

NEA PROJECT

Anime Management System (AMS)

Persia FARZANEHNIA
[Email address]

Contents

Analysis	4
Background to and Identification of the Problem	4
Description Of Current System	5
Identification of prospective users and Acceptable Limitations.....	6
Data Volumes	6
Data Flow Diagram	6
Numbered objectives of the project.....	7
Extension.....	15
Proposed Method of Solution	15
Design	16
Overview of the system	16
E-R Diagram	18
Data Dictionary	18
Table-Users.....	23
Table-Animes.....	24
Table-Studios.....	24
Table-Genres	25
Table-AnimeStudios.....	26
Table-AnimeGenres	28
.....	28
Table-AnimeStatus.....	29
Table-UserFavourites	29
Table – UserWatchList	29
Description of Data Structures	30
Description of Algorithms	30
Form: FmMenu (Used for creating tables, and inserting data from csv files)	30
Class: Encryption- Password hashing	32
Form: FmSignUp (Used to insert user details into database)	33
Form: FmLogIn (Used to let user log in to gain access to the rest of the program)	36
Form: FmAMS (Admin)	37
Form: EditAnimes (Admin Only)	46
Form: EditStudios (Admin Only)	47

Form: EditGenres (Admin Only).....	48
Form: EditAnimeStudios(Admin Only)	49
Form: EditAnimeGenres(Admin Only)	50
Form: EditAnimeStatus (Admin Only)	51
Form:DeleteUser (Admin Only).....	52
Class:DarkmodeSwitch	52
Form:FmAMS (MainForm)	53
Form: FmAnimeInformation	57
Form: FmUserFavourites	61
Form: FmUserWatchList	64
Form: FmUserStats.....	68
User interface design.....	69
System Security and Integrity of Data.....	87
Technical Solution.....	88
Name of all file/Entities in project.....	88
Complex Procedures	89
Annotated Code.....	91
Program.cs	91
Encryption.cs	92
DarkModeSwitch.cs	92
Form:FmMenu.cs.....	95
Form: FmSignUp.cs	101
Form: FmLogIn.cs.....	109
Form: FmAMS.cs	114
Form: FmAnimeInformation.cs	125
Form: FmProfile.cs.....	137
Form: FmReset.cs	141
Form: FmUserFavourites.cs.....	146
Form: FmUserWatchList.cs.....	155
Form: FmUserStats.cs	167
Form: FmAMS(Admin).cs.....	171
Form: FmEditAnimes.cs	192
Form: FmEditStudios.cs.....	197
Form: FmEditGenres.cs	201
Form: FmEditAnimeStudios.cs.....	204
Form: FmEditAnimeGenres.cs	208

Form: FmEditAnimeStatus.cs.....	212
Form: FmDeleteUser.cs	216
Testing	218
Test Plan.....	218
Test runs.....	228
Test #1.....	228
Test #2.....	229
Test #3.....	234
Test #4.....	236
Test #5.....	242
Test #6.....	244
Test #7.....	245
Test #8.....	249
Test #9.....	256
Test #10.....	264
Test #11.....	271
Test #12.....	273
Test #13.....	275
Test #14.....	276
Test #15.....	278
Test #16.....	280
Test #17.....	281
Test #18.....	283
Test #19.....	285
Test #20.....	286
Test #21.....	287
Test #22.....	289
Test #23.....	290
Test #24.....	292
Test #25.....	293
Test #26.....	294
Test #27.....	295
Test #28.....	297
Test #29.....	298
Test #30.....	299
Test #31.....	300
Test #32.....	301

Test #33.....	302
Test #34.....	305
Test #35.....	306
Test #36.....	307
Test #37.....	310
Test #38.....	311
Test #39.....	312
Test #40.....	313
Test #41.....	314
Test #42.....	315
Test #43.....	316
Test #44.....	317
Test #45.....	319
Test #46.....	320
NEA VIDEO	321
Evaluation.....	321
Evaluation against objectives	321
Feedback.....	335
Analysis of User Feedback	335
Overall Assessment of the project.....	336

Analysis

Background to and Identification of the Problem

My computer science project will be an anime management system. The reason why I did this is because I have an interest in anime and I cannot keep track of everything I watch and by making this it would be easier to keep track of.

As well as this, I am not always up to date with the latest news of what I am watching, and I wanted to see how much hours I spent watching anime. I also do not keep track of the number of episodes I am watching in a series, so if I wanted to pick the series back up from taking a break from watching it, I struggle to remember what episode I am on for that series.

Throughout my research I have found out that this is a common problem on many sites that stream anime, they do not offer you enough information about a potential sequel, they do not keep track of how much episode you have watched, and the sites are unable to keep track of the hours spent watching episodes. Furthermore, I have also found out that most sites that tend to stream anime do not mention which studio created it, this is useful information to know as you could watch more anime from that specific studio. A reason why it would be nice to know about the studio that is behind the production of specific animes would be

mostly for visual reasons. Different studios would have varied budgets to use to create these shows so for example a well-known studio would most likely have a better production than a studio that is not as well known. The studios that tend to have the most budget and better production value would most likely produce anime that would be enjoyed by many. Good, quality animation is very important in making an anime enjoyable so by being informed in reputable studios fans are more likely to watch anime produced by that studio.

Another thing that I have noticed throughout my research is that some sites don't categorize animes into genres. Being able to have categories would also be easier to find an anime you would want to watch as it links up with your interests. Also being able to compare anime through the category would be useful and making recommendations within the system would make finding out animes a lot easier. I also did some research and considered information on the length of episodes, but I deemed that as unnecessary as most episodes are the same length and the outliers would either be specials such as OVAs and movies.

Sites also don't specify the status of animes, whether they're completed or not.

These problems are not only restricted to anime sites this could occur in sites that stream tv shows, films etc. This system that I intend to create could also be adapted to fit those types of shows. As you could compare and categorize tv shows and have information on the companies that made the tv shows.

Description Of Current System

The current system is a large anime and manga database Which is not normalised and contains repeating attributes. It contains information for countless animes and mangas. The information that is contained about the anime is the synopsis, the number of episodes, statistics and the people who have worked on that series. The current system is well built and has clearly been developed over a long period of time.

However, a flaw for this site would be that it does not prompt the user to sign up and if they are not signed up the features for this site are limited. In addition to this your list of animes are automatically public instead of prompting the user for an option to publicise their list and arguably the layout for the site is bland. It also does not tell you about your watch hours or the number of episodes you have watched in total.

The screenshot shows the homepage of MyAnimeList.net. At the top, there is a navigation bar with links for Anime, Manga, Community, Industry, Watch, Read, and Help. A search bar is located at the top right. Below the navigation bar, there is a banner for 'Summer 2021 Anime' featuring several anime covers. To the right of the banner is a 'Top Airing Anime' list with five entries. Below the banner is a 'Manga Store' section with several manga covers. The 'Top Airing Anime' list is as follows:

Rank	Anime	Type	Episodes	Score	Members	Action
1	Fumetsu no Anata e	TV	20 eps	8.61	424,849 members	add
2	One Piece	TV	0 eps	8.56	1,536,134 members	add
3	Shiguang Dailiren	ONA	11 eps	8.50	34,289 members	add
4	Holo no Graffiti	ONA	0 eps	8.50	31,978 members	add
5	Kingdom 3rd Season	TV	26 eps	8.38	47,468 members	add

Identification of prospective users and Acceptable Limitations

The prospective users for this project would be available to the public however this system is intended for people who have an interest in anime.

A constraint to this would be the duration of this project as stated before the current system has been developed over the years meaning that the databases have been updated frequently in addition to this it is a fully functioning site based on html which is used to create websites whereas I am using visual studio C#.

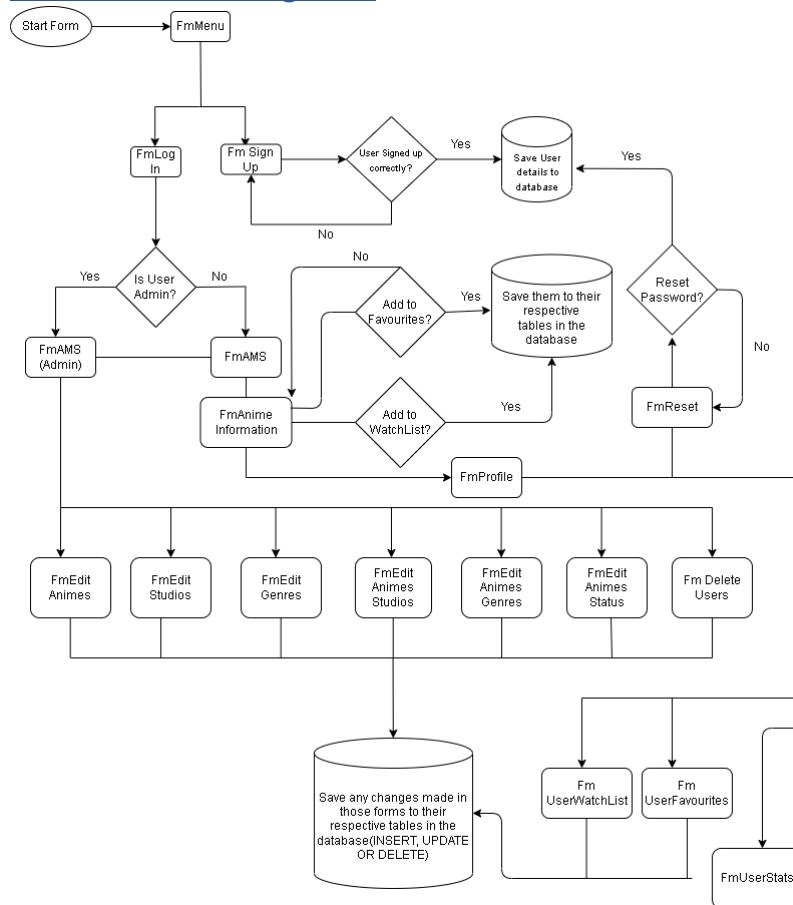
Data Volumes

This system may have to hold vast amount of data because of the amount of anime that is out there, and it will hold the information of the current users in the system and that could be added on even further with new users joining.

There will be information stored about the users including name, email, username and password. The personal details of the users will be added to the system, and they will have the option to update their details through the system.

In addition, the information stored for the animes should include the studio, the genre, the number of episodes, seasons and status (completed/not completed). The information of these animes would need to be updated as these animes could have more seasons over

Data Flow Diagram



Numbered objectives of the project

1. My program should check if the database already exists – if it doesn't exist is should create all the tables within it

1.1 There must be an 'Animes' table to store all Animes down to their titles, IDs for every Anime, Episode amount of all Animes, Pictures allocated to all Anime, Synopsis allocated to all Animes. A csv file will be used to mass insert all the values required for this table.

1.2 There must be a 'Studios' table to store all Studios and their respective IDs alongside it. . A csv file will be used to mass insert all the values required for this table

1.3 There must be a 'Genres' table to store all Genres and their respective IDs alongside it. . A csv file will be used to mass insert all the values required for this table.

1.4 There must be an 'AnimeStudios' table to split the many to many relationship between the table Animes and Studios. This table must contain all AnimeIDs from the tables 'Animes' and StudioIDs from the table 'Studio' to form a composite key. A csv file will be used to mass insert all the values required for this table

1.5 There must be an 'AnimeGenres' table to split the many to many relationship between the table Animes and Genres. This table must contain all AnimeIDs from the tables 'Animes' and GenreIDs from the table 'Genres' to form a composite key. A csv file will be used to mass insert all the values required for this table.

1.6 There must be an 'AnimeStatus' table to reduce repeating data in the main table 'Animes'. This table must contain all AnimeIDs from the table Animes and Status that was originally from the table 'Animes' to form a composite key. A csv file will be used to mass insert all the values required for this table.

1.7 There must be a 'Users' table to store all Users information to that table. The table must contain the usernames the users decide to give themselves, UserIDs as a primary key, Forename, Surname and Emails (Emails will be used to reset password via entering the email to find the user's account in this program) and the User type of the user. All passwords must be hashed for security reasons

1.8 There must be a 'UserFavourites' table that contains all UserIDs as a foreign key to link back to the table 'Users' and a unique primary key called 'UserFavID' to be assigned with the UserID, the table should also contain AnimeIDs, AnimeTitles and the user's respective rating of an Anime they have favourited

1.9 There must be a 'UserWatchList' table that contains all the UserIDs as a foreign key to link back to the table 'Users' and a unique Primary key called 'UserWatchID' to be assigned with the UserID the table should also contain AnimeIDs, AnimeTitles, the number of

episodes watched by the user and the user's respective rating of an Anime they have watched.

2. If the database already exists the application must access it

2.1 All the tables filled with values from their respective csv file must be in the database.

3. The 'FmMenu' form must allow users to navigate to the Sign-up Form, the Log-in Form and it must allow the user to exit the form.

3.1 There should be three buttons labelled 'Sign up', 'Log in' and 'Exit'.

3.2 Once a button is pressed, the corresponding form should be displayed

3.3 The menu form 'FmMenu' should close when another Form is displayed.

4. The Sign-up Form 'FmSignUp' must include a 'Return to Menu' button and a 'Save' button.

4.1 The Sign-Up form must include all the required fields the user needs to enter to create an account through textboxes and the user must select the user type through a combo box with the options 'Administrator' and 'User'

4.2 The User information that is entered by the user must be inserted into the table 'Users'

4.2.1 The passwords entered by the user must be hashed before the insert

4.2.2. User information must be inserted via a Save Button

4.2.3 A message box must be shown to let the user know that they can Log in and it should also let the user know whether they need to change something within the fields.

4.3 RegEx validation must be used for all the textbox in the Sign-up form to properly validate information

4.3.1 It should let the user know what must be include in their Sign-up form E.g. "Your username must contain numbers" etc. through error messages when trying to save information

4.3.2 An error message must show up when the required fields are empty, or anything related to RegEx validation

4.3.3 An error message must show up when the username that is entered by the user already exists in the database

4.4 If the user selects 'Administrator' in the combo box User type it must show a textbox telling the user to enter an admin key

4.4.1 An error message must be shown if the user entered the wrong key for Admin via clicking the save button

4.5 Once everything is validated and meets the requirements from the validation the user should be able to save their information.

5. The log in form 'FmLogIn' must include an 'Enter' button and a 'Return to Menu' button

5.1 The Log in form must include all the required fields for the user to enter their Username and password

5.2 The details entered by the user must be checked if the details exist in the table 'Users' through a data reader

5.2.1 The password entered by the user must be hashed to match hashed password in the table 'Users'

5.2.2 An error message must be shown if the password entered by the user does not match the password in the table 'Users'.

5.2.3. If the data reader reads that the User's UserType is 'Administrator' then the Enter button must navigate the user to the Admin Form 'FmAMS (Admin)'

5.2.4 If the data reader reads that the User's UserType is just 'User' then the Enter button must navigate the user to the Main Form 'FmAMS'

6. The Admin Form must include a 'Go to MainForm' button to navigate the user to FmAMS

6.1 The Admin Form must contain radio buttons that the admin can check to view all the tables in the database via a DataGrid Viewer

6.2 The Admin Form must contain a textbox for searching stuff in the DataGrid Viewer.

7 The DataGrid viewer must be able to show the table the user wants by clicking on the radio button

8. The Admin Form must contain an 'Edit' button which allow Admin Users to navigate to the edit forms that correspond with the table they are trying to make changes to

8.1 The Edit Form must be used in conjunction with the radio buttons to view the tables in the database. Example: if the admin user selected the radio button "List all Animes" and then pressed the edit button the Edit Form for Animes will show up with empty fields.

9. The Search bar must be able to refresh the data grid viewer and narrow down the tables in the DataGrid Viewer

9.1 The Whenever an Admin inputs a letter into the search bar the DataGrid Viewer should refresh with values that begin with that letter E.g. If the Admin clicked the radio button "List all Animes and the grid shows the table 'Animes' the admin should be able type in a single letter and the grid will refresh with everything starting with that letter.

10. The DataGrid Viewer cells should be clickable by the Admin.

10.1. If the Admin clicks on the cell, the Edit form should load up with all information gathered on the row where the cell has put them into text boxes.

11. Admin Users should be able to add Animes through the edit Form that corresponds to Animes

11.1 Adding Animes will be done through an SQL insert query

11.2 The admin user would need to get the ID of the anime through the site called MyAnimeList. For Example:

https://myanimelist.net/anime/48583/Shingeki_no_Kyojin_The_Final_Season_Part_2

The admin user would insert the ID from the URL of the anime they want to insert

11.3 The admin user will need to use that ID to make connections to different tables such as AnimeStudios, AnimeStatus, and AnimeGenres

11.4 The Anime Synopsis and the Anime Picture for that anime will have placeholder values to replace them with the actual synopsis and picture for that anime using functions in the code it is however optional to manually input the synopsis and picture link for that anime.

11.5 The admin user should be prompted with a message box saying that the anime has been added.

11.6 The admin user should be prompted with a message box saying that the anime could not be added due to an error instead of crashing

12 The admin User should be able to update Animes through the edit form that correspond to Animes

12.1 Updating Animes will be done through an SQL Update Query

12.2 Any part of the information that was loaded from the DataGrid Viewer can be updated BESIDES the ID for that anime as updating the ID will mess up the link with the other tables

12.2.1 The update query will update everything besides the ID so whatever the admin enters the ID text box will be rendered useless through the update query

13. The admin user should be able to delete Animes through the edit form that correspond to the table Animes

13.1 Deleting Animes will be done through an SQL Delete query

13.1.1 To properly delete an Anime the admin user will need delete the links to that Anime

E.g., the Admin must delete the AnimeIDs that match the anime in AnimeStudios, AnimeGenres and AnimeStatus

13.2 The admin user must be prompted by a message box confirming that they have deleted an Anime

13.3 The admin user must be prompted if the anime could not be deleted due to an error instead of crashing

14. The admin User should be able to add Genres through the edit Form that correspond to the table Genres

14.1 The Admin User Should be able to assign a new Unique ID for GenreID and input genres into the genre textbox

14.2 The Admin User Should be prompted with a message box saying that the Genre has be added

14.3 The Admin User Should be prompted with a message box saying that the Genre could not be added due to an error instead of crashing

15. The admin User Should be able to update Genres through the edit form that correspond to the table Genres

15.1 Updating Genres will be done through an SQL Update Query

15.2 The admin user should be prompted with a message box saying that the genre has been updated

15.3 The admin user should be prompted with a message box saying that the genre could not be updated due to an error

16 The admin user should be able to delete Genres through the edit form that correspond to the table Genres.

16.1 Deleting Genres will be done through an SQL delete query

16.2 The admin user Should be prompted with a message box saying that the Genre has been deleted

16.3 The admin User Should be prompted with a message box saying that the Genres could not be deleted due to an error instead of crashing

17 The admin user should be able to Add Studios through the edit form that correspond to the table 'Studios'.

17.1 Adding Studios will be done through an SQL insert query

17.2 The admin user should be prompted with a message box saying that the studio has been added

17.3 The admin user should be prompted with a message box saying that the studio could not be added because of an error instead of crashing

18. The admin user should be able to update studios through the edit form that correspond to the table 'Studios'

18.1. Updating Studios Will be done through an SQL update query

18.2. The admin user should be prompted with a message box saying that the studio has been updated

18.3 The admin user should be prompted with a message box saying that the studio could not be updated because of an error instead of crashing

19. The admin users should be able to delete studios through the edit form that correspond to the table 'Studios'

19.1 Deleting Studios Will be done through an SQL delete query

19.2. The admin user should be prompted with a message box saying that the studio has been deleted

19.3 The admin user should be prompted with a message box saying that the studio could not be deleted because of an error instead of crashing

20 The admin users should be able to add AnimeStudios through the edit form that correspond to the table 'AnimeStudios'

20.1 Adding AnimeStudios will be done through an SQL insert query

20.1.1 The Admin user must use the AnimeID that they used to add a new anime, they must also use the site MytAnimeList to see the studio behind that anime and check the database if the studio exists if the studio does not exist then then the admin user would need to add the studio and use the StudioID from it to combine with the AnimeID

20.2 The admin user should be prompted with a message box saying that the AnimeStudio has been added

20.3 The admin user should be prompted with a message box saying that the AnimeStudio has not been added due to an error instead of crashing

21. The admin users should be able to update AnimeStudios through the edit form that correspond to the table 'AnimeStudios'

21.1. Updating AnimeStudios will be done through an SQL update query

21.2 The admin user should be prompted with a message box saying that the AnimeStudio has been updated

21.3 The admin user should be prompted with a message box saying that the AnimeStudio has not been updated due to an error instead of crashing

22. The admin users should be able to delete AnimeStudios through the edit form that correspond to the table 'AnimeStudios'

22.1 Deleting AnimeStudios Will be done through an SQL delete query

22.2. The admin user should be prompted with a message box saying that the AnimeStudio has been deleted

22.3 The admin user should be prompted with a message box saying that the AnimeStudio has not been deleted due to an error instead of crashing

23. The admin users should be able to add AnimeGenres through the edit form that correspond to the table 'AnimeGenres'

23.1 Adding AnimeGenres will be done through an SQL insert query

23.1.1 The Admin user must use the AnimeID that they used to add a new anime, they must also use the site MytAnimeList to see the Genres for that anime and check the database through the search bar in the Form if the Genre exists. If the genre does not exist, then the admin user would need to add the Genre and use the GenreID from it to combine with the AnimeID

23.2 The admin user should be prompted with a message box saying that the AnimeGenre has been added

23.3 The admin user should be prompted with a message box saying that the AnimeGenres has not been added due to an error instead of crashing

24. The admin users should be able to update AnimeGenres through the edit form that correspond to the table 'AnimeGenres'

24.1 Updating AnimeGenres will be done through an SQL update query

24.2 The admin user should be prompted with a message box saying that the AnimeGenres has been updated

24.3 The admin user should be prompted with a message box saying that the Anime Genres has not been updated due to an error instead of crashing

25. The admin users should be able to delete AnimeGenres through the edit form that correspond to the table 'AnimeGenres'

- 25.1 Deleting AnimeGenres Will be done through an SQL delete query
- 25.2 The admin user should be prompted with a message box saying that the AnimeGenres has been deleted
- 25.3 The admin user should be prompted with a message box saying that the AnimeGenres has not been deleted due to an error instead of crashing
26. The admin users should be able to add AnimeStatus through the edit form that correspond to the table 'AnimeStatus'
- 26.1 Adding AnimeStatus will be done through an SQL insert query
- 26.1.1 The Admin user must use the AnimeID that they used to add a new anime, they must also use the site MytAnimeList to see the Status for that anime they can then add the status to the textbox with the AnimeID to successfully insert
- 26.2 The admin user should be prompted with a message box saying that the AnimeStatus has been added
- 26.3 The admin user should be prompted with a message box saying that the AnimeStatus has not been added due to an error instead of crashing
27. The admin users should be able to update AnimeStatus through the edit form that correspond to the table 'AnimeStatus'
- 27.1 Updating AnimeStatus will be done through an SQL update query
- 27.2 The admin user should be prompted with a message box saying that the AnimeStatus has been updated
- 27.3 The admin user should be prompted with a message box saying that the AnimeStatus has not been updated due to an error instead of crashing
28. The admin users should be able to delete AnimeStatus through the edit form that correspond to the table 'AnimeStatus'
- 28.1 Deleting AnimeStatus will be done through an SQL delete query
- 28.2 The admin user should be prompted with a message box saying that the AnimeStatus has been deleted
- 28.3 The admin user should be prompted with a message box saying that the AnimeStatus has not been deleted due to an error
29. The admin users should be able to delete through the edit form that correspond to the table 'Users'
- 29.1 Deleting User will be done through multiple SQL delete queries because the user will have data in the table 'UserFavourites' and 'UserWatchList'
- 29.2 The admin user should be prompted with a message box saying that the user has been deleted
- 29.3 The admin user should be prompted with a message box saying that the User has not been deleted due to an error instead of crashing
30. The Main Form 'FmAMS' must contain a 'search bar', a 'search button', a 'profile button', clickable Anime Pictures and a group box containing Anime News

- 30.1 If the profile button is clicked the then it will navigate the user to the profile form.
- 30.2 The profile form must contain the user's information such as UserID, Username, Forename, Surname and their UserType
- 30.3 The profile form must also contain a reset password button
- 30.4 If the Reset password button is clicked then the button should navigate the user to the Reset Password Form
- 30.5 In the Reset Password Form the user must be prompted to enter their Emails
- 30.6 If the Email is entered then the email will be checked to see if it matches the email associated to the user in the Table 'Users'
- 30.7 If the Email matches the user will see 2 textboxes appeared for them to type their new password and confirm their new password. To change passwords, they must press the 'change' button. When pressed the password will be hashed and added to the table the user can then close the form
- 31 In the Profile Form there should be a 'Favourites' button and a 'Watchlist' button
- 31.1 if the user clicks on either of those buttons it should navigate the user to the Favourites form and Watchlist Form where they can update, delete their animes in their favourites table and watchlist table.
- 31.1.1 This will be done using SQL Queries
32. In the main Form the user should be able to see Anime news that refreshes in the group box
- 32.1 Fetching AnimeNews will be done using a webscraper on the site animenewsnetwork
- 32.2 It should be able to refresh and run forever.
33. In the Main Form the user should be able to type in Animes and see a list of animes below the search bar
- 33.1 This will be done using a function that contains SQL
- 33.2 When the user clicks on the search button the Anime Picture and the title of that Anime Should be shown
- 33.3 The anime picture will be retrieved using an API and then inserted into the animes database so that it could use the database to get the pictures instead.
- 33.4 A panel will be used to show the searched Anime and a 'back' button will also be on the panel to take the user back to original state of the main form
34. In the Main Form the User should be able to click on the anime pictures and navigate them to the anime information form where all the information about the anime is on it
- 34.1 The synopsis of an anime will be retrieved via an API and then inserted into the animes database and replace the place holder value. If the User were to click on that anime again after that then it must pull the synopsis out of the database instead of using an API.

34.2 In the Anime Information form the user should be able to rate the anime, add to watchlist, add to favourites and type in the number of episodes they have watched for that specific anime

34.3 An aggregate SQL query will be used to validate the add to favourites button and add to watchlist button and make sure that it is only inserted once into the tables 'User Favourites' and 'UserWatchList'.

35. Add a statistics button in the profile form that will navigate the user to the statistics form that will show them their average ratings across their favourites list and watchlist, their total ratings, the amount of animes they're currently watching, the amount of animes yet to watch, animes dropped, total episodes watched, and hours spent watching

36. Add DarkMode to the main Form

Extension

1. Add a recommendation feature that recommends the user animes based on the genres of the animes they have favourited and watched
2. Add categories for animes based on genres and studios

Proposed Method of Solution

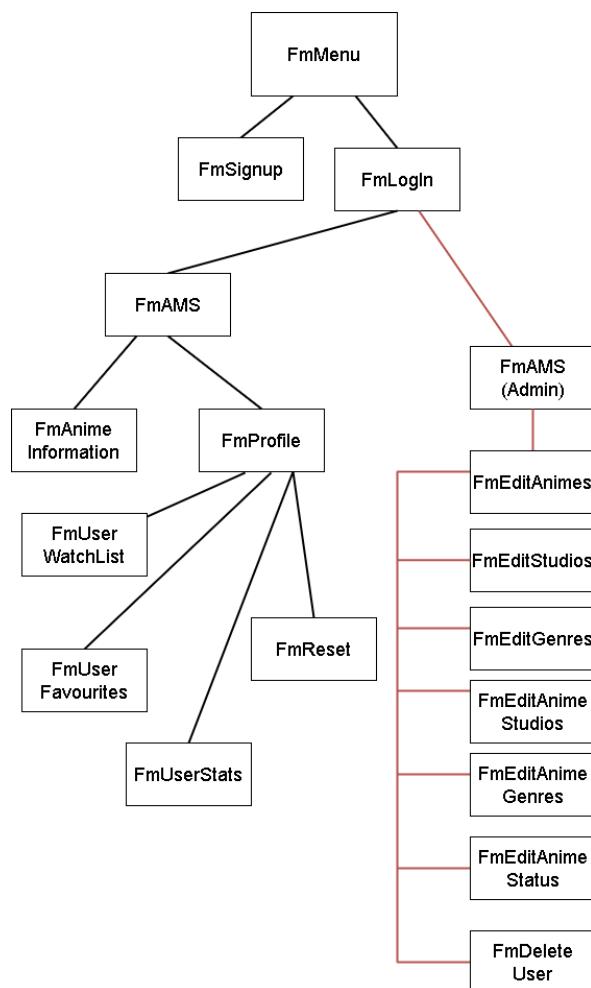
My chosen method of solution is to use visual studios to create this system to be the solution to the problem. The program will handle the user's information, the information of animes and to make it so that the user can personally manage what they watch and be informed of future seasons via the use of databases to hold the data.

Multiple pages/forms will be in place to navigate a way around the application as each form will serve a particular function. To complete this project, I have chosen the language C# as I have been using this programming language throughout college. I will also be using SQL to run queries and interact with the database.

When the program is executed for the first time the database and tables will be created. The data will be stored, written and read from tables. For the database I will be using Microsoft Access.

Design

Overview of the system



Access Key

Red = Admin

Black = All Users

Admin has access to all parts of the program

The 'FmMenu' form is the first form of the program and will be the first one displayed to the user. Upon running the program, the user should see three buttons labelled 'Sign up', 'Log in' and 'Exit'. Clicking the button 'Sign up' will bring up the Form 'FmSignUp' where the user can create their accounts and pick a user type between 'User' and 'Administrator' if the user selects 'Administrator' as their user type then they would be expected to enter the admin key which can only be found in my program. Once the user has created their accounts they can return to the 'FmMenu' form via a button in 'FmSignUp' and then click the 'Log in' button which will bring up the Form 'FmLogin' where the user can enter their username and password. If the user logs in as a 'user' then the main Form 'FmAMS' will be available to the user and all the other forms that come with the main form.

Furthermore, if the user logs in as an admin they will have access to the form 'FmAMS (Admin) and all the other forms that come with it. Admins will have access to all parts of the system.

The Form 'FmAMS' will allow users to search for an anime of their choosing. The user will have a list of animes to search from to make it easier for the user. They will be able to access the anime by clicking on their pictures.

Clicking on the anime pictures will navigate the users to the form 'FmAnimeInformation'. In this form they will be able to see the synopsis for this anime, the average rating for that anime, the number of episodes for that anime, the studios that animated the anime, the status of the anime (if it has finished airing etc) and the genres for that anime. The form will allow users the options to add the anime to their favourites alongside with their ratings for that anime and the ability to add the anime to their watchlist, they can also specify how much episodes they have watched for that show and the watch status of that anime (yet to watch, currently watching, completed and dropped).

On the main form 'FmAMS', the user will also be able to view their profile through a click of a button. Clicking on that button will show the user the Form 'FmProfile'.

In that form the user can see their user details such as their first name, second name, user type and UserID. The users can also view their watchlist, view their favourites, view their stats and the ability to reset their password. All of this will be accessed through buttons that the user can click. Clicking on those buttons will bring up the forms 'FmUserFavourites', 'FmUserWatchList', 'FmReset' and 'FmUserStats'. In the forms FmUserFavourites and FmUserWatchList the users will be able to manage their animes by updating/ deleting their lists. The users will be able to update the episodes they have watched for any anime they have in their watchlist, change the ratings of the anime in their favourites and watchlist and update the watch status for that anime. The user can also remove animes from their favourites and watchlist.

In the form 'FmReset' the user will be able to reset their passwords by entering their emails. If the email the user has entered is the same as the email, they have used to create their accounts, then the reset password form will show up letting them enter their new passwords.

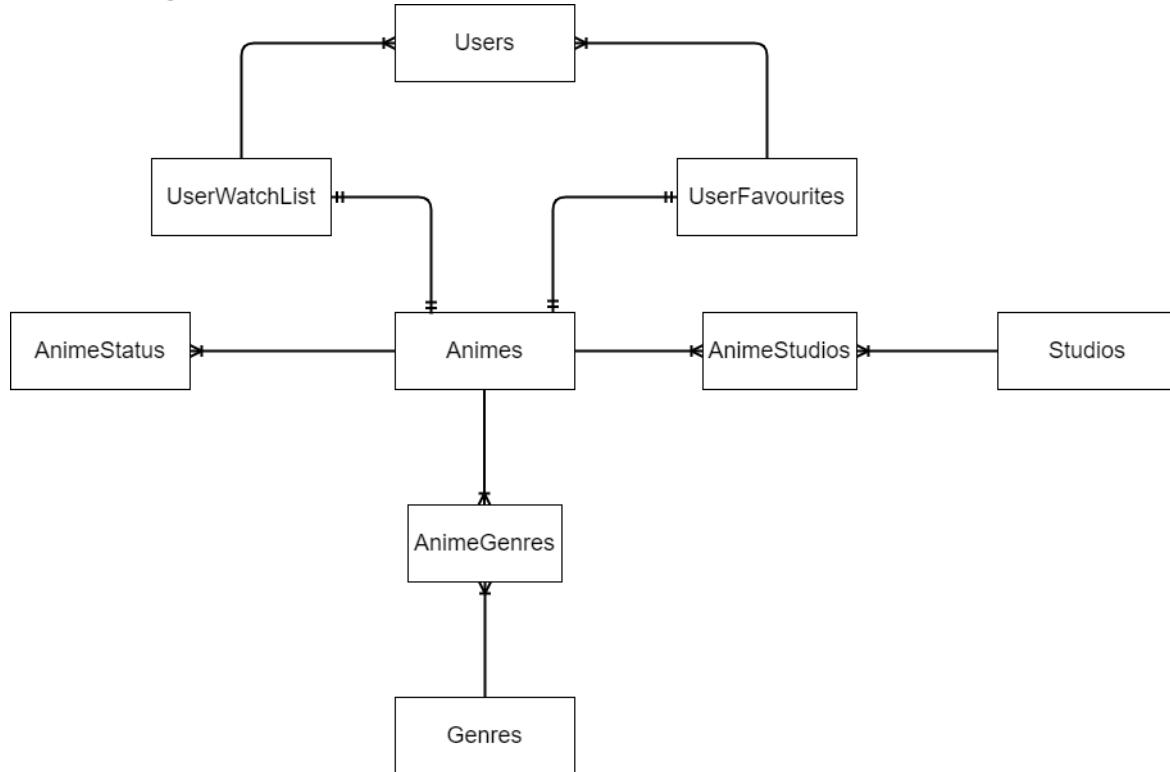
In the form 'FmUserStats' the users will be able to see their average ratings across their favourites and watchlist. The number of ratings they have made on their accounts. All the animes they are currently watching, yet to watch and dropped, the number of episodes they have watched all together and the hours spent watching on animes. The form will obviously be updated when the user has made changes to their favourites and watchlist.

For users that are admin, they will see the form 'FmAMS(Admin)' instead of the form 'FmAMS' however they can directly access that form through a click of a button and gain access to all the other forms linked to it.

In 'FmAMS (Admin)' the user will be able to view all the tables in the database in a data grid viewer with a selection of buttons corresponding to those tables. Once clicking those buttons, the user will be able to see an edit button that will bring them the forms 'FmEditAnimes', 'FmEditGenres', 'FmEditStudios', 'FmEditAnimeStudios', 'FmEditAnimeGenres', 'FmEditAnimeStatus'. In those forms the users will be able to update information, add information and delete information to the tables that correspond to those forms. The user will also have access to the form 'FmDeleteUser' where they can delete users. The admin will not be able to see the users' passwords, first name and second name

for obvious security reasons. The users will only be able to see the UserID that has been assigned to those accounts and the usernames for those accounts.

E-R Diagram



Data Dictionary

Table: Users-stored as an Access table			Primary Key: UserID		
Field Name	Data Type	Length	Validation	Example Data	Comments
UserID	AUTOINCREMENT	No length needed	Presence check Calculated by access	1	Each user will have an ID assigned to their accounts upon creation.
Username	VARCHAR	60	Presence check	Persia123	Each user must have a Username in

			Must contain at least one uppercase letter Must contain at least 1 number Uniqueness check		order to log in to the system
Pword (password)	VARCHAR	255	Presence check Must contain at least one number Must contain at least one uppercase letter and lower case Must contain at least one symbol Passwords will be hashed upon entry	Password123#	Each user must have a password in order to log in to the system
UserType	VARCHAR	30	Presence check If admin is picked the user must enter a key	'User', 'Administrator'	Each user must assign themselves a user type in order to gain access to the program
Forename	VARCHAR	60	Presence check	Persia	The user's first name
Surname	VARCHAR	60	Presence check	Farzanehnia	The user's second name
Email	VARCHAR	255	Presence check Uniqueness check	“ “@gmail.com	The user must have an email so that they can reset their passwords

Table: Studios- stored as an access table

Primary Key: StudioID

Field Name	Data Type	Length	Validation	Example Data	Comments
StudioID	INTEGER	No length needed	Presence check	20	The ID associated with studios
Studios	VARCHAR	100	Presence check	Toe Animation	The name of studios

Table: Animes- stored as an access table			Primary Key: AnimeID		
Field Name	Data Type	Length	Validation	Example Data	Comments
AnimeID	INTEGER	No length needed	Presence check	344	The ID associated with animes
AnimeTitle	VARCHAR	200	Presence check	Dragon Ball	The names of Animes
Episodes	INTEGER	No length needed	Presence check	123	The number of episodes per Anime
Ratings	INTEGER	No length needed	Presence check Must be in the range of 0 to 10 When entered by Admin user	6	The average ratings made by users on the site MyAnimeList per anime
AnimeSynopsis	LONGTEXT	4,294,967,295	Presence check (The admin user will only need to enter placeholder values and the synopsis will be implemented via another function)	"INFORMATION ABOUT THE ANIME HERE"	The summary per anime
AnimePictures	TEXT	255	Presence check (The admin user will only need to enter a placeholder value and the picture link will be implemented by a function in the program)		The picture link per anime

Table: Genres- stored as an access table			Primary Key: GenreID		
Field Name	Data Type	Length	Validation	Example Data	Comments
GenreID	INTEGER	No length needed	Presence Check	4	The ID associated with genres
Genre	TEXT	255	Presence Check	Comedy	The name of genres

Table: AnimeStudios- stored as an access table			Primary Key (CONSTRAINT): PK_AnimeStudios (AnimelD, Studioid) Foreign Keys: AnimelD, Studioid		
Field Name	Data Type	Length	Validation	Example Data	Comments
AnimelD	INTEGER	No length needed	Presence Check	2	The ID associated with Animes from the table 'Animes'
Studioid	INTEGER	No length needed	Presence Check	4	The ID associated with studios from the table 'Studios'

Table: AnimeGenres- stored as an access table			Primary Key (CONSTRAINT): PK_AnimeGenres (AnimelD, GenreID) Foreign Keys: AnimelD, GenreID		
Field Name	Data Type	Length	Validation	Example Data	Comments
AnimelD	INTEGER	No length needed	Presence Check	3	The ID associated with Animes from the table 'Animes'
GenreID	INTEGER	No length needed	Presence Check	45	The ID associated with Genres from the table 'Genres'

Table: AnimeStatus- stored as an access table			Primary Key (CONSTRAINT): PK_AnimeStatus (AnimelD, Status) Foreign Key: AnimelD		
Field Name	Data Type	Length	Validation	Example Data	Comments
AnimelD	INTEGER	No length needed	Presence Check	322	The ID associated with Animes from the table 'Animes'
Status	VARCHAR	150	Presence Check	'Finished Airing'	The status of animes

Table: UserFavourites- stored as an access table			Primary Key (CONSTRAINT) : Pk_UserCompKey (UserFavID, UserID) Foreign Key: UserID		
Field Name	Data Type	Length	Validation	Example Data	Comments
UserFavID	AUTOINCREMENT	No length needed	Presence Check Calculated by access	2	The ID for this table
UserID	INTEGER	No length needed	Presence Check	3	The ID associated with the users from the table 'Users'
AnimelD	INTEGER	No length needed	Presence Check	45	The ID associated with animes the user has added to their favourites list
AnimeTitle	VARCHAR	200	Presence Check	'Naruto'	The names of animes the user have added to their favourites list
Ratings	INTEGER	No length needed	Presence Check Must be within the range 0 and 10	6	The ratings of the users favourite animes

Table: UserWatchList- stored as an access table			Primary Key (CONSTRAINT): Pk_UserCompKey (UserWatchID, UserID) Foreign Key: UserID		
Field Name	Data Type	Length	Validation	Example Data	Comments
UserWatchID	AUTOINCREMENT	No length needed	Presence Check Calculated by access	4	The ID for this table
UserID	INTEGER	No length needed	Presence Check	3	The ID associated with the users from the table 'Users'
AnimelD	INTEGER	No length needed	Presence Check Make sure it matches the IDs of animes	2	The ID associated with animes the user has added to their watch list

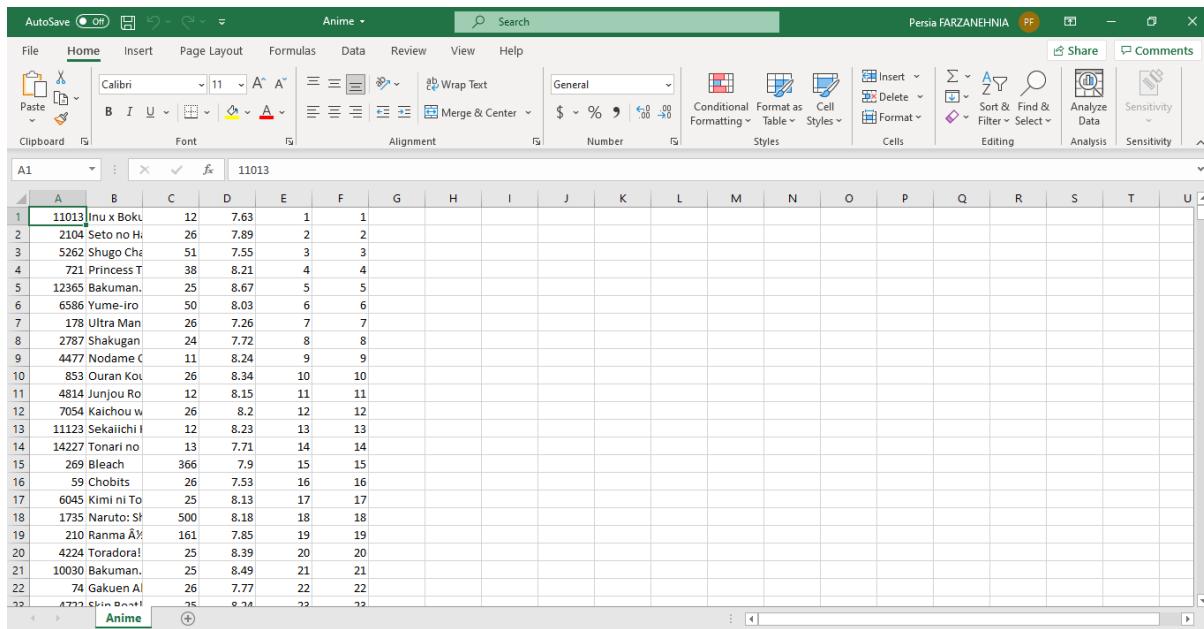
			from the table 'Animes' via a direct insert		
AnimeTitle	VARCHAR	200	Presence Check Make sure it matches the names of anime from the table 'Animes' Via a direct insert	'Bleach'	The names of animes the user have added to their watch list
WatchStatus	VARCHAR	100	Presence Check Use a checkbox to only add values from a list	'Currently Watching'	The users watch status for the animes they have added to their watch list
EpisodesWatched	INTEGER	No length needed	Presence Check Create a function to validate episodes inputted by the user	10	The amount of episodes watched of animes in the user's watch list
Ratings	INTEGER	No length needed	Presence Check Must be in the range of 0 and 10	3	The user's rating of the animes in their watch list

Table-Users

Every user will have to make an account in order to use the program, they will be prompted at the form 'FmMenu' to click on the button 'Sign Up' which will load the form 'FmSignUp'. For the users to make a 'user' account they will need to enter their information in all the required fields of that form to successfully create an account.

Furthermore, for the user to make an 'Administrator' account the user will need to enter their information in all the required fields and enter an admin key which can only be accessed within my code (under the assumption that every day users will not have access to my code). Once created the user's details will be stored in this table.

Table-Animes

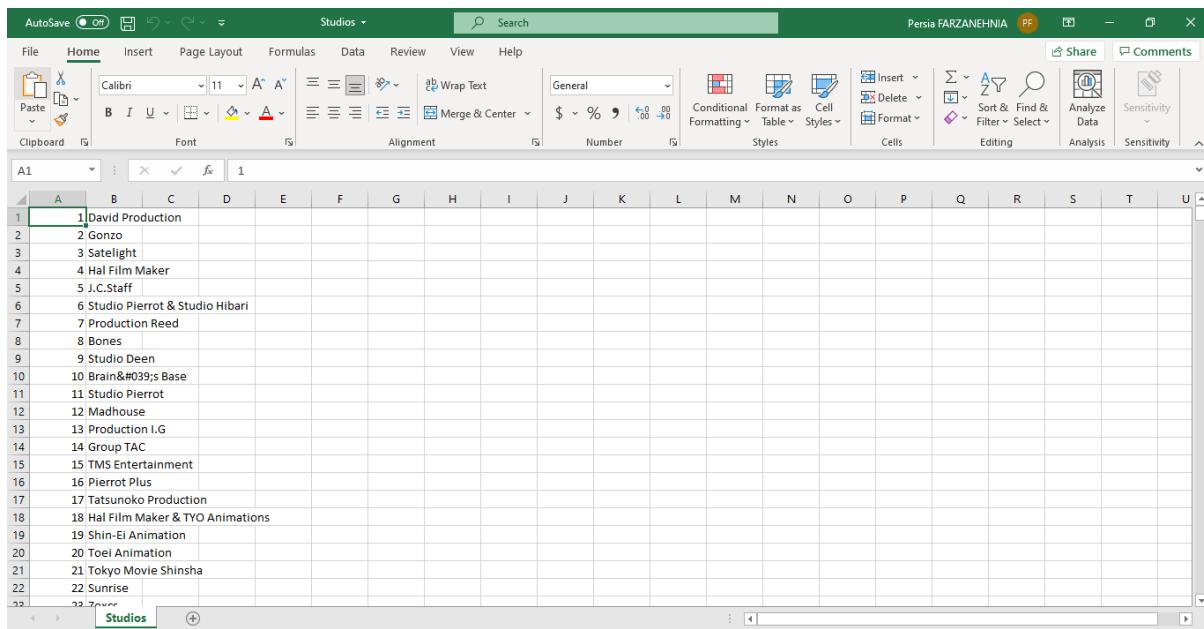


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	11013	Inu x Boku	12	7.63	1	1															
2	2104	Seto no Hi	26	7.89	2	2															
3	5262	Shugo Chi	51	7.55	3	3															
4	721	Princess T	38	8.21	4	4															
5	12365	Bakuman.	25	8.67	5	5															
6	6586	Yume-iro	50	8.03	6	6															
7	178	Ultra Man	26	7.26	7	7															
8	2787	Shakugan	24	7.72	8	8															
9	4477	Nodame C	11	8.24	9	9															
10	853	Ouran Kou	26	8.34	10	10															
11	4814	Junjou Ro	12	8.15	11	11															
12	7054	Kaichou w	26	8.2	12	12															
13	11123	Sekaiichi I	12	8.23	13	13															
14	14227	Tonari no	13	7.71	14	14															
15	269	Bleach	366	7.9	15	15															
16	59	Chobits	26	7.53	16	16															
17	6045	Kimi ni To	25	8.13	17	17															
18	1735	Naruto: Sh	500	8.18	18	18															
19	210	Ranma Ä%	161	7.85	19	19															
20	4224	Toradora!	25	8.39	20	20															
21	10030	Bakuman.	25	8.49	21	21															
22	74	Gakuen Al	26	7.77	22	22															

All the information that will be available in this table in access will be read from the csv file (Anime.csv) which will then be inserted into the table 'Animes' through a function. The csv file will be in the debug file for this program

An admin user can add update, delete information within the forms they have access to from 'FmAMS (Admin)'. All changes of this table will take place in only the table 'Animes' in access.

Table-Studios



A screenshot of Microsoft Excel showing a table of studio names. The table has 22 rows, starting with row 1 containing the text '1 David Production'. The data includes various studio names such as 'Gonzo', 'Satelight', 'Hal Film Maker', 'J.C.Staff', 'Studio Pierrot & Studio Hibari', 'Production Reed', 'Bones', 'Studio Deen', 'Brain's Base', 'Studio Pierrot', 'Madhouse', 'Production I.G', 'Group TAC', 'TMS Entertainment', 'Pierrot Plus', 'Tatsunoko Production', 'Hal Film Maker & TYO Animations', 'Shin-Ei Animation', 'Toei Animation', 'Tokyo Movie Shinsha', and 'Sunrise'. The table is located on a sheet named 'Studios'.

All the information that will be available in the table 'Studios' in access will be read from the csv file (Studios.csv) which will then be inserted into the table 'Studios' through a function. The csv file will be in the debug file for this program

An admin user can add update, delete information within the forms they have access to from 'FmAMS (Admin)'. All changes of this table will take place in only the table 'Studios' in access.

