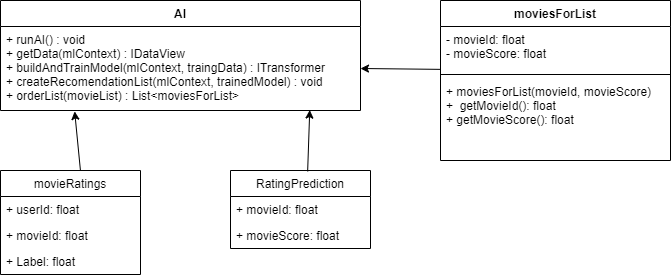
In the database, it includes empty values for important movie information like the title and year. It also includes unknown extraneous values for foreign movies and special symbols.

# Future work

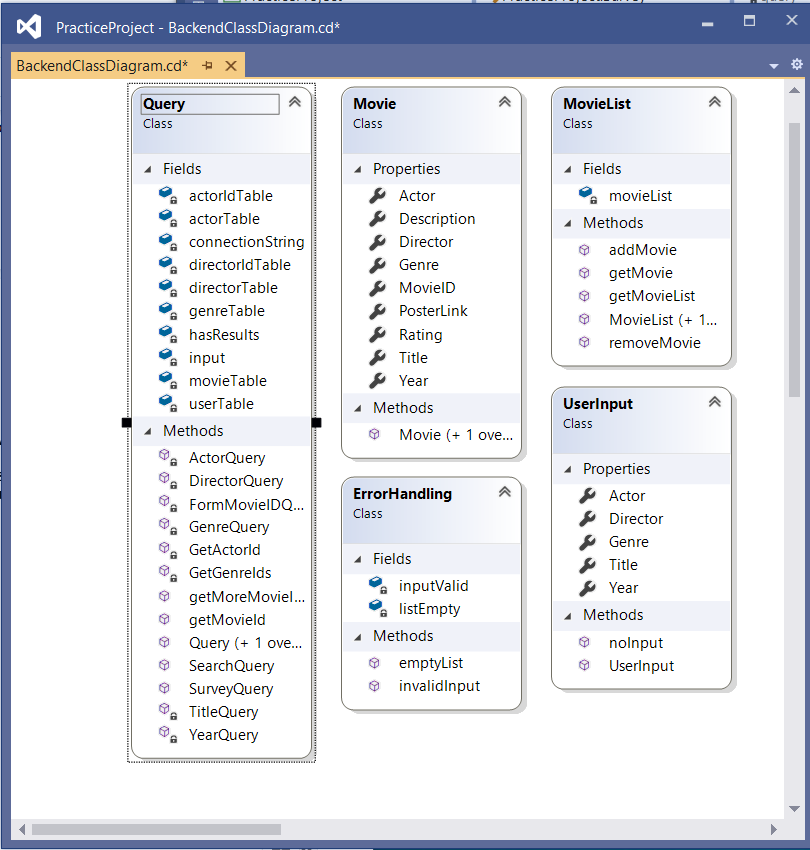
* We would add in the ability of the user to like a movie from the results page, and have that review added to the csv file for training to help optimize the recommendation system.
* Make the UI look better and more user friendly.
* Fix display issues
* Successfully implement Load More Movies/ Load Previous Movies
* Implement more functionality to improve UI experience
* Allow for use of removal of a movie from the list by the frontend.
* Build the model from the ground up.
* Keep track of liked movies by the users and make the predictions based off of that instead of having the survey.
* Get rid of the survey.
* Remove records from the database with empty values
* Manage special characters and foreign symbols in the database