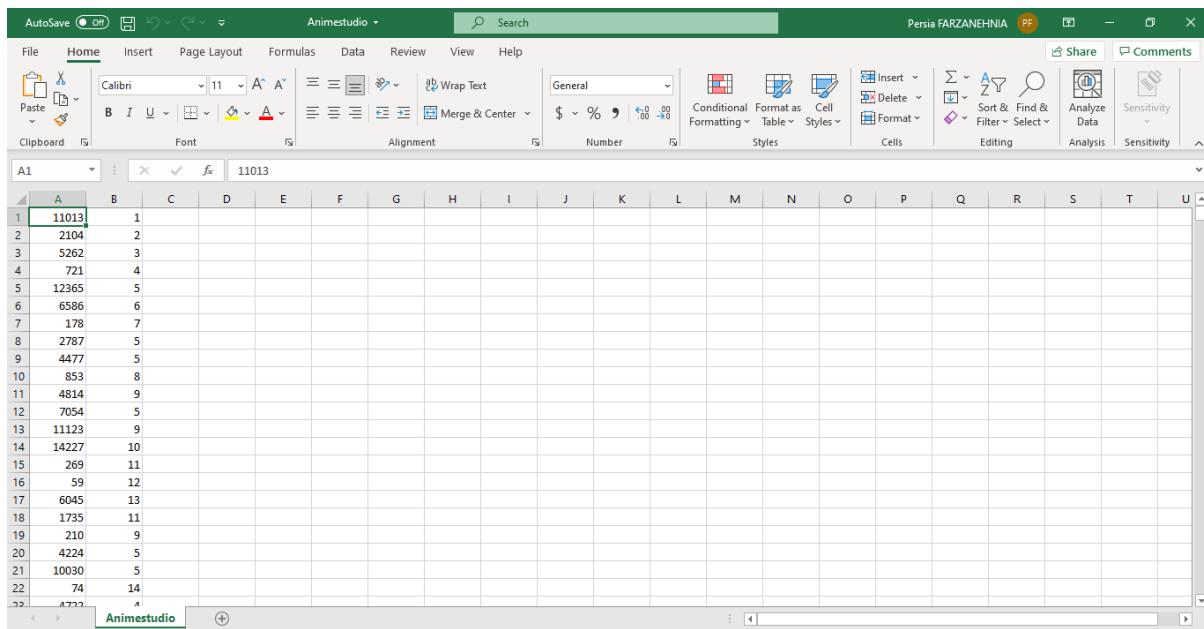
Table-Genres

	Genre
1	1 Comedy & Supernatural & Romance & Shounen
2	2 Comedy & Parody & Romance & School & Shounen
3	3 Comedy & Magic & School & Shoujo
4	4 Comedy & Drama & Magic & Romance & Fantasy
5	5 Comedy & Drama & Romance & Shounen
6	6 Kids & School & Shoujo
7	7 Magic & Comedy & Romance & School & Shoujo
8	8 Action & Drama & Fantasy & Romance & School & Supernatural
9	9 Music & Slice of Life & Comedy & Romance & Josei
10	10 Comedy & Harem & Romance & School & Shoujo
11	11 Comedy & Drama & Romance & Shounen Ai
12	12 Comedy & Romance & School & Shoujo
13	13 Slice of Life & Comedy & Romance & School & Shoujo
14	14 Action & Adventure & Comedy & Super Power & Supernatural & Shounen
15	15 Sci-Fi & Comedy & Drama & Romance & Ecchi & Seinen
16	16 Slice of Life & Drama & Romance & School & Shoujo
17	17 Action & Adventure & Comedy & Super Power & Martial Arts & Shounen
18	18 Slice of Life & Comedy & Martial Arts & Fantasy
19	19 Slice of Life & Comedy & Romance & School
20	20 Comedy & School & Shoujo & Super Power
21	21 Comedy & Drama & Romance & Shoujo

All the information that will be available in the table 'Genres' in access will be read from the csv file (Genre.csv) which will then be inserted into the table 'Studios' through a function. The csv file will be in the debug file for this program

An admin user can add update, delete information within the forms they have access to from 'FmAMS (Admin)'. All changes of this table will take place in only the table 'Genres' in access.

Table-AnimeStudios



1	11013
2	2104
3	5262
4	721
5	12365
6	6586
7	178
8	2787
9	4477
10	853
11	4814
12	7054
13	11123
14	14227
15	269
16	59
17	6045
18	1735
19	210
20	4224
21	10030
22	74

All the information that will be available in the table 'AnimeStudios' in access will be read from the csv file (Animestudio.csv) which will then be inserted into the table 'AnimeStudios' through a function. The csv file will be in the debug file for this program

An admin user can add update, delete information within the forms they have access to from 'FmAMS (Admin)'. All changes of this table will take place in only the table 'AnimeStudios' in access.

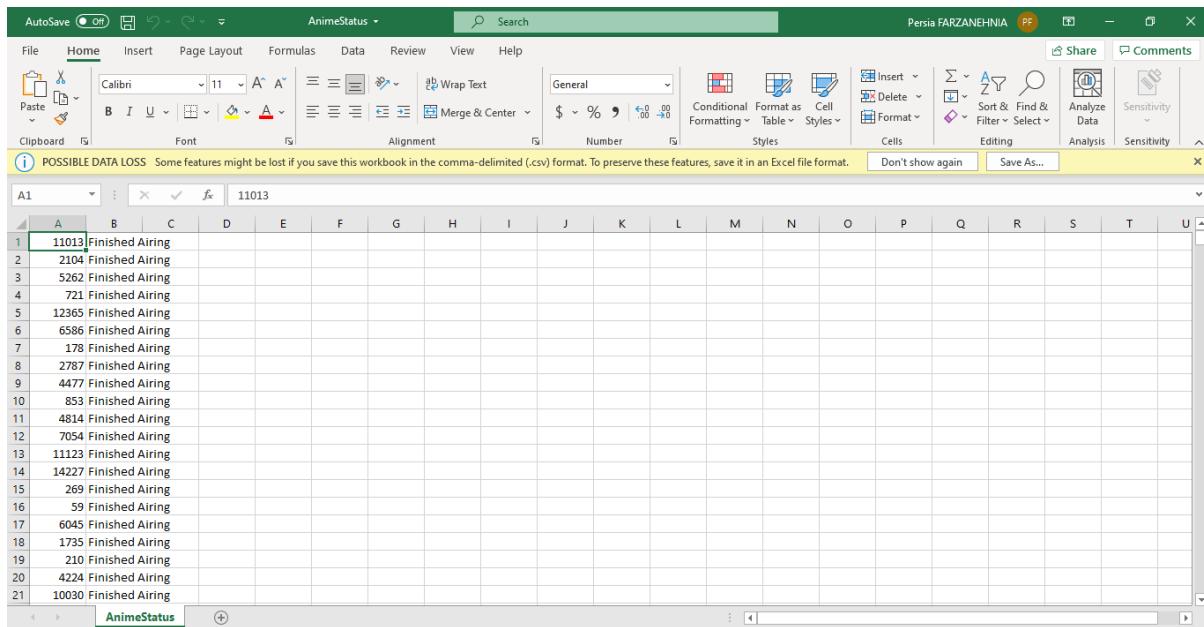
Table-AnimeGenres

Genre ID	Genre Name
11013	1
2104	2
5262	3
721	4
12365	5
6586	6
178	7
2787	8
4477	9
853	10
4814	11
7054	12
11123	11
14227	13
269	14
59	15
6045	16
1735	17
210	18
4224	19
10030	5

All the information that will be available in the table 'AnimeGenres' in access will be read from the csv file (Animegenre.csv) which will then be inserted into the table 'AnimeGenres' through a function. The csv file will be in the debug file for this program

An admin user can add update, delete information within the forms they have access to from 'FmAMS (Admin)'. All changes of this table will take place in only the table 'AnimeGenres' in access.

Table-AnimeStatus



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	11013	Finished Airing																			
2	2104	Finished Airing																			
3	5262	Finished Airing																			
4	721	Finished Airing																			
5	12365	Finished Airing																			
6	6586	Finished Airing																			
7	178	Finished Airing																			
8	2787	Finished Airing																			
9	4477	Finished Airing																			
10	853	Finished Airing																			
11	4814	Finished Airing																			
12	7054	Finished Airing																			
13	11123	Finished Airing																			
14	14227	Finished Airing																			
15	269	Finished Airing																			
16	59	Finished Airing																			
17	6045	Finished Airing																			
18	1735	Finished Airing																			
19	210	Finished Airing																			
20	4224	Finished Airing																			
21	10030	Finished Airing																			

All the information that will be available in the table 'AnimeStatus' in access will be read from the csv file (AnimeStatus.csv) which will then be inserted into the table 'AnimeStatus' through a function. The csv file will be in the debug file for this program

An admin user can add update, delete information within the forms they have access to from 'FmAMS (Admin)'. All changes of this table will take place in only the table 'AnimeStatus' in access.

Table-UserFavourites

Every user will have a user favourites table in order to use the program. Users will add to this table through the form 'FmAnimeInformation' where they can rate and add to favourites.

Users will also be able to update and delete their favourites within the form 'FmUserFavourites'.

Table – UserWatchList

Every user will have a user watchlist table in order to use the program. Users will add to this table through the form 'FmAnimeInformation' where they can rate, add the episodes watched, state their watch status and then add to their watch list.

Users will also be able to update and delete their watch list within the form 'FmUserWatchList'.

Description of Data Structures

My program will require the use of arrays to store the information from all the csv files within this project. This will make my code more efficient and prevent the writing of repeated code as each row in the csv file would have to have an SQL insert statement for every row in all the csv file.

Description of Algorithms

Form: FmMenu (Used for creating tables, and inserting data from csv files)

This is the first form that is loaded by the system therefore, when loaded the program runs the procedure 'FmMenuLoad'. If the database does not exist, the following DDL is ran to create the Database.

```
CREATE TABLE Users (UserID AUTOINCREMENT, Username VARCHAR (60), Pword  
VARCHAR (255), UserType VARCHAR (30), Forename VARCHAR (60), Surname  
VARCHAR (60), Email VARCHAR (255), PRIMARY KEY (UserID))
```

```
CREATE TABLE Studios (StudioID INTEGER, Studio VARCHAR (100), PRIMARY KEY  
(StudioID))
```

```
CREATE TABLE Animes (AnimelD INTEGER, AnimeTitle VARCHAR(200), Episodes  
INTEGER, Ratings INTEGER, AnimeSynopsis LONGTEXT, AnimePictures TEXT,  
PRIMARY KEY (AnimelD))
```

```
CREATE TABLE UserFavourites (UserFavID AUTOINCREMENT, UserID INTEGER,  
AnimelD INTEGER, AnimeTitle VARCHAR (200), Ratings INTEGER, CONSTRAINT  
PK_UserCompKey PRIMARY KEY (UserFavID, UserID) REFERENCES User(UserID))
```

```
CREATE TABLE UserWatchList (UserWatchID AUTOINCREMENT, UserID INTEGER,  
AnimelD INTEGER, AnimeTitle VARCHAR (200), WatchStatus VARCHAR (100),  
EpisodesWatched INTEGER, Ratings INTEGER, CONSTRAINT PK_UserCompKey  
PRIMARY KEY (UserWatchID, UserID) REFERENCES User (UserID))
```

```
CREATE TABLE Genres (GenreID INTEGER, Genre TEXT, PRIMARY KEY (GenreID))
```

```
CREATE TABLE AnimeStudios ( AnimeID INTEGER, StudioID INTEGER, CONSTRAINT
PK_AnimeStudios PRIMARY KEY (AnimeID, StudioID), FOREIGN KEY (AnimeID)
REFERENCES Animes(AnimeID), FOREIGN KEY (StudioID) REFERENCES
Studios(StudioID))
```

```
CREATE TABLE AnimeGenres (AnimeID INTEGER, GenreID INTEGER, CONSTRAINT
PK_AnimeGenres PRIMARY KEY (AnimeID, GenreID), FOREIGN KEY (AnimeID)
REFERENCES Animes(AnimeID), FOREIGN KEY (GenreID) REFERENCES
Genres(GenreID))
```

```
CREATE TABLE AnimeStatus( AnimeID INTEGER, Status VARCHAR (150) CONSTRAINT
PK_AnimeStatus (AnimeID, Status), FOREIGN KEY (AnimeID) REFERENCES
Animes(AnimeID))
```

SQL CSV File reader insert

After the creations of the table the program goes through multiple foreach loops to insert data into the tables

```
Lines ← File.ReadLine(Studio.csv)
```

```
FOREACH (Line in Lines)
```

```
StudioList[ ] ← Line.Split(' , ')
```

```
INSERT INTO Studios VALUES ( StudioList[0], StudioList [1])
```

```
END FOREACH
```

```
Lines ← File.ReadLine(Anime.csv)
```

```
FOREACH (Line in Lines)
```

```
AnimeList[ ] ← Line.Split(' , ')
```

```
INSERT INTO Animes VALUES ( AnimeList[0], AnimeList [1], AnimeList [2], AnimeList [3],
AnimeList[4], AnimeList [5])
```

```
END FOREACH
```

```
Lines ← File.ReadLine(Genre.csv)
```

```
FOREACH (Line in Lines)
```

```
GenreList[ ] ← Line.Split(' , ')
```

```
INSERT INTO Genres VALUES (GenreList[0], GenreList[1])
```

```
END FOREACH
```

```

Lines ← File.ReadLine(Animestudio.csv)

FOREACH (Line in Lines)

    AnimeStudioList[ ] ← Line.Split(‘ , ‘)

    INSERT INTO AnimeStudios VALUES ( AnimeStudioList [0], AnimeStudioList [1] )

END FOREACH

```

```

Lines ← File.ReadLine(Animegenre.csv)

FOREACH (Line in Lines)

    AnimeGenreList[ ] ← Line.Split(‘ , ‘)

    INSERT INTO AnimeGenres VALUES ( AnimeGenreList [0], AnimeGenreList [1] )

END FOREACH

```

```

Lines ← File.ReadLine(AnimeStatus.csv)

FOREACH (Line in Lines)

    AnimeStatusList[ ] ← Line.Split(‘ , ‘)

    INSERT INTO AnimeStatus VALUES ( AnimeStatusList [0], AnimeStatusList [1] )

END FOREACH

```

Class: Encryption- Password hashing

This class is used to hash the passwords that the user will enter in the form ‘FmSignUp’. This class can be accessed at any part of the program.

```

Public HashPassword (password)

SHA1CryptoServiceProvider sha1 ← new SHA1CryptoServiceProvider( )

Passwordbytes [ ] ← Encoding.ASCII.GetBytes(password)

Encryptedbytes [ ] ← sha1.ComputeHash(PasswordBytes)

RETURN Encryptedbytes

```

Form: FmSignUp (Used to insert user details into database)

The user can access this form by clicking the sign-up button in the menu form. The user will be prompted to enter their details and create their accounts

Username ← tbUsername.Text

Password ← tbPword.Text

Forename ← tbForename.Text

Surname ← tbSurname.Text

Email ← tbEmail.Text

HasUpperChar ← new Regex(A-Z)

HasLowerChar ← new Regex(a-z)

HasNumber ← new Regex(0-9)

HasSymbols ← new Regex (@#\$^&* +={\}|{}::<>|./?,.)

EmailValid ← new Regex ([a-zA-Z0-9_.|-+])+@(([a-zA-Z0-9|-])+.)([a-zA-Z0-9])

IF(Username is empty OR has empty space) THEN

MESSAGEBOX (Username should not be empty)

END IF

ELSE IF (Forename is empty OR has empty space) THEN

MESSAGEBOX (Forename should not be empty)

END ELSE IF

ELSE IF (Surname is Empty OR has empty space) THEN

MESSAGEBOX (Surname should not be empty)

END ELSE IF

ELSE IF (Email is Empty OR has empty space) THEN

MESSAGEBOX (Email should not be empty)

END ELSE IF

ELSE IF(Username !=HasNumber.IsMatch) THEN

MESSAGEBOX (Username should contain at least one number)

END ELSE IF

ELSE IF (Username !=HasUpperChar.IsMatch) THEN

MESSAGEBOX (Username should contain at least one upper case letter)

END ELSE IF

ELSE IF (Password is empty OR has empty space) THEN

Centre Number:32455

```
MESSAGEBOX(Password should not be empty)
END ELSE IF
ELSE IF (Password !=HasNumber.IsMatch) THEN
MESSAGEBOX (Password should contain at least one numeric value)
END ELSE IF
ELSE IF (Password !=HasUpperCase.IsMatch) THEN
MESSAGEBOX (Password should contain at least one upper case letter)
END ELSE IF
ELSE IF (Password != HasLowerCase.IsMatch) THEN
MESSAGEBOX (Password should contain at least one lower case letter)
END ELSE IF
ELSE IF (Password !=HasSymbols.IsMatch) THEN
MESSAGEBOX (Password should contain at least one special case letter)
END ELSE IF
ELSE IF ( Email != EmailValid.IsMatch) THEN
MESSAGEBOX ( Email is not valid)
END ELSE IF
ELSE
TRY
SELECT COUNT (*) FROM Users WHERE Username = Username
Count ← ExecuteScalar()
IF (Count >0) THEN
MESSAGEBOX(Username already exists!)
END IF
ELSE
IF( cbUserType.Text = 'Administrator') THEN
Key ← #?123
IF (tbAdminKey.Text != Key) THEN
MESSAGEBOX(Invalid key)
END IF
ELSE
INSERT INTO Users(Username, Pword, UserType, Forename, Surname, Email)
```

```
VALUES( tbUsername.Text, Encryption.HashPassword(tbPword.Text), tbUserType.Text,
tbForename.Text, tbSurname.Text, tbEmail.Text) //admin user insert
```

```
MESSAGEBOX(Details saved you can now log in)
```

```
END ELSE
```

```
ELSE
```

```
INSERT INTO Users(Username, Pword, UserType, Forename, Surname, Email) VALUES(
tbUsername.Text, Encryption.HashPassword(tbPword.Text), tbUserType.Text,
tbForename.Text, tbSurname.Text, tbEmail.Text) //normal user insert
```

```
MESSAGEBOX(Details saved you can now log in)
```

```
END ELSE
```

```
CATCH
```

```
MESSAGEBOX(Could not insert)
```

```
END CATCH
```

```
CATCH
```

```
MESSAGEBOX(Invalid)
```

```
END CATCH
```

Procedure: cbUserTypeSelectedIndexChanged

```
IF(cbUserType.Text = "Administrator") THEN
```

```
lbAdminKey ← Visible
```

```
tbAdminKey ← Visible
```

```
END IF
```

```
ELSE
```

```
lbAdminKey ← invisible
```

```
tbAdminKey ← invisible
```

```
END ELSE
```

Form: FmLogIn (Used to let user log in to gain access to the rest of the program)

This is the form where users can log in to access the rest of the forms in this program. When loaded the program will run the procedure 'FmLogInLoad' which will load up the user's username and password from the properties in the program if the user has clicked the checkbox 'cbRemember'. Once the user clicks on the 'enter' button the click event handler for the button will run.

GlobalVariables

Username

```
IF (properties.settings.default.username != empty) THEN
    tbUsername.Text ← properties.settings.default.username
    tbPword.Text ← properties.settings.default.password
    cbRememeber.Checked ← true
END IF
```

BtEnter.Click

TRY

```
    SELECT * FROM Users WHERE Username = tbUsername.Text
    Username ← tbUsername.Text
    DataReader reader ← ExecuteReader()
    IF (Reader Has rows) THEN
        Reader.Read()
        IF (Encryption.HashPassword(tbPword.Text) = reader [Pword]) THEN
            IF("Administratior"= reader[UserType]) THEN
                LOAD ADMIN FORM
            END IF
        ELSE
            LOAD USER FORM
        END ELSE
    ELSE
        MESSAGEBOX("Wrong password")
    END ELSE
CATCH
```

MESSAGEBOX("Could not be found")

```
IF(cbRemember.Checked = true) THEN
Properties.settings.default.Username ← tbUsername.Text
Properties.settings.Default.Password ← tbPword.Text
Properties.settings.Default.Save
END IF

IF(cbRemember.Checked = false) THEN
Properties.settings.default.Username ← EMPTY
Properties.settings.Default.Password ← EMPTY
Properties.settings.Default.Save
END IF
```

Form: FmAMS (Admin)

This is the form where only admin users can access. In this form the user can search through all tables and edit them. The tables can be seen by clicking the radio button and the show button

GLOBAL VARIABLES

AnimelD
AnimeTitle
Episodes
Status
Ratings
AnimePictures
AnimeSynopsis
StudioID
GenreID
Genre
UserID

btShow.Click

```
IF (RbAnimes.Checked // if radio button is clicked) THEN
    SELECT * FROM Animes
END IF

ELSE IF (RbGenres.Checked// if radio button is clicked) THEN
    SELECT * FROM Genres
END ELSE IF

ELSE IF (RbStudios.Checked// if radio button is clicked) THEN
    SELECT * FROM Studios
END ELSE IF

ELSE IF (RbAnimeStudios.Checked// if radio button is clicked) THEN
    SELECT * FROM AnimeStudios
END ELSE IF

ELSE IF (RbAnimeGenres.Checked// if radio button is clicked) THEN
    SELECT * FROM AnimeGenres
END ELSE IF

ELSE IF (RbAnimeStatus.Checked// if radio button is clicked) THEN
    SELECT * FROM AnimeStatus
END ELSE IF

ELSE IF (RbUsers.Checked// if radio button is clicked) THEN
    SELECT UserID, Username FROM Users
END ELSE IF

DataAdapter ← NewDataAdapter(SELECT QUERY)
DataTable ← New DataTable
DataAdapter.Fill(DataTable)
Grid.Source ← DataTable
```

tbSearch.KeyPress

TRY

```
IF(RbAnimes.Checked//radiobutton clicked) THEN
    IF(TbSearch is not empty) THEN
        Grid.AutoSizeColumnsMode ← DataGridView.Fill
```

```
DataAdapter ← new DataAdapter
DataSet ← new DataSet
DataView ← new DataView
SELECT * FROM Animes WHERE AnimeTitle LIKE tbSearch.Text OR AnimeID
LIKE tbSearch.Text
Command ← new command(SELECT QUERY, CONNECTION TO
DATABASE)
DataAdapter ← new DataAdapter(SELECT QUERY)
DataAdapter.Fill(DataSet)
DataView ← new DataView(DataSet.Table[0])
Grid.DataSource ← DataView
END IF
END IF
ELSE IF (tbSearch is empty) THEN
SELECT * FROM Animes
DataAdapter ← SELECT QUERY
DataTable ← new DataTable
DataAdapter.Fill(DataTable)
Grid.Show
END ELSE IF

ELSE IF(RbGenres.Checked//radiobutton clicked) THEN
IF(TbSearch is not empty) THEN
Grid.AutoSizeColumnsMode ← DataGridView.Fill
DataAdapter ← new DataAdapter
DataSet ← new DataSet
DataView ← new DataView
SELECT * FROM Genres WHERE GenreID LIKE tbSearch.Text OR Genre
LIKE tbSearch.Text
Command ← new command(SELECT QUERY, CONNECTION TO
DATABASE)
DataAdapter ← new DataAdapter(SELECT QUERY)
DataAdapter.Fill(DataSet)
```

```
        DataView ← new DataView(DataSet.Table[0])
        Grid.DataSource ← DataView
    END IF
END IF

ELSE IF (tbSearch is empty) THEN
    SELECT * FROM Genres
    DataAdapter ← SELECT QUERY
    DataTable ← new DataTable
    DataAdapter.Fill(DataTable)
    Grid.Show
END ELSE IF

ELSE IF(RbStudios.Checked//radiobutton clicked) THEN
    IF(TbSearch is not empty) THEN
        Grid.AutoSizeColumnsMode ← DataGridView.Fill
        DataAdapter ← new DataAdapter
        DataSet ← new DataSet
        DataView ← new DataView
        SELECT * FROM Studios WHERE StudioID LIKE tbSearch.Text OR Studios
        LIKE tbSearch.Text
        Command ← new command(SELECT QUERY, CONNECTION TO
        DATABASE)
        DataAdapter ← new DataAdapter(SELECT QUERY)
        DataAdapter.Fill(DataSet)
        DataView ← new DataView(DataSet.Table[0])
        Grid.DataSource ← DataView
    END IF
END ELSE IF

ELSE IF (tbSearch is empty) THEN
    SELECT * FROM Studios
    DataAdapter ← SELECT QUERY
    DataTable ← new DataTable
    DataAdapter.Fill(DataTable)
```

Grid.Show

END ELSE IF

ELSE IF(RbAnimeStudio.Checked//radiobutton clicked) THEN

IF(TbSearch is not empty) THEN

 Grid.AutoSizeColumnsMode ← DataGridView.Fill

 DataAdapter ← new DataAdapter

 DataSet ← new DataSet

 DataView ← new DataView

 SELECT * FROM AnimeStudios WHERE AnimeID LIKE tbSearch.Text OR StudioID LIKE tbSearch.Text

 Command ← new command(SELECT QUERY, CONNECTION TO DATABASE)

 DataAdapter ← new DataAdapter(SELECT QUERY)

 DataAdapter.Fill(DataSet)

 DataView ← new DataView(DataSet.Table[0])

 Grid.DataSource ← DataView

END IF

END ELSE IF

ELSE IF (tbSearch is empty) THEN

 SELECT * FROM AnimeStudios

 DataAdapter ← SELECT QUERY

 DataTable ← new DataTable

 DataAdapter.Fill(DataTable)

 Grid.Show

END ELSE IF

ELSE IF(RbAnimeGenres.Checked//radiobutton clicked) THEN

IF(TbSearch is not empty) THEN

 Grid.AutoSizeColumnsMode ← DataGridView.Fill

 DataAdapter ← new DataAdapter

 DataSet ← new DataSet

 DataView ← new DataView

SELECT * FROM AnimeGenres WHERE AnimeID LIKE tbSearch.Text OR
GenreID LIKE tbSearch.Text

Command ← new command(SELECT QUERY, CONNECTION TO
DATABASE)

DataAdapter ← new DataAdapter(SELECT QUERY)

DataAdapter.Fill(DataSet)

DataView ← new DataView(DataSet.Table[0])

Grid.DataSource ← DataView

END IF

END ELSE IF

ELSE IF (tbSearch is empty) THEN

SELECT * FROM AnimeGenres

DataAdapter ← SELECT QUERY

DataTable ← new DataTable

DataAdapter.Fill(DataTable)

Grid.Show

END ELSE IF

ELSE IF(RbAnimeStatus.Checked//radiobutton clicked) THEN

IF(TbSearch is not empty) THEN

Grid.AutoSizeColumnsMode ← DataGridView.Fill

DataAdapter ← new DataAdapter

DataSet ← new DataSet

DataView ← new DataView

SELECT * FROM AnimeStatus WHERE AnimeID LIKE tbSearch.Text OR Status
LIKE tbSearch.Text

Command ← new command(SELECT QUERY, CONNECTION TO
DATABASE)

DataAdapter ← new DataAdapter(SELECT QUERY)

DataAdapter.Fill(DataSet)

DataView ← new DataView(DataSet.Table[0])

Grid.DataSource ← DataView

END IF

END ELSE IF

```
ELSE IF (tbSearch is empty) THEN
    SELECT * FROM AnimeStatus
    DataAdapter ← SELECT QUERY
    DataTable ← new DataTable
    DataAdapter.Fill(DataTable)
    Grid.Show
END ELSE IF

ELSE IF (RbUsers.Checked//radiobutton clicked)
    IF(TbSearch is not empty) THEN
        Grid.AutoSizeColumnsMode ← DataGridView.Fill
        DataAdapter ← new DataAdapter
        DataSet ← new DataSet
        DataView ← new DataView
        SELECT UserID, Username FROM Users WHERE Username LIKE tbSearch.Text
        Command ← new command(SELECT QUERY, CONNECTION TO
DATABASE)
        DataAdapter ← new DataAdapter(SELECT QUERY)
        DataAdapter.Fill(DataSet)
        DataView ← new DataView(DataSet.Table[0])
        Grid.DataSource ← DataView
    END IF
END ELSE IF

ELSE IF (tbSearch is empty) THEN
    SELECT UserID, Username FROM Users
    DataAdapter ← SELECT QUERY
    DataTable ← new DataTable
    DataAdapter.Fill(DataTable)
    Grid.Show
END ELSE IF

CATCH
MESSAGEBOX("Does not Exist")
```

GridCellContent.Click

```
IF(RbAnimes.Checked// radio button clicked) THEN
    IF(Cell.RowIndex >=0) THEN
        GridView row ← Grid.Rows[Cell.RowIndex]
        AnimelD ← Row.Cells[“AnimelD”]
        AnimeTitle ← Row.Cells[“AnimeTitle”]
        Episodes ← Row.Cells[“Episodes”]
        AnimeSynopsis ← Row.Cells[“AnimeSynopsis”]
        AnimePictures ← Row.Cells[“AnimePictures”]
        LOAD EDIT FORM FOR ANIMES (FmEditAnimes)
    END IF
ELSE IF(RbStudios.Checked// radio button clicked) THEN
    IF(Cell.RowIndex >=0) THEN
        GridView row ← Grid.Rows[Cell.RowIndex]
        StudioID ← Row.Cells[“StudioID”]
        Studio ← Row.Cells[“Studio”]
        LOAD EDIT FORM FOR STUDIOS (FmEditStudio)
    END ELSE IF
ELSE IF(RbGenres.Checked// radio button clicked) THEN
    IF(Cell.RowIndex >=0) THEN
        GridView row ← Grid.Rows[Cell.RowIndex]
        GenreID ← Row.Cells[“GenreID”]
        Genre ← Row.Cells[“Genre”]
        LOAD EDIT FORM FOR GENRES (FmEditGenre)
    END ELSE IF
ELSE IF(RbAnimeGenres.Checked// radio button clicked) THEN
    IF(Cell.RowIndex >=0) THEN
        GridView row ← Grid.Rows[Cell.RowIndex]
        AnimelD ← Row.Cells[“AnimelD”]
        GenreID ← Row.Cells[“GenreID”]
        LOAD EDIT FORM FOR ANIMEGENRES (FmEditAnimeGenres)
```

END ELSE IF

ELSE IF(RbAnimeStudios.Checked// radio button clicked) THEN

IF(Cell.RowIndex >=0) THEN

GridView row ← Grid.Rows[Cell.RowIndex]

AnimelD ← Row.Cells["AnimelD"]

StudioID ← Row.Cells["StudioID"]

LOAD EDIT FORM FOR ANIMESTUDIOS (FmEditAnimeStudio)

END ELSE IF

ELSE IF(RbAnimeStatus.Checked// radio button clicked) THEN

IF(Cell.RowIndex >=0) THEN

GridView row ← Grid.Rows[Cell.RowIndex]

AnimelD ← Row.Cells["StudioID"]

Status ← Row.Cells["Status"]

LOAD EDIT FORM FOR ANIME STATUS (FmEditAnimeStatus)

END ELSE IF

ELSE IF(RbUsers.Checked// radio button clicked) THEN

IF(Cell.RowIndex >=0) THEN

GridView row ← Grid.Rows[Cell.RowIndex]

UserID ← Row.Cells["UserID"]

Username ← Row.Cells["Username"]

LOAD DELETE FORM FOR USERS (FmDeleteUser)

END ELSE IF

BtEdit.Click

IF (RbAnimes.Checked// radio button clicked) THEN

LOAD EDIT FORM FOR ANIMES (FmEditAnimes)

END IF

ELSE IF (RbStudios.Checked// radio button clicked) THEN

LOAD EDIT FORM FOR STUDIOS (FmEditStudios)

END ELSE IF

ELSE IF (RbGenres.Checked// radio button clicked) THEN

LOAD EDIT FORM FOR GENRES (FmEditGenres)

END ELSE IF

```

ELSE IF (RbAnimeStudios.Checked// radio button clicked) THEN
    LOAD EDIT FORM FOR ANIMESTUDIOS (FmEditAnimeStudios)
END ELSE IF

ELSE IF (RbAnimeGenres.Checked// radio button clicked) THEN
    LOAD EDIT FORM FOR ANIMEGENRES (FmEditAnimeGenres)
END ELSE IF

ELSE IF (RbAnimeStatus.Checked// radio button clicked) THEN
    LOAD EDIT FORM FOR ANIMESTATUS (FmEditAnimeStatus)
END ELSE IF

ELSE IF (RbUsers.Checked// radio button clicked) THEN
    LOAD DELETE FORM FOR USERS (FmDeleteUser)
END ELSE IF

```

Form: EditAnimes (Admin Only)

This form is used to make changes to the table 'Animes'. When this form is loaded the procedure 'FmEditAnimes_Load' will use the global variables that contains row data from the grid in 'FmAMS(Admin)' and output them into all the required fields in this form. The admin user will be able to update or delete this information from the table 'Anime' using SQL queries.

FmEditAnimes_Load

```

TbAnimeID.Text ← AnimeID
TbAnimeTitle.Text ← AnimeTitle
TbEpisodes.Text ← Episodes
TbRatings.Text ← Ratings
TbAnimeSynopsis.Text ← AnimeSynopsis
TbAnimePictures.Text ← AnimePictures

```

BtInsert.Click

```

TRY
    INSERT INTO Animes VALUES (tbAnimeID.Text, tbAnimeTitle.Text, tbEpisodes.Text, tb
    Ratings.Text, tbAnimeSynopsis.Text, tbAnimePictures.Text)
    MESSAGEBOX ("Information successfully added")
CATCH
    MESSAGEBOX ("Invalid")

```

BtUpdate.Click

TRY

```
UPDATE Animes SET AnimeTitle = tbAnimeTitle.Text  Episodes = tbEpisodes.Text,  
Ratings = tbRatings.Text, AnimeSynopsis = tbAnimeSynopsis,  AnimePictures=  
tbAnimePictures.Text WHERE AnimelD= tbAnimelD.Text
```

MESSAGEBOX ("Information updated!")

CATCH

MESSAGEBOX ("Could not update!")

BtDelete.Click

TRY

```
DELETE FROM Animes WHERE AnimelD = tbAnimelD.Text
```

MESSAGEBOX ("Information deleted!")

CATCH

MESSAGEBOX ("Could not delete")

BtClear.Click

TbAnimelD.Clear

TbAnimeTitle.Clear

TbEpisodes.Clear

TbRatings.Clear

TbAnimeSynopsis.Clear

TbAnimePictures.Clear

Form: EditStudios (Admin Only)

This form is used to make changes to the table 'Studios'. When this form is loaded the procedure 'FmEditStudios_Load' will use the global variables that contains row data from the grid in 'FmAMS(Admin)' and output them into all the required fields in this form. The admin user will be able to update or delete this information from the table 'Studios' using SQL queries.

FmEditStudios_Load

Centre Number:32455

TbStudioID.Text ← **StudioID**

TbStudio.Text ← **Studio**

BtInsert.Click

TRY

INSERT INTO Studios VALUES (tbStudioID.Text, tbStudio.Text)

MESSAGEBOX ("Information added")

CATCH

MESSAGEBOX ("Information could not be added")

BtUpdate.Click

TRY

UPDATE Studios SET StudioID = tbStudioID.Text, Studio = tbStudio.Text

MESSAGEBOX ("Information updated")

CATCH

MESSAGEBOX ("Information could not be updated")

BtDelete.Click

TRY

DELETE FROM Studios WHERE StudioID=tbStudioID.Text

MESSAGEBOX ("Information Deleted")

CATCH

MESSAGEBOX ("Information could not be deleted")

Form: EditGenres (Admin Only)

This form is used to make changes to the table 'Genres'. When this form is loaded the procedure 'FmEditGenres_Load' will use the global variables that contains row data from the grid in 'FmAMS(Admin)' and output them into all the required fields in this form. The admin user will be able to update or delete this information from the table 'Genres' using SQL queries.

FmEditGenres_Load

TbGenreID.Text ← **GenreID**

TbGenre.Text ← **Genre**

BtInsert.Click

Centre Number:32455

TRY

```
INSERT INTO Genres VALUES (tbGenreID.Text, tbGenre.Text)  
MESSAGEBOX ("Information added")
```

CATCH

```
MESSAGEBOX ("Information could not be added")
```

BtUpdate.Click

TRY

```
UPDATE Genres SET GenreID = tbGenreID.Text Genre = tbGenre.Text  
MESSAGEBOX ("Information updated")
```

CATCH

```
MESSAGEBOX ("Information could not be updated")
```

BtDelete.Click

TRY

```
DELETE FROM Genres WHERE GenreID=tbGenreID.Text  
MESSAGEBOX ("Information Deleted")
```

CATCH

```
MESSAGEBOX ("Information could not be deleted")
```

Form: EditAnimeStudios(Admin Only)

This form is used to make changes to the table 'AnimeStudios'. When this form is loaded the procedure 'FmEditAnimeStudios_Load' will use the global variables that contains row data from the grid in 'FmAMS(Admin)' and output them into all the required fields in this form. The admin user will be able to update or delete this information from the table 'AnimeStudios' using SQL queries.

FmEditGenres Load

TbAnimeID.Text \leftarrow **AnimID**

TbStudioID.Text \leftarrow **StudioID**

BtInsert.Click

TRY

```
INSERT INTO AnimeStudios VALUES (tbAnimID.Text, tbStudioID.Text)  
MESSAGEBOX ("Information added")
```

CATCH

```
MESSAGEBOX ("Information could not be added")
```

BtUpdate.Click

TRY

```
UPDATE Genres SET AnimeID = tbAnimeID.Text StudioID = tbStudioID.Text
MESSAGEBOX ("Information updated")
```

CATCH

```
MESSAGEBOX ("Information could not be updated")
```

BtDelete.Click

TRY

```
DELETE FROM AnimeStudios WHERE AnimeID=tbAnimeID.Text
MESSAGEBOX ("Information Deleted")
```

CATCH

```
MESSAGEBOX ("Information could not be deleted")
```

Form: EditAnimeGenres(Admin Only)

This form is used to make changes to the table 'AnimeGenres'. When this form is loaded the procedure 'FmEditAnimeGenres_Load' will use the global variables that contains row data from the grid in 'FmAMS(Admin)' and output them into all the required fields in this form. The admin user will be able to update or delete this information from the table 'AnimeGenres' using SQL queries.

FmEditAnimeGenres_Load

```
TbAnimeID.Text ←— AnimeID
```

```
TbGenreID.Text ←— GenreID
```

BtInsert.Click

TRY

```
INSERT INTO AnimeGenres VALUES (tbAnimeID.Text, tbGenreID.Text)
MESSAGEBOX ("Information added")
```

CATCH

```
MESSAGEBOX ("Information could not be added")
```

BtUpdate.Click

TRY

```
UPDATE AnimeGenres SET AnimeID = tbAnimeID.Text GenreID = tbGenreID.Text
MESSAGEBOX ("Information updated")
```

CATCH

```
MESSAGEBOX ("Information could not be updated")
```

BtDelete.Click

TRY

Centre Number:32455

50

```
DELETE FROM AnimeGenres WHERE AnimeID=tbAnimeID.Text
MESSAGEBOX ("Information Deleted")

CATCH
MESSAGEBOX ("Information could not be deleted")
```

Form: EditAnimeStatus (Admin Only)

This form is used to make changes to the table 'AnimeStatus'. When this form is loaded the procedure 'FmEditAnimeStatus_Load' will use the global variables that contains row data from the grid in 'FmAMS(Admin)' and output them into all the required fields in this form. The admin user will be able to update or delete this information from the table 'AnimeStatus' using SQL queries.

FmEditAnimeStatus_Load

```
TbAnimeID.Text ←— AnimeID
TbStatus.Text ←— Status
```

BtInsert.Click

```
TRY
INSERT INTO AnimeStatus VALUES (tbAnimeID.Text, tbStatus.Text)
MESSAGEBOX ("Information added")

CATCH
MESSAGEBOX ("Information could not be added")
```

BtUpdate.Click

```
TRY
UPDATE AnimeStatus SET Status = tb.Status.Text WHERE AnimeID = tbAnimeID.Text
MESSAGEBOX ("Information updated")

CATCH
MESSAGEBOX ("Information could not be updated")
```

BtDelete.Click

```
TRY
DELETE FROM AnimeStatus WHERE AnimeID=tbAnimeID.Text
MESSAGEBOX ("Information Deleted")

CATCH
MESSAGEBOX ("Information could not be deleted")
```

Form:DeleteUser (Admin Only)

This form is used to delete users from the table 'Users'. When this form is loaded the procedure 'FmDeleteUser_Load' will use the global variables that contains row data from the grid in 'FmAMS(Admin)' and output them into all the required fields in this form. The admin user will be able to delete this information from the table 'Users' using SQL Delete queries.

FmEditDeleteUser_Load

TbUserID.Text ← **UserID**

TbUsers.Text ← **Users**

BtDelete.Click

TRY

DELETE FROM UserFavourites WHERE UserID=tbUserID.Text

EXECUTENONQUERY

DELETE FROM UserWatchList WHERE UserID=tbUserID.Text

EXECUTENONQUERY

DELETE FROM Users WHERE UserID=tbUserID.Text

EXECUTENONQUERY

MESSAGEBOX ("Information Deleted")

CATCH

MESSAGEBOX ("Information could not be deleted")

Class:DarkmodeSwitch

GLOBAL VARIABLES

FontColour

BackgroundColour

PanelColour

TextBoxColour

SearchBoxColour

ProfileButtonColour

Function:SwitchFunction

IF(SWITCH = TRUE) THEN

 DARKMODE()

END IF

Centre Number:32455

52

ELSE

 LIGHTMODE()

END ELSE

LightMode()

FontColour ← SET RGB VALUE

BackgroundColour ← SET RGB VALUE

PanelColour ← SET RGB VALUE

GroupBoxColour ← SET RGB VALUE

TextBoxColour ← SET RGB VALUE

SearchBoxColour ← SET RGB VALUE

ProfileButtonColour ← SET RGB VALUE

Switch ← True

DarkMode()

FontColour ← SET RGB VALUE

BackgroundColour ← SET RGB VALUE

PanelColour ← SET RGB VALUE

GroupBoxColour ← SET RGB VALUE

TextBoxColour ← SET RGB VALUE

SearchBoxColour ← SET RGB VALUE

ProfileButtonColour ← SET RGB VALUE

Switch ← False

Form:FmAMS (MainForm)

In this form the user can search up animes click on pictures and access their profile through a button. When the form is 3 pictures will be loaded into the pictureboxes the webscraper to fetch anime news will also be in the load procedure. This form will also make use of the api jikan.moe which will get the picture links of animes and their synopsis. Once fetched it will then be inserted into the database by updating the placeholder values. This form will also make use of a DarkModeSwitch which will allow users to change the layout of the form.

Global Variables

AnimePic

AnimeName

FmAMS_Load

PICTUREBOX ← ANIME PICTURE URL

PICTUREBOX2 ← ANIME PICTURE URL
 PICTUREBOX 3 ← ANIME PICTURE URL
 lbUsername.Text ← Username
 AutoFillSearch()
 IF (DarkModeSwitch.Switch = true) THEN
 SetColour()
 END IF
 ELSE
 DarkModeSwitch.Switch ← FALSE
 END ELSE

SetColour

FMAMS.BackColor ← DarkModeSwitch.**BackgroundColour**
 FMAMS.ForeColor ← DarkModeSwitch.**FontColour**
 AnimePanel.BackColor ← DarkModeSwitch.**PanelColour**
 AMS.BackColor ← DarkModeSwitch.**GroupBoxColour**
 News.BackColor ← DarkModeSwitch.**TextBoxColour**
 News.ForeColor ← DarkModeSwitch.**FontColour**
 Search.BackColor ← DarkModeSwitch.**BackgroundColour**
 Search.ForeColor ← DarkModeSwitch.**FontColour**
 Back.BackColor ← DarkModeSwitch.**TextBoxColour**
 Back.ForeColor ← DarkModeSwitch.**TextBoxColour**
 Profile.BackColor ← DarkModeSwitch.**ProfileButtonColour**
 Search.BackColor ← DarkModeSwitch.**SearchBoxColour**
 Switch.BackColor ← DarkModeSwitch.**SearchBoxColour**

AutoFillSearch

TRY
 SELECT AnimeTitle FROM Animes
 AutoFill ← new AutoCompleteStringCollection
 DataReader ← ExecuteReader
 IF(DataReader.Read) THEN
 WHILE(DataReader.Read)

```
AutoFill.Add(DataReader["AnimeTitle"]  
END WHILE  
END IF  
ELSE  
MESSAGEBOX ("No AnimeTitle found!")  
END ELSE  
DataReader.Close()  
tbSearch.AutoCompleteSource ← AutoFill  
CATCH  
MESSAGEBOX("Could not load")
```

DelaySeleniumDriver

```
ChromeDriverService ← CreateDefaultService  
ChromeDriverService.HideCommandPromptWindow ← true  
ChromeOptions ← New ChromeOptions  
ChromeOptions (headless)  
ChromeDriver ← New ChromeDriver(ChromeDriverService, ChromeOptions)  
ChromeDriver.Navigate.GoToURL (animenewslink)  
Agree ← ChromeDriver.FindElement(AgreeBox)  
Agree.Click  
News ← ChromeDriver.FindElement(News)  
WHILE (TRUE)  
FOR LOOP  
TRY  
Task.Delay (Approx 4 seconds)  
NewsTitles ← ChromeDriver.FindElement(News[i])  
tbNews.Text← NewsTitles.Text  
CATCH  
END FOR LOOP  
ChromeDriver.Close ( )  
ChromeDriver.Quit( )  
END WHILE LOOP
```

BtSearch.Click

```
SELECT * FROM Animes WHERE AnimeTitle = tbSearch.Text
DataReader ← ExecuteReader()
IF (DataReader.HasRows) THEN
    DataReader.Read()
    Pic ← DataReader["AnimePicture"]
    Numeric value ← NULL
    IF( TryParse(Pic, out Numeric Value) THEN
        TRY
            AnimeID ← DataReader ["AnimeID"]
            AnimeURL ← API LINK [AnimeID] / Pictures
            JSON ← JObject.Parse( new WebClient.Download(AnimeURL)
            AnimeLink ← JSON["Pictures"] [0] ["Small"]
            PbAnimePicture4 ← LoadAsync(AnimeLink)
            lbAnimeTitle.Text ← tbSearch.Text
            AnimeName ← lbAnimeTitle.Text
            PaAnimePanel.Visible ← true
            DataReader.Close()
            UPDATE Animes SET AnimePictures = AnimeLink WHERE AnimeID =
            AnimeID
            AnimePic ← AnimeLink
        CATCH
    END IF
    ELSE
        AnimeID ← DataReader ["AnimeID"]
        DataReader.Close()
        SELECT AnimePictures FROM Animes WHERE AnimeID ← AnimeID
        EXECUTENONQUERY
        Picture ← EXECUTESCALAR
        AnimePic ← Picture
        AnimeName ← tbSearch.Text
        PbAnimePicture4.LoadAsync(AnimePic)
```

```
PaAnimePanel.Visible ← true  
END ELSE
```

BtSwitch.Click

```
DarkMode.SwitchFunction()
```

```
SetColour()
```

Form: FmAnimeInformation

This form occurs when the user clicks on the anime pictures from the main form 'FmAMS'. In this form the user will be able to add the anime to their watchlist or favourites. They will also be able to view the synopsis about this anime and all the other information associated with the anime.

AnimeInformation

```
lbAnimeName.Text ←  AnimeName
```

```
PbAnimePicture.LoadAsync( AnimePic)
```

```
SELECT AnimeID FROM Animes WHERE AnimeTitle = lbAnimeName.Text
```

```
EXECUTENONQUERY
```

```
ID ← EXECUTESCALAR
```

```
DbAnimeID.Text ← ID
```

```
SELECT Status FROM AnimeStatus WHERE AnimeID = ID
```

```
EXECUTENONQUERY
```

```
Status ← EXECUTESCALAR
```

```
DbStatus.Text ← Status
```

```
SELECT Ratings FROM Animes WHERE AnimeID = ID
```

```
EXECUTENONQUERY
```

```
Ratings ← EXECUTESCALAR
```

```
DbRatings.Text ← Ratings
```

```
SELECT GenreID FROM AnimeGenres WHERE AnimeID = ID
```

```
EXECUTENONQUERY
```

```
GenreID ← EXECUTESCALAR
```

```
SELECT Genre FROM Genres WHERE GenreID = GenreID
```

```
EXECUTENONQUERY
```

```
Genre ← EXECUTESCALAR
```

```
TbGenre.Text ← Genre
```

```

SELECT StudioID FROM AnimeStudios WHERE AnimeID = ID
EXECUTENONQUERY

SELECT Studio FROM Studios WHERE StudioID = StudioID
EXECUTENONQUERY

Studio ← EXECUTESCALAR

DbStudios.Text ← Studio

SELECT Episodes FROM Animes WHERE AnimeID = ID
EXECUTENONQUERY

Episodes ← EXECUTESCALAR

DbEpisodes.Text ← Episodes

```

SynopsisApi

```

SELECT AnimeID FROM Animes WHERE AnimeTitle= lbAnimeName.Text
EXECUTENONQUERY

ID ← EXECUTESCALAR

SELECT * FROM Animes WHERE AnimeTitle = lbAnimeName.Text
DataReader ← EXECUTEREADER

IF(DataReader.HasRows) THEN
    DataReader.Read

    Asynopsis ← DataReader [“AnimeSynopsis”]
    NumericValue ← NULL

    IF(TryParse(Asynopsis, out NumericValue) THEN
        AnimeUrl ← ApiLink/ Anime/ ID
        JSON ← JObject.Parse(newWebClient().Download(AnimeUrl))
        Text ← JSON [“data”] [“Synopsis”]
        rtbAnimeSynopsis.Text ← Text

        AnimeSynopsis ← Text
        EditedSynopsis ← AnimeSynopsis.Replace ( ““ , “ “)
        UPDATE Animes SET AnimeSynopsis= EditedSynopsis WHERE AnimeID = ID
        EXECUTENONQUERY

    END IF
    ELSE

```

```
    AnimeID ← DataReader[“AnimeID”]
    DataReader.Close()
    SELECT AnimeSynopsis FROM Animes WHERE AnimeID = AnimeID
    EXECUTENONQUERY
    Synopsis ← EXECUTESCALAR
    rtbAnimeSynopsis.Text ← Synopsis
    END ELSE
    END IF
```

BtFavourites.Click

```
TRY
    Username ← Username
    SELECT UserID FROM Users WHERE Username = Username
    UserID ← EXECUTESCALAR
    lbAnimeName.Text ← AnimeName
    SELECT COUNT(*) FROM UserFavourites WHERE AnimeTitle = lbAnimeName.Text
    AND UserID= UserID
    Count ← EXECUTESCALAR
    IF (Count > 0) THEN
        MESSAGEBOX( “Anime is already favourited” )
    END IF
    ELSE
        INSERT INTO UserFavourites(UserID, AnimeID, AnimeTitle, Ratings) VALUES
        (UserID, DbAnimeID.Text, lbAnimeTitle.Text, CbRate.Text)
        EXECUTENONQUERY
        MESSAGEBOX(“Added to Favourites!”)
    END ELSE
    CATCH
        MESSAGEBOX(“Invalid”)
```

BtWatchList.Click

```
TRY
    Username ← Username
    SELECT UserID FROM Users WHERE Username = Username
```

```

UserID← EXECUTESCALAR
IbAnimeName.Text ←  AnimeName
SELECT COUNT(*) FROM UserWatchList WHERE AnimeTitle = IbAnimeName.Text
AND UserID= UserID
Count ← EXECUTESCALAR
IF (Count > 0) THEN
    MESSAGEBOX( "Anime is already favourited" )
END IF
ELSE
    INSERT INTO UserWatchList(UserID, AnimelD, AnimeTitle, WatchStatus,
    EpisodesWatched ,Ratings) VALUES
    (UserID, DbAnimelD.Text, IbAnimeTitle.Text,cbWatchStatus.Text,
    tbEpisodesWatched.Text, cbRate.Text)
    EXECUTENONQUERY
    MESSAGEBOX("Added to WatchList!")
END ELSE
CATCH
    MESSAGEBOX("Invalid")

```

TbEpisodesWatched.Validating

```

HasSymbols ← NEW REGEX (all symbols here)
HasNumbers ← NEW REGEX ([0-9])
IF ( !HasNumbers.IsMatch(tbEpisodesWatched.Text) THEN
    MESSAGEBOX ("Please enter a number!")
END IF
ELSE IF ( HasSymbols.IsMatch(tbEpisodesWatched.Text) THEN
    MESSAGEBOX ("Please enter a number!")
END ELSE IF
ELSE
    SELECT Episodes FROM Animes WHERE AnimeTitle = AnimeTitle
    Episodes← EXECUTESCALAR
    EpisodesEntered← tbEpisodesWatched.Text

```

```

IF( EpisodesEntered > Episodes) THEN
    MESSAGEBOX("Out of range!")
    tbEpisodesWatched.Text ← Episodes
END IF

ELSE IF (EpisodesEntered < 0) THEN
    MESSAGEBOX("Out of range!")
    tbEpisodesWatched.Text ← "0"
END ELSE IF

END ELSE

```

Cb WatchStatus.SelectedIndexChanged

```

SELECT Episodes FROM Animes WHERE AnimeTitle = lbAnimeTitle.Text
Episodes ← EXECUTESCALAR

IF (CbWatchStatus.Text == "Completed") THEN
    tbEpisodesWatched.Text ← Episodes
END IF

ELSE IF (CbWatchStatus.Text == "Not Yet Watched") THEN
    tbEpisodesWatched.Text ← "0"
END ELSE IF

```

Form: FmUserFavourites

In this form the user can make changes to their favourites ranging from updating their ratings and deleting animes off their favourites. Like the admin form, the user will see a grid table that will show them their favourites.

FmUserFavourites_Load

```

lbUsername.Text ← Username

SELECT UserID FROM Users WHERE Username = lbUsername.Text
UserID ← EXECUTESCALAR

SELECT * FROM UserFavourites WHERE UserID = UserID

DataAdapter ← new DataAdapter
DataTable ← new DataTable
DataAdapter.Fill(DataTable)
UserFavouritesGrid.DataSource ← DataTable

```

UserFavouritesGrid.Show

AutoFillSearch()

AutoFillSearch

SELECT UserID FROM Users WHERE Username = lbUsername.Text

UserID ← EXECUTESCALAR

SELECT AnimeTitle FROM UserFavourites WHERE UserID= UserID

AutoFill ← new AutoCompleteStringCollection

DataReader ← EXECUTEREADER

IF(DataReader.HasRows) THEN

 WHILE(DataReader.Read)

 AutoFill.Add(DataReader["AnimeTitle"])

 END WHILE

END IF

ELSE

 MESSAGEBOX("AnimeTitle not found!")

END ELSE

UserFavouritesGrid_CellContentClick

IF (Cell.RowIndex>= 0) THEN

 DataGridViewRow ← UserFavouritesGrid.Rows[Cell.RowIndex]

 TbUserID.Text ← DataGridViewRow.Cells["UserID"]

 TbAnimeID.Text ← DataGridViewRow.Cells["AnimeID"]

 TbAnimeTitle.Text ← DataGridViewRow.Cells["AnimeTitle"]

 CbRatings.Text ← DataGridViewRow.Cells[Rating]

 SELECT AnimePictures FROM Animes WHERE AnimeTitle = tbAnimeTitle.Text

 AnimePic ← EXECUTESCALAR

 PbAnimePicture.LoadAsync(AnimePic)

END IF

TbSearch.KeyPress

SELECT UserID FROM Users WHERE Username = lbUsername.Text

UserID ← EXECUTESCALAR

Centre Number:32455

62

```
IF(tbSearch.Text != ""//if its not empty) THEN
    DataAdapter ← new DataAdapter
    DataSet ← new DataSet
    DataView ← new DataView
    SELECT * FROM UserFavourites WHERE AnimeTitle LIKE tbAnimeTitle AND UserID
    LIKE UserID
    Command ← new Command (SELECT QUERY, Conn)
    DataAdapter← new DataAdapter(Command)
    DataAdapter.Fill(DataSet)
    DataView ← new DataView(DataSet.Tables[0])
    UserFavouritesGrid.DataSource ← DataView
END IF

ELSE IF (tbSearch.Text == " " // if its empty) THEN
    SELECT * FROM UserFavourites WHERE UserID= UserID
    DataAdapter← new DataAdapter
    DataTable ← new DataTable
    DataAdapter.Fill(DataTable)
    UserFavouritesGrid.DataSource ← DataTable
END ELSE IF
```

BtUpdate.Click

```
TRY
    SELECT UserID FROM Users WHERE Username = lbUsername.Text
    UserID ← EXECUTESCALAR
    UPDATE UserFavourites SET Ratings = cbRatings.Text WHERE
    UserID= UserID AND AnimeTitle = tbAnimeTitle.Text
    EXECUTENONQUERY
    MESSAGEBOX ("Favourites Updated!")
CATCH
    MESSAGEBOX ("Could not update favourites")
```

BtDeleteClick

TRY

```
SELECT UserID FROM Users WHERE Username = IbUsername.Text  
UserID ← EXECUTESCALAR  
DELETE FROM UserFavourites WHERE  
UserID= UserID AND AnimeTitle = tbAnimeTitle.Text  
EXECUTENONQUERY  
MESSAGEBOX ("Deleted from Favourites!")
```

CATCH

```
MESSAGEBOX ("Could not delete from favourites!")
```

Form: FmUserWatchList

In this form the user can make changes to their watchlist ranging from updating their ratings, updating the episodes they have watched, updating the watchstatus and deleting animes off their watchlist. Like the admin form, the user will see a grid table that will show them their favourites.

FmUserWatchList_Load

```
IbUsername.Text ← Username
```

```
SELECT UserID FROM Users WHERE Username = IbUsername.Text  
UserID ← EXECUTESCALAR  
SELECT * FROM UserWatchList WHERE UserID= UserID  
DataAdapter ← new DataAdapter  
DataTable ← new DataTable  
DataAdapter.Fill(DataTable)  
UseWatchListGrid.DataSource ← DataTable  
UserWatchListGrid.Show  
AutoFillSearch()
```

AutoFillSearch

```
SELECT UserID FROM Users WHERE Username = IbUsername.Text  
UserID ← EXECUTESCALAR  
SELECT AnimeTitle FROM UserWatchList WHERE UserID= UserID  
AutoFill ← new AutoCompleteStringCollection  
DataReader ← EXECUTEREADER
```

```

IF(DataReader.HasRows) THEN
    WHILE(DataReader.Read)
        AutoFill.Add(DataReader[“AnimeTitle”])
    END WHILE
END IF
ELSE
    MESSAGEBOX(“AnimeTitle not found!”)
END ELSE

```

UserWatchListGrid CellContentClick

```

IF (Cell.RowIndex >= 0) THEN
    DataGridViewRow ← UserFavouritesGrid.Rows[Cell.RowIndex]
    TbUserID.Text ← DataGridViewRow.Cells[“UserID”]
    TbAnimelD.Text ← DataGridViewRow.Cells[“AnimeID”]
    TbAnimeTitle.Text ← DataGridViewRow.Cells[“AnimeTitle”]
    CbWatchStatus.Text ← DataGridViewRow.Cells[“WatchStatus”]
    TbEpisodesWatched.Text ← DataGridViewRow.Cells[“EpisodesWatched”]
    CbRatings.Text ← DataGridViewRow.Cells[“UserID”]

    SELECT AnimePictures FROM Animes WHERE AnimeTitle = tbAnimeTitle.Text
    AnimePic ← EXECUTESCALAR
    PbAnimePicture.LoadAsync(AnimePic)
END IF

```

tbSearch KeyPress

```

SELECT UserID FROM Users WHERE Username = lbUsername.Text
UserID ← EXECUTESCALAR
IF(tbSearch.Text != “”//if its not empty) THEN
    DataAdapter ← new DataAdapter
    DataSet ← new DataSet
    DataView ← new DataView
    SELECT * FROM UserWatchList WHERE AnimeTitle LIKE tbAnimeTitle AND UserID

```

```

LIKE UserID

Command ← new Command (SELECT QUERY, Conn)

DataAdapter ← new DataAdapter(Command)

DataAdapter.Fill(DataSet)

DataView ← new DataView(DataSet.Tables[0])

UserWatchListGrid.DataSource ← DataView

END IF

ELSE IF (tbSearch.Text == " " // if its empty) THEN

    SELECT * FROM UserWatchList WHERE UserID= UserID

    DataAdapter ← new DataAdapter

    DataTable ← new DataTable

    DataAdapter.Fill(DataTable)

    UserWatchListGrid.DataSource ← DataTable

END ELSE IF

```

BtUpdate.Click

```

TRY

    SELECT UserID FROM Users WHERE Username = lbUsername.Text

    UserID ← EXECUTESCALAR

    UPDATE UserFavourites SET WatchStatus = cbWatchStatus.Text, EpisodesWatched =
    tbEpisodesWatched.Text, Ratings = cbRatings.Text WHERE UserID = UserID AND
    AnimeTitle= tbAnimeTitle.Text

    EXECUTENONQUERY

    MESSAGEBOX ("Watchlist Updated!")

CATCH

    MESSAGEBOX ("Could not update Watchlist")

```

tbEpisodesWatch Validating

```

HasSymbols ← NEW REGEX (all symbols here)

HasNumbers ← NEW REGEX ([0-9])

IF ( !HasNumbers.IsMatch(tbEpisodesWatched.Text) THEN

    MESSAGEBOX ("Please enter a number!")

END IF

```

```

ELSE IF ( HasSymbols.IsMatch(tbEpisodesWatched.Text) THEN
    MESSAGEBOX ("Please enter a number!")
END ELSE IF
ELSE
    SELECT Episodes FROM Animes WHERE AnimeTitle = tbAnimeTitle.Text
    Episodes ← EXECUTESCALAR
    EpisodesEntered ← tbEpisodesWatched.Text
    IF( EpisodesEntered > Episodes) THEN
        MESSAGEBOX("Out of range!")
        tbEpisodesWatched.Text ← Episodes
    END IF
    ELSE IF (EpisodesEntered < 0) THEN
        MESSAGEBOX("Out of range!")
        tbEpisodesWatched.Text ← "0"
    END ELSE IF
END ELSE

```

CbWatchStatus_SelectedIndexChanged

```

SELECT Episodes FROM Animes WHERE AnimeTitle = tbAnimeTitle.Text
Episodes ← EXECUTESCALAR
IF (CbWatchStatus.Text == "Completed") THEN
    tbEpisodesWatched.Text ← Episodes
END IF
ELSE IF (CbWatchStatus.Text == "Not Yet Watched") THEN
    tbEpisodesWatched.Text ← "0"
END ELSE IF

```

BtDelete.Click

BtDeleteClick

TRY

```

SELECT UserID FROM Users WHERE Username = lbUsername.Text
UserID ← EXECUTESCALAR
DELETE FROM UserWatchList WHERE
UserID= UserID AND AnimeTitle = tbAnimeTitle.Text

```

```

EXECUTENONQUERY
MESSAGEBOX ("Deleted from Watchlist!")

CATCH
MESSAGEBOX ("Could not delete from Watchlist!")

```

Form: FmUserStats

In this Form the user can view their statistics such as their average ratings, their total ratings, the number of animes they have completed watching, the number of animes they're currently watching, the number of animes they have yet to watch, the number of animes they have dropped, total episodes watched and the hours that the user spent watching anime. This will be done in the load procedure.

FmUserStats Load

```

TRY
  IbUsername.Text ← Username
  SELECT UserID FROM Users WHERE Username = IbUsername.Text
  UserID ← EXECUTESCALAR
  SELECT COUNT (*) FROM UserFavourites WHERE UserID = UserID
  TotalRatingsFromFavourites ← EXECUTESCALAR
  SELECT COUNT (*) FROM UserWatchList WHERE UserID = UserID
  TotalRatingsFromWatchList ← EXECUTESCALAR
  TotalRatings ← TotalRatingsFromFavourites + TotalRatingsFromWatchList
  DbTotalRatings.Text ← TotalRatings
  SELECT AVG (UserFavourites.Ratings) + AVG(UserWatchList.Ratings) FROM
  UserFavourites, UserWatchList WHERE UserFavourites.UserID = UserID
  AND UserWatchList.UserID = UserID
  RatingsAdded ← EXECUTESCALAR
  TrueAverageRatings ← RatingsAdded / 2
  DbAverageRatings.Text ← ROUND(TrueAverageRatings, TO 1DP)
  SELECT COUNT (*) FROM UserWatchList WHERE WatchStatus = 'Completed'
  AND UserID = UserID
  Completed ← EXECUTESCALAR
  DbAnimesCompleted.Text ← Completed
  SELECT COUNT (*) FROM UserWatchList WHERE WatchStatus = 'Currently Watching'
  AND UserID = UserID

```

```

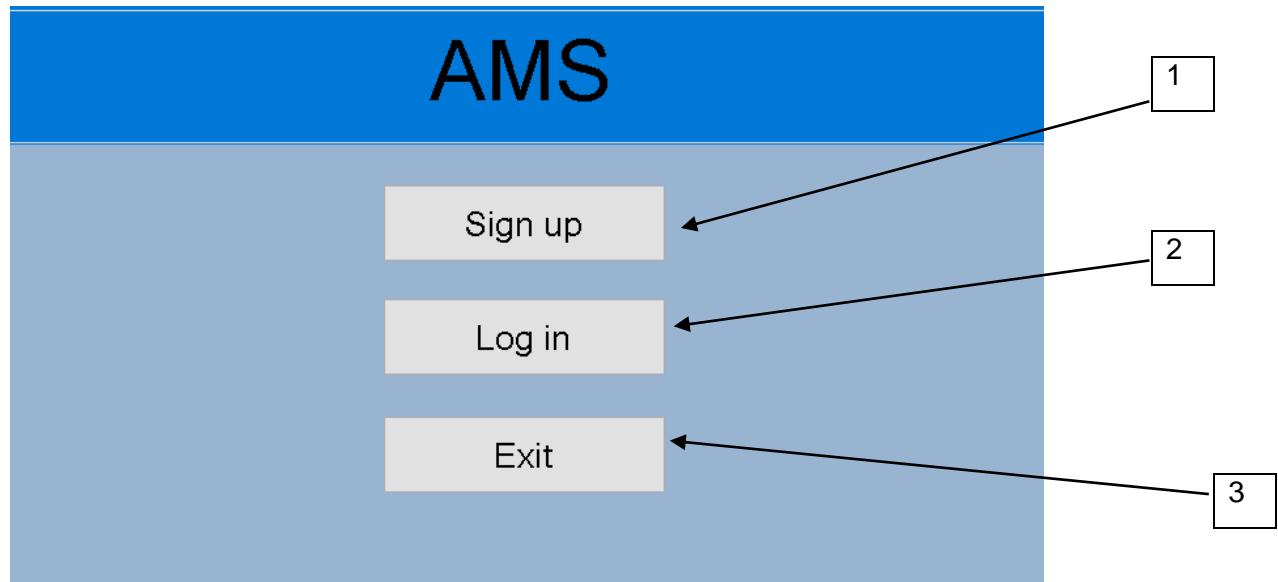
CurrentlyWatching ← EXECUTESCALAR
DbAnimesCurrentlyWatching.Text ← CurrentlyWatching
SELECT COUNT (*) FROM UserWatchList WHERE WatchStatus = 'Not Yet Watched'
AND UserID = UserID
NotYetWatched ← EXECUTESCALAR
DbAnimesYetToWatch.Text ← NotYetWatched
SELECT COUNT (*) FROM UserWatchList WHERE WatchStatus = 'Dropped'
AND UserID = UserID
Dropped ← EXECUTESCALAR
DbAnimesDropped.Text ← CurrentlyWatching
SELECT SUM(EpisodesWatched) FROM UserWatchList WHERE UserID = UserID
EpisodesWatched ← EXECUTESCALAR
DbTotalEpisodesWatched.Text ← EpisodesWatched
AnimeEpLength ← EpisodesWatched * 24
Hours ← AnimeEpLength / 60
DbHourSpent.Text ← Hours

CATCH
MESSAGEBOX ("Could not load stats!")

```

User interface design

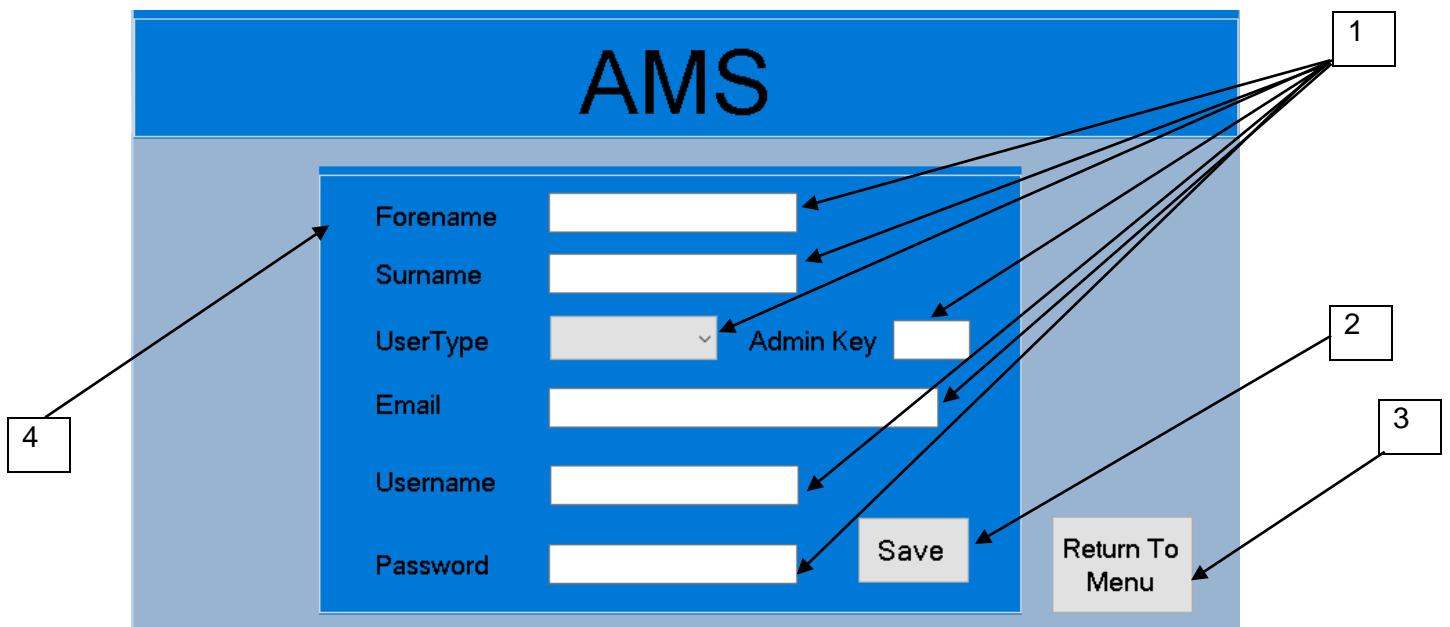
Form: FmMenu



1. **BtSignUp**. This button is used to access the Sign-Up form 'FmSignUp'
2. **BtLogin**. This button is used to access the Log In form 'FmLogin'. The user would need to create an account in 'FmSignUp' to log in
3. **BtExit**. This button is used to exit the the program from the main menu, this button will close the program completely

Form: FmSignUp

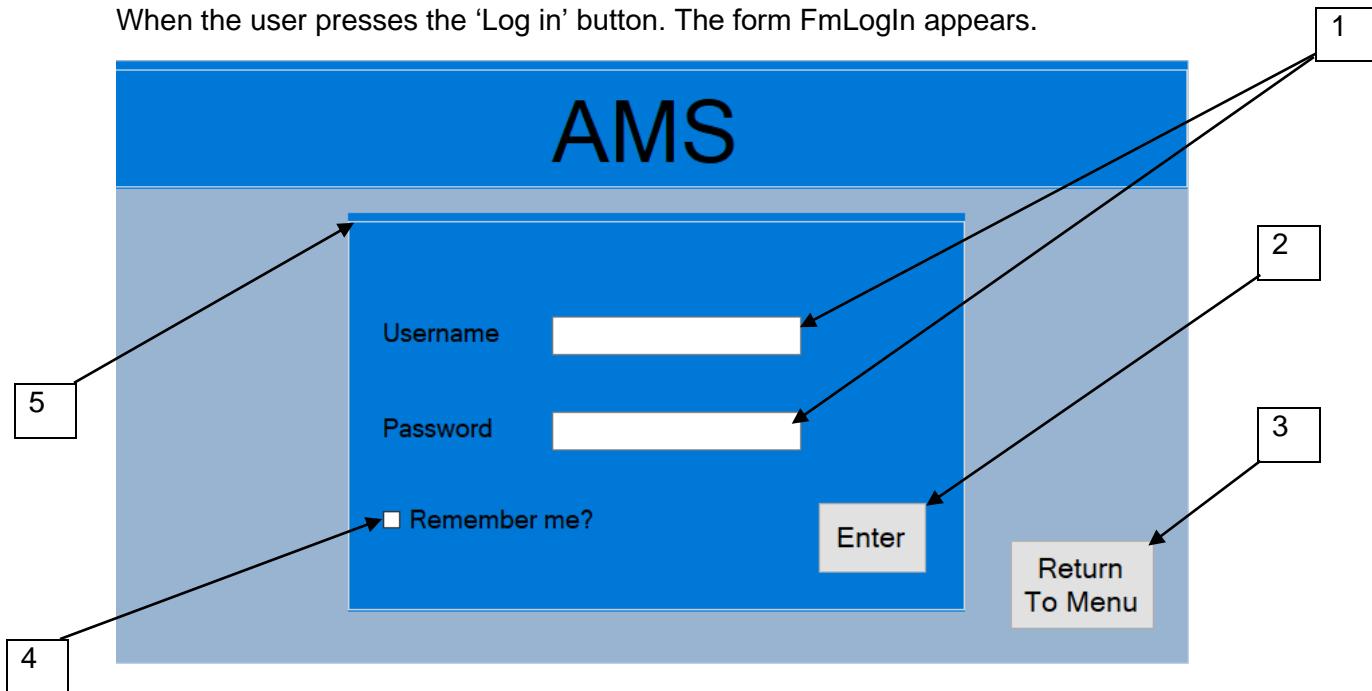
When the user presses the 'Sign Up' button. The form FmSignUp appears.



1. **TbForeName, TbSurname, CbUserType, TbAdminKey, TbEmail, TbUsername, TbPassword**. All these text fields are used to create a new account for the user. Admin Key will appear invisible upon initial load up of the form. The user would need to select the option "Administrator" for it to appear.
2. **BtSave**. This button is used to save the details the user has entered, into the table 'Users'. When this button is pressed, the program is used to validate all of the fields and if they're valid, it will be inserted into the database. The password is hashed upon entry.
3. **BtReturn**. This button is used to return the user to the main menu where they can log in.
4. **GbCreateAccount**. This groupbox is the container used for the user to create an account. It contains all the necessary inputs for the user.

Form:FmLogin

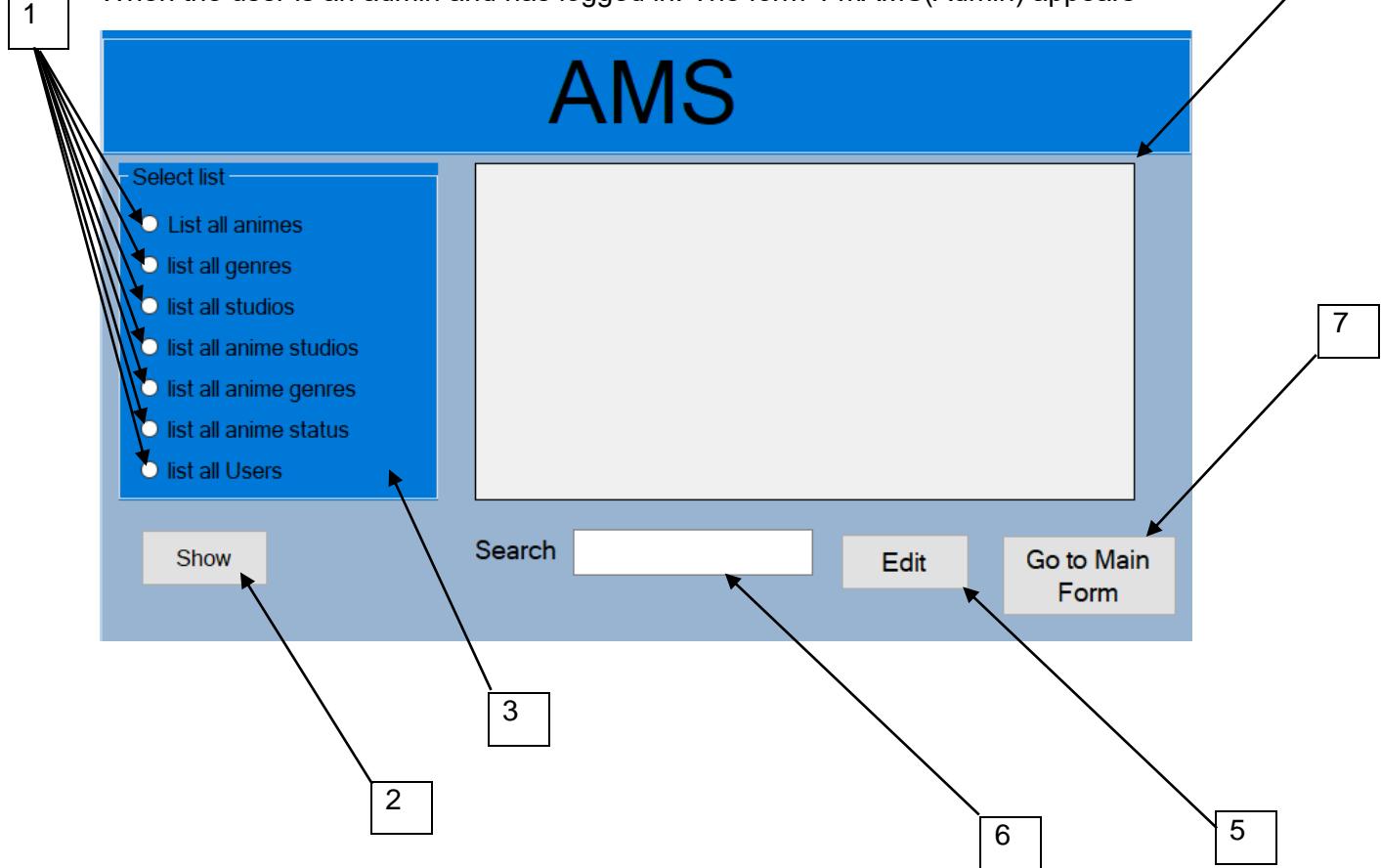
When the user presses the 'Log in' button. The form FmLogin appears.



1. **TbUsername, TbPword.** These text fields are used to login in to the form 'FmAMS' or 'FmAMS(Admin)'.
2. **BtEnter.** This button is used to submit the user's log in, when its clicked, the program checks if the user's username and password exists. If the user's credentials exist then the program will log them in. If the user is an administrator they will be sent to the form 'FmAMS(Admin)' otherwise the user will be sent to the form 'FmAMS'.
3. **BtReturn.** This button is used to return the user to the form 'FmMenu'
4. **CbRemember.** This checkbox is used to remember the credentials of the user upon next log in. If the user where to close the whole program and checked this box at this form, the next time this form loads up the user's username and password will be automatically loaded into those text fields
5. **GbUserLogin.** This groupbox is the container used for the user to log in to their accounts. It contains all the necessary inputs for the user.

Form: FmAMS(Admin)

When the user is an admin and has logged in. The form 'FmAMS(Admin)' appears



1. **RbAnimes, RbGenres, RbStudios, RbAnimeGenres, RbAnimeStudios, RbAnimeStatus and RbUsers.** These radio buttons allow the user to see the tables in the database correspondent to the radiobutton.

2. **BtShow.** This button is used in conjunction with the radio buttons to show the tables from the database onto the grid.

3. **GbSelect.** This groupbox is the container used for the radio buttons. It contains all the necessary buttons for the user.

4. **Grid.** This datagridviewer outputs the tables from the database when the user clicks on one of the radio buttons and clicks on the button "Show". This grid is also clickable. Clicking a cell on this grid will bring up all the information of that row into an edit form correspondent to the radio buttons.

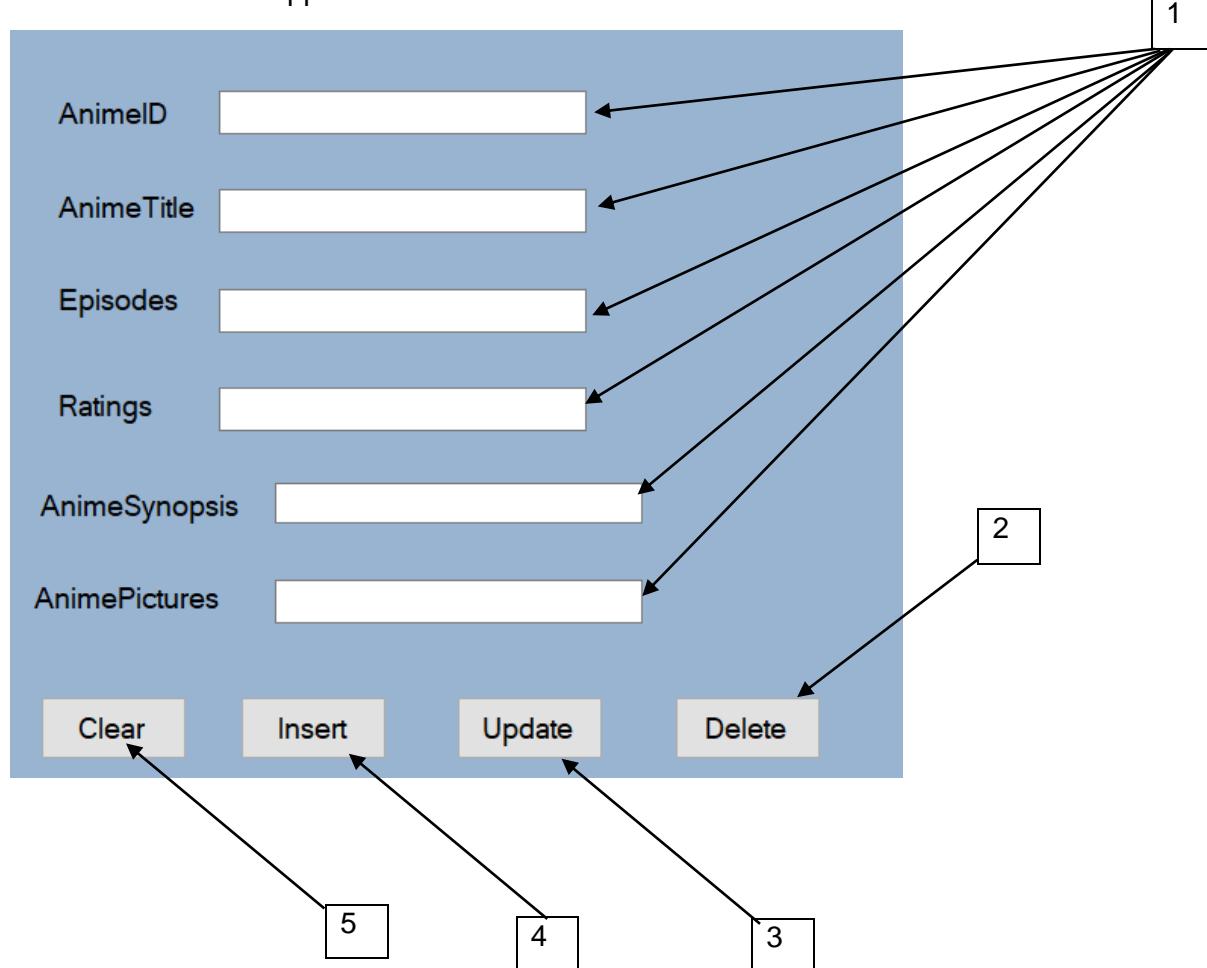
5. **BtEdit.** This button will bring up a form correspondent to the radiobuttons so that the user can add anything to the tables in the database.

6. **TbSearch.** This textbox is used for the user to search through the tables correspondent to the radio buttons.

7. **BtMainForm.** This button allows the user to go on to the form 'FmAMS'.

Form: FmEditAnimes

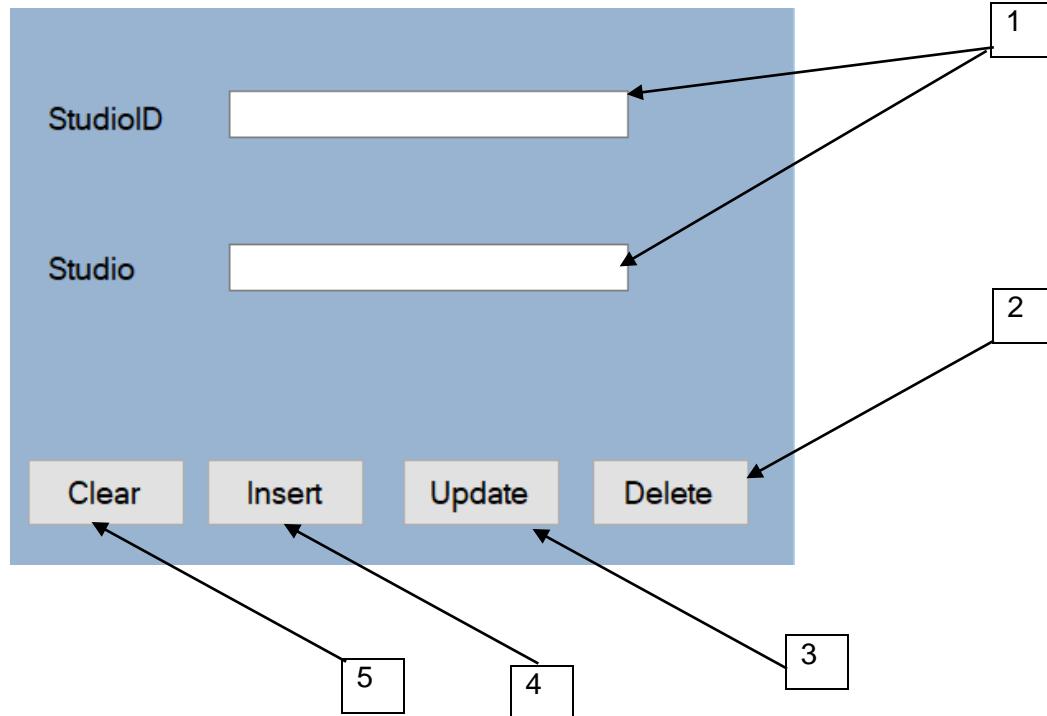
When the radio button 'rbAnimes' is checked, and the Edit button is clicked The form 'FmEditAnimes' will appear



1. **TbAnimelD, TbAnimeTitle, TbEpisodes, TbRatings, TbAnimeSynopsis, TbAnimePictures.** These text fields allow the user to type in all the specific information to alter the table 'Animes'
2. **BtDelete.** Clicking on this button will delete the Anime from the table 'Animes'.
3. **BtUpdate.** Clicking on this button will update the information about the anime in the table 'Animes'
4. **BtInsert.** Clicking on this button will add the anime to the table 'Animes'
5. **BtClear.** Clicking on this button will clear all the text fields in this form.

Form: FmEditStudios

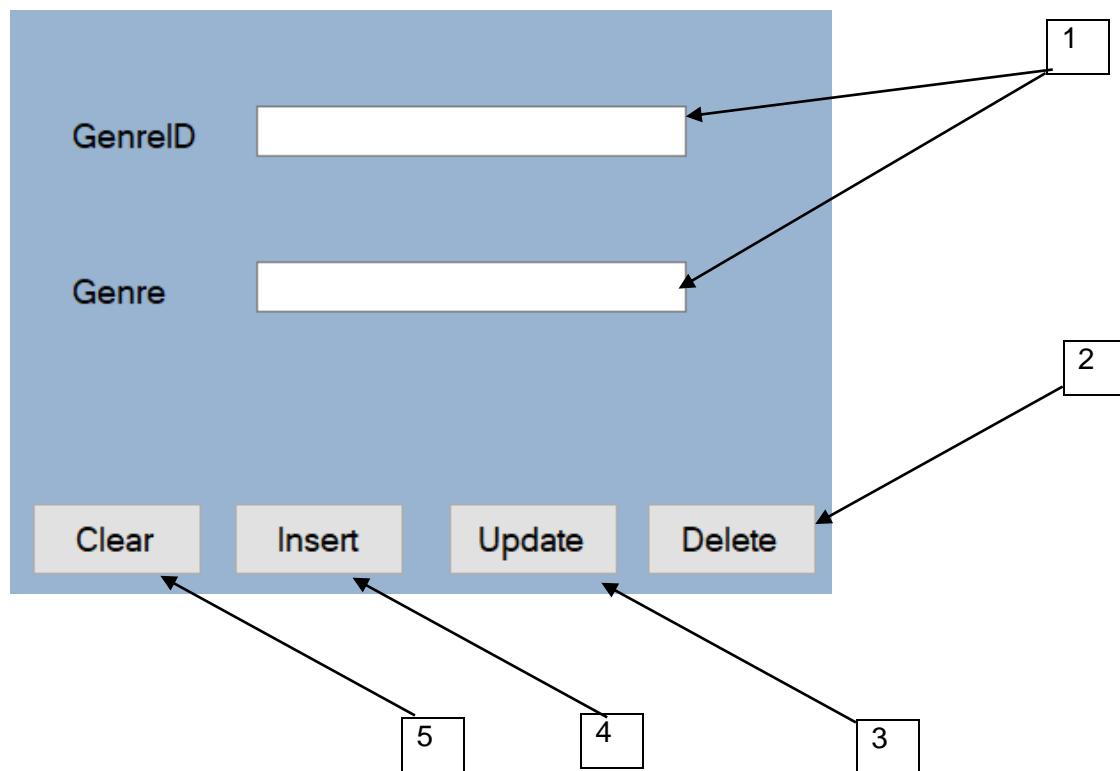
When the radio button 'rbStudios' is checked and the edit button is clicked, the form 'FmEditStudios' will appear.



1. **TbStudioID, TbStudio.** These text fields allow the user to type in all the specific information to alter the table 'Studios'.
2. **BtDelete.** Clicking on this button will delete the Studio from the table 'Studios'.
3. **BtUpdate.** Clicking on this button will update the information about the Studio in the table 'Studios'.
4. **BtInsert.** Clicking on this button will add the Studio to the table 'Studios'.
5. **BtClear.** Clicking on this button will clear all the text fields in this form.

Form: FmEditGenres

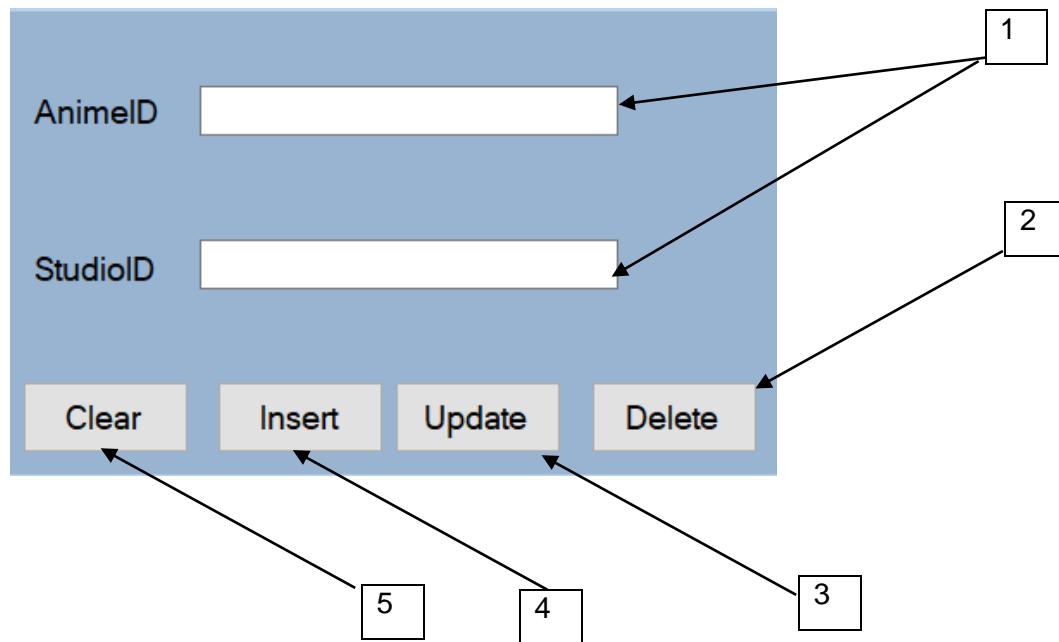
When the radio button 'rbGenres' is checked and the edit button is clicked, the form 'FmEditGenres' will appear.



1. **TbGenreID, TbGenre.** These text fields allow the user to type in all the specific information to alter the table 'Genres'.
2. **BtDelete.** Clicking on this button will delete the Genre from the table 'Genres'.
3. **BtUpdate.** Clicking on this button will update the information about the Genre in the table 'Genres'.
4. **BtInsert.** Clicking on this button will add the Genre to the table 'Genres'
5. **BtClear.** Clicking on this button will clear all the text fields in this form.

Form: FmEditAnimeStudios

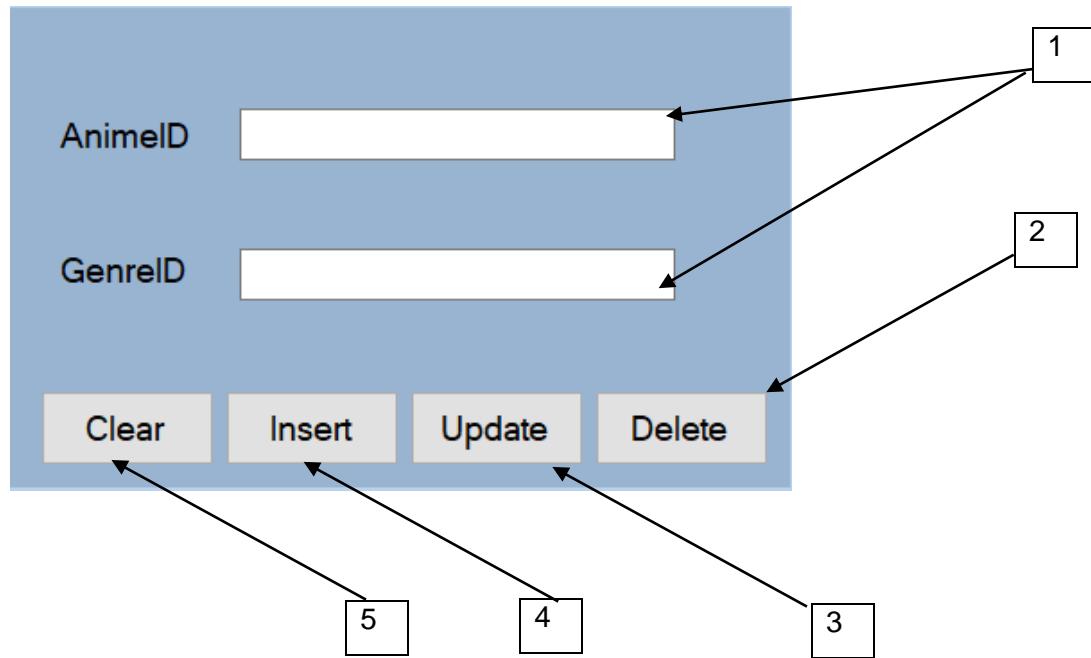
When the radio button 'rbAnimeStudios' is checked and the edit button is clicked, the form 'FmEditAnimeStudios' will appear.



1. **TbAnimelD, TbStudioID.** These text fields allow the user to type in all the specific information to alter the table 'AnimeStudios'.
2. **BtDelete.** Clicking on this button will delete the AnimeStudio from the table 'AnimeStudios'.
3. **BtUpdate.** Clicking on this button will update the information about the AnimeStudio in the table 'AnimeStudios'.
4. **BtInsert.** Clicking on this button will add the AnimeStudio to the table 'AnimeStudios'
5. **BtClear.** Clicking on this button will clear all the text fields in this form.

Form: FmEditAnimeGenres

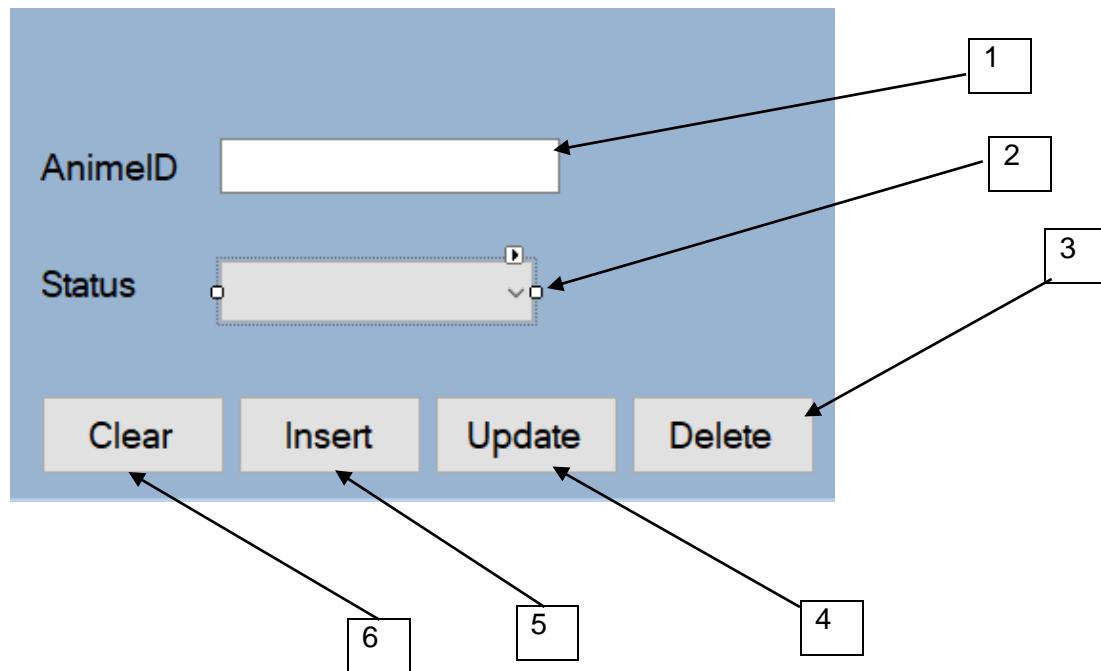
When the radio button 'rbAnimeGenres' is checked and the edit button is clicked, the form 'FmEditAnimeGenres' will appear.



1. **TbAnimelD, TbGenreID.** These text fields allow the user to type in all the specific information to alter the table 'AnimeGenres'.
2. **BtDelete.** Clicking on this button will delete the AnimeGenre from the table 'AnimeGenres'
3. **BtUpdate.** Clicking on this button will update the information about the AnimeGenre in the table 'AnimeGenres'.
4. **BtInsert.** Clicking on this button will add the AnimeGenre to the table 'AnimeGenres'
5. **BtClear.** Clicking on this button will clear all the text fields in this form.

Form: FmAnimeStatus

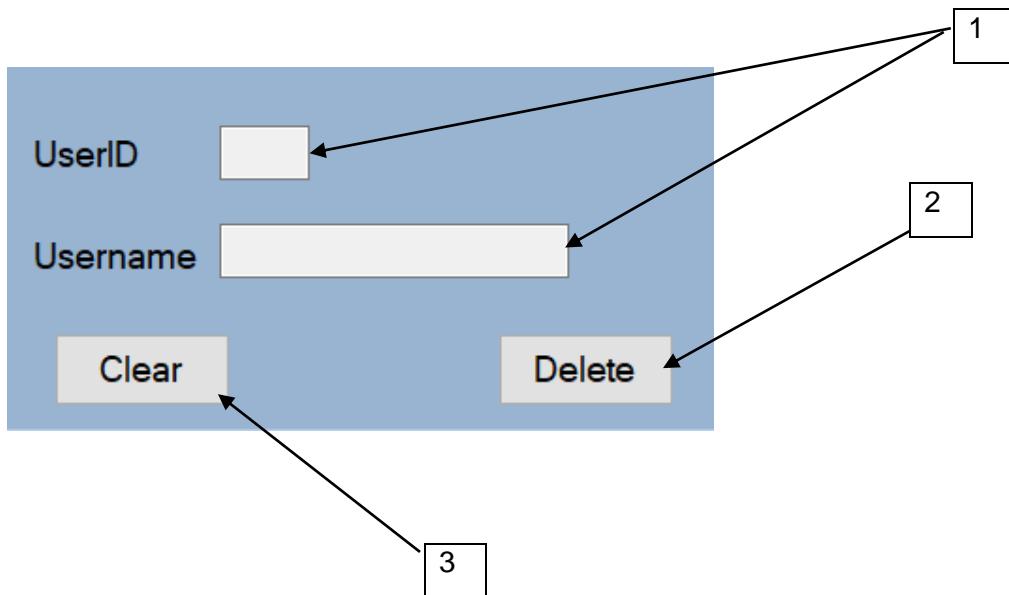
When the radio button 'rbAnimeStatus' is checked and the edit button is clicked, the form 'FmEditAnimeStatus' will appear.



1. **TbAnimelD.** This textbox will allow the user to type in any AnimelD in order to alter the table 'AnimeStatus'.
2. **CbStatus.** This combobox will allow the user to select the status of the Anime in order to alter the table 'AnimeStatus'.
3. **BtDelete.** Clicking on this button will delete the AnimeStatus from the table 'AnimeStatus'
4. **BtUpdate.** Clicking on this button will update the information about the Anime in the table 'AnimeStatus'
5. **BtInsert.** Clicking on this button will add the AnimeStatus to the table 'AnimeStatus'
6. **BtClear.** Clicking on this button will clear all the text fields in this form.