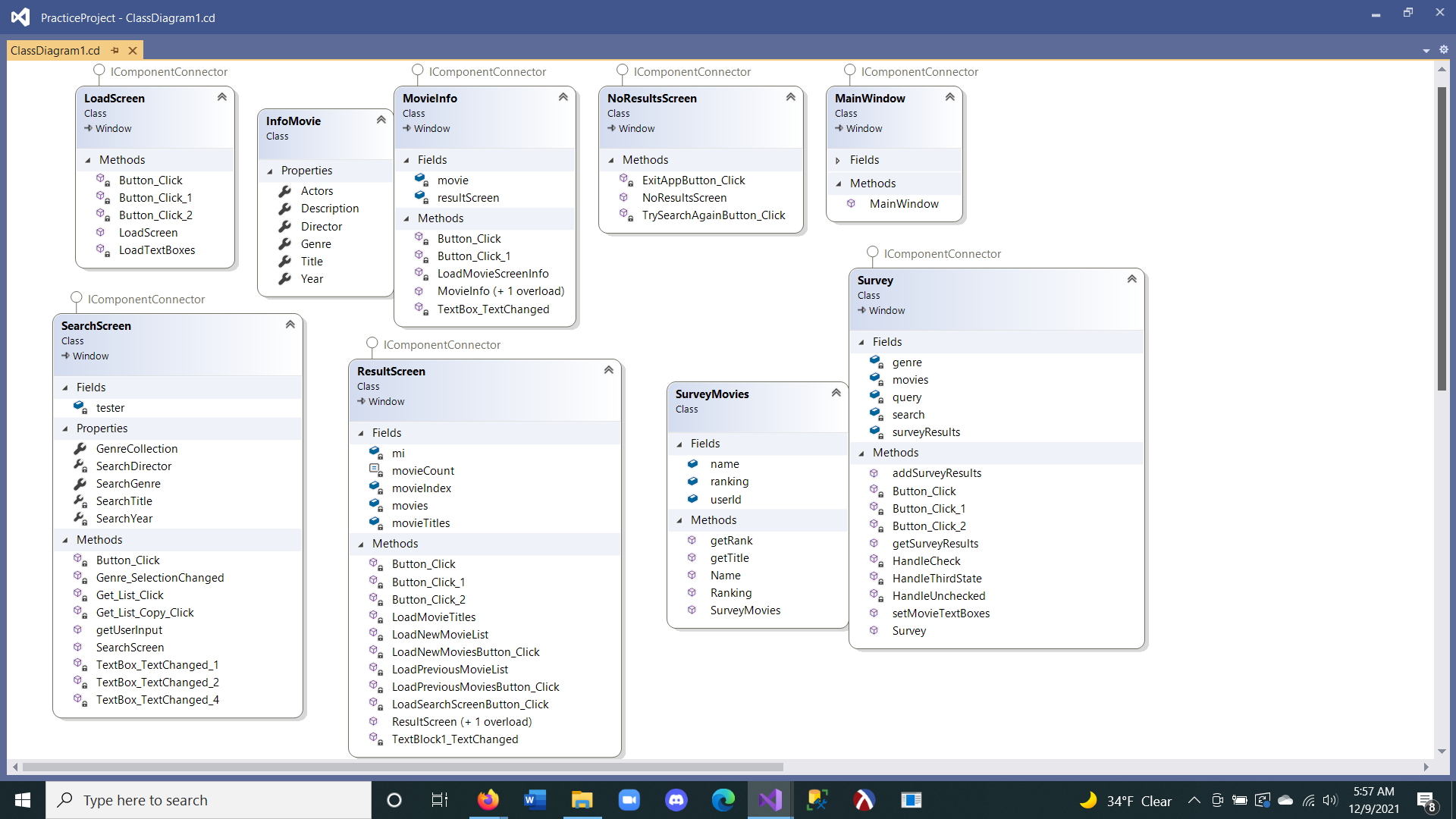
Appendix

AI Class Diagram



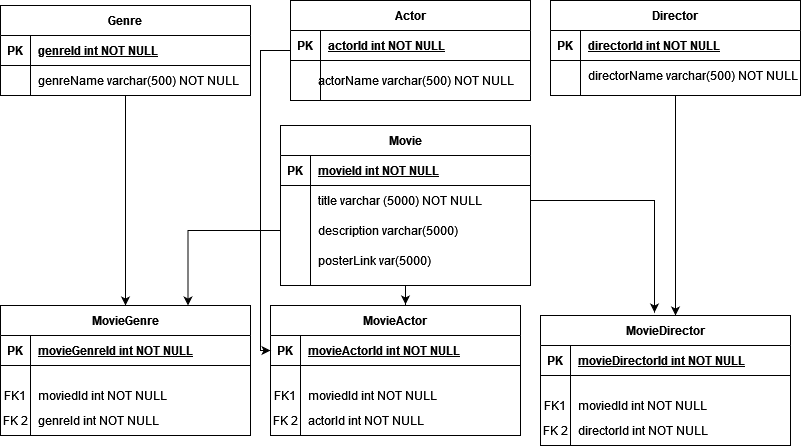
Backend Class Diagram