Form: FmDeleteUser

When the radio button 'rbUsers' is checked and the edit button is clicked, the form 'FmDeleteUser' will appear.

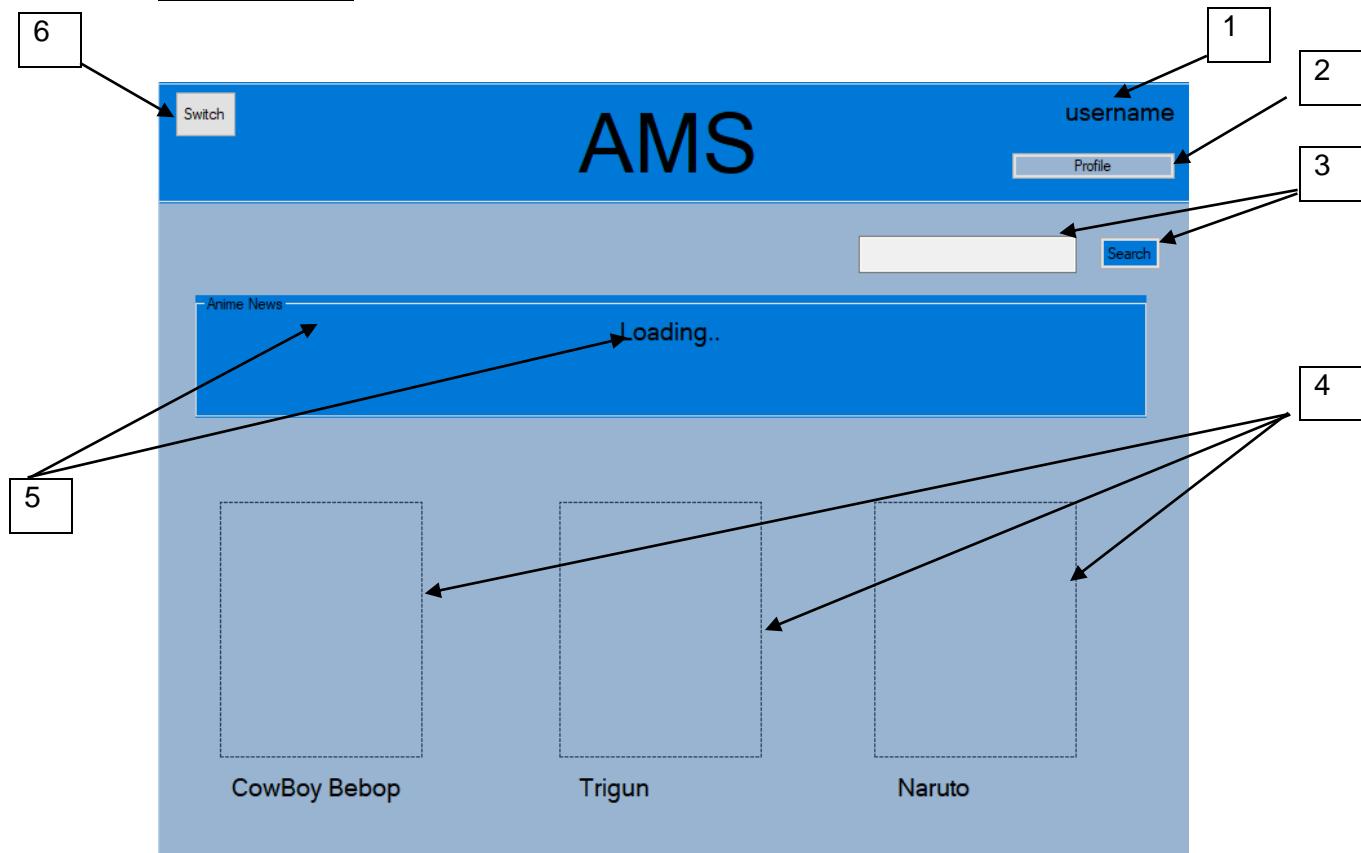


1. **TbUserID, TbUsername.** These text fields allow the user to type in all the specific information to alter the table 'Users', 'UserFavourites' and 'UserWatchList'.
2. **BtDelete.** Clicking on this button will delete the User from the tables 'Users', 'UserFavourites' and 'UserWatchList'
3. **BtClear.** Clicking on this button will clear all the text fields in this form.

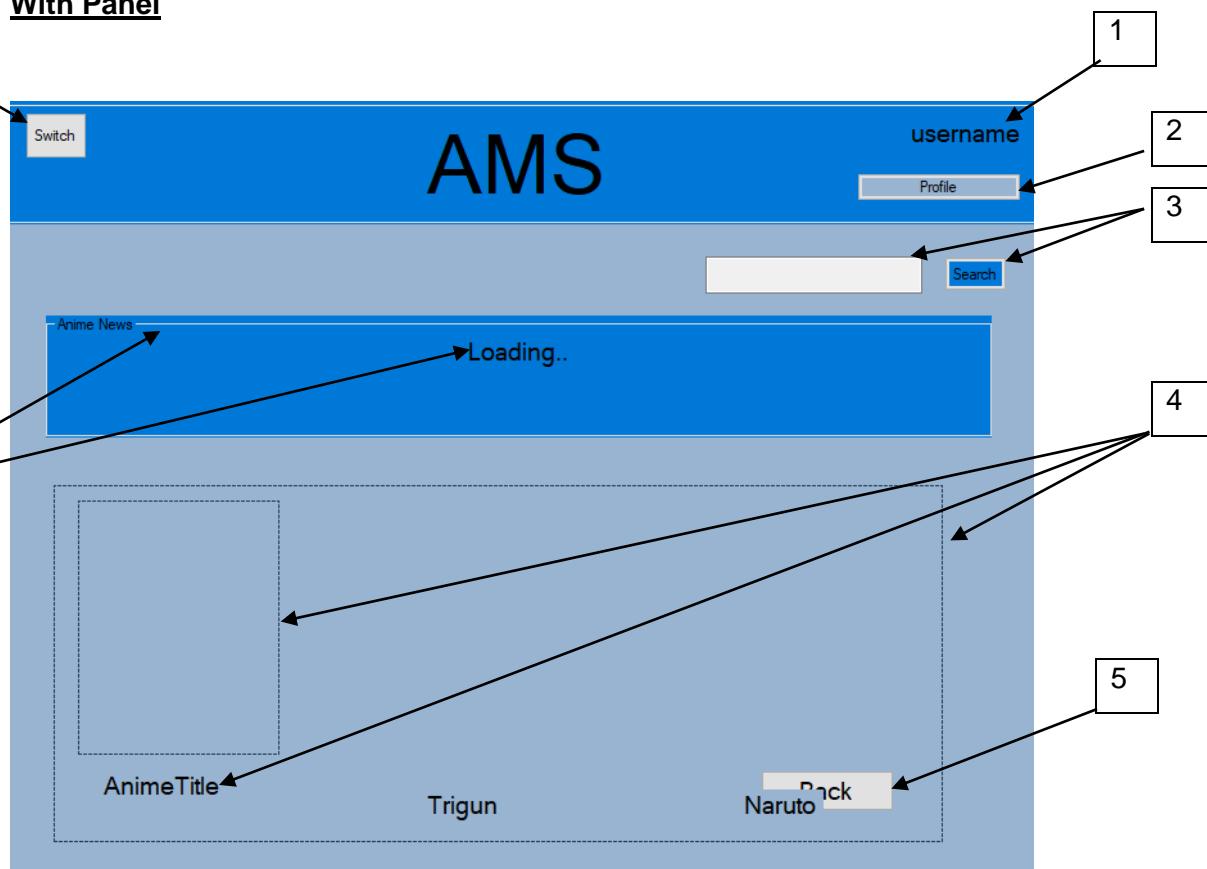
Form: FmAMS

When the user has logged in successfully. The form 'FmAMS' appears

Without Panel



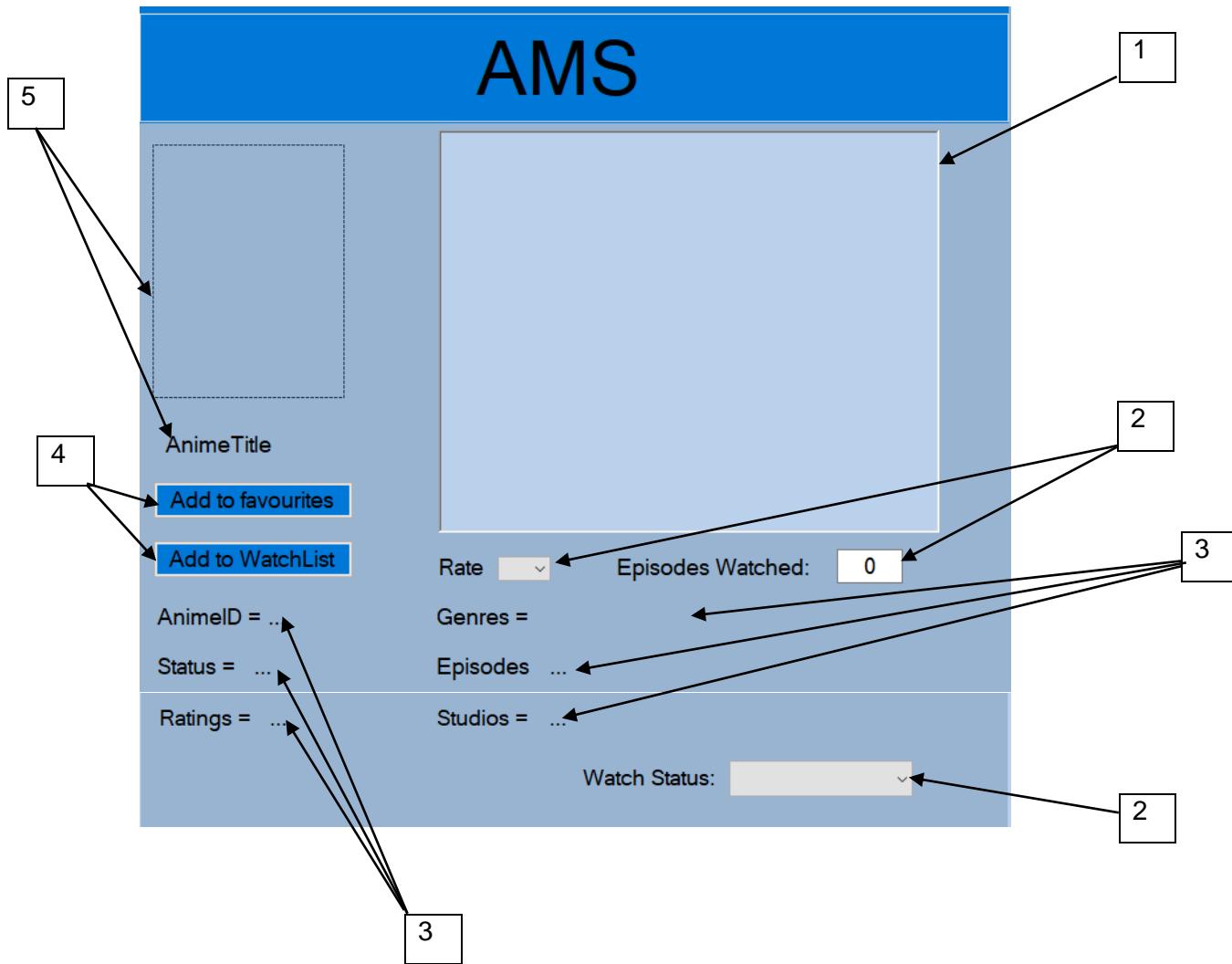
- 1. LbUsername.** This label will show the username of the user that is logged into the system by making use of the global variable defined in 'FmLogIn'
- 2. BtProfile.** This button will allow the user to see their profile which will show them all of their information.
- 3. TbSearch, BtSearch.** The textbox will allow the user to search animes. The button will showcase the result of the search.
- 4. PbAnimePicture1, PbAnimePicture2, PbAnimePicture3.** The pictureboxes will show the pictures of animes that corresponds to the labels below them
- 5. GbAnimeNews, TbNews.** The group box is the container of all news that will be shown through the textbox.
- 6. BtSwitch.** This button will allow the user to change the whole colour scheme of this form to DarkMode.

With Panel

1. **LbUsername.** This label will show the username of the user that is logged into the system by making use of the global variable defined in 'FmLogIn'
2. **BtProfile.** This button will take the user to the form 'FmProfile' which will show them all of their information.
3. **TbSearch, BtSearch.** The textbox will allow the user to search animes. The button will showcase the result of the search.
4. **PbAnimePicture4,LbAnimeTitle,PaAnimePanel.** When the user presses the search button after typing the anime they want to find, the panel and everything within it will turn visible to the user and the 3 pictureboxes and their labels will turn invisible and show the user the picture of the anime they have searched through the picturebox and the name of the anime beneath it through the label.
5. **BtBack.** When the user presses the back button, the panel and everything that is contained in the panel will be invisible and thus return to the state of what the form was like when it was first loaded.
6. **GbAnimeNews, TbNews.** The group box is the container of all news that will be shown through the textbox.
7. **BtSwitch.** This button will allow the user to change the whole colour scheme of this form to DarkMode.

Form: FmAnimeInformation

This form loads when any picturebox is clicked in 'FmAMS'



1. RtbAnimeSynopsis. This rich text box will contain the synopsis of the anime the user has searched for and the picturebox that was clicked on.

2. CbRate, CbWatchStatus, TbEpisodesWatched. The user can input information into all of these fields. The comboBoxes will contain a list of things the user can pick from. For the comboBox cbRate the user can pick a number between 1-10 and for cbWatchStatus the user can select from: "Completed", "Currently Watching", "Not yet Watched" and "Dropped". The user can enter the number of episodes watched into the textbox tbEpisodesWatched.

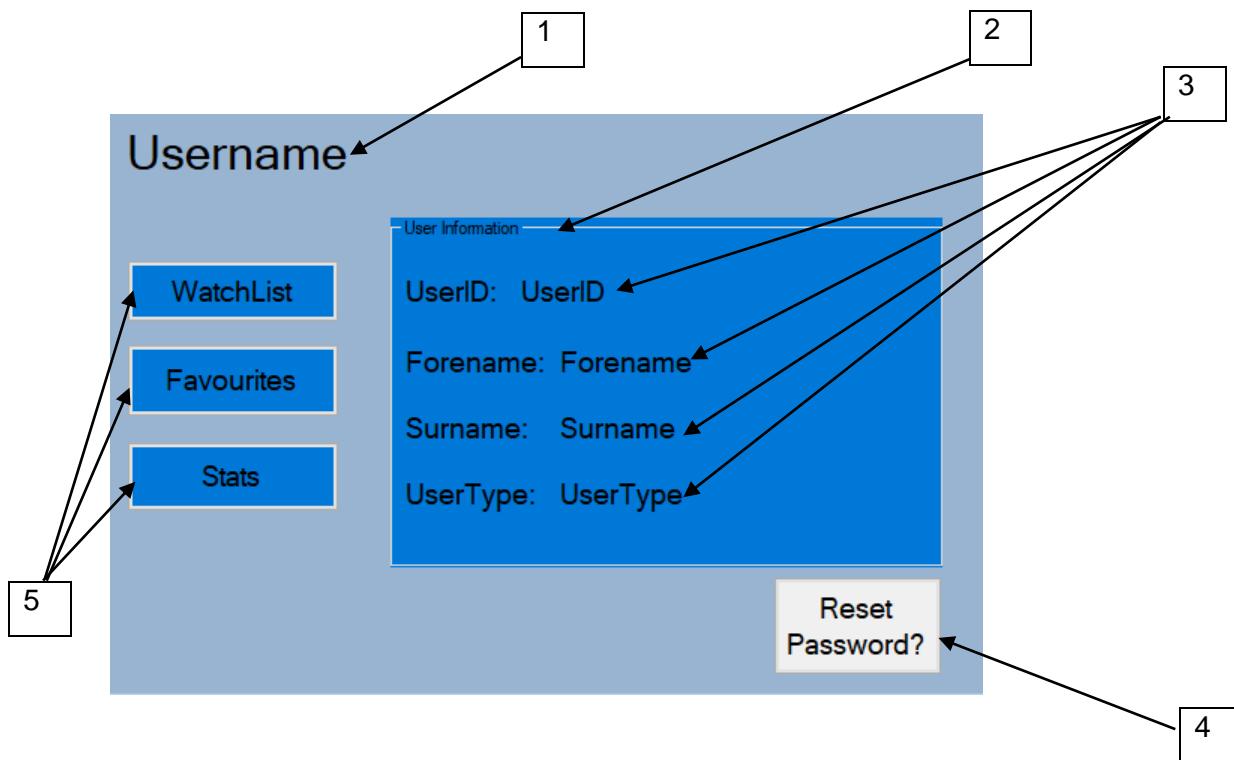
3. DbAnimeID, DbStatus, DbRatings, TbGenres, DbEpisodes, DbStudios. All of these fields will contain information about the anime that will be loaded in through the load procedure.

4. BtFavourites, BtWatchList. These buttons will allow the user to add the anime to their favourites or to their Watchlist

5. PbAnimePicture, IbAnimeTitle. The picturebox and the label will contain the picture and the name of the anime that will be loaded in through the load procedure.

Form: FmProfile

If the user clicks on the 'profile' button on the form 'FmAMS' then the form 'FmProfile' will load.



1. **LbUsername.** This label will show the username of the user that is logged into the system by making use of the global variable defined in 'FmLogIn'.

2. **GbUserInformation.** This groupbox will contain user's information in the corresponding fields within the group box.

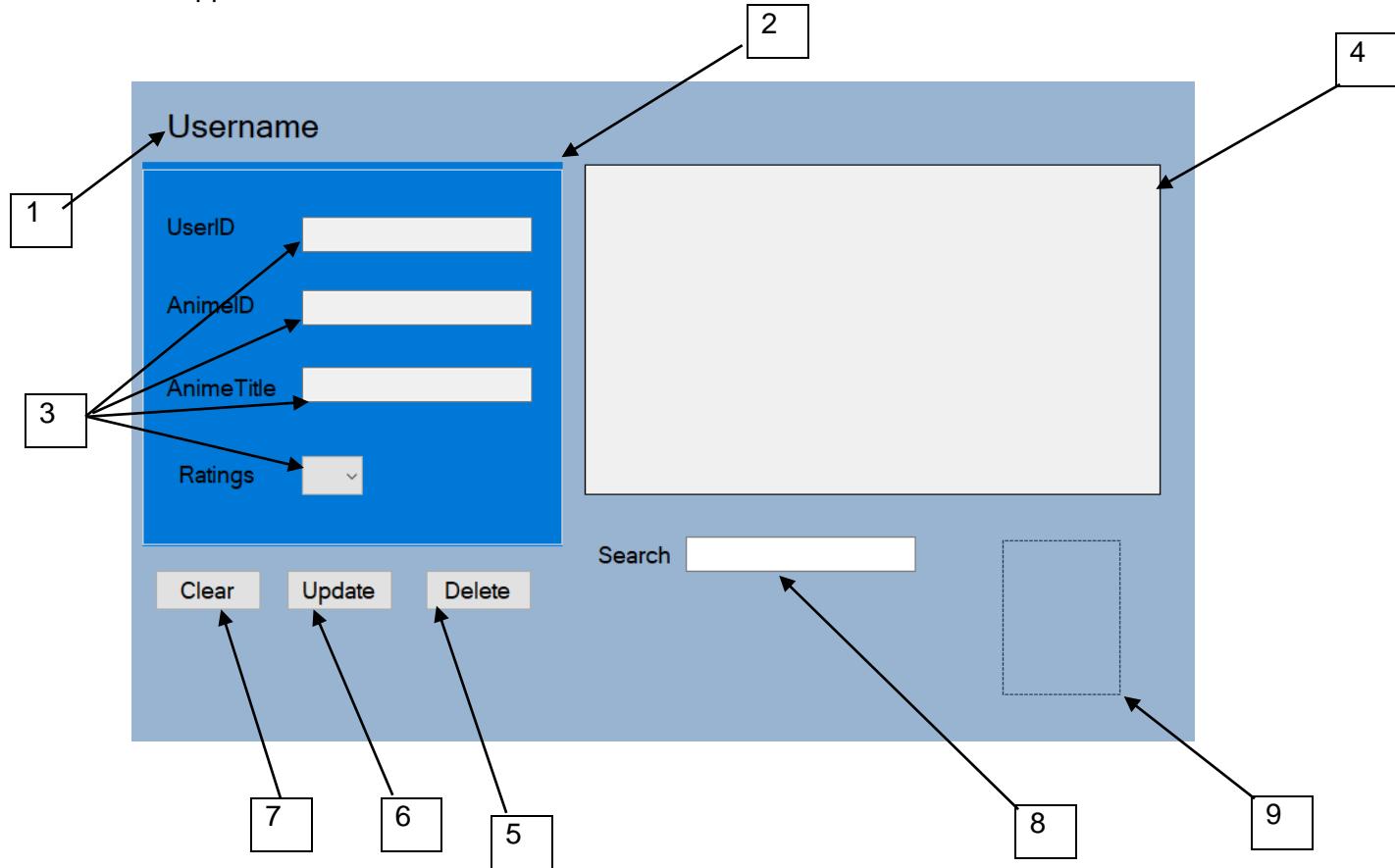
3. **DbUserID, DbForename, DbSurname, DbUserType.** These labels will contain the user's information that will be loaded in with the load procedure for this form

4. **BtReset.** This button will take the user to the form 'FmReset' so that they can reset their password.

5. **BtFavourites, BtWatchList, BtStats.** These buttons will take the user to the forms 'FmUserWatchList', 'FmUserFavourites' and 'FmUserStats' where they can view their watchlist and make changes, view their favourites and make changes and view their stats.

Form: UserFavourites

If the user clicks on the 'Favourites' button in 'FmProfile' then the form 'FmUserFavourites' will appear.



1. LbUsername. This label will show the username of the user that is logged into the system by making use of the global variable defined in 'FmLogIn'.

2. GbUserFavourites. This groupbox will contain all the fields that contain information from the grid when a cell is clicked on.

3. TbUserID, TbAnimeID, TbAnimeTitle, CbRatings. The user will be able to alter the only the combobox CbRatings. The textboxes will be ReadOnly.

4. UserFavouritesGrid. The grid will output the user's favourites list upon the load procedure for this form

5. BtDelete. Clicking on this button will delete the Anime from the table 'UserFavourites'.

6. BtUpdate. Clicking on this button will update the Anime's rating from the table 'UserFavourites'

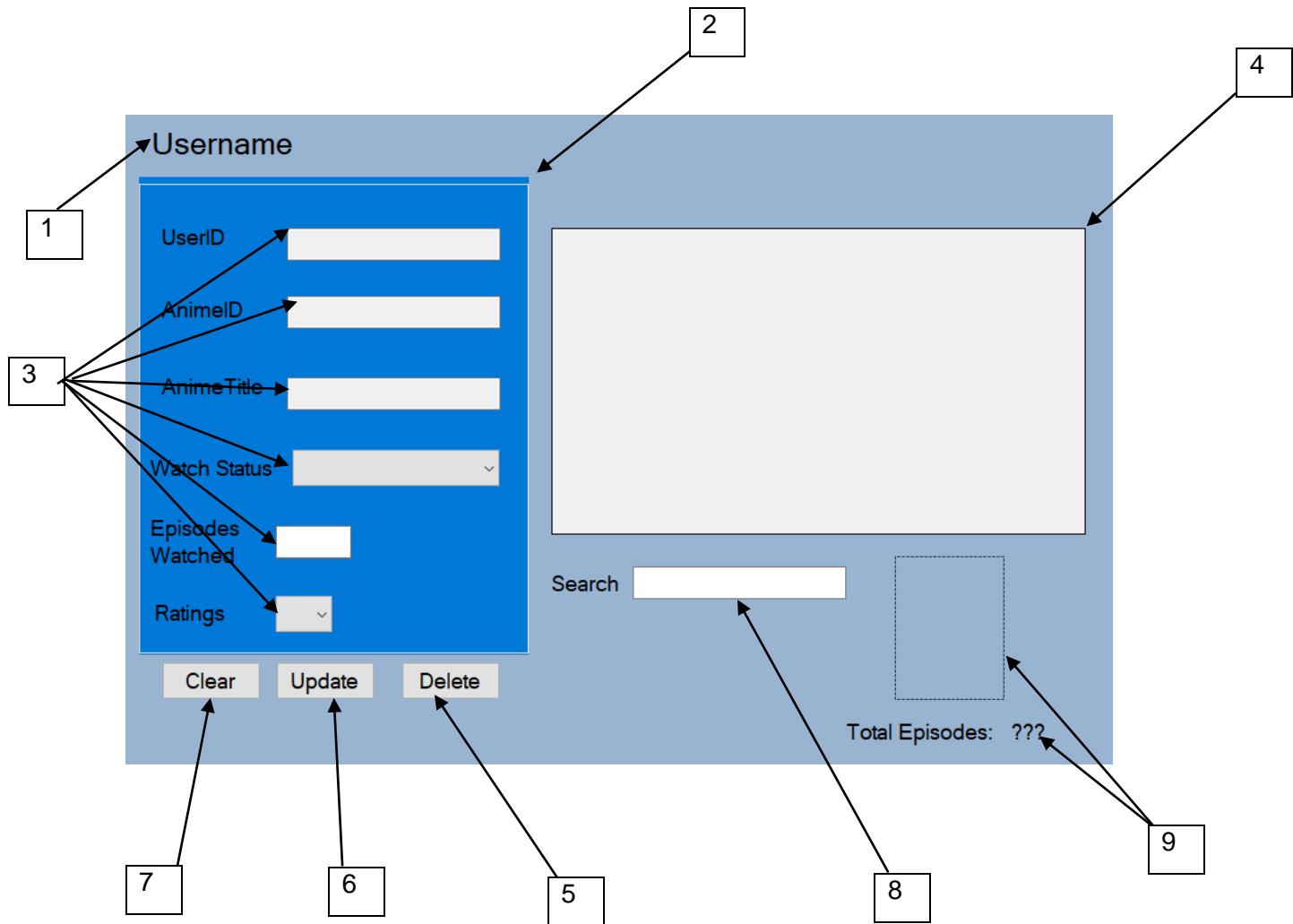
7. BtClear. Clicking on this button will clear all the fields in GbUserFavourites.

8. TbSearch. The user can search through the table by inputting an AnimeName from their favourites.

9. PbAnimePicture. If the user clicks on a cell. The entire row will be inserted into the textboxes in gbUserFavourites and the picture of the anime will be loaded into the Picturebox

Form: UserWatchList

If the user clicks on the 'WatchList' button in 'FmProfile' then the form 'FmUserWatchList' will appear.



1. LbUsername. This label will show the username of the user that is logged into the system by making use of the global variable defined in 'FmLogIn'.

2. GbUserWatchList. This groupbox will contain all the fields that contain information from the grid when a cell is clicked on.

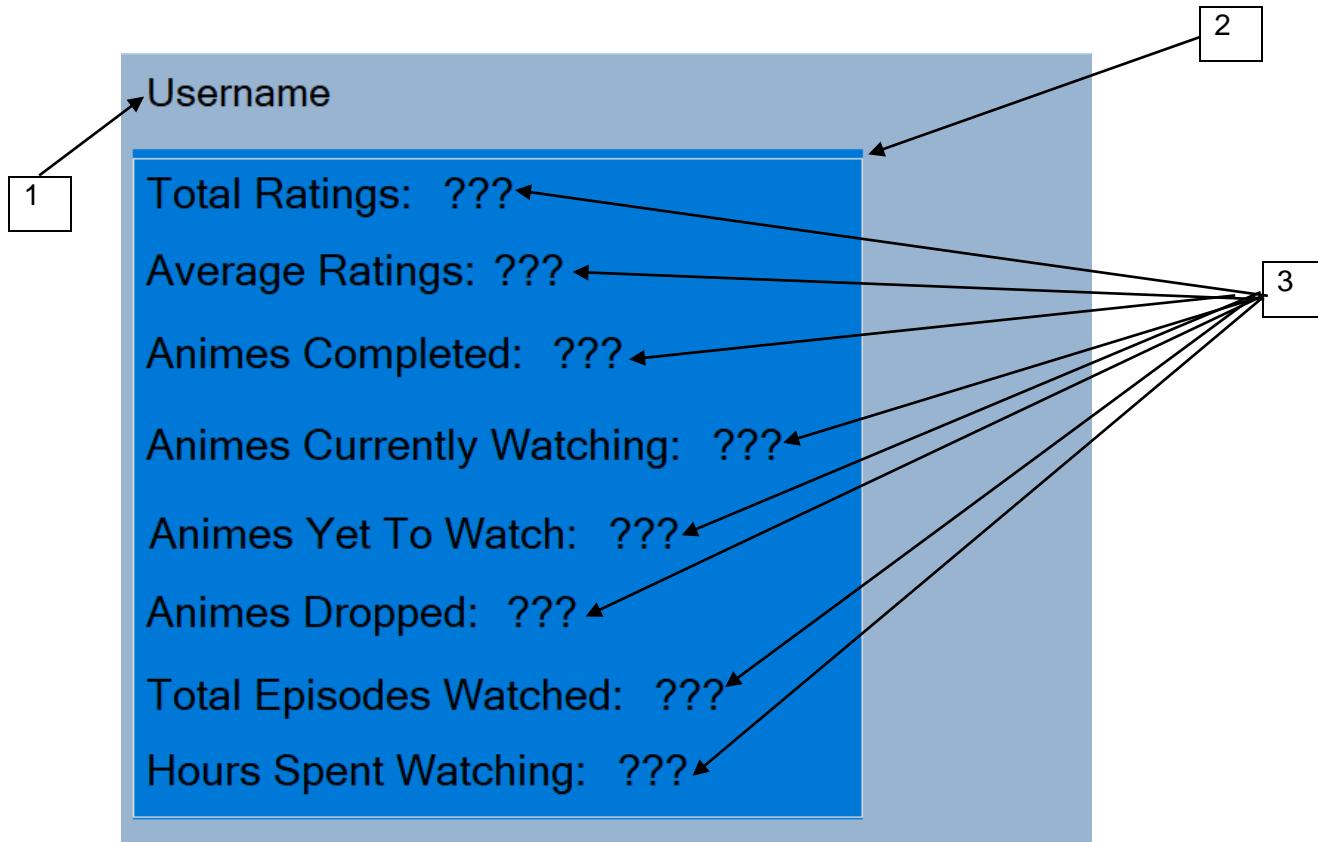
3. TbUserID, TbAnimelD, TbAnimeTitle, CbRatings, CbWatchStatus,

TbEpisodesWatched. The user will be able to alter the comboboxes CbRatings and CbWatchStatus and the textbox TbEpisodesWatched the rest of the textboxes will be ReadOnly.

4. **UserWatchListGrid.** The grid will output the user's watch list upon the load procedure for this form
5. **BtDelete.** Clicking on this button will delete the Anime from the table 'UserWatchList'.
6. **BtUpdate.** Clicking on this button will update the Anime's rating, watch status and number of episodes watched from the table 'UserWatchList'
7. **BtClear.** Clicking on this button will clear all the fields in GbUserWatchList.
8. **TbSearch.** The user can search through the table by inputting an AnimeName from their watch list.
9. **PbAnimePicture.** If the user clicks on a cell. The entire row will be inserted into the textboxes in gbUserWatchList and the picture of the anime will be loaded into the Picturebox

Form: UserStats

If the user clicks on the 'Stats' button in 'FmProfile' then the form 'FmUserStats' will appear.

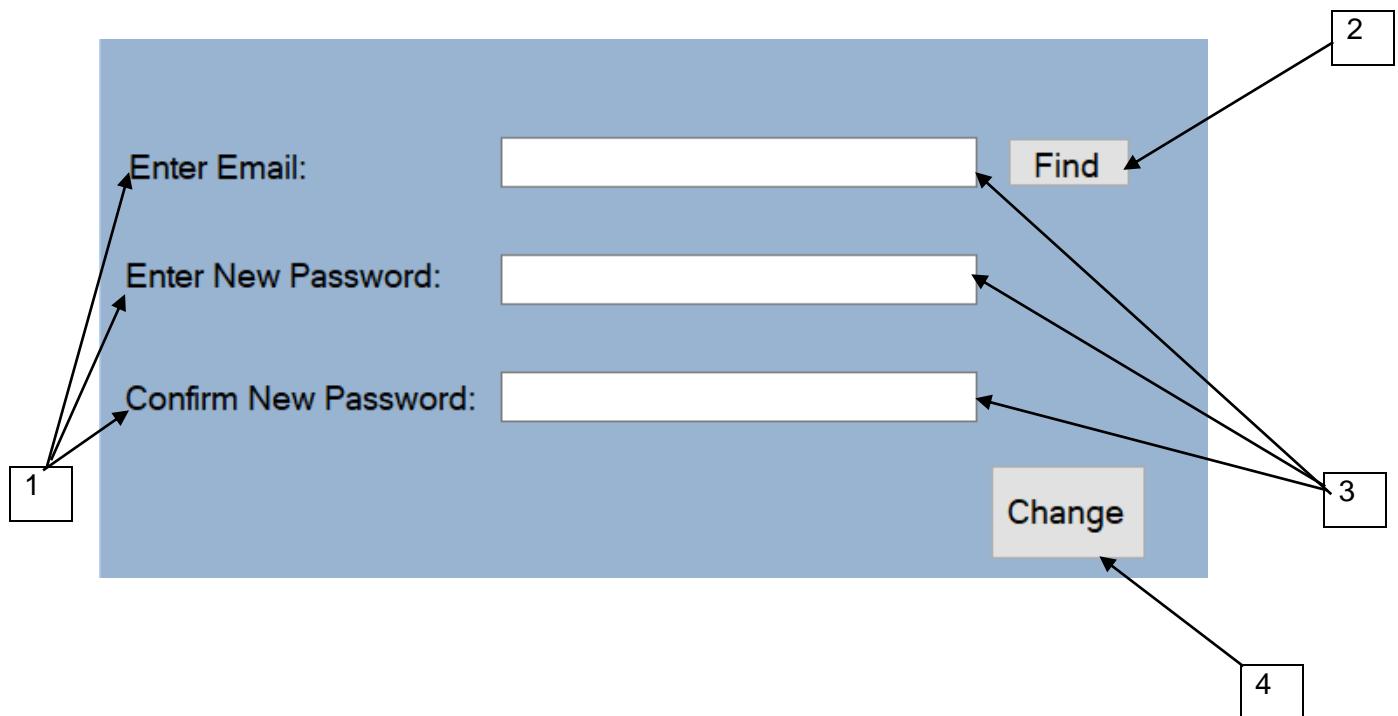


1. **LbUsername.** This label will show the username of the user that is logged into the system by making use of the global variable defined in 'FmLogIn'.
2. **GbUserWatchList.** This groupbox is the container used to show the User's statistics.
3. **DbTotalRatings, DbAverageRatings, DbAnimesCompleted, DbAnimesCurrentlyWatching, DbAnimesYetToWatch, DbAnimesDropped, DbTotalEpisodesWatched, DbHoursSpent.** All of these labels will contain information from

the tables 'UserFavourites' and 'UserWatchList'. They will all be loaded in through the load procedure of this form.

Form: FmReset

If the user clicks on the 'reset' button in 'FmProfile' then the form 'FmReset' will appear.



1. **LbEmail, LbNewPassword, LbConfirmNewPassword.** These labels are used to specify the purpose of the textboxes. The labels LbNewPassword and LbConfirmNewPassword will stay hidden until the email that the user enters is found in the table 'Users'

2. **BtFind.** This button will allow the user to find their email in the table 'Users'.

3. **TbEmail, TbNewPassword, TbConfirmNewPassword.** These textboxes allow the user to enter their email and new password for their account. The textboxes TbNewPassword and TbConfirmNewPassword upon loading this form they will appear if the email entered by the user is found in the table 'Users' through the click of the button btFind.

4. **BtChange.** This button will allow the user to change their old password to the new one they have made

System Security and Integrity of Data

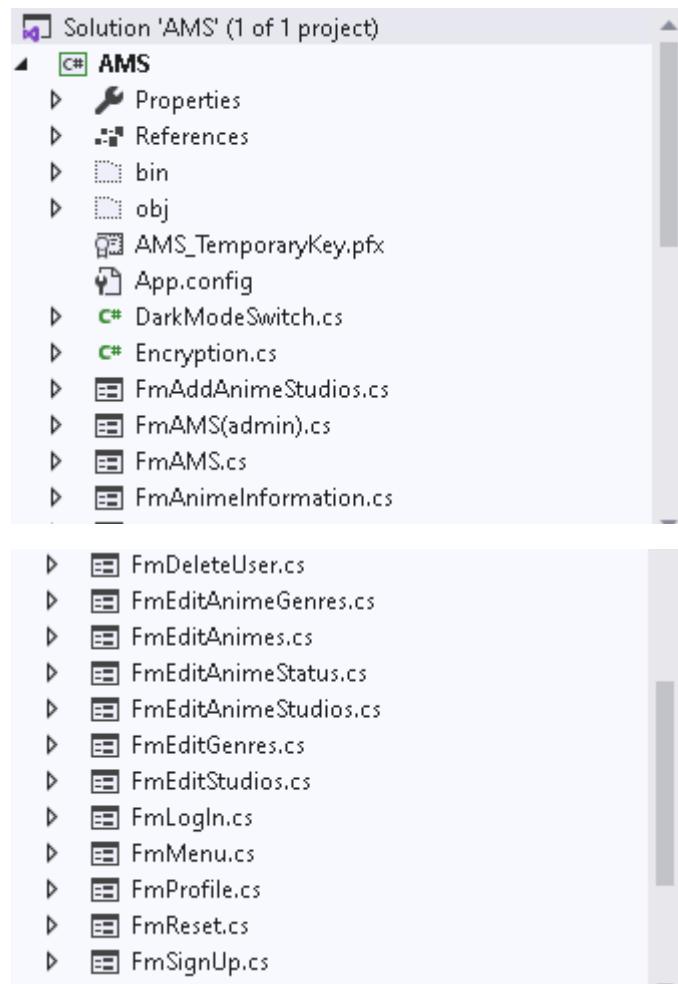
Certain measures will need to be incorporated into the system in order to keep the data accurate and secure. The system will carry out a combination of validation checks built into the program and display visually to the user if the validation fails.

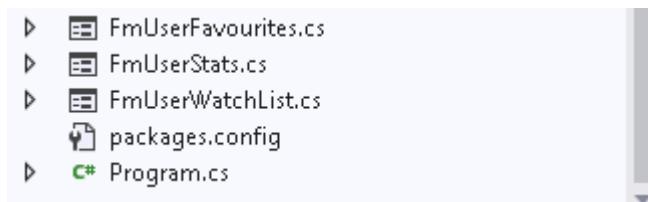
When entering information in the sign up form each textbox is validated with regular expression to ensure the information is in the correct format. The passwords are encrypted to ensure that user information cannot be accessed maliciously. The encryption method used for this was hashing. The admin users can not view the user's password, real name and email.

The admin users will be alerted if the information they have inputted in the edit forms is not in the correct format.

Technical Solution

Name of all file/Entities in project





OneDrive - Xaverian College > Desktop > programming > NEA > AMS

Name	Status	Date modified	Type	Size
.vs	○ R	18/12/2021 14:18	File folder	
AMS	○ R	07/02/2022 21:09	File folder	
packages	○ R	24/01/2022 16:45	File folder	
AMS.sln	○ R	13/12/2021 09:10	Visual Studio Solu...	2 KB

Drive - Xaverian College > Desktop > programming > NEA > AMS > AMS > bin > Debug

Name	Status	Date modified	Type	Size
app.publish	○ R	07/02/2022 17:21	File folder	
AMS	○ R	06/02/2022 22:38	Microsoft Access ...	28,676 KB
AMS	○ R	07/02/2022 11:23	Application Manif...	2 KB
AMS	○ R	07/02/2022 11:23	Application	121 KB
AMS.exe.config	○ R	24/01/2022 22:50	XML Configuratio...	2 KB
AMS.exe.manifest	○ R	07/02/2022 11:23	MANIFEST File	6 KB
AMS.pdb	○ R	07/02/2022 11:23	Program Debug D...	246 KB
Anime	○ R	21/01/2022 15:33	Microsoft Excel C...	648 KB
Animegenre	○ R	23/01/2022 01:12	Microsoft Excel C...	138 KB
AnimeStatus	○ R	20/01/2022 18:53	Microsoft Excel C...	288 KB
Animestudio	○ R	28/12/2021 18:37	Microsoft Excel C...	124 KB
chromedriver	○ R	10/11/2021 17:25	Application	11,106 KB
Genre	○ R	23/01/2022 01:12	Microsoft Excel C...	197 KB
Newtonsoft.Json.dll	○ R	17/03/2021 20:03	Application exten...	686 KB
Newtonsoft.Json	○ R	17/03/2021 19:58	XML Document	694 KB
Studios	○ R	30/12/2021 16:16	Microsoft Excel C...	18 KB
WebDriver.dll	○ R	22/11/2021 15:30	Application exten...	7,674 KB

Complex Procedures

Procedures	Description	Page
HashPassword() (Encryption.cs)	Accepts the password entered by the user and returns the hash value of that password	92
FmMenu_Load() (FmMenu.cs)	Creates all the tables required for this program and inserts the values needed for those tables with multiple foreach loops	97-101

BtSave.Click() (FmSignUp.cs)	Validates the entries made by the user to create an account to save the details to the table 'Users '.	102-109
BtEnter.Click() (FmLogIn.cs)	Validates the entries of the user and determines the user type of the user to send them to the admin form or the normal form	111-113
AutoFillSearch() (FmAMS.cs) (FmUserFavourites.cs) (FmUserWatchList.cs)	Gets the column AnimeTitles and saves the entire column as a string collection for the textbox tbsearch	117-118 147-149 157-158
DelaySeleniumDriver() + SeleniumDriver() (FmAMS.cs)	Gets news from a site called animenewsnetwork and outputs the news every 4 seconds	118-119
btSearch.Click() (FmAMS.cs)	Uses an api to get a json file and parses it. Gets the AnimePicture associated with what the user has searched and loads it into a picture box, then updates the placeholder value with the picture link in the column AnimePictures so that next time the user searches up the same anime it will select the link from the database instead of using the api and load it into the picture box	120-122
AnimelInformation() (FmAnimelInformation.cs)	Gets all the information associated with the anime loaded up through multiple select queries	126-129
SynopsisApi() (FmAnimelInformation.cs)	Uses an api to get a json file and parses it. Gets the Synopsis associated with the anime that is in the form and loads it into a rich text box then updates the placeholder value with the synopsis in the column AnimeSynopsis so that next time the user searches up the same anime it will select the synopsis from the database instead of using the api and load it into the picture box	129-131
BtFavourites.click() (FmAnimelInformation.cs)	Uses an aggregate sql query to validate the button	131-133

	so that it only adds to favourites once anime and per user	
BtWatchList.Click() (FmAnimeInformation.cs)	Uses an aggregate sql query to validate the button so that it only adds to watchlist once per anime and per user	133-135
tbEpisodesWatched.validating() (FmAnimeInformation.cs) (FmUserWatchList.cs)	Uses an aggregate sql query and range validation to validate input from the user	135-137 163-165
FmProfile_Load()	Uses multiple sql select queries to load up user information	138-140
tbSearch_KeyPress() (FmUserFavourites) (FmUserWatchList) (FmAMS(Admin))	Uses the user input to refresh the datagrid viewer to narrow down the search through a SQL select LIKE query	150-152 160-162 175-187
FmUserStats_Load	Uses a complex SQL aggregate query to get the average ratings and uses multiple SQL select queries to get other statistics of the user.	168-171

Annotated Code

Program.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Data.OleDb;

namespace AMS
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application
        /// </summary>
        [STAThread]
        static void Main()
    }
}

```

```

    {
        DarkModeSwitch.SwitchFunction();
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new FmMenu());
    }
    public const string connString = "Provider =Microsoft.ACE.OLEDB.12.0; Data Source =
AMS.accdb";
}
}

```

Encryption.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Security.Cryptography;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in <c>Class:
    Encryption</c>
    /// </summary>
    public class Encryption
    {
        /// <summary>
        /// The resulting method for where Encryption takes place
        /// </summary>
        /// <param name="password"> The <see cref="System.Object"/> that represents the
        user's password </param>
        public static string HashPassword(string password)
        {
            //Declaring the hash function variable
            SHA1CryptoServiceProvider sha1 = new SHA1CryptoServiceProvider();

            //Gets the bytes of the ASCII value of the password
            byte[] Passwordbytes = Encoding.ASCII.GetBytes(password);

            //hashes the bytes of the password
            byte[] Encryptedbytes = sha1.ComputeHash(Passwordbytes);

            //returns the password as a string
            return Convert.ToString(Encryptedbytes);
        }
    }
}

```

DarkModeSwitch.cs

```

using System;
using System.Collections.Generic;
Centre Number:32455
92

```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AMS
{/// <summary>
 /// Handles all inputs and validation of functions that will take place in <c>Class:
DarkModeSwitch</c>
/// </summary>
public class DarkModeSwitch
{
    /// <summary>
    /// Global variable
    /// </summary>
    public static Color FontColour;
    /// <summary>
    /// Global variable
    /// </summary>
    public static Color BackgroundColour;
    /// <summary>
    /// Global variable
    /// </summary>
    public static Color PanelColour;
    /// <summary>
    /// Global variable
    /// </summary>
    public static Color GroupBoxColour;
    /// <summary>
    /// Global variable
    /// </summary>
    public static Color TextBoxColour;
    /// <summary>
    /// Global variable
    /// </summary>
    public static Color SearchBoxColour;
    /// <summary>
    /// Global variable
    /// </summary>
    public static Color ProfileButtonColour;
    /// <summary>
    /// Global variable
    /// </summary>
    public static bool Switch = false;

    /// <summary>
    /// The resulting method for when switch is in use
    /// </summary>
    public static void SwitchFunction()
    {
        //if the button is clicked
        if(Switch==true)
        {
            //Use the function DarkMode
    
```

```
        DarkMode();  
    }  
  
    //otherwise  
    else  
    {  
        //Use the function LightMode  
        LightMode();  
    }  
}  
/// <summary>  
/// The resulting method for setting the form to lightmode  
/// </summary>  
private static void LightMode()  
{  
    /// Sets the colour of any text to black  
    FontColour = Color.Black;  
  
    //Sets the Background colour an rgb value (the colours equivalent to the form  
    fmAMS)  
    BackgroundColour = Color.FromArgb(153, 180, 209);  
  
    //Sets the panel colour an rgb value (the colours equivalent to the form FmAMS)  
    PanelColour = Color.FromArgb(153, 180, 209);  
  
    //Sets the group box colour an rgb value (the colours equivalent to the form FmAMS)  
    GroupBoxColour = Color.FromArgb(0, 120, 215);  
  
    //Sets the textbox colour an rgb value (the colours equivalent to the form FmAMS)  
    TextBoxColour = Color.FromArgb(0, 120, 215);  
  
    //Sets the Search colour an rgb value (the colours equivalent to the form FmAMS)  
    SearchBoxColour = Color.FromArgb(255, 236, 216);  
  
    //Sets the Background colour an rgb value (the colours equivalent to the form  
    FmAMS)  
    ProfileButtonColour = Color.FromArgb(153, 180, 209);  
  
    //Sets the button that is represented as a bool to true  
    Switch = true;  
}  
  
/// <summary>  
/// The resulting method for setting the form to DarkMode  
/// </summary>  
private static void DarkMode()  
{  
    /// Sets the colour of any text to white  
    FontColour = Color.White;  
  
    //Sets the Background colour to an rgb value for a dark colour (For the form:  
    fmAMS)  
    BackgroundColour = Color.FromArgb(51, 51, 51);
```

```

//Sets the panel colour to an rgb value for a dark colour (For the form: FmAMS)
PanelColour = Color.FromArgb(51, 51, 51);

//Sets the group box colour to an rgb value for a dark colour (For the form: FmAMS)
GroupBoxColour = Color.FromArgb(38, 38, 38);

//Sets the text box colour to an rgb value for a dark colour (For the form: FmAMS)
TextBoxColour = Color.FromArgb(38, 38, 38);

//Sets the search box colour to an rgb value for a dark colour (For the form: FmAMS)
SearchBoxColour = Color.FromArgb(52, 52, 51);

//Sets the button colour to an rgb value for a dark colour(For the form: FmAMS)
ProfileButtonColour = Color.FromArgb(51, 51, 51);

//Sets the button that is represented as a bool to false
Switch = false;
}

}

}

```

Form:FmMenu.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Data.OleDb;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in <c>FmMenu</c>
    /// </summary>
    public partial class FmMenu : Form
    {

        /// <summary>
        /// Required Method for Designer support for <c>FmMenu</c>
        /// </summary>
        public FmMenu()
        {
            InitializeComponent();
        }

        /// <summary>
        /// The resulting method when the user clicks on the Sign-in button
        /// </summary>
    }
}