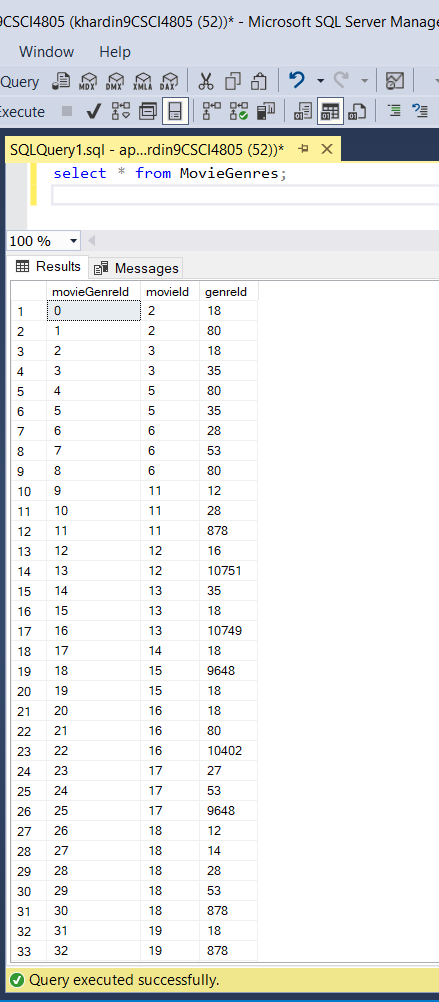
Frontend Class Diagram

\*The lock by a field or method means private.