```

```
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
///
private void btSignUp_Click(object sender, EventArgs e)
{
    //Hides the form from the user
    Hide();

    //Creates a variable from the Form FmSignUp
    FmSignUp FmSign = new FmSignUp();

    // Shows the Form FmSign
    FmSign.ShowDialog();

}

/// <summary>
/// The resulting method for when the user clicks on the Log-in button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btLogIn_Click(object sender, EventArgs e)
{
    //Hides the form from the user
    Hide();

    //Creates a variable from the Form
    FmLogIn FmLog = new FmLogIn();

    // Shows the Form FmLog
    FmLog.ShowDialog();

}

/// <summary>
/// The resulting method for when the user clicks the Exit button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btExit_Click(object sender, EventArgs e)
{
    //When exit button is clicked the form will close.
    Close();

}

/// <summary>
```

```
/// The resulting method for when the Form loads
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void FmMenu_Load(object sender, EventArgs e)
{
    // Checks if database exists, if it doesn't it will create the database
    if (File.Exists("AMS.accdb") == false)

    {
        ADOX.Catalog cat = new ADOX.Catalog();

        // Uses constant defined in Program.cs to create database
        cat.Create(Program.connString);

        // Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        // Opens Connection to the database
        Conn.Open();

        // Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        // Creates a connection with the Command Object
        Cmd.Connection = Conn;

        // Creates User Table
        Cmd.CommandText = "CREATE TABLE Users(UserID AUTOINCREMENT,
Username VARCHAR(60), Pword VARCHAR(255), UserType VARCHAR(30), Forename
VARCHAR(60), Surname VARCHAR(60), Email VARCHAR(255), PRIMARY KEY
(UserID))";

        // Executes the SQL Query
        Cmd.ExecuteNonQuery();

        // Creates Studios table
        Cmd.CommandText = "CREATE TABLE Studios(StudioID INTEGER, Studio
VARCHAR(100), PRIMARY KEY (StudioID))";

        // Executes the SQL Query
        Cmd.ExecuteNonQuery();

        // Creates Animes table
        Cmd.CommandText = "CREATE TABLE Animes(AnimeID INTEGER, AnimeTitle
VARCHAR(200), Episodes INTEGER, Ratings INTEGER, AnimeSynopsis LONGTEXT,
AnimePictures TEXT, PRIMARY KEY (AnimeID))";

        // Executes the SQL Query
        Cmd.ExecuteNonQuery();

        // Creates UserFavourites Table
    }
}
```

```
Cmd.CommandText = "CREATE TABLE UserFavourites(UserFavID  
AUTOINCREMENT, UserID INTEGER, AnimeID INTEGER, AnimeTitle VARCHAR(200),  
Ratings INTEGER, CONSTRAINT PK_UserCompKey PRIMARY KEY(UserFavID, UserID),  
FOREIGN KEY (UserID) REFERENCES Users(UserID))";  
  
//Executes the SQL Query  
Cmd.ExecuteNonQuery();  
  
//Creates UserWatchList Table  
Cmd.CommandText = "CREATE TABLE UserWatchList(UserWatchID  
AUTOINCREMENT, UserID INTEGER, AnimeID INTEGER, AnimeTitle VARCHAR (200),  
WatchStatus VARCHAR(100), EpisodesWatched INTEGER, Ratings INTEGER,  
CONSTRAINT PK_UserCompKey PRIMARY KEY(UserWatchID, UserID), FOREIGN KEY  
(UserID) REFERENCES Users(UserID))";  
  
//Executes the SQL Query  
Cmd.ExecuteNonQuery();  
  
//Creates Genres Table  
Cmd.CommandText = "CREATE TABLE Genres(GenreID INTEGER,Genre TEXT,  
PRIMARY KEY (GenreID))";  
  
//Executes the SQL Query  
Cmd.ExecuteNonQuery();  
  
// Creates AnimeStudios table (Creates a junction table between Animes and  
Studio to eliminate Many to Many relationship / Composite Key)  
Cmd.CommandText = "CREATE TABLE AnimeStudios(AnimeID INTEGER,  
StudioID INTEGER, CONSTRAINT PK_AnimeStudios PRIMARY KEY (AnimeID, StudioID),  
FOREIGN KEY (AnimeID) REFERENCES Animes(AnimeID), FOREIGN KEY (StudioID)  
REFERENCES Studios(StudioID))";  
  
//Executes the SQL Query  
Cmd.ExecuteNonQuery();  
  
// Creates AnimeGenres table (Creates a junction table between Animes and  
Genre to eliminate Many to Many relationship / Composite Key)  
Cmd.CommandText = "CREATE TABLE AnimeGenres(AnimeID INTEGER,  
GenreID INTEGER, CONSTRAINT PK_AnimeGenres PRIMARY KEY (AnimeID, GenreID),  
FOREIGN KEY (AnimeID) REFERENCES Animes(AnimeID), FOREIGN KEY (GenreID)  
REFERENCES Genres(GenreID))";  
  
//Executes the SQL Query  
Cmd.ExecuteNonQuery();  
  
// Creates AnimeStatus table (Composite key)  
Cmd.CommandText = "CREATE TABLE AnimeStatus(AnimeID INTEGER, Status  
VARCHAR(150), CONSTRAINT PK_AnimeStatus PRIMARY KEY (AnimeID, Status),  
FOREIGN KEY(AnimeID) REFERENCES Animes(AnimeID))";  
  
//Executes the SQL Query  
Cmd.ExecuteNonQuery();
```

```
// Assigned a variable to read the Studio.csv file
var lines = File.ReadLines("Studios.csv");

// Iterates the Studio.csv file with a condition associated to the line variable
foreach (string line in lines)

{
    // splits the Studio.csv file into values that are associated with the Table column
    // for Studios and stores it into an array
    string[] StudioList = line.Split(',');

    // Inserts the values into the database from Studio.csv file that are contained in
    // the array
    Cmd.CommandText = "INSERT INTO Studios VALUES(" + StudioList[0] + ", "
    + StudioList[1] + ")";

    //Execute SQL command
    Cmd.ExecuteNonQuery();

    // will keep inserting until the end of the Studio.csv file
}

// Assigned a variable to read the Anime.csv file
lines = File.ReadLines("Anime.csv");

// Iterates the Anime.csv file with a condition associated to the line variable
foreach (string line in lines)

{
    // splits the Anime.csv file into values that are associated with the Table column
    // for Animes and stores it into an array
    string[] AnimeList = line.Split(',');

    // Inserts the value into the database from Anime.csv file that are contained in
    // the array
    Cmd.CommandText = "INSERT INTO Animes VALUES(" + AnimeList[0] + ", "
    + AnimeList[1] + ", " + AnimeList[2] + ", " + AnimeList[3] + ", "
    + AnimeList[4] + ", " + AnimeList[5] + ")";

    //Execute SQL command
    Cmd.ExecuteNonQuery();

    // will keep inserting until the end of the Anime.csv file
}

// Assigned a variable to read the Genre.csv file
lines = File.ReadLines("Genre.csv");

// Iterates the Genre.csv file with a condition associated to the line variable
foreach (string line in lines)

{
```

```
// splits the Genre.csv file into values that are associated with the Table column
for Genres and stores it into an array
    string[] GenreList = line.Split(',');

    // Inserts the values into the database from Genre.csv file that are contained in
the array
    Cmd.CommandText = "INSERT INTO Genres VALUES('" + GenreList[0] + "','" +
GenreList[1] + "')";

    //Execute SQL command
    Cmd.ExecuteNonQuery();

    // will keep inserting until the end of the csv file

}

// Assigned a variable to read the Animestudio.csv file
lines = File.ReadLines("Animestudio.csv");

// Iterates the Animestudio.csv file with a condition associated to the line variable
foreach (string line in lines)

{
    // splits the Animestudio.csv file into values that are associated with the Table
column for AnimeStudios and stores it into an array
    string[] AnimeStudioList = line.Split(',');

    // Insert the values into the database from Animestudio.csv file that are
contained in the array
    Cmd.CommandText = "INSERT INTO AnimeStudios VALUES('" +
AnimeStudioList[0] + "','" + AnimeStudioList[1] + "')";

    //Execute SQL command
    Cmd.ExecuteNonQuery();

    // will keep inserting until the end of the csv file

}

// Assigned a variable to read the Animegenre.csv file
lines = File.ReadLines("Animegenre.csv");

// Iterates the Animegenre.csv file with a condition associated to the line variable
foreach (string line in lines)

{
    // splits the Animegenre.csv file into values that are associated with the Table
column for AnimeGenres and stores it into an array
    string[] AnimeGenreList = line.Split(',');

    // Insert the values into the database from Animegenre.csv file that are
contained in the array
    Cmd.CommandText = "INSERT INTO AnimeGenres VALUES('" +
AnimeGenreList[0] + "','" + AnimeGenreList[1] + "')";
```

```
//Execute SQL command
Cmd.ExecuteNonQuery();

// will keep inserting until the end of the csv file

}

// Assigned a variable to read the AnimeStatus.csv file
lines = File.ReadLines("AnimeStatus.csv");

// Iterates the AnimeStatus.csv file with a condition associated to the line variable
foreach (string line in lines)

{
    // splits the AnimeStatus.csv file into values that are associated with the Table
    // column for AnimeGenres and stores it into an array
    string[] AnimeStatuslist = line.Split(',');

    // Insert the values into the database from AnimeStatus.csv file that are
    // contained in the array
    Cmd.CommandText = "INSERT INTO AnimeStatus VALUES(" +
    AnimeStatuslist[0] + ", " + AnimeStatuslist[1] + ")";

    //Execute SQL command
    Cmd.ExecuteNonQuery();

    // will keep inserting until the end of the csv file
}

// Closes the connection to the database
Conn.Close();

}

}
```

Form: FmSignUp.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Data.OleDb;
using System.Text.RegularExpressions;
```

Centre Number:32455

101

```
namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in <c>FmSignIn</c>
    /// </summary>
    public partial class FmSignUp : Form
    {

        /// <summary>
        /// Required Method for Designer support for <c>FmSignIn</c>
        /// </summary>
        public FmSignUp()
        {
            InitializeComponent();
        }

        /// <summary>
        /// The resulting method when the user clicks on the Return button
        /// </summary>
        /// <param name="sender"> The <see cref="System.Object"/> that represents the
        sender instance</param>
        /// <param name="e">The <see cref="System.EventArgs"/> that represents the event
        arguments passed
        /// to the event handler</param>
        private void btReturn_Click(object sender, EventArgs e)
        {
            //Closes the form FmSignIn
            Close();

            //Creates a variable from the Form FmMenu
            FmMenu fmMenu = new FmMenu();

            // Shows the Form FmMenu
            fmMenu.Show();
        }

        /// <summary>
        /// The resulting method when the user clicks on the save button
        /// </summary>
        /// <param name="sender"> The <see cref="System.Object"/> that represents the
        sender instance</param>
        /// <param name="e">The <see cref="System.EventArgs"/> that represents the event
        arguments passed
        /// to the event handler</param>
        private void btSave_Click(object sender, EventArgs e)
        {
            //Exception handling
            try
            {

                //Sets Username to the text of textbox tbUsername
            }
        }
    }
}
```

```
var Username = tbUsername.Text;

//Sets the variable Password to the text of textbox tbPword
var Password = tbPword.Text;

//Sets the variable Forname to the text of textbox tbForename
var Forename = tbForename.Text;

//Sets the variable Surname to the text of textbox tbSurname
var Surname = tbSurname.Text;

//Sets the variable Email to the text of textbox tbEmail
var Email = tbEmail.Text;

//Regex validation for upper case characters
var hasUpperChar = new Regex(@"[A-Z]+");

//Regex validation for lower case characters
var hasLowerChar = new Regex(@"[a-z]+");

//Regex validation for numbers
var hasNumber = new Regex(@"[0-9]+");

//Regex validation for symbols
var hasSymbols = new Regex(@"[!@#$^&*()_+=\[\{\}];:<>|./?,-]+");

//Regex validation for Emails
var EmailValid = new Regex(@"^([a-zA-Z0-9_.\|-]+)@(([a-zA-Z0-9\-.])+(\.[a-zA-Z]{2,3})+$");

//if the variable Username is empty
if(Username == ""|| Username ==" ")
{
    //Displays a message window stating that the User's Username is empty
    MessageBox.Show("Usernames should not be empty!");
}

//if the variable Forname is empty
else if (Forename ==""|| Forename ==" ")
{
    //Displays a message window stating that the User's Forename should not be
empty
    MessageBox.Show("Forename should not be empty!");
}

//if the variable Surname is empty
else if (Surname=="" || Surname ==" ")
{
    //Displays a message window stating that the User's Surname should not be
empty
    MessageBox.Show("Surname should not be empty!");
}
```

```
//if the variable Email is empty
else if (Email=="|| Email ==" ")
{
    //Displays a message window stating that the User's Email should not be empty
    MessageBox.Show("Email should not be empty");

}

//if the Variable Username does not match the regex validation for numbers
else if(!hasNumber.IsMatch(Username))
{
    //Displays a message window stating that the User's Username
    //should contain atleast one number
    MessageBox.Show("Username should contain atleast one number");

    //Clears the text that was entered in the textbox tbUsername
    tbUsername.Clear();

}

//if the Variable Username does not match the regex validation for upper case
characters
else if (!hasUpperChar.IsMatch(Username))
{
    //Displays a message window stating that the User's Username
    //should contain atleast one uppercase letter
    MessageBox.Show("Username should contain atleast one uppercase letter!");

    //Clears the text that was entered in the textbox tbUsername
    tbUsername.Clear();

}

//if the variable Password is empty
else if (Password ==""||Password ==" ")

{
    //Displays a message window stating that the User's Email should not be empty
    MessageBox.Show("Password should not be empty!");

}

//if the Variable Password does not match the regex validation for numbers
else if (!hasNumber.IsMatch(Password))
{
    //Displays a message window stating that the User's Password
    //should contain atleast one numeric value
    MessageBox.Show("Password shoud contain atleast one numeric value!");

    //Clears the text that was entered in the textbox tbPword
    tbPword.Clear();
}
```

```
//if the Variable Password does not match the regex validation for upper case
characters
if (!hasUpperChar.IsMatch>Password)
{
    //Displays a message window stating that the User's Password
    //should contain atleast one upper case letter
    MessageBox.Show("Password should contain atleast one upper case letter!");

    //Clears the text that was entered in the textbox tbPword
    tbPword.Clear();
}
//if the Variable Username does not match the regex validation for lower case characters
else if (!hasLowerChar.IsMatch>Password)
{
    //Displays a message window stating that the User's Password
    //should contain atleast one lower case letter
    MessageBox.Show("Password should contain atleast one lower case letter!");

    //Clears the text that was entered in the textbox tbPword
    tbPword.Clear();
}

//if the Variable Username does not match the regex validation for symbols
else if (!hasSymbols.IsMatch>Password)
{
    //Displays a message window stating that the User's Password
    //should contain atleast one special case letter
    MessageBox.Show("Password should contain atleast one special case
character!");

    //Clears the text that was entered in the textbox tbPword
    tbPword.Clear();
}

//if the Variable Email does not match the regex validation for Emails
else if (!EmailValid.IsMatch>Email)
{
    //Displays a message window stating that the User's Email is not valid
    MessageBox.Show("Email is not valid");

    //Clears the text that was entered in the textbox tbEmail
    tbEmail.Clear();
}
else if(cbUserType.SelectedIndex == -1)
{
    MessageBox.Show("Please select a user type!");
}

//if all the variables meet all the requirements
else
{
    try // Exception Handling
```

```
{  
  
    //Create a database connection object with the constant defined in  
    program.cs  
    OleDbConnection Conn = new OleDbConnection(Program.connString);  
  
    //Opens Connection to the database  
    Conn.Open();  
  
    //Creates a database command object  
    OleDbCommand Cmd = new OleDbCommand();  
  
    //Creates a connection with the Command Object  
    Cmd.Connection = Conn;  
  
    //An Aggregate SQL query that selects the Count From the table users  
    //where username is given by the user  
    //This is done to check if the Username exists in the table Users  
    Cmd.CommandText = "SELECT count(*) From Users WHERE Username =  
@Username";  
  
    //Set the parameter for @Username to the username given by the user  
    Cmd.Parameters.Add("@Username", OleDbType.Char).Value =  
tbUsername.Text;  
  
    //Gets the result from the Select query and sets the count to the that result  
    int Count = Convert.ToInt32(Cmd.ExecuteScalar());  
  
    //if the Username Exists  
    if (Count > 0)  
    {  
        //Displays a MessageBox saying that the username entered  
        //by the user already exists  
        MessageBox.Show("Username already exists!");  
    }  
  
    //If the Username doesn't exist in the table  
    else  
    {  
        //if the user selected Administrator from the combobox cbUserType  
        if (cbUserType.Text == "Administrator")  
        {  
            //declares a key set as a string that contains the key  
            string key = "#?123";  
  
            //if the text that was entered by the user in the textbox tbAdminKey  
            //Does not match the key  
            if (tbAdminKey.Text != key)  
            {  
                //Displays a MessageBox stating that the key is invalid  
                MessageBox.Show("Invalid key");  
            }  
        }  
    }  
}
```

```
        //Clears the text that was entered by the user in the textbot
tbAdminKey
    tbAdminKey.Clear();
}

//if the text that was entered by the user in the textbox tbAdminKey
//Matches the key
else
{
    //Inserts the values that were entered by the user
    // the password that the user entered is hashed and then inserted
    Cmd.CommandText = "INSERT INTO Users(Username, Pword,
    UserType, Forename, Surname, Email) VALUES(" + tbUsername.Text + "','" +
Encryption.HashPassword(tbPword.Text) + "','" + cbUserType.Text + "','" + tbForename.Text
+ "','" + tbSurname.Text + "','" + tbEmail.Text + ")";

    //Executes the SQL Query
    Cmd.ExecuteNonQuery();
// Closes the connection to the database
Conn.Close();

    //Displays a message window stating that the User's Details
    //are now saved and that the user can log in
    MessageBox.Show("Details saved, you can now log in!");

    //Clears the text that was entered in the textbox tbForename
    tbForename.Clear();

    //Clears the text that was entered in the textbox tbSurname
    tbSurname.Clear();

    //Clears the text that was entered in the textbox tbEmail
    tbEmail.Clear();

    //Clears the text that was entered in the textbox tbUsername
    tbUsername.Clear();

    //Clears the text that was entered in the textbox tbPword
    tbPword.Clear();

    //Clears the text that was entered in the textbox tbAdminKey
    tbAdminKey.Clear();

    //Resets the combo box so that nothing is selected
    ResetSelection();

    //Set the visibility of the label lbAdminKey to false
    //Makes it invisible
    lbAdminKey.Visible = false;

    //Set the visibility of the label tbAdminKey to false
    //Makes it invisible
    tbAdminKey.Visible = false;
```

```
        }
    }
else
{
    //Inserts the values that were entered by the user
    // the password that the user entered is hashed and then inserted
    Cmd.CommandText = "INSERT INTO Users(Username, Pword,
    UserType, Forename, Surname, Email) VALUES('" + tbUsername.Text + "', '" +
    Encryption.HashPassword(tbPword.Text) + "','" + cbUserType.Text + "','" + tbForename.Text +
    "','" + tbSurname.Text + "','" + tbEmail.Text + "')";

    //Executes the SQL Query
    Cmd.ExecuteNonQuery();

    // Closes the connection to the database
    Conn.Close();

    //Displays a message window stating that the User's Details
    //are now saved and that the user can log in
    MessageBox.Show("Details saved, you can now log in!");

    //Clears the text that was entered in the textbox tbForename
    tbForename.Clear();

    //Clears the text that was entered in the textbox tbSurname
    tbSurname.Clear();

    //Clears the text that was entered in the textbox tbEmail
    tbEmail.Clear();

    //Clears the text that was entered in the textbox tbUsername
    tbUsername.Clear();

    //Clears the text that was entered in the textbox tbPword
    tbPword.Clear();

    //Resets the combo box so that nothing is selected
    ResetSelection();

}

}

//If an Exception is caught in the sql queries
catch(Exception)
{
    //Displays a message window saying it Could not insert
    MessageBox.Show("Could not insert");
}
}
```

```
//if an exception is caught
catch (Exception)
{
    //Displays a message window saying it is invalid
    MessageBox.Show("Invalid");
}

}
/// <summary>
/// The resulting method when the user clicks on the ComboBox cbUserType and
selects something from the DropDownList
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>
private void cbUserType_SelectedIndexChanged(object sender, EventArgs e)
{
    //if the User Selects Administrator on the ComboBox cbUserType
    if(cbUserType.Text=="Administrator")
    {
        // Sets the visibility of the label lbAdminKey to true
        //Makes it visible(It is set as false in the label properties so the
        //Form will load up with this label hidden
        lbAdminKey.Visible = true;

        // Sets the visibility of the textbox tbAdminKey to true
        //Makes it visible(It is set as false in the label properties so the
        //Form will load up with this textbox hidden
        tbAdminKey.Visible = true;
    }
    else
    {
        // Sets the visibility of the label lbAdminKey to false
        //Makes it invisible
        lbAdminKey.Visible = false;

        // Sets the visibility of the label tbAdminKey to false
        //Makes it invisible
        tbAdminKey.Visible = false;
    }
}

private void ResetSelection()
{
    cbUserType.SelectedIndex = -1;
}
}
```

Form: FmLogIn.cs

```
using System;
Centre Number:32455
109
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Data.OleDb;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in <c>FmLogIn</c>
    /// </summary>
    public partial class FmLogIn : Form
    {
        /// <summary>
        /// Global variables
        /// </summary>

        public static string Username="";
        public static FmLogIn FmLogInInstance;

        /// <summary>
        /// Required Method for Designer support for <c>FmLogIn</c>
        /// </summary>
        public FmLogIn()
        {
            InitializeComponent();
        }

        /// <summary>
        /// The resulting method for when the Form loads
        /// </summary>
        /// <param name="sender"> The <see cref="System.Object"/> that represents the
        /// sender instance </param>
        /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
        /// arguments passed to the event handler </param>
        private void FmLogIn_Load(object sender, EventArgs e)
        {
            // if the Username that is located in properties is not empty and contains a string
            if(Properties.Settings.Default.Username != string.Empty)

            {
                // Sets tbUsername(textbox) field value to the value of the Username stored in
                // properties
                tbUsername.Text = Properties.Settings.Default.Username;

                //Sets tbPword(textbox) value to the value of Password stored in properties
                tbPword.Text = Properties.Settings.Default.Password;

                // Sets cbRemember(combo box) to true when all conditions are met
            }
        }
    }
}
```

```

        cbRemember.Checked = true;
    }
}
/// <summary>
/// The resulting method when the user clicks on the Enter button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments
/// passed to the event handler</param>
private void btEnter_Click(object sender, EventArgs e)
{
    try //Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //Selecting everthing from the table Users where the username is entered in the
        textbox tbUsername
        Cmd.CommandText = "SELECT * FROM Users WHERE Username ='" +
        tbUsername.Text + "'";

        //Sets the global variable Username that is defined in FmMenu to the value
        //of the username that is entered in the textbox tbUsername
        Username = tbUsername.Text;

        //Creates a reader object and Executes the SQL Query
        OleDbDataReader reader = Cmd.ExecuteReader();

        // if the SQL query contains one or more rows
        if (reader.HasRows)

        {
            //read all the rows
            reader.Read();

            // if the hashedvalue of the password that is entered in the textbox tbPword
            // matches the hashedPassword in the User table
            if (Encryption.HashPassword(tbPword.Text) == reader["Pword"].ToString())

            {
                //if the UserType of the User is Administrator
                if ("Administrator" == reader["UserType"].ToString())

```

```
        this.Visible = false;

        //Creates a variable from the Form FmAMS(admin)
        FmAMS_admin_ f4 = new FmAMS_admin_();

        // Shows the Form FmAms(Admin) through the variable
        f4.ShowDialog();

        this.Visible = true;
    }

    //if the UserType of the user is not Administrator
    else

    {
        this.Visible = false;

        //Creates a variable from the Form FmAMS
        FmAMS f4 = new FmAMS();

        // Shows the Form FmAMS through the variable
        f4.ShowDialog();

        this.Visible = true;
    }

    // if the hashedvalue of the password that is entered in the textbox tbPword
    // does not match the hashedPassword in the User table
    else

    {
        //Displays a message window stating that the User's password is wrong
        MessageBox.Show("Wrong password");

        //Clears the text that was enter in the textbox tbUsername
        tbUsername.Clear();

        //Clears the text that was enter in the textbox tbPword
        tbPword.Clear();
    }
}

}

catch (Exception) //Catches the error
{
    //Displays a message box saying it could not be found
    MessageBox.Show(" Could not be found");
}
```

```
//if the checkbox cbRemember is checked
if(cbRemember.Checked == true)

{
    //Set the value of Username that is located in Properties to the value of the textbox
    tbUsername
    Properties.Settings.Default.Username = tbUsername.Text;

    //Set the value of Username that is located in Properties to the value of the textbox
    tbPword
    Properties.Settings.Default.Password = tbPword.Text;

    //Save the values in Properties
    Properties.Settings.Default.Save();

}

//if the checkbox cbRemember is not checked
if (cbRemember.Checked == false)

{
    //Set the value of Username that is located in Properties to an empty string
    Properties.Settings.Default.Username = "";

    //Set the value of Username that is located in Properties to an empty string
    Properties.Settings.Default.Password = "";

    //Save the values in Properties
    Properties.Settings.Default.Save();
}

}

/// <summary>
/// The resulting method when the user clicks on the Return button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>

private void btReturn_Click(object sender, EventArgs e)
{
    //Closes the form FmLogin
    Close();

    //Creates a variable from the Form FmMenu
    FmMenu fmMenu = new FmMenu();

    // Shows the Form FmMenu
    fmMenu.ShowDialog();
}

}

}

Centre Number:32455
113
```

Form: FmAMS.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Threading;
using System.Net;
using Newtonsoft.Json.Linq;
using System.IO;
using System.Data.OleDb;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using System.Net.Http;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in <c>FmAMS</c>
    /// </summary>

    public partial class FmAMS : Form
    {
        /// <summary>
        /// Global variable
        /// </summary>
        public static FmAMS FmAmsInstance;
        /// <summary>
        /// Global variables
        /// </summary>
        public static string animePic = "";
        /// <summary>
        /// Global variables
        /// </summary>
        public static string AnimeName = "";

        public static FmAMS fmAmsInstance;

        /// <summary>
        /// Required Method for Designer support for <c>FmAMS</c>
        /// </summary>
        public FmAMS()
        {
            InitializeComponent();
            SetColour();

        }
        /// <summary>
        /// The resulting method for determining the colour of the form FmAMS
    }
}
```

```
/// </summary>
private void SetColour()
{
    //Sets the background colour of this form to the background colour declared in class:
    DarkModeSwitch
    this.BackColor = DarkModeSwitch.BackgroundColour;

    //Sets the text colour of this form to the font colour declared in class:
    DarkModeSwitch
    this.ForeColor = DarkModeSwitch.FontColour;

    //Sets the background colour for the panel PaAnimePanel of this form to the panel
    colour declared in class: DarkModeSwitch
    PaAnimePanel.BackColor = DarkModeSwitch.PanelColour;

    //Sets the background colour for the groupbox gbAMS of this form to the group box
    colour declared in class: DarkModeSwitch
    gbAMS.BackColor = DarkModeSwitch.GroupBoxColour;

    //Sets the background colour for the groupbox gbAnimeNews of this form to the group box
    colour declared in class: DarkModeSwitch
    gbAnimeNews.BackColor = DarkModeSwitch.GroupBoxColour;

    //Sets the text colour for the groupbox gbAnimeNews of this form to the font colour
    declared in class: DarkModeSwitch
    gbAnimeNews.ForeColor = DarkModeSwitch.FontColour;

    //Sets the background colour for the textbox tbNews of this form to the text box
    colour declared in class: DarkModeSwitch
    tbNews.BackColor = DarkModeSwitch.TextBoxColour;

    //Sets the text colour for the textbox tbNews of this form to the font colour declared in
    class: DarkModeSwitch
    tbNews.ForeColor = DarkModeSwitch.FontColour;

    //Sets the background colour for the textbox tbSearch of this form to the search box
    colour declared in class: DarkModeSwitch
    tbSearch.BackColor = DarkModeSwitch.SearchBoxColour;

    //Sets the text colour for the textbox tbSearch of this form to the font colour declared
    in class: DarkModeSwitch
    tbSearch.ForeColor = DarkModeSwitch.FontColour;

    //Sets the background colour for the button btBack of this form to the text box colour
    declared in class: DarkModeSwitch
    btBack.BackColor = DarkModeSwitch.TextBoxColour;

    //Sets the background colour for the button btProfile of this form to the profile button
    colour declared in class: DarkModeSwitch
    btProfile.BackColor = DarkModeSwitch.ProfileButtonColour;

    //Sets the background colour for the button btSearch of this form to the search box
    colour declared in class: DarkModeSwitch
    btSearch.BackColor = DarkModeSwitch.SearchBoxColour;
```

```
//Sets the background colour for the button btSwitch of this form to the search box
colour declared in class: DarkModeSwitch
btSwitch.BackColor = DarkModeSwitch.SearchBoxColour;

//Sets the background colour for the button btSearch of this form to the text box
colour declared in class: DarkModeSwitch
btSearch.BackColor = DarkModeSwitch.TextBoxColour;

//Sets the text colour for the button btBack of this form to the font colour declared in
class: DarkModeSwitch
btBack.ForeColor = DarkModeSwitch.FontColour;
}

/// <summary>
/// The resulting method for when the Form loads
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void FmAMS_Load(object sender, EventArgs e)
{
    //loads the image from the link to the picture box PbAnimePicture1
PbAnimePicture1.LoadAsync(@"https://cdn.myanimelist.net/images/anime/7/37911.jpg");

    //loads the image from the link to the picture box PbAnimePicture2
PbAnimePicture2.LoadAsync(@"https://cdn.myanimelist.net/images/anime/8/199681.jpg");

    //loads the image from the link to the picture box PbAnimePicture3
PbAnimePicture3.LoadAsync(@"https://cdn.myanimelist.net/images/anime/1/201.jpg");

    //Sets the label lbUsername to the global variable declared in the form FmLogin which
contains the user's username
    lbUsername.Text = FmLogin.Username;

    //Runs the procedure AutoFillSearch
    AutoFillSearch();

    //if the global variable switch is true in the class: DarkModeSwitch
    if(DarkModeSwitch.Switch==true)
    {
        //Runs the procedure SetColour
        SetColour();
    }

    //otherwise
    else
    {
        //Set the global variable switch to false in the class: DarkModeSwitch
    }
}
```

```
    _ = DarkModeSwitch.Switch == false;  
}  
  
//Runs the procedure SeleniumDriver  
SeleniumDriver();  
  
//Sets the visibility of the panel PaAnimePanel to false  
PaAnimePanel.Visible = false;  
}  
  
/// <summary>  
/// The resulting method for the procedure AutoFillSearch  
/// </summary>  
private void AutoFillSearch()  
{  
  
    try//Exception Handling  
    {  
  
        //Create a database connection object with the constant defined in program.cs  
        OleDbConnection Conn = new OleDbConnection(Program.connString);  
  
        //Opens Connection to the database  
        Conn.Open();  
  
        //Creates a database command object  
        OleDbCommand Cmd = new OleDbCommand();  
  
        //Creates a connection with the Command Object  
        Cmd.Connection = Conn;  
  
        //A select query to select the column AnimeTitle from the table Animes  
        Cmd.CommandText = "SELECT AnimeTitle FROM Animes";  
  
        //Defines the autocompletefeature that holds a collection of strings AutoFill  
        AutoCompleteStringCollection AutoFill = new AutoCompleteStringCollection();  
  
        //Creates a reader object and Executes the SQL Query  
        OleDbDataReader reader = Cmd.ExecuteReader();  
  
        //if the reader object has found rows  
        if (reader.HasRows)  
        {  
            //while the reader object is reading the rows in the column AnimeTitle  
            while (reader.Read())  
            {  
                //Add the rows that are in the column AnimeTitle to AutoFill  
                AutoFill.Add(reader["AnimeTitle"].ToString());  
            }  
  
        }  
  
        //otherwise  
        else  
        {  
    }
```

```
    //Display a messagebox informing the user that there was no AnimeTitles found
    MessageBox.Show("Anime title not found!");
}

//Closes the reader object.
reader.Close();

//Sets the autocomplete source of the textbox tbSearch to AutoFill which contains
all the AnimeTitles
tbSearch.AutoCompleteCustomSource = AutoFill;

//Closes the connection to the database
Conn.Close();
}

catch(Exception) //catches error
{
    //Displays a message box informing the user that it could not loadAutoFill
    MessageBox.Show("Could not load");
}

}

/// <summary>
/// The resulting method for the procedure SeleniumDriver
/// </summary>
private async void SeleniumDriver()
{
    //Suspends the function DelaySeleniumDriver
    await DelaySeleniumDriver();
}

/// <summary>
/// The resulting method for the procedure DelaySeleniumDriver
/// </summary>
async Task DelaySeleniumDriver()
{
    //Declaring service as a chromediverservice variable
    ChromeDriverService service = ChromeDriverService.CreateDefaultService();

    //hides the command prompt window so its running in the background
    service.HideCommandPromptWindow = true;

    //Declaring chromeoptions as a new variable
    var chromeOptions = new ChromeOptions();

    //hides google chrome
    chromeOptions.AddArgument("headless");

    //declaring the variable driver with the variables service and chromeoptions
    var driver = new ChromeDriver(service, chromeOptions);

    //the driver navigates to the link in google chrome
    driver.Navigate().GoToUrl("https://www.animenewsnetwork.com/news/?ann-
edition=uk");
}
```

```

//finds the main source of the news via css selector
var News = driver.FindElements(By.CssSelector("#mainfeed > div.mainfeed-
section.herald-boxes"));

// runs forever
while (true)
{
    //until i=80 (the rough amount of news on the site)
    for (int i = 1; i < 80; i++)
    {
        try //Exception Handling
        {
            //delay the 4 loop by 4 seconds
            await Task.Delay(4000);

            //gets the news titles via xpath and will keep getting it until i=80
            var NewsTitles =
driver.FindElement(By.XPath("//*[@id='mainfeed']/div[2]/div[" + i + "]/div[3]/div/h3/a"));

            //output the newtitles through the textbox tbNews
            tbNews.Text = NewsTitles.Text;

        }
        catch (Exception) // catches error
        {
            //nothing to display for the user so that there's no disruption
        }
    }

    //closes the driver
    driver.Close();

    //quits the driver
    driver.Quit();

}

/// <summary>
/// The resulting method for when the search button is clicked by the user
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>

private void btSearch_Click(object sender, EventArgs e)
{
    //Create a database connection object with the constant defined in program.cs
    OleDbConnection Conn = new OleDbConnection(Program.connString);

    //Opens Connection to the database

```

```
Conn.Open();

//Creates a database command object
OleDbCommand Cmd = new OleDbCommand();

//Creates a connection with the Command Object
Cmd.Connection = Conn;

//A select query to get everything from the table animes from where the animetitle is
entered by the user
Cmd.CommandText = "SELECT * FROM Animes WHERE AnimeTitle=
""+tbSearch.Text+"";

//Creates a reader object and Executes the SQL Query
OleDbDataReader reader = Cmd.ExecuteReader();

//sets the visibility of the label lbCowboyBepop to false (invisible)
lbCowboybebop.Visible = false;

//sets the visibility of the label lbTrigun to false (invisible)
lbTrigun.Visible = false;

//sets the visibility of the label lbNaruto to false (invisible)
lbNaruto.Visible = false;

// if the SQL query contains one or more rows
if (reader.HasRows)
{
    //read all the rows
    reader.Read();

    //sets the variable pic with the value AnimePictures from the table
    var pic = reader["AnimePictures"].ToString();

    //declaring numeric value as an integer
    int numericvalue;

    //if the pic can be parsed as an integer meaning that the variable pic contains no
    url but the placeholder value
    if (int.TryParse(pic, out numericvalue))
    {

        try//Exception handling
        {
            //sets the variable animeID with the value AnimelD from the table
            var animeID = reader["AnimelD"].ToString();

            // Declaring anime url as a string which contains an api link with the variable
            animeID embedded into it
            string animeURL = "https://api.jikan.moe/v3/anime/" + animeID + "/pictures";

            //parses the url as a jsonfile and downloads the jsonfile
            JObject json = JObject.Parse(new WebClient().DownloadString(animeURL));
        }
    }
}
```

```

        //declaring the variable animelink with the value json where it contains a small
picture
        var animelink = json["pictures"][0]["small"];

        //loads the link from the variable animelink into the picturebox
PbAnimePicture4
        PbAnimePicture4.LoadAsync(animelink.ToString());

        //sets the label lbAnimeTitle with the same text as the textbpx tbSearch
lbAnimeTitle.Text = tbSearch.Text;

        //Sets the global variable AnimeName with the label lbAnimeTitle
AnimeName = lbAnimeTitle.Text;

        //sets the visibility of the panel PaAnimePanel to true
PaAnimePanel.Visible = true;

        //Closes the reader object.
reader.Close();

        //Update AnimePictures with the link from animelink. This update will take
place where the AnimeID is the same as the variable animeID
        Cmd.CommandText = "UPDATE Animes SET AnimePictures= "+
animelink.ToString()+" WHERE AnimeID = " + animeID.ToString() + "";

        //Sets the global variable animePic with the variable animelink
animePic = animelink.ToString();

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

    }

    catch(Exception ) //catches error
    {
        //Displays a message box informing the user that the anime could not be
found
        MessageBox.Show("Could not be found");
    }
}

else //if it could not be parsed as an integer this means there is a Url in the table
{
    //sets the visibility of the label lbCowboyBepop to false (invisible)
lbCowboybebop.Visible = false;

    //sets the visibility of the label lbTrigun to false (invisible)
lbTrigun.Visible = false;

    //sets the visibility of the label lbNaruto to false (invisible)
lbNaruto.Visible = false;

    //sets the variable animeID with the value AnimeID from the table
var animeID = reader["AnimeID"].ToString();

```

```

        //closes the reader object
        reader.Close();

        //select the url from the table Animes where the AnimeID is located
        Cmd.CommandText ="SELECT AnimePictures FROM Animes WHERE
        AnimeID=" + animeID + "";

        //Exceutes the SQL query
        Cmd.ExecuteNonQuery();

        //sets the variable animeID with the value from the select query
        var Picture = Cmd.ExecuteScalar();

        //sets the global variable animePic with the url in the varaiable Picture
        animePic = Picture.ToString();

        //loads the url in the global variable animePic in the picturebox PbAnimePicture4
        PbAnimePicture4.LoadAsync(animePic);

        //Sets the global variable AnimeName with the text from the textbox tbSearch
        AnimeName = tbSearch.Text;

        //sets the label IbAnimeTitle with the text from the textbox tbSearch
        IbAnimeTitle.Text = tbSearch.Text;

        //sets the visibility of the panel PaAnimePanel to true (visible)
        PaAnimePanel.Visible = true;

    }

}

/// <summary>
/// The resulting method for when the picture is clicked by the user
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>

private void PbAnimePicture1_Click(object sender, EventArgs e)
{
    //sets the global variable animePic with url as its value
    animePic = @"https://cdn.myanimelist.net/images/anime/7/37911.jpg";

    //sets the global variable AnimeName with the text from the label IbCowboyBepop as
    value
    AnimeName = IbCowboybebop.Text;

    //Creates a variable from the Form FmAnimeInformation
    FmAnimeInformation FmAnime = new FmAnimeInformation();

    // Shows the Form FmAnimeInformation through the variable

```

```
FmAnime.ShowDialog();  
  
}  
/// <summary>  
/// The resulting method for when the picture is clicked by the user  
/// </summary>  
/// <param name="sender"> The <see cref="System.Object"/> that represents the  
sender instance </param>  
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event  
arguments passed to the event handler </param>  
  
private void PbAnimePicture2_Click(object sender, EventArgs e)  
{  
    //sets the global variable animePic with url as its value  
    animePic = @"https://cdn.myanimelist.net/images/anime/8/199681.jpg";  
  
    //sets the global variable AnimeName with the text from the label lbTrigun as value  
    AnimeName = lbTrigun.Text;  
  
    //Creates a variable from the Form FmAnimeInformation  
    FmAnimeInformation FmAnime = new FmAnimeInformation();  
  
    // Shows the Form FmAnimeInformation through the variable  
    FmAnime.ShowDialog();  
}  
  
/// <summary>  
/// The resulting method for when the picture is clicked by the user  
/// </summary>  
/// <param name="sender"> The <see cref="System.Object"/> that represents the  
sender instance </param>  
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event  
arguments passed to the event handler </param>  
  
private void PbAnimePicture3_Click(object sender, EventArgs e)  
{  
    //sets the global variable animePic with url as its value  
    animePic = @"https://cdn.myanimelist.net/images/anime/1/201.jpg";  
  
    // sets the global variable AnimeName with the text from the label lbNaruto as value  
    AnimeName = lbNaruto.Text;  
  
    //Creates a variable from the Form FmAnimeInformation  
    FmAnimeInformation FmAnime = new FmAnimeInformation();  
  
    // Shows the Form FmAnimeInformation through the variable  
    FmAnime.ShowDialog();  
}  
  
/// <summary>  
/// The resulting method for when the picture is clicked by the user  
/// </summary>
```

```
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>

private void PbAnimePicture4_Click(object sender, EventArgs e)
{
    //Creates a variable from the Form FmAnimeInformation
    FmAnimeInformation FmAnime = new FmAnimeInformation();

    // Shows the Form FmAnimeInformation through the variable
    FmAnime.ShowDialog();

}

/// <summary>
/// The resulting method for when the profile button is clicked by the user
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>

private void btProfile_Click(object sender, EventArgs e)
{
    //Creates a variable from the Form FmProfile
    FmProfile FmProfile = new FmProfile();

    // Shows the Form FmProfile through the variable
    FmProfile.ShowDialog();
}

/// <summary>
/// The resulting method for when the back button is clicked by the user
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>

private void btBack_Click(object sender, EventArgs e)
{
    //sets the visibility of the panel PaAnimePanel to false (invisible)
    PaAnimePanel.Visible = false;

    //sets the visibility of the label IbNaruto to true (visible)
    IbNaruto.Visible = true;

    //sets the visibility of the label IbTrigun to true (visible)
    IbTrigun.Visible = true;

    //sets the visibility of the label IbCowboybebop to true (visible)
    IbCowboybebop.Visible = true;
```

```

    }

/// <summary>
/// The resulting method for when the switch button is clicked by the user
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>

public void btSwitch_Click(object sender, EventArgs e)
{
    //Refers to the class DarkmodeSwitch and runs the procedure
    DarkModeSwitch.SwitchFunction();

    //runs the procedure alongside the class DarkModeSwitch
    SetColour();
}

}
}

```

Form: FmAnimeInformation.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;
using Newtonsoft.Json.Linq;
using System.Net;
using System.Text.RegularExpressions;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in
    <c>FmAnimeInformation</c>
    /// </summary>
    public partial class FmAnimeInformation : Form
    {
        /// <summary>
        /// Global variable
        /// </summary>
        public static string AnimePicLoad;
        /// <summary>
        /// Global variable that is declared in <c>FmAnimeInformation</c>to make the form
        accessible to other forms
        /// </summary>
    }
}

```

```
public static FmAnimeInformation fmAnimeInformationInstance;

/// <summary>
///   Required Method for Designer support for <c>FmAnimeInformation</c>
/// </summary>
public FmAnimeInformation()
{
    InitializeComponent();
}

/// <summary>
/// The resulting method when the form loads
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>
private void FmAnimeInformation_Load(object sender, EventArgs e)
{
    //runs the procedure AnimeInformation
    AnimeInformation();

    //runs the procedure SynopsisApi
    SynopsisApi();

}

/// <summary>
/// The resulting method for the procedure AnimeInformation
/// </summary>
private void AnimeInformation()
{
    try
    {
        //sets the label lbAnimeName to the global variable in FmAMS AnimeName
        lbAnimeName.Text = FmAMS.AnimeName;

        //loads the link from the global variable in FmAMS to the picturebox
        PbAnimePicture
        PbAnimePicture.LoadAsync(FmAMS.animePic.ToString());

        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;
    }
}
```

```
//A select query to select the AnimeID from the table Where the AnimeTitle is the
//same as the text in the label lbAnimeName
Cmd.CommandText = "SELECT AnimeID FROM Animes WHERE AnimeTitle= " +
lbAnimeName.Text + "";

//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable ID to the result of the sql query
var ID = Cmd.ExecuteScalar();

//sets the label DbAnimID text to the value ID in string form
DbAnimID.Text = ID.ToString();

//A select query to select the Status from the table Animes where the AnimeID is
//the same as the variable ID
Cmd.CommandText = "SELECT Status FROM AnimeStatus WHERE AnimeID= "
+ ID + "";

//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable status to the result of the sql query
var Status = Cmd.ExecuteScalar();

//sets the label DbStatus text to the value Status in string form
DbStatus.Text = Status.ToString();

//A select query to select the Ratings from the table Animes where the AnimeID is
//the same as the variable ID
Cmd.CommandText = "SELECT Ratings From Animes WHERE AnimeID= " + ID +
"";

//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable Ratings to the result of the sql query
var Ratings = Cmd.ExecuteScalar();

//sets the label DbRatings text to the value Ratings in string form
DbRatings.Text = Ratings.ToString();

//A select query to select the GenreID from the table AnimeGenres where the
//AnimeID is the same as the variable ID
Cmd.CommandText = "SELECT GenreID FROM AnimeGenres WHERE
AnimeID= " + ID + "";

//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable GenreID to the result of the sql query
var GenreID = Cmd.ExecuteScalar();
```

```
//A select query to select the Genre from the table Genres where the GenreID is
//the same as the variable GenreID
Cmd.CommandText = "SELECT Genre FROM Genres WHERE GenreID= " +
GenreID + "";

//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable Genre to the result of the sql query
var Genre = Cmd.ExecuteScalar();

//sets the textbox tbGenres text to the value Genre in string form
tbGenres.Text = Genre.ToString();

//A select query to select the StudioID from the table AnimeStudios where the
//AnimelD is the same as the variable ID
Cmd.CommandText = "SELECT StudioID FROM AnimeStudios WHERE
AnimelD= " + ID + "";

//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable StudioID to the result of the sql query
var StudioID = Cmd.ExecuteScalar();

//A select query to select the Studio from the table Studios where the StudioID is
//the same as the variable StudioID
Cmd.CommandText = "SELECT Studio From Studios WHERE StudioID= " +
StudioID + "";

//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable Studio to the result of the sql query
var Studio = Cmd.ExecuteScalar();

//sets the label DbStudios text to the value Studio in string form
DbStudios.Text = Studio.ToString();

//A select query to select the Episodes from the table Animes where the AnimelD
//is the same as the variable ID
Cmd.CommandText = "SELECT Episodes FROM Animes WHERE AnimelD=" +
ID + "";

//sets the variable Episodes to the result of the sql query
var Episodes = Cmd.ExecuteScalar();

//sets the label DbEpisodes text to the value Episodes in string form
DbEpisodes.Text = Episodes.ToString();
}

catch //catches error
{
    //Displays a message box informing the user that nothing was found
    MessageBox.Show("Nothing was found");
}
```

```
        }
    }
/// <summary>
/// The resulting method for the procedure SynopsisApi
/// </summary>
private void SynopsisApi()
{
    //Create a database connection object with the constant defined in program.cs
    OleDbConnection Conn = new OleDbConnection(Program.connString);

    //Opens Connection to the database
    Conn.Open();

    //Creates a database command object
    OleDbCommand Cmd = new OleDbCommand();

    //Creates a connection with the Command Object
    Cmd.Connection = Conn;

    //A select query to select the AnimeID from the table Animes Where the anime title is
    //the same as the label lbAnimeName
    Cmd.CommandText = "SELECT AnimeID FROM Animes WHERE AnimeTitle= " +
        lbAnimeName.Text + "';

    //Executes the SQL query
    Cmd.ExecuteNonQuery();

    //sets the variable ID to the result of the sql query
    var ID = Cmd.ExecuteScalar();

    //A select query to select everything from the table Anime where the location of the
    //AnimeTitle is (which contains the label lbAnimeName)
    Cmd.CommandText = "SELECT * FROM Animes WHERE AnimeTitle= " +
        lbAnimeName.Text + "';

    //Creates a reader object and Executes the SQL Query
    OleDbDataReader reader = Cmd.ExecuteReader();

    // if the SQL query contains one or more rows
    if (reader.HasRows)
    {
        //read all the rows
        reader.Read();

        //sets the variable pic with the value AnimeSynopsis from the table Animes
        var Asynopsis = reader["AnimeSynopsis"].ToString();

        //declaring numeric value as an integer
        int numericvalue;

        //if the Asynopsis can be parsed as an integer meaning that the variable Asynopsis
        //contains no text but the placeholder value
        if (int.TryParse(Asynopsis, out numericvalue))
        {

```

```

try //Exception Handling
{
    // Declaring anime url as a string which contains an api link with the variable
    ID embedded into it
    string animeURL = "https://api.jikan.moe/v4/anime/" + ID.ToString();

    //parses the url as a jsonfile and downloads the jsonfile
    JObject json = JObject.Parse(new WebClient().DownloadString(animeURL));

    //declaring the variable text with the value json where it contains synopsis
    var text = json["data"]["synopsis"];

    //Sets the rich text box rtbAnimeSynopsis to the variable text in string form
    rtbAnimeSynopsis.Text = text.ToString();

    //sets the variable AnimeSynopsis to the value of the variable text in string
    form
    var AnimeSynopsis = text.ToString();

    //Closes the reader object
    reader.Close();

    // sets the variable EditedSynopsis with the value AnimeSynopsis where all
    the apostrophe
    // is replaced with an empty space so that the sql query can work without an
    error
    var EditedSynopsis = AnimeSynopsis.Replace("'", " ");

    //Update AnimeSynopsis with the synopsis from EditedSynopsis. This update
    will take place where the location of the AnimeID is (which contains the variable ID)
    Cmd.CommandText = "UPDATE Animes SET AnimeSynopsis= " +
    EditedSynopsis + " WHERE AnimeID = " + ID.ToString() + "";

    //Executes the SQL query
    Cmd.ExecuteNonQuery();
}
catch(Exception) //catches error
{
    //Displays a message box informing the user that the synopsis could not be
    found
    MessageBox.Show("The synopsis could not be found");
}

//if it could not be parsed as an integer this means that there is a synopsis in the
table
else
{
    //sets the variable animeID with the value AnimeID from the table
    var animeID = reader["AnimeID"].ToString();

    //closes the reader object
    reader.Close();
}

```

```
//A select query to select the AnimeSynopsis in where the location of the
AnimeID is the same as the variable animeID
Cmd.CommandText = "SELECT AnimeSynopsis FROM Animes WHERE
AnimeID=" + animeID + "";

//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable Synopsis to the result of the sql query
var Synopsis = Cmd.ExecuteScalar();

//Sets the value of the rich text box rtbAnimeSynopsis to value of the variable
Synopsis in string form
rtbAnimeSynopsis.Text = Synopsis.ToString();
}

}

/// <summary>
/// The resulting method for when the 'Add to favourites' button is clicked by the user
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>

private void btFavourites_Click(object sender, EventArgs e)
{
    try //catches error
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //if the user did not select a rating from the combo box cbRatings
        if (cbRate.SelectedIndex == -1)
        {
            //Displays a message box informing the user to select a rating
            MessageBox.Show("Please select a rating!");
        }

        else
        {
            //sets the variable Username to the global variable Username from FmLogin
            var Username = FmLogin.Username;
        }
    }
}
```

```
//A select query to select the UserID in the location where the username is and
it is the same value as the variable Username
Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" +
Username + "";