ERD Diagram

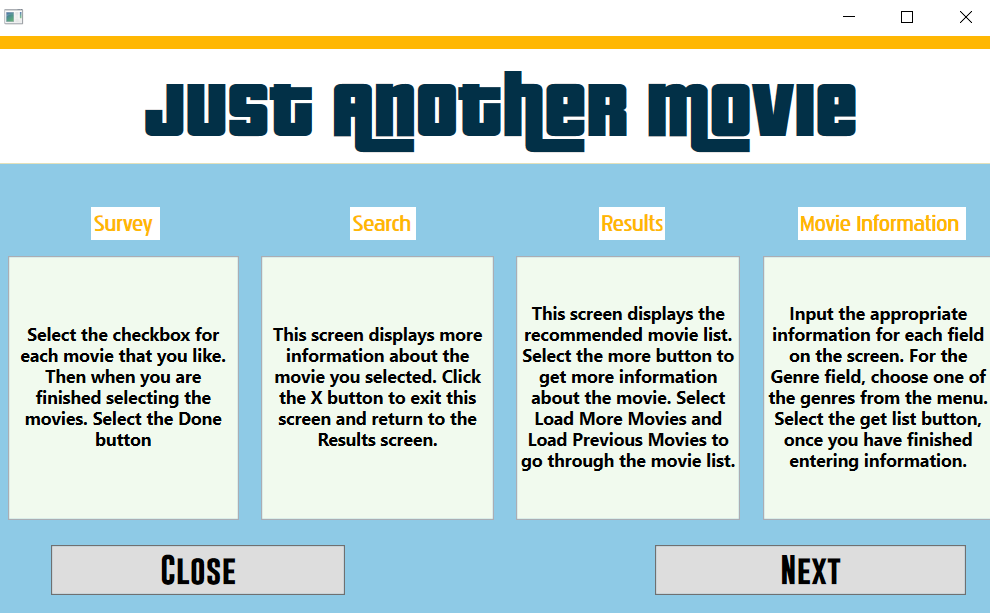


Sample of MovieGenres

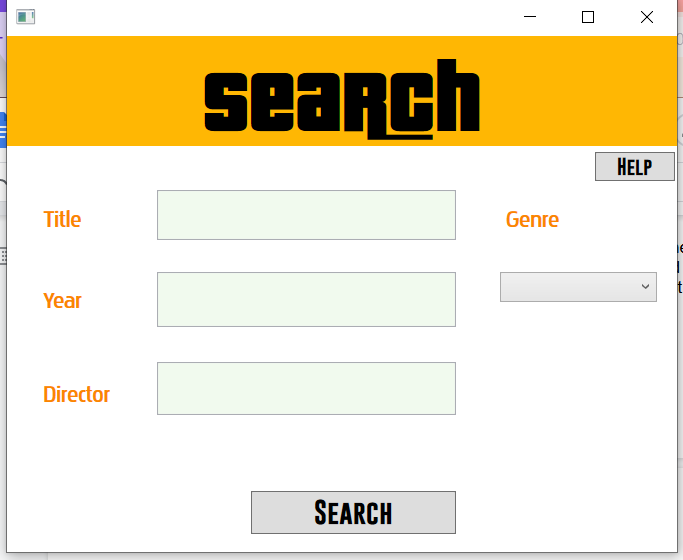


Screenshots of Application