//sets the variable UserID to the result of the sql query
var UserID = Cmd.ExecuteScalar();

//sets the label lbAnimeName to the global variable AnimeName from FmAMS
lbAnimeName.Text = FmAMS.AnimeName;

//A select query to count the entries where it contains the AnimeTitle that is the
//same as lbAnimeName and the userID which is the same as the variable
userID
Cmd.CommandText = "SELECT count(*) From UserFavourites WHERE
AnimeTitle=" + lbAnimeName.Text + " AND UserID=" + UserID + "";

//sets the integer count to the result of the sql query in an integer format
int Count = Convert.ToInt32(Cmd.ExecuteScalar());

//if the count is more than 0 meaning it already exists in the table UserFavourites
if (Count > 0)
{
    //Displays a message box informing the user that the Anime is already in their
favourites
    MessageBox.Show("Anime is already favourited");
}

//if it does not exist in the table UserFavourites
else
{
    //An insert query to insert the variable UserID, the label DbAnimeID, the label
lbAnimeName and the combo box value from cbRate into the table UserFavourites
    Cmd.CommandText = "INSERT INTO UserFavourites(UserID, AnimID,
AnimeTitle, Ratings) VALUES(" + UserID + "," + DbAnimeID.Text + "," + lbAnimeName.Text
+ "," + cbRate.Text + ")";

    //Executes the SQL query
    Cmd.ExecuteNonQuery();

    //Displays a message box informing the user that the anime has been added
to WatchList
    MessageBox.Show("Added to favourites!");
}

}

catch(Exception) //catches an error
{
    //Displays a messagebox informing the user that it was invalid.
    MessageBox.Show("Invalid");
}
```

```

        }

/// <summary>
/// The resulting method for when the 'Add to Watchlist' button is clicked by the user
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>
private void btWatchList_Click(object sender, EventArgs e)
{
    try//Catches error
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //if user did not select a watch status from the combobox cbWatchStatus
        if (cbWatchStatus.SelectedIndex == -1)

        {
            //Displays a message box informing the user to select a watch status
            MessageBox.Show("Please select a watch status");
        }

        //if the user did not select a rating
        else if (cbRate.SelectedIndex == -1)
        {
            //Displays a message box informing the user to select a rating
            MessageBox.Show("Please select a rating");
        }

        //otherwise
        else
        {
            //sets the variable Username to the global variable Username from FmLogin
            var Username = FmLogin.Username;

            //A select query to select the UserID in the location where the username is and
            it is the same value as the variable Username
            Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" +
            Username + "''";

            //Executes the SQL query
        }
    }
}

```

```

        Cmd.ExecuteNonQuery();

        //sets the variable UserID to the result of the sql query
        var UserID = Cmd.ExecuteScalar();

        //A select query to count the entries where it contains the AnimeTitle that is the
        //same as lbAnimeName and the userID which is the same as the variable
        userID
        Cmd.CommandText = "SELECT count(*) From UserWatchList WHERE
        AnimeTitle= @AnimeTitle AND UserID= @UserID";

        //Parameterised SQL which contains the label lbAnimeName
        Cmd.Parameters.Add("@AnimeTitle", OleDbType.Char).Value =
        lbAnimeName.Text;

        //Parameterised SQL which contains the variable UserID
        Cmd.Parameters.Add("@UserID", OleDbType.Integer).Value = UserID;

        //sets the integer count to the result of the sql query in an integer format
        int Count = Convert.ToInt32(Cmd.ExecuteScalar());

        //if the count is more than 0 meaning it already exists in the table UserWatchList
        if (Count > 0)
        {
            //Displays a message box informing the user that the Anime is already in the
            user's watch list
            MessageBox.Show("Anime is already in Watchlist!");
        }

        //if it does not exist in the table UserFavourites
        else
        {
            //An insert query to insert the variable UserID, the label DbAnimeID, the label
            lbAnimeName, the combo box value
            //from cbWatchStatus, the textbox value tbEpisodesWatched and the combo
            box value from cbRate into the table UserWatchList
            Cmd.CommandText = "INSERT INTO UserWatchList(UserID, AnimeID,
            AnimeTitle, WatchStatus, EpisodesWatched, Ratings) VALUES(" + UserID + ", " +
            DbAnimeID.Text + "," + lbAnimeName.Text + "," + cbWatchStatus.Text + ", " +
            tbEpisodesWatched.Text + "," + cbRate.Text + ")";

            //Executes the sql query
            Cmd.ExecuteNonQuery();

            //Displays a message box informing the user that the anime has been added
            to their watch list
            MessageBox.Show("Added to watchlist!");

        }
    }
    catch(Exception) //catches error
    {
        //Displays a messagebox informing the user that it was invalid.
    }
}

```

```
        MessageBox.Show("Invalid");
    }

/// <summary>
/// The resulting method for when the textbox tbEpisodeWatched is validating the user's
input
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>
private void tbEpisodeWatched_Validating(object sender, CancelEventArgs e)
{
    //Regex validation for symbols
    var hasSymbols = new Regex(@"[!@#$^&*()_+=\{\}];:<>|./?,-]+");

    //Regex validation for numbers
    var hasNumber = new Regex(@"[0-9]+");

    //if the text in the textbox tbEpisodesWatched does not contain numbers
    if (!hasNumber.IsMatch(tbEpisodesWatched.Text))
    {

        //Displays a message box informing the user to enter a number
        MessageBox.Show("Please enter a number!");

        //sets the text in the textbox tbEpisodesWatched to the text 0.
        tbEpisodesWatched.Text = "0";
    }

    //if the text in the textbox tbEpisodes Watched contains symbols
    else if (hasSymbols.IsMatch(tbEpisodesWatched.Text))
    {
        //Displays a message box informing the user to enter a number
        MessageBox.Show("Please enter a number!");

        //sets the text in the textbox tbEpisodesWatched to the text 0.
        tbEpisodesWatched.Text = "0";
    }

    //if the input is fully valid
    else
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
    }
}
```

```
Cmd.Connection = Conn;

//A select query to select episodes from the table Animes from the location of
the AnimeTitle
Cmd.CommandText = "SELECT Episodes FROM Animes WHERE AnimeTitle=
@AnimeTitle";

//Parameteriesed SQL that contains the label lbAnimeName
Cmd.Parameters.Add("@AnimeTitle", OleDbType.Char).Value =
lbAnimeName.Text;

//sets the integer Episodes to the result of the sql query in an integer format
int Episodes = Convert.ToInt32(Cmd.ExecuteScalar());

//sets the integer EpisodesEntered to the value of the text
//tbEpisodesWatched in an integer format
int EpisodesEntered = Convert.ToInt32(tbEpisodesWatched.Text);

// if the episodes entered by the user is greater than the total episodes of the
anime
if (EpisodesEntered > Episodes)
{
    //Displays a message box informing the user that the episodes entered are
out range
    MessageBox.Show("Out of range!");

    //sets the text from the textbox tbEpisodesWatched with the variable
    //which contains the total episodes of the anime
    tbEpisodesWatched.Text = Episodes.ToString();
}

//if the episodes entered by the user is less than 0
else if (EpisodesEntered < 0)
{
    //Displays a message box informing the user that the episodes entered are
out range
    MessageBox.Show("Out of range!");

    //sets the text from the textbox tbEpisodesWatched to 0 in text form
    tbEpisodesWatched.Text = "0";
}

//Closes the connection to the database
Conn.Close();
}

/// <summary>
/// The resulting method for when the combo box cbWatchStatus is selected
/// </summary>
```

```
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>
private void cbWatchStatus_SelectedIndexChanged(object sender, EventArgs e)
{
    //Create a database connection object with the constant defined in program.cs
    OleDbConnection Conn = new OleDbConnection(Program.connString);

    //Opens Connection to the database
    Conn.Open();

    //Creates a database command object
    OleDbCommand Cmd = new OleDbCommand();

    //Creates a connection with the Command Object
    Cmd.Connection = Conn;

    //A select query to select episodes from the table Animes from the location of the
    AnimeTitle
    Cmd.CommandText = "SELECT Episodes FROM Animes WHERE AnimeTitle=@AnimeTitle";

    //Parameterised SQL that contains the label lbAnimeName
    Cmd.Parameters.AddWithValue("@AnimeTitle", OleDbType.Char).Value = lbAnimeName.Text;

    //sets the integer Episodes to the result of the sql query in an integer format
    int Episodes = Convert.ToInt32(Cmd.ExecuteScalar());

    //if the user selects "Completed" in the drop down list for cbWatchStatus
    if (cbWatchStatus.Text == "Completed")
    {
        //sets the text from the textbox tbEpisodesWatched with the variable Episodes
        //which contains the total episodes of the anime
        tbEpisodesWatched.Text = Episodes.ToString();
    }

    //if the user selects "Not Yet Watched" in the drop down list for cbWatchStatus
    else if (cbWatchStatus.Text == "Not Yet Watched")
    {
        //sets the text from the textbox tbEpisodesWatched to 0 in text form
        tbEpisodesWatched.Text = "0";
    }
}
```

Form: FmProfile.cs

```
using System;
Centre Number:32455
137
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in <c>FmProfile</c>
    /// </summary>
    public partial class FmProfile : Form
    {
        /// <summary>
        /// Required Method for Designer support for <c>FmAMS</c>
        /// </summary>
        public FmProfile()
        {
            InitializeComponent();
        }

        /// <summary>
        /// The resulting method for when the Form loads
        /// </summary>
        /// <param name="sender"> The <see cref="System.Object"/> that represents the
        sender instance </param>
        /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
        arguments passed to the event handler </param>
        private void FmProfile_Load(object sender, EventArgs e)
        {
            //Sets the label lbUsername to the global variable declared
            //in the form FmLogin which contains the user's username
            lbUsername.Text = FmLogin.Username;

            //Create a database connection object with the constant defined in program.cs
            OleDbConnection Conn = new OleDbConnection(Program.connString);

            //Opens Connection to the database
            Conn.Open();

            //Creates a database command object
            OleDbCommand Cmd = new OleDbCommand();

            //Creates a connection with the Command Object
            Cmd.Connection = Conn;

            //A select query to select the UserID From the table users where the location of
            //the username is (the username is the global variable from FmLogin Username)
            Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" + 
            FmLogin.Username + "";
        }
    }
}
```

```
//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable UserID to the result of the Sql query
var UserID = Cmd.ExecuteScalar();

//sets the label DbUserID to the variable UserID
DbUserID.Text = UserID.ToString();

//A select query to select the Forename From the table users where the location of
//the username is (the username is the global variable from FmLogIn Username)
Cmd.CommandText = "SELECT Forename FROM Users WHERE Username=" +
FmLogIn.Username + "';

//Executes the sql query
Cmd.ExecuteNonQuery();

//sets the variable Forename to the result of the Sql query
var Forename = Cmd.ExecuteScalar();

//sets the label DbForename to the variable Forename
DbForename.Text = Forename.ToString();

//A select query to select the Surname From the table users where the location of
//the username is (the username is the global variable from FmLogIn Username)
Cmd.CommandText = "SELECT Surname FROM Users WHERE Username=" +
FmLogIn.Username + "';

//Executes the sql query
Cmd.ExecuteNonQuery();

//sets the variable Surname to the result of the Sql query
var Surname = Cmd.ExecuteScalar();

//sets the label DbSurname to the variable Forename
DbSurname.Text = Surname.ToString();

//A select query to select the UserType From the table users where the location of
//the username is (the username is the global variable from FmLogIn Username)
Cmd.CommandText = "SELECT UserType FROM Users WHERE Username=" +
FmLogIn.Username + "';

//Executes the sql query
Cmd.ExecuteNonQuery();

//sets the variable UserType to the result of the Sql query
var UserType = Cmd.ExecuteScalar();

//sets the label DbUserType to the variable Forename
DbUserType.Text = UserType.ToString();
}

/// <summary>
/// The resulting method for when the reset button is clicked

```

```
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btReset_Click(object sender, EventArgs e)
{
    //Creates a variable from the Form FmReset
    FmReset fmReset = new FmReset();

    // Shows the Form FmReset through the variable fmReset
    fmReset.ShowDialog();
}

/// <summary>
/// The resulting method for when the watch list button is clicked
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btWatchList_Click(object sender, EventArgs e)
{
    //Creates a variable from the Form FmUserWatchList
    FmUserWatchList fmUserWatchList = new FmUserWatchList();

    // Shows the Form FmUserWatchList through the variable fmUserWatchList
    fmUserWatchList.ShowDialog();
}

/// <summary>
/// The resulting method for when the favourites button is clicked
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btFavourites_Click(object sender, EventArgs e)
{
    //Creates a variable from the Form FmUserFavourites
    FmUserFavourites fmUserFavourites = new FmUserFavourites();

    // Shows the Form FmUserWatchList through the variable fmUserFavourites
    fmUserFavourites.ShowDialog();
}

/// <summary>
/// The resulting method for when the stats button is clicked
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
```

```
private void btStats_Click(object sender, EventArgs e)
{
    //Creates a variable from the Form FmUserStats
    FmUserStats fmUserStats = new FmUserStats();

    // Shows the Form FmUserStats through the variable fmUserStats
    fmUserStats.ShowDialog();
}
```

Form: FmReset.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;
using System.Text.RegularExpressions;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in <c>FmReset</c>
    /// </summary>
    public partial class FmReset : Form
    {
        /// <summary>
        /// Required Method for Designer support for <c>FmReset</c>
        /// </summary>
        public FmReset()
        {
            InitializeComponent();
        }
        /// <summary>
        /// The resulting method when the user clicks on the Find button
        /// </summary>
        /// <param name="sender"> The <see cref="System.Object"/> that represents the
        sender instance </param>
        /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
        arguments passed to the event handler </param>
        ///
        private void btFind_Click(object sender, EventArgs e)
        {
            //Regex validation for Emails
            var EmailValid = new Regex(@"^([a-zA-Z0-9_.\|-]+)+@(([a-zA-Z0-9\-])+\.)([a-zA-Z]{2,3})+$");
            //Create a database connection object with the constant defined in program.cs
        }
    }
}
```

```
OleDbConnection Conn = new OleDbConnection(Program.connString);

//Opens Connection to the database
Conn.Open();

//Creates a database command object
OleDbCommand Cmd = new OleDbCommand();

//Creates a connection with the Command Object
Cmd.Connection = Conn;

//Sets the Variable UserName to the global variable Username from FmLogIn
var UserName = FmLogIn.Username.ToString();

//Select email from the table users from the username that is given
Cmd.CommandText = "SELECT Email FROM Users WHERE Username=" + 
UserName + """;

//Creates a reader object and Executes the SQL Query
OleDbDataReader reader = Cmd.ExecuteReader();

// if the SQL query contains one or more rows
if (reader.HasRows)
{
    //read all the rows
    reader.Read();

    //If the email that is entered in the textbox tbEmail
    //does not match with the Selected Email
    if (tbEmail.Text != reader["Email"].ToString())
    {
        //Displays a message window stating that the User's Entered Email
        //is not correct/does not match up with the selected email
        MessageBox.Show("This Email is incorrect");

        //Clears the text that was entered in the textbox tbEmail
        tbEmail.Clear();
    }

    // if the email that is entered in the textbox tbEmail
    //does not match regex validation for emails
    else if (!EmailValid.IsMatch(tbEmail.Text))
    {
        //Displays a message window stating that the User's
        //entered email is not valid
        MessageBox.Show("This is not a valid email address");

    }

    //If the entered Email matches the selected email and
    //matches regex validation for emails
    else

```

```
{  
    //Set the visibility of the label lbNewPassword to true(makes it visible to the user)  
    lbNewPassword.Visible = true;  
  
    //Set the visibility of the label lbConfrimNewPassword to true(makes it visible to  
    the user)  
    lbConfirmNewPassword.Visible = true;  
  
    //Set the visibility of the textbox tbNewPassword to true(makes it visible to the  
    user)  
    tbNewPassword.Visible = true;  
  
    //Set the visibility of the textbox tbConfirmNewPassword to true(makes it visible  
    to the user)  
    tbConfirmNewPassword.Visible = true;  
  
    //Closes the reader command object  
    reader.Close();  
}  
}  
  
}  
  
/// <summary>  
/// The resulting method when the user clicks on the Change button  
/// </summary>  
/// <param name="sender"> The <see cref="System.Object"/> that represents the  
sender instance </param>  
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event  
arguments passed to the event handler </param>  
///  
private void btChange_Click(object sender, EventArgs e)  
{  
    //Regex validation for upper case letters  
    var hasUpperChar = new Regex(@"[A-Z]+");  
  
    //Regex validation for lower case letters  
    var hasLowerChar = new Regex(@"[a-z]+");  
  
    //Regex validation for numbers  
    var hasNumber = new Regex(@"[0-9]+");  
  
    //Regex validation for symbols  
    var hasSymbols = new Regex(@"[!@#$^&*()_+=\{\}\};<>./?,,-]+");  
  
    //Create a database connection object with the constant defined in program.cs  
    OleDbConnection Conn = new OleDbConnection(Program.connString);  
  
    //Opens Connection to the database  
    Conn.Open();  
  
    //Creates a database command object  
    OleDbCommand Cmd = new OleDbCommand();  
  
    //Creates a connection with the Command Object
```

```
Cmd.Connection = Conn;

//if the text entered in the textbox tbNewPassword does not match with the textbox
tbConfirmNewPassword
if (tbNewPassword.Text != tbConfirmNewPassword.Text)
{
    //Displays a message window stating that the User
    //needs to enter the same password in both textboxes
    MessageBox.Show("Please enter the same password in both fields");
}

//if the text entered in the textbox tbNewPassword is empty
else if (tbNewPassword.Text == "")
{
    //Displays a message window stating that the User
    //needs to enter a password
    MessageBox.Show("Please enter a password");
}

//if the text entered in the textbox tbConfirmNewPassword is empty
else if (tbConfirmNewPassword.Text == "")
{
    //Displays a message window stating that the User
    //needs to enter a confirmation of new password
    MessageBox.Show("Enter confirmation of new password");
}

//if the text entered in the textbox tbConfirmNewPassword does not match the regex
validation for
//upper case letters or if the text entered in the textbox tbNewPassword
//does not match the regex validation for Upper case letters
else if (!hasUpperChar.IsMatch(tbConfirmNewPassword.Text) ||
!hasUpperChar.IsMatch(tbNewPassword.Text))
{
    //Displays a message window stating that the User's Entered password
    //must contain atleast one upper case letter
    MessageBox.Show("Your new password must contain atleast one upper case
letter");
}

//if the text entered in the textbox tbConfirmNewPassword does not match the regex
validation for
//lower case letters or if the text entered in the textbox tbNewPassword
//does not match the regex validation for lower case letters
else if (!hasLowerChar.IsMatch(tbConfirmNewPassword.Text) ||
!hasLowerChar.IsMatch(tbNewPassword.Text))
{
    //Displays a message window stating that the User's Entered password
    //must contain atleast one lower case letter
    MessageBox.Show("Your new password must contain atleast one lower case
letter");
}
```

```
//if the text entered in the textbox tbConfirmNewPassword does not match the regex
validation for
//numbers or if the text entered in the textbox tbNewPassword
//does not match the regex validation for numbers
else if (!hasNumber.IsMatch(tbConfirmNewPassword.Text)||

!hasNumber.IsMatch(tbNewPassword.Text))
{
    //Displays a message window stating that the User's Entered password
    //must contain atleast one numeric value
    MessageBox.Show("Your new password must contain atleast one upper numeric
value");

}

//if the text entered in the textbox tbConfirmNewPassword does not match the regex
validation for
//symbols or if the text entered in the textbox tbNewPassword
//does not match the regex validation for symbols
else if
(!hasSymbols.IsMatch(tbConfirmNewPassword.Text)||!hasSymbols.IsMatch(tbNewPasswor
d.Text))
{
    //Displays a message window stating that the User's Entered password
    //must contain atleast one symbol
    MessageBox.Show("Your new password must contain atleast one symbol");
}

//if all the requirements are met for regex validation and the textboxes are not empty
else
{
    //Set the variable username to the global variable in FmLogin
    var UserName = FmLogin.Username.ToString();

    //Update the users password with the new hashed version of the confirmed
    //password where the username is given
    Cmd.CommandText = "UPDATE Users SET Pword=" +
Encryption.HashPassword(tbConfirmNewPassword.Text) + "WHERE Username=" +
UserName + "';

    //Execute the sql statement
    Cmd.ExecuteNonQuery();

    //Displays a message window stating that the User's password
    //has been changed successfully
    MessageBox.Show("Password successfully changed");
}
}
}
```

Form: FmUserFavourites.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in
    <c>FmUserFavourites</c>
    /// </summary>
    public partial class FmUserFavourites : Form
    {
        /// <summary>
        /// Required Method for Designer support for <c>FmUserFavourites</c>
        /// </summary>
        public FmUserFavourites()
        {
            InitializeComponent();
        }

        /// <summary>
        /// The resulting method for when the Form loads
        /// </summary>
        /// <param name="sender"> The <see cref="System.Object"/> that represents the
        sender instance </param>
        /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
        arguments passed to the event handler </param>
        private void FmUserFavourites_Load(object sender, EventArgs e)
        {
            //Sets the label lbUsername to the global variable declared in the form FmLogin
            which contains th user's username
            lbUsername.Text = FmLogin.Username;

            //Create a database connection object with the constant defined in program.cs
            OleDbConnection Conn = new OleDbConnection(Program.connString);

            //Opens Connection to the database
            Conn.Open();

            //Creates a database command object
            OleDbCommand Cmd = new OleDbCommand();

            //Creates a connection with the Command Object
            Cmd.Connection = Conn;

            //A select query to select the UserID From the table users where the location of

```

```
//the username is (the username is the label lbUsername)
Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" + 
lbUsername.Text + "";

//sets the variable UserID to the result of the Sql query
var UserID = Cmd.ExecuteScalar();

//A select query to select everything from the table UserFavourites where
//the location of the UserID is declared as the variable UserID
Cmd.CommandText = "SELECT * From UserFavourites WHERE UserID=" + UserID
+ "";

//Creates A connection between the datasource(select query) and dataset to Add
rows or refresh rows
OleDbDataAdapter da = new OleDbDataAdapter(Cmd);

//Creates a table that will be filled with data from the select query
DataTable table = new DataTable();

//the data adapter da adds rows to the table
da.Fill(table);

// sets the datagridviewer UserFavouritesGrid source to that of the DataTable table
// that is filled with data from the select query
UserFavouritesGrid.DataSource = table;

// Closes the connection to the database
Conn.Close();

//Displays the DataTable
UserFavouritesGrid.Show();

//Runs the procedure AutoFillSearch
AutoFillSearch();
}

/// <summary>
/// The resulting method for the procedure AutoFillSearch
/// </summary>
private void AutoFillSearch()
{
    try //Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;
    }
}
```

```
//A select query to select the UserID From the table users where the location of
//the username is (the username is the label lbUsername)
Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" +
lbUsername.Text + "";

//sets the variable UserID to the result of the Sql query
var UserID = Cmd.ExecuteScalar();

//A select query to select AnimeTitle from the table UserFavourites where
//the location of the UserID is declared as the variable UserID
Cmd.CommandText = "SELECT AnimeTitle FROM UserFavourites WHERE
UserID=" + UserID + "";

//Defines the autocompletefeature that holds a collection of strings, AutoFill
AutoCompleteStringCollection AutoFill = new AutoCompleteStringCollection();

//Creates a reader object and Executes the SQL Query
OleDbDataReader reader = Cmd.ExecuteReader();

//if the reader object has found rows
if (reader.HasRows)
{
    //while the reader object is reading the rows in the column AnimeTitle
    while (reader.Read())
    {

        //Add the rows that are in the column AnimeTitle to AutoFill
        AutoFill.Add(reader["AnimeTitle"].ToString());
    }
}

//otherwise
else
{
    //Display a messagebox informing the user that there was no AnimeTitles found
    MessageBox.Show("Anime title not found!");
}

//Closes the reader object.
reader.Close();

//Sets the autocomplete source of the textbox tbSearch to AutoFill which contains
all the AnimeTitles
tbSearch.AutoCompleteCustomSource = AutoFill;

//Closes the connection to the database
Conn.Close();
}
catch(Exception) //catches error
{
    //Displays a message box informing the user that it could not loadAutoFill
    MessageBox.Show("Could not load");
}
```

```

        }

/// <summary>
/// The resulting method when the user clicks on a cell from the grid
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>
private void UserFavouritesGrid_CellContentClick(object sender,
DataGridViewCellEventArgs e)
{
    //Create a database connection object with the constant defined in program.cs
    OleDbConnection Conn = new OleDbConnection(Program.connString);

    //Opens Connection to the database
    Conn.Open();

    //Creates a database command object
    OleDbCommand Cmd = new OleDbCommand();

    //Creates a connection with the Command Object
    Cmd.Connection = Conn;

    //if the cell's row is greater than or equal to 0
    if (e.RowIndex >= 0)
    {

        //stores the cell's row into a datagridviewrow variable 'row'
        DataGridViewRow row = UserFavouritesGrid.Rows[e.RowIndex];

        //Sets the textbox tbUserID to the cell value under the column UserID
        tbUserID.Text = row.Cells["UserID"].Value.ToString();

        //Sets the textbox tbAnimelD to the cell value under the column AnimeID
        tbAnimelD.Text = row.Cells["AnimelD"].Value.ToString();

        //Sets the textbox tbAnimeTitle to the cell value under the column AnimeTitle
        tbAnimeTitle.Text = row.Cells["AnimeTitle"].Value.ToString();

        //Sets the combobox cbRatings to the cell value under the column Ratings
        cbRatings.Text = row.Cells["Ratings"].Value.ToString();

        //A select query to select the AnimePictures from the table Animes where the
        //location of the AnimeTitle is declared as tbAnimeTitle
        Cmd.CommandText = "SELECT AnimePictures FROM Animes WHERE
        AnimeTitle=" + tbAnimeTitle.Text + "';

        //sets the variable AnimePic to the result of the Sql query
        var AnimePic = Cmd.ExecuteScalar();

        //loads the url in the variable AnimePic to the picturebox PbAnimePicture
    }
}

```

```

        PbAnimePicture.LoadAsync(AnimePic.ToString()));

    }

/// <summary>
/// The resulting method when the user presses a key in the textbox tbSearch
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>
private void tbSearch_KeyPress(object sender, KeyPressEventArgs e)
{
    //Create a database connection object with the constant defined in program.cs
    OleDbConnection Conn = new OleDbConnection(Program.connString);

    //Opens Connection to the database
    Conn.Open();

    //Creates a database command object
    OleDbCommand Cmd = new OleDbCommand();

    //Opens Connection to the database
    Cmd.Connection = Conn;

    //A select query to select the UserID From the table users where the location of
    //the username is (the username is the label lbUsername)
    Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" +
    lbUsername.Text + "';

    //sets the variable UserID to the result of the Sql query
    var UserID = Cmd.ExecuteScalar();

    //if the textbox tbSearch is not empty
    if (tbSearch.Text != "")
    {
        try //Exception Handling
        {
            // Gets value entered from the user to indicate the size of the columns and fill it
            // with data
            UserFavouritesGrid.AutoSizeColumnsMode =
            DataGridViewAutoSizeColumnsMode.Fill;

            //Creates A connection between the datasource(select query) and dataset to
            //Add rows or refresh rows
            OleDbDataAdapter Adapt = new OleDbDataAdapter();

            //Creates an empty set for data to be filled
            DataSet DS = new DataSet();

            // Creates view of the the data stored in the dataset
            DataView DV = new DataView();

```

```
// A select query that gets everything from the table UserFavourites where
// the location of the UserID is, and is declared as the variable UserID, when the
user
    // types a letter it is checked and compared to entries in the table
    // to determine whether a specific character string matches a specified pattern
    // in the column AnimeTitle
    string SQL = "SELECT * FROM UserFavourites WHERE AnimeTitle LIKE '" +
tbSearch.Text + "%' AND UserID=" + UserID + "";
    //Creates a database command object that contains the connection to the
    //database and the string containing the select query
    Cmd = new OleDbCommand(SQL, Conn);
    //Creates A connection between the datasource(select query) and dataset to
    //refresh rows from the command
    Adapt = new OleDbDataAdapter(Cmd);
    //Fills the DataSet with information from the select query
    Adapt.Fill(DS);
    // Creates a view of the data stored in the dataset
    DV = new DataView(DS.Tables[0]);
    // sets the datagridviewer UserFavouritesGrid source to that of the DataView
    // that is filled with data from the select query
    UserFavouritesGrid.DataSource = DV;
    //Closes the connection to the database
    Conn.Close();
}
catch(Exception) //catches error
{
    //Displays a message box informing the user that their input is invalid
    MessageBox.Show("Invalid input");
}
}

//if the textbox tbSearch is empty
else if (tbSearch.Text == "")
{
    //A select query to select everything from the table UserFavourites where
    //the location of the UserID is declared as the variable UserID
    Cmd.CommandText = "SELECT * FROM UserFavourites WHERE UserID=" +
UserID + "";
    //Creates A connection between the datasource(select query) and dataset
    //to Add rows or refresh rows
    OleDbDataAdapter da = new OleDbDataAdapter(Cmd);
    //Creates a table that will be filled with data from the select query
    DataTable table = new DataTable();
    //the data adapter da adds rows to the table
    da.Fill(table);
}
```

```
// sets the datagridviewer UserFavouritesGrid source to that of the DataTable table
// that is filled with data from the select query
UserFavouritesGrid.DataSource = table;

//Closes the connection to the database
Conn.Close();

//Displays the DataTable
UserFavouritesGrid.Show();
}

}

/// <summary>
/// The resulting method for when the user clicks on the update button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>

private void btUpdate_Click(object sender, EventArgs e)
{
    try //Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //if the user did not select a rating
        if (cbRatings.SelectedIndex == -1)
        {
            //Displays a message box informing the user to select a rating
            MessageBox.Show("Please select a rating");
        }
        else
        {
            //A select query to select the UserID From the table users where the location of
            //the username is (the username is the label lbUsername)
            Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" +
            lbUsername.Text + "";

            //sets the variable UserID to the result of the Sql query
            var UserID = Cmd.ExecuteScalar();

            //Updates the information contained in Ratings with the combobox cbRatings.
        }
    }
}
```

```

        //This update will take place in the location where the UserID is as the same as
        the variable UserID
        //and where the AnimeID is same as the textbox tbAnimeID
        Cmd.CommandText = "UPDATE UserFavourites SET Ratings=" +
cbRatings.Text + " WHERE UserID=" + UserID + " AND AnimeTitle=" + tbAnimeTitle.Text + +
""";
        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a messagebox informing the user that their favourites list has been
        updated
        MessageBox.Show(" Favourites Updated!");
    }
}

catch (Exception) //catches error
{
    //Displays a messagebox informing the user that their favourites list could not be
    updated
    MessageBox.Show("Could not update Favourites!");
}
}

/// <summary>
/// The resulting method for when the user clicks on the Delete button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
    sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
    arguments passed to the event handler </param>
private void btDelete_Click(object sender, EventArgs e)
{
    try //Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //A select query to select the UserID From the table users where the location of
        //the username is (the username is the label lbUsername)
        Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" +
lbUsername.Text + "}";

        //sets the variable UserID to the result of the Sql query
        var UserID = Cmd.ExecuteScalar();
    }
}

```

```
//The delete query will happen on the row containing the UserID
//that's the same as the variable UserID and AnimeTitle that's the same as the
textbox tbAnimeTitle
Cmd.CommandText = "DELETE FROM UserFavourites WHERE UserID=" +
UserID + " AND AnimeTitle=" + tbAnimeTitle.Text + "";
//Executes the SQL Query
Cmd.ExecuteNonQuery();

//Displays a message box informing the user that the anime has been deleted
from their favourites list
MessageBox.Show("Deleted from Favourites!");
}
catch (Exception) //catches error
{
//Displays a message box informing the user that the anime could not be deleted
from their favourites list
MessageBox.Show("Could not delete from favourites!");
}
}
/// <summary>
/// The resulting method for clearing the combobox and setting it to default
/// </summary>
private void ResetSelection()
{
//Sets the combo box to no element selected when - by 1
cbRatings.SelectedIndex = -1;
}

/// <summary>
/// The resulting method for when the user clicks on the Clear button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btClear_Click(object sender, EventArgs e)
{
//Clears the text that was entered in the textbox tbUserID
tbUserID.Clear();

//Clears the text that was entered in the textbox tbAnimeID
tbAnimeID.Clear();

//Clears the text that was entered in the textbox tbAnimeTitle
tbAnimeTitle.Clear();

//Runs the procedure ResetSelection
ResetSelection();
}
}
```

Form: FmUserWatchList.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;
using System.Text.RegularExpressions;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in
    <c>FmUserWatchList</c>
    /// </summary>
    public partial class FmUserWatchList : Form
    {
        /// <summary>
        /// Required Method for Designer support for <c>FmUserWatchList</c>
        /// </summary>
        public FmUserWatchList()
        {
            InitializeComponent();
        }

        /// <summary>
        /// The resulting method for when the Form loads
        /// </summary>
        /// <param name="sender"> The <see cref="System.Object"/> that represents the
        sender instance </param>
        /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
        arguments passed to the event handler </param>
        private void FmUserWatchList_Load(object sender, EventArgs e)
        {
            //Sets the label lbUsername to the global variable declared in the form FmLogIn
            which contains th user's username
            lbUsername.Text = FmLogIn.Username;

            //sets the visibility of the label lbTotalEpisodes to false (invisible)
            lbTotalEpisodes.Visible = false;

            //sets the visibility of the label lbEpisodeCount to false (invisible)
            lbEpisodeCount.Visible = false;

            //Create a database connection object with the constant defined in program.cs
            OleDbConnection Conn = new OleDbConnection(Program.connString);

            //Opens Connection to the database
            Conn.Open();
        }
    }
}
```

```
//Creates a database command object
OleDbCommand Cmd = new OleDbCommand();

//Creates a connection with the Command Object
Cmd.Connection = Conn;

//A select query to select the UserID From the table users where the location of
//the username is (the username is the label lbUsername)
Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" +
lbUsername.Text + "";

//sets the variable UserID to the result of the Sql query
var UserID = Cmd.ExecuteScalar();

//A select query to select everything from the table UserWatchList where
//the location of the UserID is declared as the variable UserID
Cmd.CommandText = "SELECT * From UserWatchList WHERE UserID=" + UserID
+ "";

//Creates A connection between the datasource(select query) and dataset to Add
rows or refresh rows
OleDbDataAdapter da = new OleDbDataAdapter(Cmd);

//Creates a table that will be filled with data from the select query
DataTable table = new DataTable();

//the data adapter da adds rows to the table
da.Fill(table);

// sets the datagridviewer UserWatchListGrid source to that of the DataTable table
// that is filled with data from the select query
UserWatchListGrid.DataSource = table;

// Closes the connection to the database
Conn.Close();

//Displays the DataTable
UserWatchListGrid.Show();

//Runs the procedure AutoFillSearch
AutoFillSearch();

}

/// <summary>
/// The resulting method for the procedure AutoFillSearch
/// </summary>
private void AutoFillSearch()
{
    try//Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);
```

```
//Opens Connection to the database
Conn.Open();

//Creates a database command object
OleDbCommand Cmd = new OleDbCommand();

//Creates a connection with the Command Object
Cmd.Connection = Conn;

//A select query to select the UserID From the table users where the location of
//the username is (the username is the label lbUsername)
Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" +
lbUsername.Text + "";

//sets the variable UserID to the result of the Sql query
var UserID = Cmd.ExecuteScalar();

//A select query to select AnimeTitle from the table UserWatchList where
//the location of the UserID is declared as the variable UserID
Cmd.CommandText = "SELECT AnimeTitle FROM UserWatchList WHERE
UserID=" + UserID + "";

//Defines the autocompletefeature that holds a collection of strings, AutoFill
AutoCompleteStringCollection AutoFill = new AutoCompleteStringCollection();

//Creates a reader object and Executes the SQL Query
OleDbDataReader reader = Cmd.ExecuteReader();

//if the reader object has found rows
if (reader.HasRows)
{
    //while the reader object is reading the rows in the column AnimeTitle
    while (reader.Read())
    {

        //Add the rows that are in the column AnimeTitle to AutoFill
        AutoFill.Add(reader["AnimeTitle"].ToString());
    }
}

//otherwise
else
{

    //Display a messagebox informing the user that there was no AnimeTitles found
    MessageBox.Show("Anime title not found!");
}

//Closes the reader object.
reader.Close();

//Sets the autocomplete source of the textbox tbSearch to AutoFill which contains
all the AnimeTitles
```

```
tbSearch.AutoCompleteCustomSource = AutoFill;

    //Closes the connection to the database
    Conn.Close();
}
catch(Exception) //catches error
{

    //Displays a message box informing the user that it could not loadAutoFill
    MessageBox.Show("Anime title not found!");
}
}

/// <summary>
/// The resulting method when the user clicks on a cell from the grid
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>

private void UserWatchListGrid_CellContentClick(object sender,
DataGridViewCellEventArgs e)
{
    //Create a database connection object with the constant defined in program.cs
    OleDbConnection Conn = new OleDbConnection(Program.connString);

    //Opens Connection to the database
    Conn.Open();

    //Creates a database command object
    OleDbCommand Cmd = new OleDbCommand();

    //Creates a connection with the Command Object
    Cmd.Connection = Conn;

    //if the cell's row is greater than or equal to 0
    if (e.RowIndex >= 0)
    {

        //stores the cell's row into a datagridviewrow variable 'row'
        DataGridViewRow row = UserWatchListGrid.Rows[e.RowIndex];

        //Sets the textbox tbUserID to the cell value under the column UserID
        tbUserID.Text = row.Cells["UserID"].Value.ToString();

        //Sets the textbox tbAnimeID to the cell value under the column AnimeID
        tbAnimeID.Text = row.Cells["AnimeID"].Value.ToString();

        //Sets the textbox tbAnimeTitle to the cell value under the column AnimeTitle
        tbAnimeTitle.Text = row.Cells["AnimeTitle"].Value.ToString();

        //Sets the combobox cbWatchStatus to the cell value under the column
        WatchStatus
    }
}
```

```

cbWatchStatus.Text = row.Cells["WatchStatus"].Value.ToString();

//Sets the textbox tbEpisodesWatched to the cell value under the column
EpisodesWatched
tbEpisodesWatched.Text = row.Cells["EpisodesWatched"].Value.ToString();

//Sets the combobox cbRatings to the cell value under the column Ratings
cbRatings.Text = row.Cells["Ratings"].Value.ToString();

//A select query to select the AnimePictures from the table Animes where the
//location of the AnimeTitle is declared as tbAnimeTitle
Cmd.CommandText="SELECT AnimePictures FROM Animes WHERE
AnimeTitle='"+tbAnimeTitle.Text+"'";

//sets the variable AnimePic to the result of the Sql query
var AnimePic = Cmd.ExecuteScalar();

//loads the url in the variable AnimePic to the picturebox PbAnimePicture
PbAnimePicture.LoadAsync(AnimePic.ToString());

//A select query to select the Episodes from the table Animes where the
//location of the AnimeTitle is declared as tbAnimeTitle
Cmd.CommandText ="SELECT Episodes FROM Animes WHERE
AnimeTitle='"+tbAnimeTitle.Text+"'";

//sets the variable Episodes to the result of the Sql query
var Episodes = Cmd.ExecuteScalar();

//sets the label lbEpisodeCount text to the value of the variable Episodes
lbEpisodeCount.Text = Episodes.ToString();

//sets the visibility of the label lbTotalEpisodes to true (visible)
lbTotalEpisodes.Visible = true;

//sets the visibility of the label lbEpisodeCount to true (visible)
lbEpisodeCount.Visible = true;

}

}

/// <summary>
/// The resulting method when the user presses a key in the textbox tbSearch
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>

private void tbSearch_KeyPress(object sender, KeyPressEventArgs e)
{
    //Create a database connection object with the constant defined in program.cs
    OleDbConnection Conn = new OleDbConnection(Program.connString);
}

```

```
//Opens Connection to the database
Conn.Open();

//Creates a database command object
OleDbCommand Cmd = new OleDbCommand();

//Opens Connection to the database
Cmd.Connection = Conn;

//A select query to select the UserID From the table users where the location of
//the username is (the username is the label lbUsername)
Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" + 
lbUsername.Text + "";

//sets the variable UserID to the result of the Sql query
var UserID = Cmd.ExecuteScalar();

//if the textbox tbSearch is not empty
if (tbSearch.Text != "")
{
    try
    {
        // Gets value entered from the user to indicate the size of the columns and fill it
        with data
        UserWatchListGrid.AutoSizeColumnsMode =
        DataGridViewAutoSizeColumnsMode.Fill;

        //Creates A connection between the datasource(select query) and dataset to
        Add rows or refresh rows
        OleDbDataAdapter Adapt = new OleDbDataAdapter();

        //Creates an empty set for data to be filled
        DataSet DS = new DataSet();

        // Creates view of the the data stored in the dataset
        DataView DV = new DataView();

        // A select query that gets everything from the table UserWatchList where
        // the location of the UserID is, and is declared as the variable UserID, when the
        user
        // types a letter it is checked and compared to entries in the table
        // to determine whetere a specific character string matches a specified pattern
        //in the column AnimeTitle
        string SQL = "SELECT * FROM UserWatchList WHERE AnimeTitle LIKE " +
        tbSearch.Text + "%" AND UserID=" + UserID + "";

        //Creates a database command object that contains the connection to the
        //database and the string containing the select query
        Cmd = new OleDbCommand(SQL, Conn);

        //Creates A connection between the datasource(select query) and dataset to
        refresh rows from the command
        Adapt = new OleDbDataAdapter(Cmd);
```

```
//Fills the DataSet with information from the select query
Adapt.Fill(DS);

// Creates a view of the data stored in the dataset
DV = new DataView(DS.Tables[0]);

// sets the datagridviewer UserWatchListGrid source to that of the DataView
// that is filled with data from the select query
UserWatchListGrid.DataSource = DV;

//Closes the connection to the database
Conn.Close();
}

catch(Exception) //catches error
{
    //Displays a message box informing the user that their input is invalid
    MessageBox.Show("Invalid input");
}

//if the textbox tbSearch is empty
else if (tbSearch.Text == "")
{

    //A select query to select everything from the table UserWatchList where
    //the location of the UserID is declared as the variable UserID
    Cmd.CommandText = "SELECT * FROM UserWatchList WHERE
UserID=" + UserID + "";

    //Creates A connection between the datasource(select query) and dataset
    //to Add rows or refresh rows
    OleDbDataAdapter da = new OleDbDataAdapter(Cmd);

    //Creates a table that will be filled with data from the select query
    DataTable table = new DataTable();

    //the data adapter da adds rows to the table
    da.Fill(table);

    // sets the datagridviewer UserFavouritesGrid source to that of the DataTable table
    // that is filled with data from the select query
    UserWatchListGrid.DataSource = table;

    //Closes the connection to the database
    Conn.Close();

    //Displays the DataTable
    UserWatchListGrid.Show();
}

/// <summary>
/// The resulting method for when the user clicks on the update button
```

```
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btUpdate_Click(object sender, EventArgs e)

try //Exception Handling
{
    //Create a database connection object with the constant defined in program.cs
    OleDbConnection Conn = new OleDbConnection(Program.connString);

    //Opens Connection to the database
    Conn.Open();

    //Creates a database command object
    OleDbCommand Cmd = new OleDbCommand();

    //if the user did not select a watch status
    if (cbWatchStatus.SelectedIndex == -1)
    {
        //Displays a message box informing the user to select a watch status
        MessageBox.Show("Please select a watch status!");
    }

    //if the user did not select a rating
    else if (cbRatings.SelectedIndex == -1)
    {
        //Displays a message box informing the user to select a rating
        MessageBox.Show("Please select a rating");
    }
    else
    {
        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //A select query to select the UserID From the table users where the location of
        //the username is (the username is the label lbUsername)
        Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" +
        lbUsername.Text + "';

        //sets the variable UserID to the result of the Sql query
        var UserID = Cmd.ExecuteScalar();

        //Updates the information contained in WatchStatus, EpisodesWatched and
        Ratings with the entries,
        //cbWatchStatus, tbEpisodesWatched and cbRatings
        //This update will take place in the location where the UserID is as the same as
        the variable UserID
        //and where the AnimelD is same as the textbox tbAnimelD
        Cmd.CommandText = "UPDATE UserWatchlist SET WatchStatus=" +
        cbWatchStatus.Text + ", EpisodesWatched=" + tbEpisodesWatched.Text + ", Ratings = " +
```

```
cbRatings.Text + " WHERE UserID=" + UserID + " AND AnimeTitle=" + tbAnimeTitle.Text +  
"";  
        //Executes the SQL Query  
        Cmd.ExecuteNonQuery();  
  
        //Displays a messagebox informing the user that their watch list has been  
        updated  
        MessageBox.Show("WatchList Updated!");  
    }  
}  
catch(Exception) //catches error  
{  
    //Displays a messagebox informing the user that their favourites list could not be  
    updated  
    MessageBox.Show("Could not update WatchList!");  
}  
}  
/// <summary>  
/// The resulting method for when the textbox tbEpisodeWatched is validating the user's  
input  
/// </summary>  
/// <param name="sender"> The <see cref="System.Object"/> that represents the  
sender instance</param>  
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event  
arguments passed  
/// to the event handler</param>  
private void tbEpisodesWatched_Validating(object sender, CancelEventArgs e)  
{  
    //Regex validation for symbols  
    var hasSymbols = new Regex(@"[!@#$^&*()_+=\{\}];:<>|./?,.-]+");  
  
    //Regex validation for numbers  
    var hasNumber = new Regex(@"[0-9]+");  
  
    //if the text in the textbox tbEpisodesWatched does not contain numbers  
    if (!hasNumber.IsMatch(tbEpisodesWatched.Text))  
    {  
  
        //Displays a message box informing the user to enter a number  
        MessageBox.Show("Please enter a number!");  
    }  
  
    //if the text in the textboxtbEpisodes Watched contains symbols  
    else if(hasSymbols.IsMatch(tbEpisodesWatched.Text))  
    {  
        //Displays a message box informing the user to enter a number  
        MessageBox.Show("Please Enter a number!");  
    }  
  
    //if the input is fully valid  
    else  
    {  
        //Create a database connection object with the constant defined in program.cs
```

```
OleDbConnection Conn = new OleDbConnection(Program.connString);

//Opens Connection to the database
Conn.Open();

//Creates a database command object
OleDbCommand Cmd = new OleDbCommand();

//Creates a connection with the Command Object
Cmd.Connection = Conn;

//A select query to select episodes from the table Animes from the location of
the AnimeTitle
Cmd.CommandText = "SELECT Episodes FROM Animes WHERE
AnimeTitle=" + tbAnimeTitle.Text + "";

//sets the integer Episodes to the result of the sql query in an integer format
int Episodes = Convert.ToInt32(Cmd.ExecuteScalar());

//sets the integer EpisodesEntered to the value of the text
//tbEpisodesWatched in an integer format
int EpisodesEntered = Convert.ToInt32(tbEpisodesWatched.Text);

// if the episodes entered by the user is greater than the total episodes of the
anime
if (EpisodesEntered > Episodes)
{
    //Displays a message box informing the user that the episodes entered are out
    range
    MessageBox.Show("Out of range!");

    //sets the text from the textbox tbEpisodesWatched with the variable Episodes
    //which contains the total episodes of the anime
    tbEpisodesWatched.Text = Episodes.ToString();
}

//if the episodes entered by the user is less than 0
else if (EpisodesEntered < 0)
{
    //Displays a message box informing the user that the episodes entered are out
    range
    MessageBox.Show("Out of range!");

    //sets the text from the textbox tbEpisodesWatched to 0 in text form
    tbEpisodesWatched.Text = "0";
}

//Closes the connection to the database
Conn.Close();
}

/// <summary>
```

```
/// The resulting method for when the user clicks on the Delete button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btDelete_Click(object sender, EventArgs e)
{
    try //Exception Handling
    {

        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //A select query to select the UserID From the table users where the location of
        //the username is (the username is the label lbUsername)
        Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" +
        lbUsername.Text + "'';

        //sets the variable UserID to the result of the Sql query
        var UserID = Cmd.ExecuteScalar();

        //The delete query will happen on the row containing the UserID
        //that's the same as the variable UserID and AnimeTitle that's the same as the
        //textbox tbAnimeTitle
        Cmd.CommandText = "DELETE FROM UserWatchList WHERE UserID=" +
        UserID + " AND AnimeTitle=" + tbAnimeTitle.Text + "'';

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box informing the user that the anime has been deleted
        //from their watch list
        MessageBox.Show("Deleted from WatchList");
    }
    catch(Exception) //catches error
    {
        //Displays a message box informing the user that the anime could not be deleted
        //from their watch list
        MessageBox.Show("Could not delete from WatchList");
    }
}
/// <summary>
/// The resulting method for clearing the combobox and setting it to default
/// </summary>
```

```
private void ResetSelection()
{
    // Sets the combo box cbRatings to no element selected when -by 1
    cbRatings.SelectedIndex = -1;

    // Sets the combo box cbWatchStatus to no element selected when -by 1
    cbWatchStatus.SelectedIndex = -1;
}

/// <summary>
/// The resulting method for when the user clicks on the Clear button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btClear_Click(object sender, EventArgs e)
{
    //Clears the text that was entered in the textbox tbUserID
    tbUserID.Clear();

    //Clears the text that was entered in the textbox tbAnimeID
    tbAnimeID.Clear();

    //Clears the text that was entered in the textbox tbAnimeTitle
    tbAnimeTitle.Clear();

    //Clears the text that was entered in the textbox tbEpisodesWatched
    tbEpisodesWatched.Clear();

    //Runs the procedure ResetSelection
    ResetSelection();
}

/// <summary>
/// The resulting method for when the combo box cbWatchStatus is selected
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>

private void cbWatchStatus_SelectedIndexChanged(object sender, EventArgs e)
{
    //Create a database connection object with the constant defined in program.cs
    OleDbConnection Conn = new OleDbConnection(Program.connString);

    //Creates a database command object
    Conn.Open();

    //Creates a connection with the Command Object
    OleDbCommand Cmd = new OleDbCommand();
}
```

```
//Creates a connection with the Command Object
Cmd.Connection = Conn;

//A select query to select episodes from the table Animes from the location of the
AnimeTitle
Cmd.CommandText = "SELECT Episodes FROM Animes WHERE AnimeTitle=
@AnimeTitle";

//Parameteriesed SQL that contains the label lbAnimeName
Cmd.Parameters.Add("@AnimeTitle", OleDbType.Char).Value = tbAnimeTitle.Text;

//sets the integer Episodes to the result of the sql query in an integer format
int Episodes = Convert.ToInt32(Cmd.ExecuteScalar());

//if the user selects "Completed" in the drop down list for cbWatchStatus
if (cbWatchStatus.Text == "Completed")
{
    //sets the text from the textbox tbEpisodesWatched with the variable Episodes
    //which contains the total episodes of the anime
    tbEpisodesWatched.Text = Episodes.ToString();

}

//if the user selects "Not Yet Watched" in the drop down list for cbWatchStatus
else if (cbWatchStatus.Text == "Not Yet Watched")
{
    //sets the text from the textbox tbEpisodesWatched to 0 in text form
    tbEpisodesWatched.Text = "0";
}

}
```

Form: FmUserStats.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in
    <c>FmUserStats</c>
    /// </summary>
```

```
public partial class FmUserStats : Form
{
    /// <summary>
    /// Required Method for Designer support for <c>FmUserStats</c>
    /// </summary>
    public FmUserStats()
    {
        InitializeComponent();
    }

    /// <summary>
    /// The resulting method for when the Form loads
    /// </summary>
    /// <param name="sender"> The <see cref="System.Object"/> that represents the
    sender instance </param>
    /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
    arguments passed to the event handler </param>
    private void FmUserStats_Load(object sender, EventArgs e)
    {
        try //Exception Handling
        {

            //Sets the label lbUsername to the global variable declared in the form FmLogIn
            which contains th user's username
            lbUsername.Text = FmLogIn.Username;

            //Create a database connection object with the constant defined in program.cs
            OleDbConnection Conn = new OleDbConnection(Program.connString);

            //Opens Connection to the database
            Conn.Open();

            //Creates a database command object
            OleDbCommand Cmd = new OleDbCommand();

            //Creates a connection with the Command Object
            Cmd.Connection = Conn;

            //A select query to select the UserID From the table users where the location of
            //the username is (the username is the label lbUsername)
            Cmd.CommandText = "SELECT UserID FROM Users WHERE Username=" +
            lbUsername.Text + "'';

            //sets the variable UserID to the result of the Sql query
            var UserID = Cmd.ExecuteScalar();

            //A select query to count the entries where it contains enteries that has
            //the userID which is the same as the variable userID in itin the table
            UserFavourites
            Cmd.CommandText = "SELECT COUNT(*) FROM UserFavourites WHERE
            UserID=" + UserID + "'';

            //Executes the SQL query
            Cmd.ExecuteNonQuery();
        }
    }
}
```

```
//sets the variable TotalRatingsFromFavourites to the result of the Sql query in
integer form
int TotalRatingsFromFavourites = Convert.ToInt32(Cmd.ExecuteScalar());

//A select query to count the entries where it contains enteries that has
//the userID which is the same as the variable userID in the table UserWatchList
Cmd.CommandText = "SELECT COUNT(*) FROM UserWatchList WHERE
UserID=" + UserID + "";
Cmd.ExecuteNonQuery();

//sets the variable TotalRatingsFromWatchList to the result of the Sql query in
integer form
int TotalRatingsFromWatchList = Convert.ToInt32(Cmd.ExecuteScalar());

//Sets the integer TotalRatings to the sum of the two variables
//TotalRatingsFromFavourites and TotalRatingsFromWatchList
int TotalRatings = TotalRatingsFromFavourites + TotalRatingsFromWatchList;

//sets the label DbTotalRatings to the variable TotalRatings in string form
DbTotalRatings.Text = TotalRatings.ToString();

//A select query to select the average ratings from the table UserFavourites and
UserWatchList where the UserIDs in both tables contain the variable UserID
Cmd.CommandText = "SELECT AVG(UserFavourites.Ratings) +
AVG(UserWatchList.Ratings) FROM UserFavourites, UserWatchList WHERE
UserFavourites.UserID=" + UserID + " AND UserWatchList.UserID=" + UserID + "";

//Executes the SQL query
Cmd.ExecuteNonQuery();

// sets the variable RatingsAdded to the result of the Sql query in double form
double RatingsAdded = Convert.ToDouble(Cmd.ExecuteScalar());

//Sets the variable TrueAvgRatings to half of the variable RatingsAdded
double TrueAvgRatings = RatingsAdded / 2;

//Rounds the variable TrueAvgRatings to 1 decimal place and is set as the label
DbAverageRatings
DbAverageRatings.Text = Math.Round(TrueAvgRatings, 1).ToString();

//A select query to count the entries where it contains enteries that has
//the WatchStatus set as completed where the location of the UserID is (which
contains the variable UserID)
Cmd.CommandText = "SELECT COUNT(*) FROM UserWatchList WHERE
WatchStatus = 'Completed' AND UserID=" + UserID + "";

//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable Completed to the result of the Sql query in integer form
int Completed = Convert.ToInt32(Cmd.ExecuteScalar());

//Sets the label DbAnimesCompleted to variable Completed in string form
```

```
DbAnimesCompleted.Text = Completed.ToString();

//A select query to count the entries where it contains enteries that has
//the WatchStatus set as CurrentlyWatching where the location of the UserID is
(which contains the variable UserID)
Cmd.CommandText = "SELECT COUNT(*) FROM UserWatchList WHERE
WatchStatus = 'Currently Watching' AND UserID=" + UserID + "";

//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable Currently Watching to the result of the Sql query in integer form
int CurrentlyWatching = Convert.ToInt32(Cmd.ExecuteScalar());

//Sets the label DbAnimesCurrentlyWatching to the variable CurrentlyWatching in
string form
DbAnimesCurrentlyWatching.Text = CurrentlyWatching.ToString();

//A select query to count the entries where it contains enteries that has
//the WatchStatus set as Not yet Watched where the location of the UserID is
(which contains the variable UserID)
Cmd.CommandText = "SELECT COUNT(*) FROM UserWatchList WHERE
WatchStatus = 'Not Yet Watched' AND UserID=" + UserID + "";

//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable NotYetWatched to the result of the Sql query in integer form
int NotYetWatched = Convert.ToInt32(Cmd.ExecuteScalar());

//Sets the label DbAnimesYetToWatch to the variable NotYetWatched in string
form
DbAnimesYetToWatch.Text = NotYetWatched.ToString();

//A select query to count the entries where it contains enteries that has
//the WatchStatus set as Dropped where the location of the UserID is (which
contains the variable UserID)
Cmd.CommandText = "SELECT COUNT(*) FROM UserWatchList WHERE
WatchStatus = 'Dropped' AND UserID=" + UserID + "";

//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable Dropped to the result of the Sql query in integer form
int Dropped = Convert.ToInt32(Cmd.ExecuteScalar());

//Sets the label DbAnimesDropped to the variable Dropped in string form
DbAnimesDropped.Text = Dropped.ToString();

//A select query to select the sum of all episodeswatched in UserWatchList in
where the
//location of the UserID is (which contains the variable UserID)
Cmd.CommandText = "SELECT SUM(EpisodesWatched) FROM UserWatchList
WHERE UserID=" + UserID + ";
```

```
//Executes the SQL query
Cmd.ExecuteNonQuery();

//sets the variable EpisodesWatched to the result of the Sql query in integer form
int EpisodesWatched = Convert.ToInt32(Cmd.ExecuteScalar());

//Sets the label DbTotalEpisodesWatched to the variable Dropped in string form
DbTotalEpisodesWatched.Text = EpisodesWatched.ToString();

//sets the variable AnimeEpLength to the result of the multiplication between
//the variable EpisodesWatched and the number 24
int AnimeEpLength = EpisodesWatched * 24;

//sets the variable Hours to the result of the division between
//the variable AnimeEpLength and the number 60
int Hours = AnimeEpLength / 60;

//Sets the label DbHourSpent to the variable Hours in string form
DbHourSpent.Text = Hours.ToString();
}

catch(Exception) //catches error
{
    //Displays a message box informing the user that their stats could not be loaded.
    MessageBox.Show("Could not load stats");
}

}
```

Form: FmAMS(Admin).cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;
using System.IO;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in
    <c>FmAMS(Admin)</c>
    /// </summary>
    public partial class FmAMS_admin_ : Form
    {

```

```
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c>to make this variable
accessible to other forms
/// </summary>
public static string AnimeID = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c>to make this variable
accessible to other forms
/// </summary>
public static string AnimeTitle = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c>to make this variable
accessible to other forms
/// </summary>
public static string Episodes = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c>to make this variable
accessible to other forms
/// </summary>
public static string Status = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c>to make this variable
accessible to other forms
/// </summary>
public static string Ratings = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c>to make this variable
accessible to other forms
/// </summary>
public static string AnimePictures = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c>to make this variable
accessible to other forms
/// </summary>
public static string AnimeSynopsis = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c>to make this variable
accessible to other forms
/// </summary>
public static string StudioID = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c>to make this variable
accessible to other forms
/// </summary>
public static string Studio = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c>to make this variable
accessible to other forms
/// </summary>
public static string GenreID = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c>to make this variable
accessible to other forms
/// </summary>
```

```
public static string Genre = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c> to make this variable
accessible to other forms
/// </summary>
public static string UserID = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c> to make this variable
accessible to other forms
/// </summary>
public static string Username = "";
/// <summary>
/// Global variable that is declared in <c>FmAms(Admin)</c> to make the form
accessible to other forms
/// </summary>
public static FmAMS_admin_ FmAmsAdminInstance;

/// <summary>
/// Required Method for Designer support for <c>FmAms(Admin)</c>
/// </summary>
public FmAMS_admin_()

{
    InitializeComponent();
    FmAmsAdminInstance = this;

}

/// <summary>
/// The resulting method when the user clicks on the Show button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>
private void btShow_Click(object sender, EventArgs e)
{
    //Create a database connection object with the constant defined in program.cs
    OleDbConnection Conn = new OleDbConnection(Program.connString);

    //Opens Connection to the database
    Conn.Open();

    //Creates a database command object
    OleDbCommand Cmd = new OleDbCommand();

    //Creates a connection with the Command Object
    Cmd.Connection = Conn;

    //if the radiobutton rbAnimes is clicked by the User
    if(rbAnimes.Checked)
    {
```

```
//A select query to select everything in the table called Animes
Cmd.CommandText = "SELECT * FROM Animes";

}

//if the radiobutton rbGenres is clicked by the User
else if(rbGenres.Checked)
{
    //A select query to select everything in the table called Genres
    Cmd.CommandText = "SELECT * FROM Genres";

}

//if the radiobutton rbStudios is clicked by the User
else if (rbStudios.Checked)
{
    //A select query to select everything in the table called Studios
    Cmd.CommandText = "SELECT * FROM Studios";

}

//if the radiobutton rbAnimeStudios is clicked by the User
else if (rbAnimeStudios.Checked)
{
    //A select query to select everything in the table called AnimeStudios
    Cmd.CommandText = "SELECT * FROM AnimeStudios";

}

//if the radiobutton rbAnimeGenres is clicked by the User
else if (rbAnimeGenres.Checked)
{
    //A select query to select everything in the table called AnimeGenres
    Cmd.CommandText = "SELECT * FROM AnimeGenres";

}

//if the radiobutton rbAnimeStatus is clicked by the User
else if (rbAnimeStatus.Checked)
{
    //A select query to select everything in the table called AnimeStatus
    Cmd.CommandText = "SELECT * FROM AnimeStatus";

}

//if the radiobutton rbUsers is clicked by the User
else if (rbUsers.Checked)
{
    //A select query to select everything in the table called AnimeStatus
    Cmd.CommandText = "SELECT UserID, Username FROM Users";

}

//Creates A connection between the datasource(select query) and dataset to Add
rows or refresh rows
OleDbDataAdapter da = new OleDbDataAdapter(Cmd);
```

```
//Creates a table that will be filled with data from the select query
DataTable table = new DataTable();

//the data adapter da adds rows to the table
da.Fill(table);

// sets the datagridviewer Grid source to that of the DataTable table
// that is filled with data from the select query
Grid.DataSource = table;

// Closes the connection to the database
Conn.Close();

//Displays the DataTable
Grid.Show();

}

/// <summary>
/// The resulting method when the user presses a key in the textbox tbSearch
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>
private void tbSearch_KeyPress(object sender, KeyPressEventArgs e)
{
    //Exception Handling
    try
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Opens Connection to the database
        Cmd.Connection = Conn;

        //if the radiobutton rbAnimes is clicked by the User
        if (rbAnimes.Checked)
        {
            //if the textbox tbSearch is not empty
            if (tbSearch.Text != "")
            {
                try //Exception Handling
                {
                    //Gets value entered from the user to indicate the size of the columns and
                    fill it with data
                    Grid.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

                    //Creates A connection between the datasource(select query) and dataset
                    to Add rows or refresh rows
                }
            }
        }
    }
}
```

```
OleDbDataAdapter Adapt = new OleDbDataAdapter();

//Creates an empty set for data to be filled
DataSet DS = new DataSet();

// Creates view of the the data stored in the dataset
DataView DV = new DataView();

// A select query that gets everything from the table Animes, when the user
// types a letter it is checked and compared to entries in the table
// to determine whetere a specific character string matches a specified
pattern
//in either the column AnimeTitle or AnimeID
string SQL = "SELECT * FROM Animes WHERE AnimeTitle LIKE '" +
tbSearch.Text + "'% OR AnimeID LIKE '" + tbSearch.Text + "%'";

//Opens a connection to the database
Conn.Open();

//Creates a database command object that contains the connection to the
//database and the string containing the select query
Cmd = new OleDbCommand(SQL, Conn);

//Creates A connection between the datasource(select query) and dataset
// to refresh rows from the command
Adapt = new OleDbDataAdapter(Cmd);

//Fills the DataSet with information from the select query
Adapt.Fill(DS);

// Creates a view of the data stored in the dataset
DV = new DataView(DS.Tables[0]);

// sets the datagridviewer Grid source to that of the DataView
// that is filled with data from the select query
Grid.DataSource = DV;

//Closes the connection to the database
Conn.Close();
}
catch(Exception) //catches error
{
    //Displays a message box informing the user that their input is invalid
    MessageBox.Show("Invalid input");
}
}

//if the textbox tbSearch is empty
else if (tbSearch.Text == "")
{
    //A select query to select everything in the table called Animes
    Cmd.CommandText = "SELECT * FROM Animes";

    //Creates A connection between the datasource(select query) and dataset
```

```
//to Add rows or refresh rows
OleDbDataAdapter da = new OleDbDataAdapter(Cmd);

//Creates a table that will be filled with data from the select query
DataTable table = new DataTable();

//the data adapter da adds rows to the table
da.Fill(table);

// sets the datagridviewer Grid source to that of the DataTable table
// that is filled with data from the select query
Grid.DataSource = table;

//Closes the connection to the database
Conn.Close();

//Displays the DataTable
Grid.Show();
}

} //if the radiobutton rbGenres is clicked by the User
else if (rbGenres.Checked)
{
    //if the textbox tbSearch is not empty
    if (tbSearch.Text != "")
    {
        try //Exception Handling
        {
            //Gets value entered from the user to indicate the size of the columns and
            fill it with data
            Grid.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

            //Creates A connection between the datasource(select query) and dataset
            to Add rows or refresh rows
            OleDbDataAdapter Adapt = new OleDbDataAdapter();

            //Creates an empty set for data to be filled
            DataSet DS = new DataSet();

            // Creates view of the data stored in the dataset
            DataView DV = new DataView();

            // A select query that gets everything from the table Genres, when the user
            // types a letter it is checked and compared to entries in the table
            // to determine whetere a specific character string matches a specified
            pattern
            //in either the column GenreID or Genre
            string SQL = "SELECT * FROM Genres WHERE GenreID LIKE " +
            tbSearch.Text + "%" + " OR Genre LIKE " + tbSearch.Text + "%";

            //Opens a connection to the database
            Conn.Open();

            //Creates a database command object that contains the connection to the
            //database and the string containing the select query
```

```
Cmd = new OleDbCommand(SQL, Conn);

    //Creates A connection between the datasource(select query) and dataset
    //to Add rows or refresh rows from the command
    Adapt = new OleDbDataAdapter(Cmd);

    //Fills the DataSet with information from the select query
    Adapt.Fill(DS);

    // Creates a view of the data stored in the dataset
    DV = new DataView(DS.Tables[0]);

    // sets the datagridviewer Grid source to that of the DataView
    // that is filled with data from the select query
    Grid.DataSource = DV;

    //Closes the connection to the database
    Conn.Close();
}

catch (Exception) //catches error
{
    //Displays a message box informing the user that their input is invalid
    MessageBox.Show("Invalid input");
}

}

//if the textbox tbSearch is empty
else if (tbSearch.Text == "")
{
    //A select query to select everything in the table called Genres
    Cmd.CommandText = "SELECT * FROM Genres";

    //Creates A connection between the datasource(select query) and dataset
    //to Add rows or refresh rows
    OleDbDataAdapter da = new OleDbDataAdapter(Cmd);

    //Creates a table that will be filled with data from the data adapter
    DataTable table = new DataTable();

    //the data adapter da adds rows to the table
    da.Fill(table);

    // sets the datagridviewer Grid source to that of the DataTable table
    // that is filled with data from the select query
    Grid.DataSource = table;

    //Closes the connection to the database
    Conn.Close();

    //Displays the DataTable
    Grid.Show();
}
}
```

```
//if the radiobutton rbStudios is clicked by the User
else if (rbStudios.Checked)
{
    //if the textbox tbSearch is not empty
    if (tbSearch.Text != "")
    {
        try //Exception Handling
        {
            //Gets value entered from the user to indicate the size of the columns and
            fill it with data
            Grid.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

            //Creates A connection between the datasource(select query) and dataset
            to Add rows or refresh rows
            OleDbDataAdapter Adapt = new OleDbDataAdapter();

            //Creates an empty set for data to be filled
            DataSet DS = new DataSet();

            // Creates view of the data stored in the dataset
            DataView DV = new DataView();

            // A select query that gets everything from the table Studios, when the user
            // types a letter it is checked and compared to entries in the table
            // to determine whetere a specific character string matches a specified
            pattern
            //in either the column StudioID or Studio
            string SQL = "SELECT * FROM Studios WHERE StudioID LIKE '" +
            tbSearch.Text + "%" + " OR Studio LIKE'" + tbSearch.Text + "%'";

            //Opens a connection to the database
            Conn.Open();

            //Creates a database command object that contains the connection to the
            //database and the string containing the select query
            Cmd = new OleDbCommand(SQL, Conn);

            //Creates A connection between the datasource(select query) and dataset
            to refresh rows from the command
            Adapt = new OleDbDataAdapter(Cmd);

            //Fills the DataSet with information from the select query
            Adapt.Fill(DS);

            // Creates a view of the data stored in the dataset
            DV = new DataView(DS.Tables[0]);

            // sets the datagridviewer Grid source to that of the DataView
            // that is filled with data from the select query
            Grid.DataSource = DV;

            //Closes the connection to the database
            Conn.Close();
        }
    }
}
```

```
        }
        catch(Exception) //catches error
        {
            //Displays a message box informing the user that their input is invalid
            MessageBox.Show("Invalid input");
        }
    }

    //if the textbox tbSearch is empty
    else if (tbSearch.Text == "")
    {
        //A select query to select everything in the table called Studios
        Cmd.CommandText = "SELECT * FROM Studios";

        //Creates A connection between the datasource(select query) and dataset
        //to Add rows or refresh rows
        OleDbDataAdapter da = new OleDbDataAdapter(Cmd);

        //Creates a table that will be filled with data from the data adapter
        DataTable table = new DataTable();

        //the data adapter da adds rows to the table
        da.Fill(table);

        // sets the datagridviewer Grid source to that of the DataTable table
        // that is filled with data from the select query
        Grid.DataSource = table;

        //Closes the connection to the database
        Conn.Close();

        //Displays the DataTable
        Grid.Show();
    }
}

//if the radiobutton rbAnimeStudios is clicked by the User
else if (rbAnimeStudios.Checked)
{
    //if the textbox tbSearch is not empty
    if (tbSearch.Text != "")
    {
        try //Exception Handling
        {
            //Gets value entered from the user to indicate the size of the columns and
            fill it with data
            Grid.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

            //Creates A connection between the datasource(select query) and dataset
            to Add rows or refresh rows
            OleDbDataAdapter Adapt = new OleDbDataAdapter();

            //Creates an empty set for data to be filled
            DataSet DS = new DataSet();
        }
    }
}
```

```

// Creates view of the data stored in the dataset
DataView DV = new DataView();

// A select query that gets everything from the table AnimeStudios, when
the user
// types a letter it is checked and compared to entries in the table
// to determine whether a specific character string matches a specified
pattern
//in either the column AnimeID or StudioID
string SQL = "SELECT * FROM AnimeStudios WHERE AnimeID LIKE '" +
tbSearch.Text + "%" OR StudioID LIKE'" + tbSearch.Text + "%';

//Opens a connection to the database
Conn.Open();

//Creates a database command object that contains the connection to the
//database and the string containing the select query
Cmd = new OleDbCommand(SQL, Conn);

//Creates A connection between the datasource(select query) and dataset
to refresh rows from the command
Adapt = new OleDbDataAdapter(Cmd);

//Fills the DataSet with information from the select query
Adapt.Fill(DS);

// Creates a view of the data stored in the dataset
DV = new DataView(DS.Tables[0]);

// sets the datagridviewer Grid source to that of the DataTable table
// that is filled with data from the select query
Grid.DataSource = DV;

//Closes connection to the database
Conn.Close();
}
catch(Exception) //catches error
{
    //Displays a message box informing the user that their input is invalid
    MessageBox.Show("Invalid input");
}
}

//if the textbox tbSearch is empty
else if (tbSearch.Text == "")
{
    //A select query to select everything in the table called AnimeStudios
    Cmd.CommandText = "SELECT * FROM AnimeStudios";

    //Creates A connection between the datasource(select query) and dataset
    //to Add rows or refresh rows
    OleDbDataAdapter da = new OleDbDataAdapter(Cmd);

    //Creates a table that will be filled with data from the data adapter
}

```

```

DataTable table = new DataTable();

//the data adapter da adds rows to the table
da.Fill(table);

// sets the datagridviewer Grid source to that of the DataTable table
// that is filled with data from the select query
Grid.DataSource = table;

//Closes the connection to the database
Conn.Close();

//Displays the DataTable
Grid.Show();
}

}

//if the radiobutton rbAnimeGenres is clicked by the User
else if (rbAnimeGenres.Checked)
{
    //if the textbox tbSearch is not empty
    if (tbSearch.Text != "")
    {
        try //catches the error
        {
            //Gets value entered from the user to indicate the size of the columns and
            fill it with data
            Grid.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

            //Creates A connection between the datasource(select query) and dataset
            to Add rows or refresh rows
            OleDbDataAdapter Adapt = new OleDbDataAdapter();

            //Creates an empty set for data to be filled
            DataSet DS = new DataSet();

            // Creates view of the the data stored in the dataset
            DataView DV = new DataView();

            // A select query that gets everything from the table AnimeGenres, when
            the user
            // types a letter it is checked and compared to entries in the table
            // to determine whetere a specific character string matches a specified
            pattern
            //in either the column AnimelD or GenrelD
            string SQL = "SELECT * FROM AnimeGenres WHERE AnimelD LIKE " +
            tbSearch.Text + "%" OR GenrelD LIKE " + tbSearch.Text + "%";

            //Opens a connection to the database
            Conn.Open();

            //Creates a database command object that contains the connection to the
            //database and the string containing the select query
            Cmd = new OleDbCommand(SQL, Conn);

```

```
//Creates A connection between the datasource(select query) and dataset
to refresh rows from the command
Adapt = new OleDbDataAdapter(Cmd);

//Fills the DataSet with information from the select query
Adapt.Fill(DS);

// Creates a view of the data stored in the dataset
DV = new DataView(DS.Tables[0]);

// sets the datagridviewer Grid source to that of the DataTable table
// that is filled with data from the select query
Grid.DataSource = DV;

//Closes the connection to the database
Conn.Close();
}

catch(Exception) //catches error
{
    //Displays a message box informing the user that their input is invalid
    MessageBox.Show("Invalid Input!");
}

}

//if the textbox tbSearch is empty
else if (tbSearch.Text == "")
{
    //A select query to select everything in the table called AnimeGenres
    Cmd.CommandText = "SELECT * FROM AnimeGenres";

    //Creates A connection between the datasource(select query) and dataset
    //to Add rows or refresh rows
    OleDbDataAdapter da = new OleDbDataAdapter(Cmd);

    //Creates a table that will be filled with data from the data adapter
    DataTable table = new DataTable();

    //the data adapter da adds rows to the table
    da.Fill(table);

    // sets the datagridviewer Grid source to that of the DataTable table
    // that is filled with data from the select query
    Grid.DataSource = table;

    //Closes the connection to the database
    Conn.Close();

    //Displays the DataTable
    Grid.Show();
}

}

//if the radiobutton rbAnimeStatus is clicked by the User
if (rbAnimeStatus.Checked)
```

```

{
  //if the textbox tbSearch is not empty
  if (tbSearch.Text != "")
  {
    try //Exception handling
    {
      //Gets value entered from the user to indicate the size of the columns and
      fill it with data
      Grid.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

      //Creates A connection between the datasource(select query) and dataset
      to Add rows or refresh rows
      OleDbDataAdapter Adapt = new OleDbDataAdapter();

      //Creates an empty set for data to be filled
      DataSet DS = new DataSet();

      //Creates an empty set for data to be filled
      DataView DV = new DataView();

      // A select query that gets everything from the table AnimeStatus, when the
      user
      // types a letter it is checked and compared to entries in the table
      // to determine whetere a specific character string matches a specified
      pattern
      //in either the column AnimeID or Status
      string SQL = "SELECT * FROM AnimeStatus WHERE AnimeID LIKE " +
      tbSearch.Text + "%" OR Status LIKE " + tbSearch.Text + "%";

      //Opens a connection to the database
      Conn.Open();

      //Creates a database command object that contains the connection to the
      //database and the string containing the select query
      Cmd = new OleDbCommand(SQL, Conn);

      //Creates A connection between the datasource(select query) and dataset
      to refresh rows from the command
      Adapt = new OleDbDataAdapter(Cmd);

      //Fills the DataSet with information from the select query
      Adapt.Fill(DS);

      // Creates a view of the the data stored in the dataset
      DV = new DataView(DS.Tables[0]);

      // sets the datagridviewer Grid source to that of the DataTable table
      // that is filled with data from the select query
      Grid.DataSource = DV;

      //Closes the connection to the database
      Conn.Close();
    }
    catch(Exception) //catches error
  }
}

```

```
{  
    //Displays a message box informing the user that their input is invalid  
    MessageBox.Show("Invalid input");  
}  
}  
  
//if the textbox tbSearch is empty  
else if (tbSearch.Text == "")  
{  
    //A select query to select everything from the table called AnimeStatus  
    Cmd.CommandText = "SELECT * FROM AnimeStatus";  
  
    //Creates A connection between the datasource(select query) and dataset  
    //to Add rows or refresh rows  
    OleDbDataAdapter da = new OleDbDataAdapter(Cmd);  
  
    //Creates a table that will be filled with data from the data adapter  
    DataTable table = new DataTable();  
  
    //the data adapter da adds rows to the table  
    da.Fill(table);  
  
    // sets the datagridviewer Grid source to that of the DataTable table  
    // that is filled with data from the select query  
    Grid.DataSource = table;  
  
    //Closes the connection to the database  
    Conn.Close();  
  
    //Displays the DataTable  
    Grid.Show();  
}  
}  
  
//if the radiobutton rbUsers is clicked by the User  
else if (rbUsers.Checked)  
{  
    //if the textbox tbSearch is not empty  
    if (tbSearch.Text != "")  
    {  
        try //Exception handling  
        {  
            //Gets value entered from the user to indicate the size of the columns and  
            fill it with data  
            Grid.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;  
  
            //Creates A connection between the datasource(select query) and dataset  
            to Add rows or refresh rows  
            OleDbDataAdapter Adapt = new OleDbDataAdapter();  
  
            //Creates an empty set for data to be filled  
            DataSet DS = new DataSet();  
  
            //Creates an empty set for data to be filled
```

```
        DataView DV = new DataView();  
  
        // A select query that gets the UserID and Username from the table Users,  
when the user  
        // types a letter it is checked and compared to entries in the table  
        // to determine whether a specific character string matches a specified  
pattern  
        //in the column Username  
        string SQL = "SELECT UserID, Username FROM Users WHERE  
Username LIKE" + tbSearch.Text + "%";  
  
        //Opens a connection to the database  
        Conn.Open();  
  
        //Creates a database command object that contains the connection to the  
        //database and the string containing the select query  
        Cmd = new OleDbCommand(SQL, Conn);  
  
        //Creates A connection between the datasource(select query) and dataset  
        //to refresh rows from the command  
        Adapt = new OleDbDataAdapter(Cmd);  
  
        //Fills the DataSet with information from the select query  
        Adapt.Fill(DS);  
  
        // Creates a view of the the data stored in the dataset  
        DV = new DataView(DS.Tables[0]);  
  
        // sets the datagridviewer Grid source to that of the DataTable table  
        // that is filled with data from the select query  
        Grid.DataSource = DV;  
  
        //Closes the connection to the database  
        Conn.Close();  
    }  
    catch(Exception) //catches error  
    {  
        //Displays a message box informing the user that their input is invalid  
        MessageBox.Show("Invalid input");  
    }  
}  
  
//if the textbox tbSearch is empty  
else if (tbSearch.Text == "")  
{  
    //A select query to select the UserID and Username from the table called  
Users  
    Cmd.CommandText = "SELECT UserID, Username FROM Users";  
  
    //Creates A connection between the datasource(select query) and dataset  
    //to Add rows or refresh rows  
    OleDbDataAdapter da = new OleDbDataAdapter(Cmd);  
  
    //Creates a table that will be filled with data from the data adapter
```

```

DataTable table = new DataTable();

//the data adapter da adds rows to the table
da.Fill(table);

// sets the datagridviewer Grid source to that of the DataTable table
// that is filled with data from the select query
Grid.DataSource = table;

//Closes the connection to the database
Conn.Close();

//Displays the DataTable
Grid.Show();
}

}

}

catch(Exception)// catches the error
{
    //Displays a message box saying it does not exist
    MessageBox.Show("Does not exist");
}
}

/// <summary>
/// The resulting method when the user clicks on a cell from the grid
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>
private void Grid_CellContentClick(object sender, DataGridViewCellEventArgs e)
{
    //if the radiobutton rbAnimes is clicked by the User
    if (rbAnimes.Checked)
    {
        //if the cell's row is greater than or equal to 0
        if(e.RowIndex >=0)
        {
            //stores the cell's row into a datagridviewrow variable 'row'
            DataGridViewRow row = Grid.Rows[e.RowIndex];

            //Sets the global variable AnimeID to the cell value under the column AnimeID
            AnimeID= row.Cells["AnimeID"].Value.ToString();

            //Sets the global variable AnimeTitle to the cell value under the column
            AnimeTitle
            AnimeTitle = row.Cells["AnimeTitle"].Value.ToString();

            //Sets the global variable Episodes to the cell value under the column Episodes
            Episodes = row.Cells["Episodes"].Value.ToString();
        }
    }
}

```

```
//Sets the global variable Ratings to the cell value under the column Ratings
Ratings = row.Cells["Ratings"].Value.ToString();

//Sets the global variable AnimeSynopsis to the cell value under the column
//AnimeSynopsis
AnimeSynopsis = row.Cells["AnimeSynopsis"].Value.ToString();

//Sets the global variable AnimePictures to the cell value under the column
//AnimePictures
AnimePictures = row.Cells["AnimePictures"].Value.ToString();

//Creates a variable of the Form FmEditAnimes
FmEditAnimes fmEditAnimes = new FmEditAnimes();

//Shows the form FmEditAnimes through the variable
fmEditAnimes.ShowDialog();

}

}

//if the radiobutton rbStudios is clicked by the User
else if (rbStudios.Checked)
{
    //if the cell's row is greater than or equal to 0
    if (e.RowIndex >=0)
    {
        //stores the cell's row into a datagridviewrow variable 'row'
        DataGridViewRow row = Grid.Rows[e.RowIndex];

        //Sets the global variable StudioID to the cell value under the column StudioID
        StudioID = row.Cells["StudioID"].Value.ToString();

        //Sets the global variable Studio to the cell value under the column Studio
        Studio = row.Cells["Studio"].Value.ToString();

        //Creates a variable of the Form FmEditStudios
        FmEditStudios fmEditStudios = new FmEditStudios();

        //Shows the form FmEditStudios through the variable
        fmEditStudios.ShowDialog();

    }
}

//if the radiobutton rbGenres is clicked by the User
else if (rbGenres.Checked)
{
    //if the cell's row is greater than or equal to 0
    if (e.RowIndex >=0)
    {
        //stores the cell's row into a datagridviewrow variable 'row'
        DataGridViewRow row = Grid.Rows[e.RowIndex];
```

```
//Sets the global variable GenreID to the cell value under the column GenreID
GenreID = row.Cells["GenreID"].Value.ToString();

//Sets the global variable Genre to the cell value under the column Genre
Genre = row.Cells["Genre"].Value.ToString();

//Creates a variable of the Form FmEditGenres
FmEditGenres fmEditGenres = new FmEditGenres();

//Shows the form FmEditGenres through the variable
fmEditGenres.ShowDialog();
}

}

//if the radiobutton rbAnimeGenres is clicked by the User
else if (rbAnimeGenres.Checked)
{
    //if the cell's row is greater than or equal to 0
    if (e.RowIndex >= 0)
    {
        //stores the cell's row into a datagridviewrow variable 'row'
        DataGridViewRow row = Grid.Rows[e.RowIndex];

        //Sets the global variable AnimelD to the cell value under the column AnimelD
        AnimelD = row.Cells["AnimelD"].Value.ToString();

        //Sets the global variable GenreID to the cell value under the column GenreID
        GenreID = row.Cells["GenreID"].Value.ToString();

        //Creates a variable of the Form FmEditAnimeGenres
        FmEditAnimeGenres fmEditAnimeGenres = new FmEditAnimeGenres();

        //Shows the form FmEditAnimeGenres through the variable
        fmEditAnimeGenres.ShowDialog();
    }
}

//if the radiobutton rbAnimeStudios is clicked by the User
else if (rbAnimeStudios.Checked)
{
    //if the cell's row is greater than or equal to 0
    if (e.RowIndex >= 0)
    {
        //stores the cell's row into a datagridviewrow variable 'row'
        DataGridViewRow row = Grid.Rows[e.RowIndex];

        //Sets the global variable AnimelD to the cell value under the column AnimelD
        AnimelD = row.Cells["AnimelD"].Value.ToString();

        //Sets the global variable StudioID to the cell value under the column StudioID
        StudioID = row.Cells["StudioID"].Value.ToString();

        //Creates a variable of the Form FmEditAnimeStudios
        FmEditAnimeStudios fmEditAnimeStudios = new FmEditAnimeStudios();
```

```

        //Shows the form FmEditAnimeStudios through the variable
        fmEditAnimeStudios.ShowDialog();
    }

}

//if the radiobutton rbAnimeStatus is clicked by the User
else if (rbAnimeStatus.Checked)
{

    //if the cell's row is greater than or equal to 0
    if (e.RowIndex >=0)
    {

        //stores the cell's row into a datagridviewrow variable 'row'
        DataGridViewRow row = Grid.Rows[e.RowIndex];

        //Sets the global variable AnimeID to the cell value under the column AnimeID
        AnimeID = row.Cells["AnimeID"].Value.ToString();

        //Sets the global variable Status to the cell value under the column Status
        Status = row.Cells["Status"].Value.ToString();

        //Creates a variable of the Form FmEditAnimeStatus
        FmEditAnimeStatus fmEditAnimeStatus = new FmEditAnimeStatus();

        //Shows the form FmEditAnimeStatus through the variable
        fmEditAnimeStatus.ShowDialog();

    }
}

//if the radiobutton rbUsers is clicked by the User
else if (rbUsers.Checked)
{
    //if the cell's row is greater than or equal to 0
    if (e.RowIndex >= 0)
    {

        //stores the cell's row into a datagridviewrow variable 'row'
        DataGridViewRow row = Grid.Rows[e.RowIndex];

        //Sets the global variable UserID to the cell value under the column AnimeID
        UserID = row.Cells["UserID"].Value.ToString();

        //Sets the global variable Username to the cell value under the column
        Username
        Username = row.Cells["Username"].Value.ToString();

        //Creates a variable of the Form FmDeleteUser
        FmDeleteUser fmDeleteUser = new FmDeleteUser();

        //Shows the form FmDeleteUser through the variable
        fmDeleteUser.ShowDialog();

    }
}

```

```
/// <summary>
/// The resulting method when the user clicks on the Edit button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>
private void btEdit_Click(object sender, EventArgs e)
{
    //if the radiobutton rbAnimes is clicked by the User
    if (rbAnimes.Checked)
    {
        //Creates a variable of the Form FmEditAnimes
        FmEditAnimes fmEditAnimes = new FmEditAnimes();

        //Shows the form FmEditAnimes through the variable
        fmEditAnimes.ShowDialog();

    }

    //if the radiobutton rbStudios is clicked by the User
    else if (rbStudios.Checked)
    {
        //Creates a variable of the Form FmEditStudios
        FmEditStudios fmEditStudios = new FmEditStudios();

        //Shows the form FmEditStudios through the variable
        fmEditStudios.ShowDialog();

    }

    //if the radiobutton rbGenres is clicked by the User
    else if (rbGenres.Checked)
    {
        //Creates a variable of the Form FmEditGenres
        FmEditGenres fmEditGenres = new FmEditGenres();

        //Shows the form FmEditGenres through the variable
        fmEditGenres.ShowDialog();

    }

    //if the radiobutton rbAnimeStudios is clicked by the User
    else if (rbAnimeStudios.Checked)
    {
        //Creates a variable of the Form FmEditAnimeStudios
        FmEditAnimeStudios fmEditAnimeStudios = new FmEditAnimeStudios();

        //Shows the form FmEditAnimeStudios through the variable
        fmEditAnimeStudios.ShowDialog();

    }

    //if the radiobutton rbAnimeGenres is clicked by the User
    else if (rbAnimeGenres.Checked)
    {
        //Creates a variable of the Form FmEditAnimeGenres
```

```
FmEditAnimeGenres fmEditAnimeGenres = new FmEditAnimeGenres();

//Shows the form FmEditAnimeGenres through the variable
fmEditAnimeGenres.ShowDialog();
}

//if the radiobutton rbAnimeStatus is clicked by the User
else if (rbAnimeStatus.Checked)
{
    //Creates a variable of the Form FmEditAnimeStatus
    FmEditAnimeStatus fmEditAnimeStatus = new FmEditAnimeStatus();

    //Shows the form FmEditAnimeStatus through the variable
    fmEditAnimeStatus.ShowDialog();
}

//if the radiobutton rbUsers is clicked by the User
else if (rbUsers.Checked)
{
    //Creates a variable of the Form FmDeleteUser
    FmDeleteUser fmDeleteUser = new FmDeleteUser();

    //Shows the form FmDeleteUser through the variable
    fmDeleteUser.ShowDialog();
}

}

/// <summary>
/// The resulting method when the user clicks on the MainForm button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance</param>
/// <param name="e">The <see cref="System.EventArgs"/> that represents the event
arguments passed
/// to the event handler</param>
private void btMainForm_Click(object sender, EventArgs e)
{
    //Creates a variable of the Form FmAMS
    FmAMS fmAMS = new FmAMS();

    //Shows the form FmAMS through the variable
    fmAMS.ShowDialog();
}
}
```

Form: FmEditAnimes.cs

```
using System;
using System.Collections.Generic;
```

Centre Number:32455

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in
    <c>FmEditAnimes</c>
    /// </summary>
    public partial class FmEditAnimes : Form
    {
        /// <summary>
        /// Required Method for Designer support for <c>FmEditAnimes</c>
        /// </summary>
        public FmEditAnimes()
        {
            InitializeComponent();
        }

        /// <summary>
        /// The resulting method for when the Form loads
        /// </summary>
        /// <param name="sender"> The <see cref="System.Object"/> that represents the
        sender instance </param>
        /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
        arguments passed to the event handler </param>

        private void FmEditAnimes_Load(object sender, EventArgs e)
        {
            //Sets the textbox tbAnimelD to the value of the global variable AnimelD from
            FmAMS(Admin) which contains a cell
            tbAnimelD.Text = FmAMS_admin_.AnimelD;

            //Sets the textbox tbAnimeTitle to the value of the global variable AnimeTitle from
            FmAMS(Admin) which contains a cell
            tbAnimeTitle.Text = FmAMS_admin_.AnimeTitle;

            //Sets the textbox tbEpisodes to the value of the global variable Episodes from
            FmAMS(Admin) which contains a cell
            tbEpisodes.Text = FmAMS_admin_.Episodes;

            //Sets the textbox tbRatings to the value of the global variable Ratings from
            FmAMS(Admin) which contains a cell
            tbRatings.Text = FmAMS_admin_.Ratings;

            //Sets the textbox tbAnimeSynopsis to the value of the global variable
            AnimeSynopsis from FmAMS(Admin) which contains a cell
            tbAnimeSynopsis.Text = FmAMS_admin_.AnimeSynopsis;
        }
    }
}
```

```
//Sets the textbox tbAnimePictures to the value of the global variable AnimeID from
FmAMS(Admin) which contains a cell
tbAnimePictures.Text = FmAMS_admin_.AnimePictures;
}
/// <summary>
/// The resulting method for when the user clicks on the insert button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btInsert_Click(object sender, EventArgs e)
{
    //Exception Handling
    try
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //Insert the textboxes, tbAnimeID, tbAnimeTitle, tbEpisodes, tbRatings,
        tbAnimeSynopsis and tbAnimePictures to the table Animes.
        Cmd.CommandText = "INSERT INTO Animes
VALUES(" + tbAnimeID.Text + "," + tbAnimeTitle.Text + "," + tbEpisodes.Text
+ "," + tbRatings.Text + "," + tbAnimeSynopsis.Text + "," + tbAnimePictures.Text + ")";

        //Opens Connection to the database
        Conn.Open();

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        // Closes the connection to the database
        Conn.Close();

        //Displays a message box informing the user that everything from the textbox has
        been added!
        MessageBox.Show("Information successfully added");
    }
    catch(Exception) //Catches error
    {
        //Displays a message box informing the user that it could not be added
        MessageBox.Show("Could not be added");
    }
}
/// <summary>
/// The resulting method for when the user clicks on the update button
```

```
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btUpdate_Click(object sender, EventArgs e)
{
    try//Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //Updates the information contained in AnimeTitle, Episodes, Ratings,
        AnimeSynopsis and AnimePictures with the textboxes tbAnimeTitle, tbEpisodes, tbRatings,
        tbAnimeSynopsis and tbAnimePictures. This update will happen where AnimeID is the same
        as the value in textbox tbAnimeID
        Cmd.CommandText = "UPDATE Animes SET AnimeTitle='" + tbAnimeTitle.Text +
        "', Episodes=" + tbEpisodes.Text + ", Ratings=" + tbRatings.Text + ", AnimeSynopsis=" +
        tbAnimeSynopsis.Text + ", AnimePictures=" + tbAnimePictures.Text + " WHERE
        AnimeID=" + tbAnimelD.Text + "";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box informing the user that the information has been
updated
        MessageBox.Show("Information updated!");

        // Closes the connection to the database
        Conn.Close();
    }
    catch(Exception ) //Catches the error
    {
        //Displays a message box informing the user that the update could not happen
        MessageBox.Show("Could not update!");
    }
}

/// <summary>
/// The resulting method for when the user clicks on the Delete button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
```

```
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btDelete_Click(object sender, EventArgs e)
{
    try//Exception Handling
    {

        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //The delete query will happen on the row containing the AnimelD that's the same
        as the textbox tbAnimelD
        Cmd.CommandText = "DELETE FROM Animes WHERE AnimelD=" +
        tbAnimelD.Text + "";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box telling the user that the information is now deleted
        MessageBox.Show("Information is now deleted!");

        // Closes the connection to the database
        Conn.Close();
    }
    catch(Exception)//Catches error
    {
        //Displays a message box telling the user that the information could not be deleted
        MessageBox.Show("Information could not be deleted!");
    }
}

/// <summary>
/// The resulting method for when the user clicks on the Clear button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btClear_Click(object sender, EventArgs e)
{
    //Clears the text that was entered in the textbox tbAnimelD
    tbAnimelD.Clear();

    //Clears the text that was entered in the textbox tbAnimeTitle
}
```

```
tbAnimeTitle.Clear();

//Clears the text that was entered in the textbox tbEpisodes
tbEpisodes.Clear();

//Clears the text that was entered in the textbox tbRatings
tbRatings.Clear();

//Clears the text that was entered in the textbox tbAnimePictures
tbAnimePictures.Clear();

//Clears the text that was entered in the textbox tbAnimeSynopsis
tbAnimeSynopsis.Clear();

}

}
```

Form: FmEditStudios.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in
    <c>FmEditStudios</c>
    /// </summary>
    public partial class FmEditStudios : Form
    {
        /// <summary>
        /// Required Method for Designer support for <c>FmEditStudios</c>
        /// </summary>
        public FmEditStudios()
        {
            InitializeComponent();
        }
        /// <summary>
        /// The resulting method for when the Form loads
        /// </summary>
        /// <param name="sender"> The <see cref="System.Object"/> that represents the
        sender instance </param>
        /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
        arguments passed to the event handler </param>
```

```
private void FmEditStudios_Load(object sender, EventArgs e)
{
    //Sets the textbox tbStudioID to the value of the global variable StudioID from
    FmAMS(Admin) which contains a cell
    tbStudioID.Text = FmAMS_admin_.StudioID;

    //Sets the textbox tbStudio to the value of the global variable Studio from
    FmAMS(Admin) which contains a cell
    tbStudio.Text = FmAMS_admin_.Studio;
}

/// <summary>
/// The resulting method for when the user clicks on the insert button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btInsert_Click(object sender, EventArgs e)
{
    try//Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //Insert the textboxes, tbStudioID, tbStudios to the table Studios.
        Cmd.CommandText = "INSERT INTO Studios
Values("+tbStudioID.Text+",""+tbStudio.Text+")";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box telling the user that everything from the textbox has
been added!
        MessageBox.Show("Information successfully added!");

        // Closes the connection to the database
        Conn.Close();
    }

    catch(Exception) //Catches error
    {
        //Displays a message box informing the user that it could not be added
        MessageBox.Show("Information could not be added!");
    }
}
```

```
}

/// <summary>
/// The resulting method for when the user clicks on the update button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btUpdate_Click(object sender, EventArgs e)
{
    try//Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //Updates the information contained in Studio with the textbox tbStudio. This
        update will take place in where the StudioID is same as the textbox tbStudioID
        Cmd.CommandText = "UPDATE Studios SET Studio= "+tbStudio.Text+" WHERE
        StudioID="+tbStudioID.Text+"";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box informing the user that the information has been
        updated
        MessageBox.Show("Information updated!");

        // Closes the connection to the database
        Conn.Close();
    }
    catch (Exception)//Catches Error
    {
        //Displays a message box informing the user that the update could not happen
        MessageBox.Show("Information could not be updated!");
    }
}

/// <summary>
/// The resulting method for when the user clicks on the Delete button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btDelete_Click(object sender, EventArgs e)
```

```
{  
    try //Exception Handling  
    {  
        //Create a database connection object with the constant defined in program.cs  
        OleDbConnection Conn = new OleDbConnection(Program.connString);  
  
        //Opens Connection to the database  
        Conn.Open();  
  
        //Creates a database command object  
        OleDbCommand Cmd = new OleDbCommand();  
  
        //Creates a connection with the Command Object  
        Cmd.Connection = Conn;  
  
        //The delete query will happen on the row containing the StudioID that's the same  
        as the textbox tbStudioID  
        Cmd.CommandText = "DELETE FROM Studios WHERE StudioID="  
        "+tbStudioID.Text+";  
  
        //Executes the SQL Query  
        Cmd.ExecuteNonQuery();  
  
        //Displays a message box informing the user that the information is now deleted  
        MessageBox.Show("Information deleted!");  
  
        // Closes the connection to the database  
        Conn.Close();  
  
    }  
    catch (Exception) //Catches error  
    {  
  
        //Displays a message box informing the user that the information could not be  
        deleted  
        MessageBox.Show("Information could not be deleted!");  
    }  
}  
  
/// <summary>  
/// The resulting method for when the user clicks on the Clear button  
/// </summary>  
/// <param name="sender"> The <see cref="System.Object"/> that represents the  
sender instance </param>  
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event  
arguments passed to the event handler </param>  
private void btClear_Click(object sender, EventArgs e)  
{  
    //Clears the text that was entered in the textbox tbStudioID  
    tbStudioID.Clear();  
  
    //Clears the text that was entered in the textbox tbStudio  
    tbStudio.Clear();
```

```
    }  
}  
}
```

Form: FmEditGenres.cs

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
using System.Data.OleDb;  
  
namespace AMS  
{  
    /// <summary>  
    /// Handles all inputs and validation of functions that will take place in  
<c>FmEditGenres</c>  
    /// </summary>  
    public partial class FmEditGenres : Form  
    {  
        /// <summary>  
        /// Required Method for Designer support for <c>FmEditGenres</c>  
        /// </summary>  
        public FmEditGenres()  
        {  
            InitializeComponent();  
  
        }  
        /// <summary>  
        /// The resulting method for when the Form loads  
        /// </summary>  
        /// <param name="sender"> The <see cref="System.Object"/> that represents the  
sender instance </param>  
        /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event  
arguments passed to the event handler </param>  
  
        private void FmEditGenres_Load(object sender, EventArgs e)  
        {  
            //Sets the textbox tbGenreID to the value of the global variable GenreID from  
FmAMS(Admin) which contains a cell  
            tbGenreID.Text = FmAMS_admin_.GenreID;  
  
            //Sets the textbox tbGenre to the value of the global variable Genre from  
FmAMS(Admin) which contains a cell  
            tbGenre.Text = FmAMS_admin_.Genre;  
        }  
    }
```