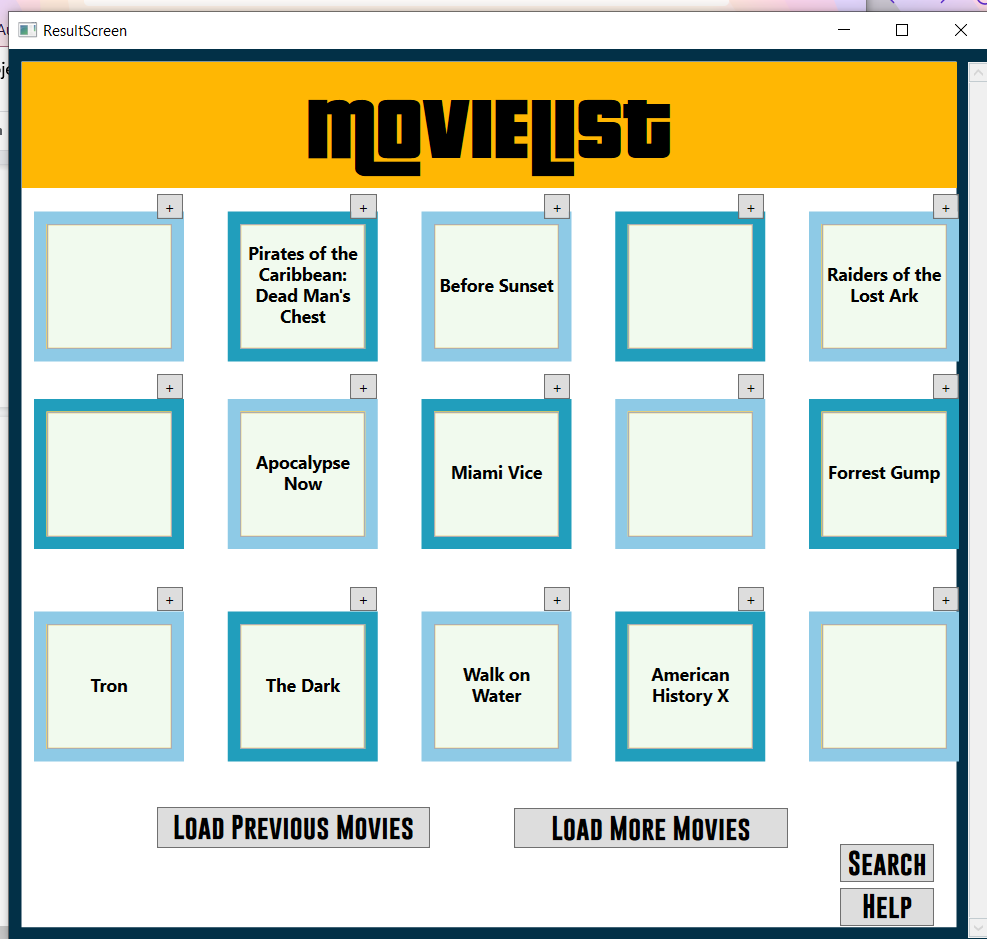
LoadScreen



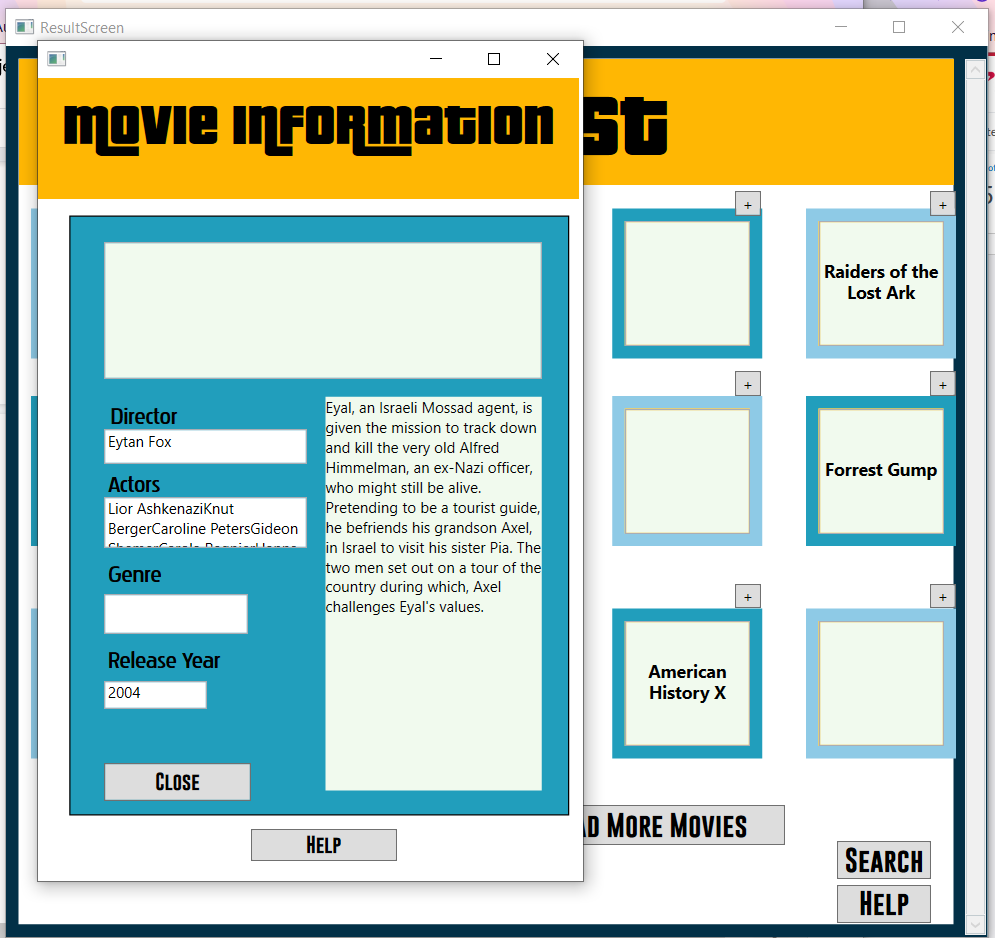
SearchScreen



ResultScreen



MovieInfoScreen



Survey