```
/// <summary>
/// The resulting method for when the user clicks on the insert button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btInsert_Click(object sender, EventArgs e)
{
    try//Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        // Insert the textboxes, tbGenreID, tbGenre to the table Genres.
        Cmd.CommandText = "INSERT INTO Genres VALUES(" + tbGenreID.Text + ", " + 
tbGenre.Text + ")";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box telling the user that everything from the textbox has
been added!
        MessageBox.Show("Information successfully added!");

        // Closes the connection to the database
        Conn.Close();

    }
    catch(Exception) //catches error
    {
        //Displays a message box informing the user that it could not be added
        MessageBox.Show("Information could not be added!");
    }
}
/// <summary>
/// The resulting method for when the user clicks on the update button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>

private void btUpdate_Click(object sender, EventArgs e)
```

```
{  
    try //Exception Handling  
    {  
        //Create a database connection object with the constant defined in program.cs  
        OleDbConnection Conn = new OleDbConnection(Program.connString);  
  
        //Opens Connection to the database  
        Conn.Open();  
  
        //Creates a database command object  
        OleDbCommand Cmd = new OleDbCommand();  
  
        //Creates a connection with the Command Object  
        Cmd.Connection = Conn;  
  
        //Updates the information contained in Studio with the textbox tbStudio. This  
        update will take place in where the GenrelID is same as the textbox tbGenrelID  
        Cmd.CommandText ="UPDATE Genres SET Genre = "+tbGenre.Text+" WHERE  
        GenrelID="+tbGenrelID.Text+";  
  
        //Executes the SQL Query  
        Cmd.ExecuteNonQuery();  
  
        //Displays a message box informing the user that the information has been  
        updated  
        MessageBox.Show("Information updated!");  
  
        // Closes the connection to the database  
        Conn.Close();  
    }  
    catch(Exception) //catches error  
    {  
        //Displays a message box informing the user that the update could not happen  
        MessageBox.Show("Information could not be updated!");  
    }  
}  
/// <summary>  
/// The resulting method for when the user clicks on the Delete button  
/// </summary>  
/// <param name="sender"> The <see cref="System.Object"/> that represents the  
sender instance </param>  
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event  
arguments passed to the event handler </param>  
private void btDelete_Click(object sender, EventArgs e)  
{  
    try//Exception Handling  
    {  
  
        //Create a database connection object with the constant defined in program.cs  
        OleDbConnection Conn = new OleDbConnection(Program.connString);  
  
        //Opens Connection to the database  
        Conn.Open();  
    }
```

```
//Creates a database command object
OleDbCommand Cmd = new OleDbCommand();

//Creates a connection with the Command Object
Cmd.Connection = Conn;

//The delete query will happen on the row containing the GenreID that's the same
as the textbox tbGenreID
Cmd.CommandText = "DELETE FROM Genres WHERE GenreID=
"+tbGenreID.Text+"";

//Executes the SQL Query
Cmd.ExecuteNonQuery();

//Displays a message box informing the user that the information is now deleted
MessageBox.Show("Information deleted!");

// Closes the connection to the database
Conn.Close();

}

catch(Exception) //catches error
{
    //Displays a message box informing the user that the information could not be
    deleted
    MessageBox.Show("Information could not be deleted!");
}

}

/// <summary>
/// The resulting method for when the user clicks on the Clear button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btClear_Click(object sender, EventArgs e)
{
    //Clears the text that was entered in the textbox tbGenreID
    tbGenreID.Clear();

    //Clears the text that was entered in the textbox tbGenre
    tbGenre.Clear();
}
```

Form: FmEditAnimeStudios.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in
    <c>FmEditAnimeStudios</c>
    /// </summary>
    public partial class FmEditAnimeStudios : Form
    {
        /// <summary>
        /// Required Method for Designer support for <c>FmEditAnimeStudios</c>
        /// </summary>
        public FmEditAnimeStudios()
        {
            InitializeComponent();
        }
        /// <summary>
        /// The resulting method for when the Form loads
        /// </summary>
        /// <param name="sender"> The <see cref="System.Object"/> that represents the
        sender instance </param>
        /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
        arguments passed to the event handler </param>
        private void FmEditAnimeStudios_Load(object sender, EventArgs e)
        {
            //Sets the textbox tbAnimeID to the value of the global variable AnimeID from
            FmAMS(Admin) which contains a cell
            tbAnimeID.Text = FmAMS_admin_.AnimeID;

            //Sets the textbox tbStudioID to the value of the global variable StudioID from
            FmAMS(Admin) which contains a cell
            tbStudioID.Text = FmAMS_admin_.StudioID;
        }
        /// <summary>
        /// The resulting method for when the user clicks on the insert button
        /// </summary>
        /// <param name="sender"> The <see cref="System.Object"/> that represents the
        sender instance </param>
        /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
        arguments passed to the event handler </param>
        private void btInsert_Click(object sender, EventArgs e)
        {
            try//Exception Handling
            {
                //Create a database connection object with the constant defined in program.cs
                OleDbConnection Conn = new OleDbConnection(Program.connString);

                //Opens Connection to the database
                Conn.Open();
            }
        }
    }
}
```

```
//Creates a database command object
OleDbCommand Cmd = new OleDbCommand();

//Creates a connection with the Command Object
Cmd.Connection = Conn;

//Insert the textboxes, tbAnimelD and tbStudioID to the table AnimeStudios.
Cmd.CommandText = "INSERT INTO AnimeStudios
VALUES("+tbAnimelD.Text+","+tbStudioID.Text +)";

//Executes the SQL Query
Cmd.ExecuteNonQuery();

//Displays a message box informing the user that everything from the textbox and
the combobox has been added!
MessageBox.Show("Information successfully inserted!");

// Closes the connection to the database
Conn.Close();

}

catch(Exception) //catches error
{
    //Displays a message box informing the user that it could not be added
    MessageBox.Show("Could not be inserted");
}

}

/// <summary>
/// The resulting method for when the user clicks on the update button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btUpdate_Click(object sender, EventArgs e)
{
    try //Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //Updates the information contained in StudioID with the textbox tbStudioID. This
        update will take place in where the AnimelD is same as the textbox tbAnimelD
```

```
Cmd.CommandText = "UPDATE AnimeStudios SET StudioID=" + tbStudioID.Text
+" WHERE AnimeID=" + tbAnimeID.Text +";

//Executes the SQL Query
Cmd.ExecuteNonQuery();

//Displays a message box informing the user that the information has been
updated
MessageBox.Show("Information successfully updated!");

// Closes the connection to the database
Conn.Close();
}

catch(Exception) //catches error
{
    //Displays a message box informing the user that the update could not happen
    MessageBox.Show("Information could not be updated!");
}

/// <summary>
/// The resulting method for when the user clicks on the Delete button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btDelete_Click(object sender, EventArgs e)
{
    try//Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //The delete query will happen on the row containing the AnimeID that's the same
        as the textbox tbAnimeID
        Cmd.CommandText = "DELETE FROM AnimeStudios WHERE AnimeID=
" + tbAnimeID.Text +";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box informing the user that the information is now deleted
        MessageBox.Show("Information deleted!");

        // Closes the connection to the database
    }
}
```

```
Conn.Close();
}
catch(Exception) //catches error
{
    //Displays a message box informing the user that the information could not be
    //deleted
    MessageBox.Show("Information Could not be deleted!");
}

/// <summary>
/// The resulting method for when the user clicks on the Clear button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btClear_Click(object sender, EventArgs e)
{
    //Clears the text that was entered in the textbox tbAnimeID
    tbAnimeID.Clear();

    //Clears the text that was entered in the textbox tbStudioID
    tbStudioID.Clear();
}
}
```

Form: FmEditAnimeGenres.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in
    <c>FmEditAnimeGenres</c>
    /// </summary>
    public partial class FmEditAnimeGenres : Form
    {
        /// <summary>
        /// Required Method for Designer support for <c>FmEditAnimeGenres</c>
        /// </summary>
        public FmEditAnimeGenres()
        {

```

```
    InitializeComponent();
}
/// <summary>
/// The resulting method for when the Form loads
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void FmEditAnimeGenres_Load(object sender, EventArgs e)
{
    //Sets the textbox tbAnimeID to the value of the global variable AnimeID from
FmAMS(Admin) which contains a cell
    tbAnimeID.Text = FmAMS_admin_.AnimeID;

    //Sets the textbox tbGenreID to the value of the global variable GenreID from
FmAMS(Admin) which contains a cell
    tbGenreID.Text = FmAMS_admin_.GenreID;
}
/// <summary>
/// The resulting method for when the user clicks on the insert button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btInsert_Click(object sender, EventArgs e)
{
    try// Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //Insert the textboxes, tbAnimeID and tbGenreID to the table AnimeGenres.
        Cmd.CommandText = "INSERT INTO AnimeGenres VALUES(" + tbAnimeID.Text
+ "," + tbGenreID.Text + ")";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box informing the user that everything from the textbox and
the combobox has been added!
        MessageBox.Show("Information successfully added!");

        // Closes the connection to the database
    }
}
```

```
        Conn.Close();

    }

    catch (Exception) //catches error
    {
        //Displays a message box informing the user that it could not be added
        MessageBox.Show("Information could not be added!");
    }
}

/// <summary>
/// The resulting method for when the user clicks on the update button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btUpdate_Click(object sender, EventArgs e)
{
    try//Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //Updates the information contained in GenreID with the textbox tbGenreID. This
        update will take place in where the AnimeID is same as the textbox tbAnimeID
        Cmd.CommandText = "UPDATE AnimeGenres SET GenreID= "+tbGenreID.Text+
        WHERE AnimeID="+tbAnimeID.Text+"";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box informing the user that the information has been
        updated
        MessageBox.Show("Information Updated!");

        // Closes the connection to the database
        Conn.Close();
    }

    catch (Exception) //catches error
    {
        //Displays a message box informing the user that the update could not happen
        MessageBox.Show("Information could not be updated!");
    }
}
```

```
}

/// <summary>
/// The resulting method for when the user clicks on the Delete button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btDelete_Click(object sender, EventArgs e)
{
    try//Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //The delete query will happen on the row containing the AnimeID that's the same
        as the textbox tbAnimeID
        Cmd.CommandText = "DELETE FROM AnimeGenres WHERE AnimeID=
" +tbAnimeID.Text+"";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box informing the user that the information is now deleted
        MessageBox.Show("Information Deleted!");

        // Closes the connection to the database
        Conn.Close();
    }
    catch (Exception) //catches error
    {
        //Displays a message box informing the user that the information could not be
        deleted
        MessageBox.Show("Information could not be deleted!");
    }
}

/// <summary>
/// The resulting method for when the user clicks on the Clear button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
```

```
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btClear_Click(object sender, EventArgs e)
{
    //Clears the text that was entered in the textbox tbAnimelD
    tbAnimelD.Clear();

    //Clears the text that was entered in the textbox tbGenreID
    tbGenreID.Clear();
}
}
```

Form: FmEditAnimeStatus.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in
    <c>FmEditAnimeStatus</c>
    /// </summary>
    public partial class FmEditAnimeStatus : Form
    {
        /// <summary>
        /// Required Method for Designer support for <c>FmEditAnimeStatus</c>
        /// </summary>
        public FmEditAnimeStatus()
        {
            InitializeComponent();
        }
        /// <summary>
        /// The resulting method for when the Form loads
        /// </summary>
        /// <param name="sender">The <see cref="System.Object"/> that represents the
        sender instance </param>
        /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
        arguments passed to the event handler </param>
        private void FmEditAnimeStatus_Load(object sender, EventArgs e)
        {
            //Sets the textbox tbAnimeID to the value of the global variable AnimeID from
            FmAMS(Admin) which contains a cell
            tbAnimeID.Text = FmAMS.admin.AnimeID;
        }
    }
}
```

```
//Sets the combobox cbStatus to the value of the global variable Status from
FmAMS(Admin) which contains a cell
cbStatus.Text = FmAMS_admin_.Status;
}
/// <summary>
/// The resulting method for clearing the combobox and setting it to default
/// </summary>
private void ResetSelection()
{
    //Sets the combo box to no element selected when - by 1
    cbStatus.SelectedIndex = -1;

    //Sets the text of the combobox to Select Status
    cbStatus.Text = "Select Status";
}

/// <summary>
/// The resulting method for when the user clicks on the insert button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btInsert_Click(object sender, EventArgs e)
{
    try//Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //Insert the textbox, tbAnimeID and combobox cbStatus to the table AnimeStatus.
        Cmd.CommandText = "INSERT INTO AnimeStatus VALUES(" + tbAnimeID.Text +
", " + cbStatus.Text + ")";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box informing the user that everything from the textbox and
        the combobox has been added!
        MessageBox.Show("Information successfully added!");

        // Closes the connection to the database
        Conn.Close();
    }
}
```

```
        catch(Exception) //Catches Error
    {
        //Displays a message box informing the user that it could not be added
        MessageBox.Show("Could not be added");
    }
}
/// <summary>
/// The resulting method for when the user clicks on the update button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btUpdate_Click(object sender, EventArgs e)
{
    try//Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //Updates the information contained in Status with the combobox cbStatus. This
update will take place in the location where the AnimeID is same as the textbox tbAnimeID
        Cmd.CommandText = "UPDATE AnimeStatus SET Status='" + cbStatus.Text + "'"
WHERE AnimeID='"+tbAnimeID.Text+"'";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box informing the user that the information has been
updated
        MessageBox.Show("Information Updated!");

        // Closes the connection to the database
        Conn.Close();
    }
    catch(Exception) //Catches error
    {
        //Displays a message box informing the user that the update could not happen
        MessageBox.Show("Could not update!");
    }
}
/// <summary>
/// The resulting method for when the user clicks on the Delete button
/// </summary>
```

```
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>

private void btDelete_Click(object sender, EventArgs e)
{
    try//Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //The delete query will happen on the row containing the AnimeID that's the same
        as the textbox tbAnimeID
        Cmd.CommandText = "DELETE FROM AnimeStatus WHERE AnimeID= " +
        tbAnimeID.Text + "";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box informing the user that the information is now deleted
        MessageBox.Show("Information Deleted!");

        // Closes the connection to the database
        Conn.Close();
    }
    catch(Exception) //Catches error
    {
        //Displays a message box informing the user that the information could not be
        deleted
        MessageBox.Show("Information could not be deleted");
    }
}

/// <summary>
/// The resulting method for when the user clicks on the Clear button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btClear_Click(object sender, EventArgs e)
{
    //Clears the text that was entered in the textbox tbAnimeID
    tbAnimeID.Clear();
}
```

```
        //Runs the function ResetSelection
        ResetSelection();
    }
}
```

Form: FmDeleteUser.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;

namespace AMS
{
    /// <summary>
    /// Handles all inputs and validation of functions that will take place in
    <c>FmDeleteUser</c>
    /// </summary>
    public partial class FmDeleteUser : Form
    {
        /// <summary>
        /// Required Method for Designer support for <c>FmDeleteUser</c>
        /// </summary>
        public FmDeleteUser()
        {
            InitializeComponent();
        }

        /// <summary>
        /// The resulting method for when the Form loads
        /// </summary>
        /// <param name="sender"> The <see cref="System.Object"/> that represents the
        sender instance </param>
        /// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
        arguments passed to the event handler </param>

        private void FmDeleteUser_Load(object sender, EventArgs e)
        {
            //Sets the textbox tbUserID to the value of the global variable UserID from
            FmAMS(Admin) which contains a cell
            tbUserID.Text = FmAMS_admin_.UserID;

            //Sets the textbox tbUsername to the value of the global variable Studio from
            FmLogin which contains the username of the user
            tbUsername.Text = FmAMS_admin_.Username;
        }
    }
}
```

```
/// <summary>
/// The resulting method for when the user clicks on the Delete button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void btDelete_Click(object sender, EventArgs e)
{
    try//Exception Handling
    {
        //Create a database connection object with the constant defined in program.cs
        OleDbConnection Conn = new OleDbConnection(Program.connString);

        //Opens Connection to the database
        Conn.Open();

        //Creates a database command object
        OleDbCommand Cmd = new OleDbCommand();

        //Creates a connection with the Command Object
        Cmd.Connection = Conn;

        //The delete query will happen on the row containing the UserID that's the same as
        the textbox tbUserID in UserFavourites
        Cmd.CommandText = "DELETE FROM UserFavourites WHERE
UserID=" + tbUserID.Text + "";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //The delete query will happen on the row containing the UserID that's the same as
        the textbox tbUserID in UserWatchList
        Cmd.CommandText = "DELETE FROM UserWatchList WHERE UserID=" +
        tbUserID.Text + "";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //The delete query will happen on the row containing the UserID that's the same as
        the textbox tbUserID in Users
        Cmd.CommandText = "DELETE FROM Users WHERE UserID=" + tbUserID.Text
        + " AND Username=" + tbUsername.Text + "";

        //Executes the SQL Query
        Cmd.ExecuteNonQuery();

        //Displays a message box informing the user that the user is now deleted
        MessageBox.Show("User is now deleted!");
    }
    catch (Exception) //catches error
    {
        //Displays a message box informing the user that the user could not be deleted
    }
}
```

```
        MessageBox.Show("User could not be deleted!");
    }

}

/// <summary>
/// The resulting method for when the user clicks on the Clear button
/// </summary>
/// <param name="sender"> The <see cref="System.Object"/> that represents the
sender instance </param>
/// <param name="e">The <see cref="System.EventArgs"/> that Represents the event
arguments passed to the event handler </param>
private void BtClear_Click(object sender, EventArgs e)
{
    //Clears the text that was entered in the textbox tbUserID
    tbUserID.Clear();

    //Clears the text that was entered in the textbox tbUsername
    tbUsername.Clear();
}

}
```

Testing

Test Plan

Test number	Links to objectives	Description of test	Expected Outcome	Actual Outcome	Page number
#1	1, 1.1, 1.2 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9	Running the program for the first time	Database creation.	As Expected.	226-227
#2	2, 2.1	Checking to see if there is content in the tables	All the tables should contain the data from the csv file except the tables Users, UserFavourites and UserWatchList.	As Expected.	227-232
#3	3, 3.1, 3.2, 3.3	Check to see if the form FmMenu has vbs 3 buttons that will will navigate me to the forms	The forms 'FmLogIn' and FmSignUp' will appear	As Expected.	232-233
#4	4, 4.1, 4.2, 4.2.1, 4.2.2, 4.2.3, 4.3, 4.3.2, 4.3.3, 4.4, 4.4.1, 4.5	Check to see if the validations work in the form FmSignUp and create an account	Message boxes will appear if I didn't put in my details in the right format, and	As Expected.	234-239

			a messagebox telling me that I have created an account		
#5	5, 5.1, 5.2, 5.2.1, 5.2.2, 5.2.3, 5.2.4	Check to see if the form 'FmLogIn' has everything in the form required to log in, successfully log in as a user and log in as an admin to see and see 2 different forms correspondent to the user type	The form FmAMS will show up for a normal user and the form FmAMS(Admin) will show up for an admin user	As Expected.	240-242
#6	6	Check to see if the form FmAMS(Admin) goes to the main form FmAMS through the button btMainForm	The form FmAMS will appear	As Expected.	242-243
#7	6.1, 6.2, 7	Check to see if all the tables that are necessary for the form FmAMS(Admin) will appear in the datagridviewer for by using the radio button in conjunction with the 'show' button	All the tables will appear	As Expected	243-247
#8	8, 8.1	Check to see if the edit button in FmAMS(admin) will navigate the user to the edit forms when used in conjunction with the radio buttons	All the edit forms will appear.	As Expected.	247-254
#9	9, 9.1	Check to see if the search bar works with the tables and check to see if it refreshes the tables in the datagridviewer	The search button will narrow down the table to what I input in	As Expected.	254-261
#10	10,10.1	Check to see if the cells in the data grid viewer in FmAMS(admin) are clickable and will navigate the user to the edit form with all the text fields filled with the cell row	The edit forms will contain all the information from the cell's row and will be put into the correct textboxes	As Expected	262-269
#11	11.11.2,11.3,11.4, 11.5	Adding an anime to the database	The anime will be added successfully,	As Expected.	269-270

		(FmAMS (Admin)). Test to see if the exception handling works	and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not add an anime due to an input issue		
#12	12.1,12.2,12.2.1	Updating anime information (FmAMS (Admin)). Test to see if the exception handling works	The information about an anime will be updated and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not update anime information due to an input issue	As Expected	271-272
#13	13, 13.1, 13.1.1, 13.2, 13.3	Deleting an anime of the database by deleting everything that is required to get rid of the anime. Test to see if the exception handling works	The deletion of the anime will happen, and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not delete an anime due to an input issue	As Expected	273-274
#14	14, 14.1, 14.2,14.3	Adding a genre to the database (FmAMS(Admin)). Test to see if the exception handling works	The genre will be added successfully, and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not add a genre due to an input issue	As Expected.	274-275

#15	15, 15.1, 15.2, 15.3	Updating genre information. Test to see if the exception handling works	The information about the genre will be updated and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not update the genre due to an input issue	As Expected.	276-277
#16	16, 16.1, 16.2, 16.3	Deleting a genre off the database (FmAMS (Admin)). Test to see if the exception handling works	The deletion of the genre will happen, and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not delete a genre due to an input issue	As Expected.	278-279
#17	17, 17.1, 17.2, 17.3	Adding a studio to the database (FmAMS (Admin)). Test to see if the exception handling works	The studio will be added successfully, and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not add a studio due to an input issue	As Expected.	279-280
#18	18, 18.1, 18.2, 18.3	Updating studio information. Test to see if the exception handling works	The information about the studio will be updated and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not update	As Expected.	281-282

			a studio due to an input issue		
#19	19, 19.2, 19.3	Deleting a studio off the database (FmAMS (Admin)). Test to see if the exception handling works	The deletion of the studio will happen, and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not delete the studio due to an input issue	As Expected.	283-284
#20	20.1, 20.1.1, 20.2, 20.3	Adding an AnimeStudio key to the database (link between AnimelD and StudioID). Test to see if the exception handling works	The AnimeStudio key will be added successfully, and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not add an AnimeStudio key due to an input issue	As Expected.	284-285
#21	21, 21.1, 21.2, 21.3	Updating an AnimeStudio key by changing the StudioID. Test to see if the exception handling works	The key will be updated and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not update the AnimeStudio key due to an input issue	As Expected.	285-286
#22	22, 22.1, 22.2, 22.3	Deleting an AnimeStudio key off the database. Test to see if the exception handling works	The deletion of the AnimeStudio Key will happen and I will be prompted with a message box confirming that. I will also be	As Expected.	287-288

			prompted with a message box confirming that I could not delete the AnimeStudio key due to an input issue		
#23	23, 23.1, 23.1.1, 23.2, 23.3	Adding an AnimeGenre key to the database. Test to see if the exception handling works	The AnimeGenre key will be added successfully, and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not add an AnimeGenre due to an input issue	As Expected.	288-289
#24	24, 24.1, 24.2, 24.3	Updating an AnimeGenre key by changing the GenreID. Test to see if the exception handling works	The AnimeGenre key will be updated and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not update the AnimeGenre key due to an input issue	As Expected.	290-291
#25	25, 25.1, 25.2, 25.3	Deleting an AnimeGenre key off the database. Test to see if the exception handling works	The deletion of the AnimeGenre Key will happen and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not delete the AnimeGenre key due to an input issue	As Expected.	291-292

#26	26, 26.1, 26.1.1, 26.2, 26.3	Adding an AnimeStatus key to the database. Test to see if the exception handling works	The AnimeStatus key will be added successfully, and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not add an AnimeStatus due to an input issue	As Expected.	292-293
#27	27, 27.1, 27.2, 27.3	Updating an AnimeStatus key by changing the GenreID. Test to see if the exception handling works	The AnimeStatus key will be updated, and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not update the AnimeStatus key due to an input issue	As Expected.	293-294
#28	28, 28.1, 28.2, 28.3	Deleting an AnimeStatus key off the database. Test to see if the exception handling works	The deletion of the AnimeStatus Key will happen, and I will be prompted with a message box confirming that. I will also be prompted with a message box confirming that I could not delete the AnimeStatus key due to an input issue	As Expected.	295-296
#29	29, 29.1, 29.2, 29.3	Deleting a user off the database. Test to see if the exception handling works	The deletion of a user will happen, and I will be prompted with a message box confirming that. I will also be prompted with a	As Expected	296-297

			message box confirming that I could not delete a user due to an input issue		
#30	32, 32.1, 32.2	Test to see if the news function refreshes in FmAMS	The news function will refresh	As Expected.	297
#31	33, 33.1, 33.2, 33.3, 33.4	Test to see if the search bar works and the search button works in FmAMS	The search bar will show the user a list of anime titles related to what they 're searching and the search button will show the picture associated with that anime	As Expected	298
#32	34, 34.1	Test to see if the form 'FmAnimeInformation' will show up after clicking on the anime picture	The form will load up	As Expected	299
#33	34.2, 34.3	Rating the anime, Adding the anime to favourites and watchlist. Test to see if it will only add the anime to favourites and watchlist once	Rating the anime will work by adding to favourites or watchlist after, I will be prompted with a message box telling me that the anime is already in watchlist and favourites if I try to add the anime again	As Expected	300-302
#34	30.1, 30.2	Test to see if the form 'FmProfile' loads up and contains user information	The form will load up and show the following information about the user, UserID, Username, Forename, Surname and UserType.	As Expected	303
#35	30.3, 30.4	Test to see if the form FmReset will appear after clicking on the reset button	The form will appear	As Expected	304

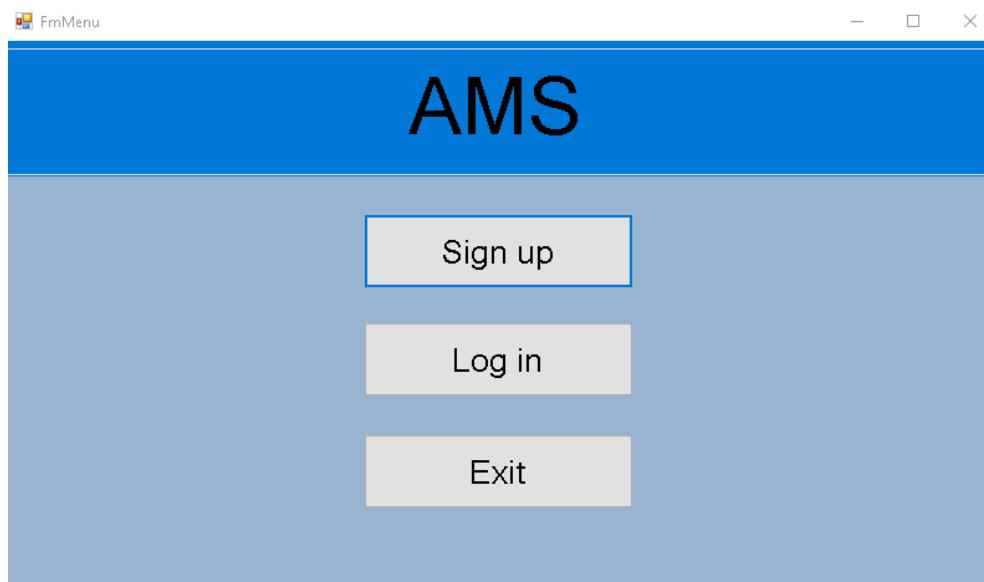
#36	30.5, 30.6, 30.7	Test to see if the email function works by entering the email to see the textboxes tbNewPassword and tbConfirmPassword. Test to see if the password changes. Test to see if the Exception handling works.	The textboxes will appear after entering the correct email and I will be prompted with a message box confirming the password change. I will also be prompted with message boxes about the passwords not being in the right format.	As Expected	305-308
#37	31, 31.1	Test to see if the forms 'FmUserFavourites' and 'FmUserWatchList' appears after clicking on the button Favourites and Watchlist	The forms will appear	As Expected	308-309
#38	31.1	Test to see if the cells in the data grid viewer is clickable and to see if the all the information of the cell row is put into all the textboxes. Check to see if an anime picture of the anime that is clicked is shown UserFavourites	The cells from the data grid will be clickable and all the textboxes will be filled with information from the cell row.	As Expected	309-310
#39	31, 31.1, 31.1.1	Test to see if you can update the ratings of animes in your favourites list in the form UserFavourites	The ratings will change, and I will be prompted with a message box confirming that change	As Expected	310-311
#40	31, 31.1, 31.1.1	Test to see if you can delete an anime off your favourites lists	The anime will be deleted, and I will be prompted with a message confirming the deletion	As Expected.	311-312
#41	31, 31.1, 31.1.1	Test to see if the cells in the data grid viewer is clickable and to see if the all	The cells from the data grid will be clickable and all the textboxes	As Expected	312-313

		the information of the cell row is put into all the textboxes. Check to see if an anime picture of the anime that is clicked is shown UserWatchList	will be filled with information from the cell row.		
#42	31, 31.1, 31.1.1	Test to see if you can update the watch status, ratings, and episodes watched of your animes in your watchlist	The ratings, watch status, episodes watched will change and I will be prompted with a message box confirming those changes	As Expected.	313-314
#43	31, 31.1, 31.1.1	Test to see if you can delete an anime off your watch list	The anime will be deleted, and I will be prompted with a message confirming the deletion	As Expected.	314-315
#44		Test to see if the search function works in UserFavourites and UserWatchlist	The search bar will show me a list of animes in favourites in the user favourites form and list of animes in the user watchlist form the search bar will refresh the data grid in both forms via input	As Expected.	315-317
#45	35	Test to see if the form FmUserStats will appear by clicking the stats button in user favourites	The form will appear with stats telling me my average rating, my total ratings, the amount of animes I am currently watching, the amount of animes I have completed, the amount of animes I have dropped, the amount of animes I have yet to watch, the	As Expected.	317-318

			total number of episodes I have watched and the amount of hours I have spent watching		
#46	36	Test to see if darkmode works	The form FmAMS will change to the dark mode theme through the click of the 'switch' button	As Expected	318-319

Test runs

Test #1



All Access Obj... ⊗ «

Search...

Tables

- AnimeGenres
- Animes
- AnimeStatus
- AnimeStudios
- Genres
- Studios
- UserFavourites
- Users
- UserWatchList

Test #2

All Access Obj...  «

Tables 

Tables

- AnimeGenres**
- Animes
- AnimeStatus
- AnimeStudios
- Genres
- Studios
- UserFavourites
- Users
- UserWatchList

AnimeGenres

AnimeID	GenreID	Click to Add
1	3279	
5	3684	
6	2250	
7	3495	
8	3011	
15	2418	
16	702	
17	551	
18	255	
19	2904	
20	17	
21	32	
22	200	
23	1396	
24	557	
25	1339	
26	4058	
27	428	
28	169	
29	3825	
30	2082	

Record: 14  1 of 13102    

All Access Obj...  «

Tables 

Tables

- AnimeGenres**
- Animes**
- AnimeStatus
- AnimeStudios
- Genres
- Studios
- UserFavourites
- Users
- UserWatchList

Animes

AnimeID	AnimeTitle	Episodes	Ratings	AnimeSynopsis	AnimePicture	Click to Add
1	Cowboy Bebop	26	9 8140	8227		
5	Cowboy Bebop	1	8 9692	9795		
6	Trigun	26	8 4823	4874		
7	Witch Hunter F	26	7 8953	9052		
8	Beet the Vand	52	7 8298	8386		
15	EYESHIELD 21	145	8 5255	5309		
16	Hachimitsu to	24	8 1030	1039		
17	Hungry Heart:	52	8 792	801		
18	Initial D Fourth	24	8 310	314		
19	Monster	74	9 6802	6872		
20	Naruto	220	8 4017	4064		
21	One Piece	0	9 35	35		
22	Tennis no Ouji	178	8 234	237		
23	Ring ni Kakeru	12	7 8678	8770		
24	School Rumble	26	8 833	841		
25	Sunabouzu	24	8 2348	2373		
26	Texhnolyze	22	8 11176	11299		
27	Trinity Blood	24	7 9872	9977		
28	Yakitate!! Japa	69	8 2565	2590		
29	Zipang	26	8 10289	10401		
30	Neon Genesis	26	8 4366	4411		

Record: 14  1 of 13105    

All Access Obj... ⊞ «

Search...

Tables

- AnimeGenres
- Animes
- AnimeStatus**
- AnimeStudios
- Genres
- Studios
- UserFavourites
- Users
- UserWatchList

AnimeStatus

AnimID	Status	Click to Add
1	Finished Airing	
5	Finished Airing	
6	Finished Airing	
7	Finished Airing	
8	Finished Airing	
15	Finished Airing	
16	Finished Airing	
17	Finished Airing	
18	Finished Airing	
19	Finished Airing	
20	Finished Airing	
21	Currently Airing	
22	Finished Airing	
23	Finished Airing	
24	Finished Airing	
25	Finished Airing	
26	Finished Airing	
27	Finished Airing	
28	Finished Airing	
29	Finished Airing	
30	Finished Airing	

Record: 1 of 13105 No Filter

All Access Obj... ⊞ «

Search...

Tables

- AnimeGenres
- Animes
- AnimeStatus
- AnimeStudios**
- Genres
- Studios
- UserFavourites
- Users
- UserWatchList

AnimeStudios

AnimID	StudioID	Click to Add
1	22	
5	8	
6	12	
7	22	
8	20	
15	143	
16	5	
17	39	
18	97	
19	12	
20	11	
21	20	
22	83	
23	20	
24	174	
25	2	
26	12	
27	2	
28	22	
29	9	
30	437	

Record: 1 of 13105 No Filter

All Access Obj... □ «

Search... 

Tables

- AnimeGenres
- Animes
- AnimeStatus
- AnimeStudios
- Genres **Selected**
- Studios
- UserFavourites
- Users
- UserWatchList

Genres

GenreID	Genre	Click to Add
1	Comedy & Superhero	
2	Comedy & Parody	
3	Comedy & Magic	
4	Comedy & Drama	
5	Comedy & Drama	
6	Kids & School	
7	Magic & Comedy	
8	Action & Drama	
9	Music & Slice of Life	
10	Comedy & Horror	
11	Comedy & Drama	
12	Comedy & Romance	
13	Slice of Life & Romance	
14	Action & Adventure	
15	Sci-Fi & Comedy	
16	Slice of Life & Romance	
17	Action & Adventure	
18	Slice of Life & Romance	
19	Slice of Life & Romance	
20	Comedy & Science Fiction	
21	Comedy & Drama	

Record: 14 | 1 of 4372 | ► | ► | ► | ► | No Filter | Search

All Access Obj... □ «

Search... 

Tables

- AnimeGenres
- Animes
- AnimeStatus
- AnimeStudios
- Genres
- Studios **Selected**
- UserFavourites
- Users
- UserWatchList

Studios

StudioID	Studio	Click to Add
1	David Production	
2	Gonzo	
3	Satelight	
4	Hal Film Maker	
5	J.C.Staff	
6	Studio Pierrot	
7	Production Reel	
8	Bones	
9	Studio Deen	
10	Brain's P	
11	Studio Pierrot	
12	Madhouse	
13	Production I.G	
14	Group TAC	
15	TMS Entertainment	
16	Pierrot Plus	
17	Tatsunoko Pro	
18	Hal Film Maker	
19	Shin-Ei Animation	
20	Toei Animation	
21	Tokyo Movie Studio	

Record: 14 | 1 of 729 | ► | ► | ► | ► | No Filter | Search

All Access Obj... ⊞ «

Search...

Tables

- AnimeGenres
- Animes
- AnimeStatus
- AnimeStudios
- Genres
- Studios
- UserFavourites
- Users
- UserWatchList

UserFavourites

UserFavID	UserID	AnimeID	AnimeTitle	Ratings	Click to Add
*	(New)				

Record: 1 of 1 No Filter Search

All Access Obj... ⊞ «

Search...

Tables

- AnimeGenres
- Animes
- AnimeStatus
- AnimeStudios
- Genres
- Studios
- UserFavourites
- Users
- UserWatchList

Users

UserID	Username	Pword	UserType	Forename	Surname	Email	Click to Add
*	(New)						

Record: 1 of 1 No Filter Search

All Access Obj...

Search...

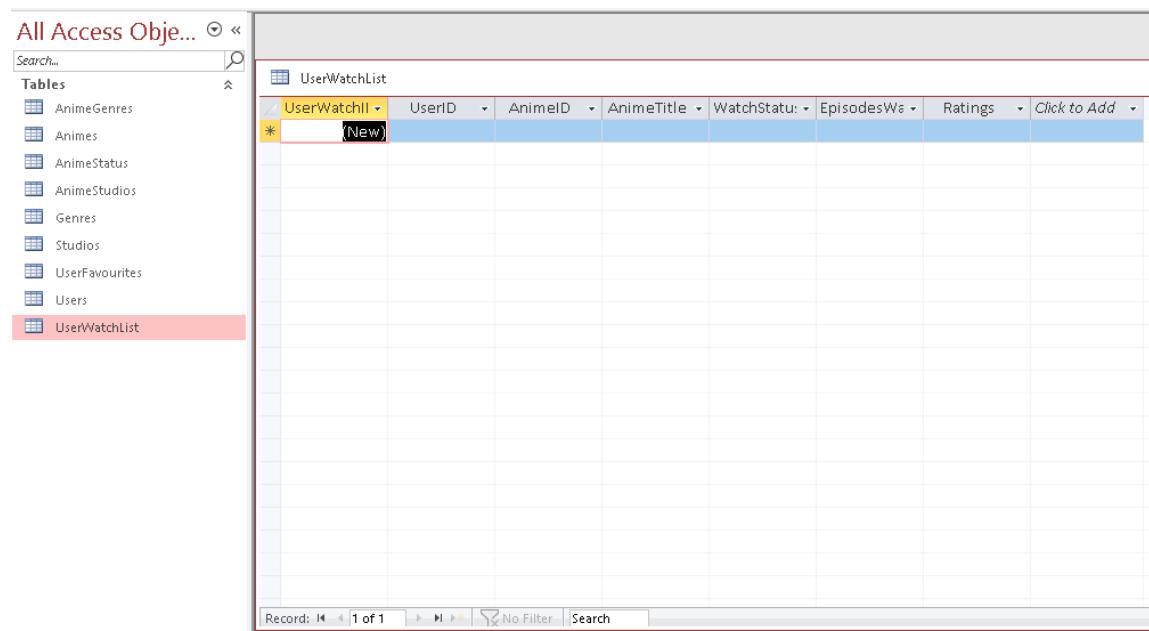
Tables

- AnimeGenres
- Animes
- AnimeStatus
- AnimeStudios
- Genres
- Studios
- UserFavourites
- Users
- UserWatchList

UserWatchList

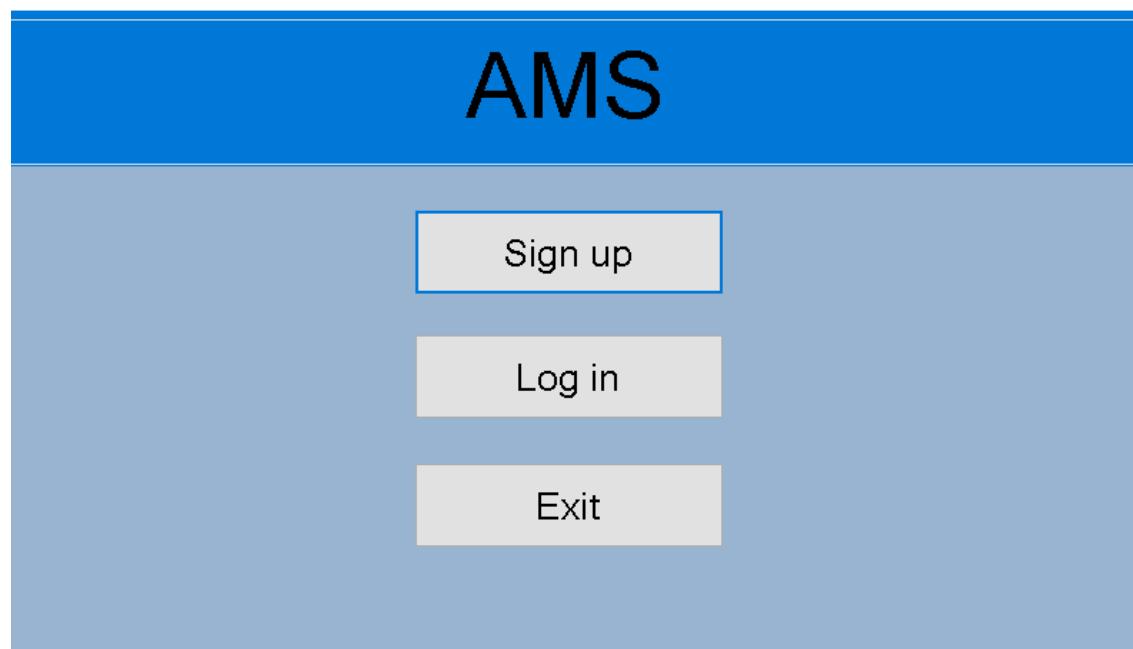
UserWatchList	UserID	AnimeID	AnimeTitle	WatchStatus	EpisodesWatched	Ratings	Click to Add
*	(New)						

Record: 1 of 1



Test #3

FmMenu



FrmSignUp

AMS

Forename	<input type="text"/>
Surname	<input type="text"/>
UserType	<input type="text"/>
Email	<input type="text"/>
Username	<input type="text"/>
Password	<input type="text"/>

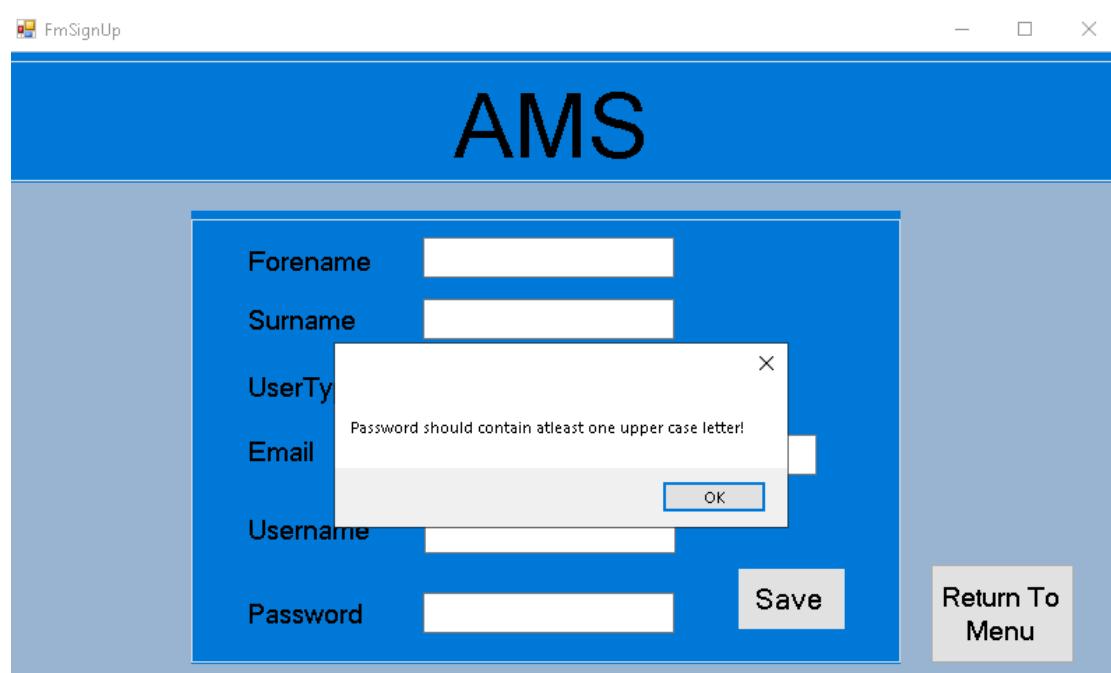
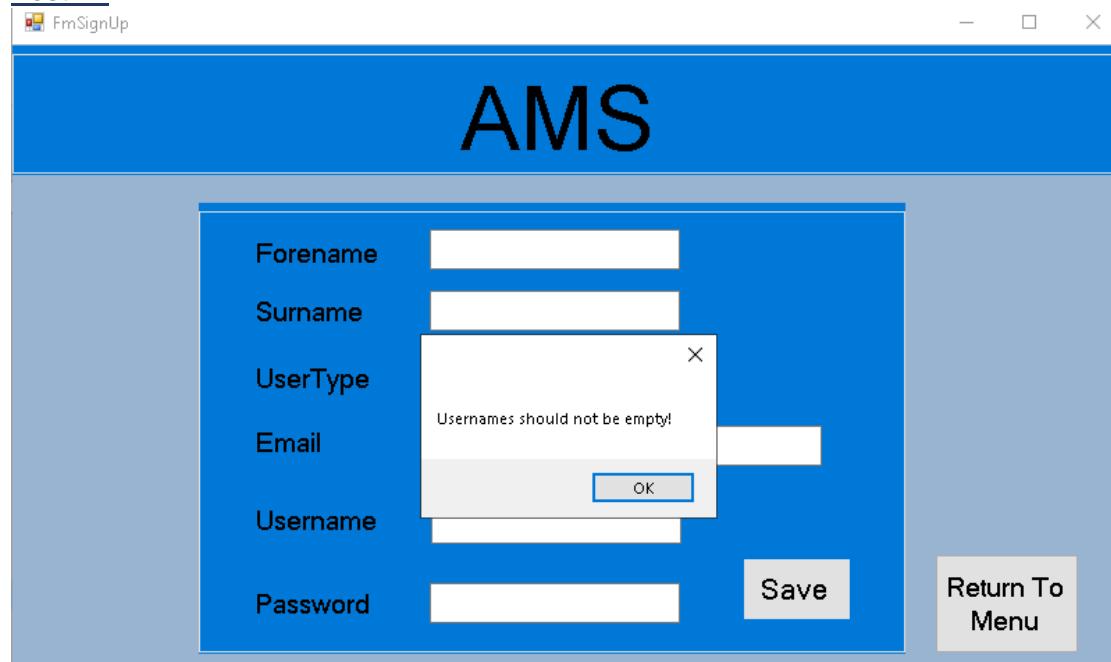
FrmLogin

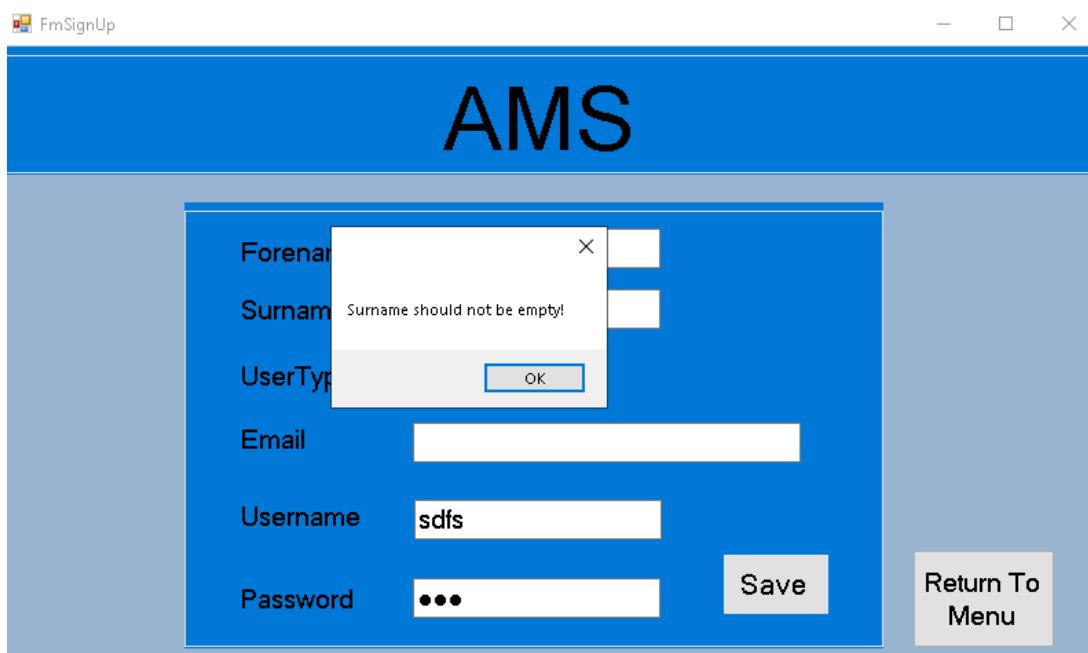
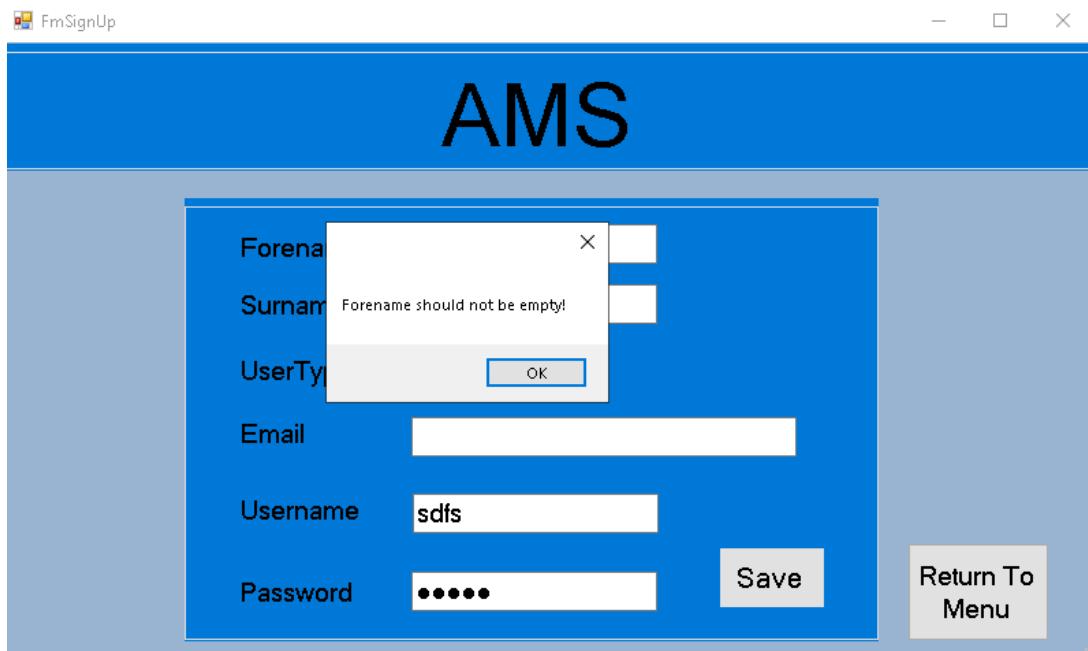
AMS

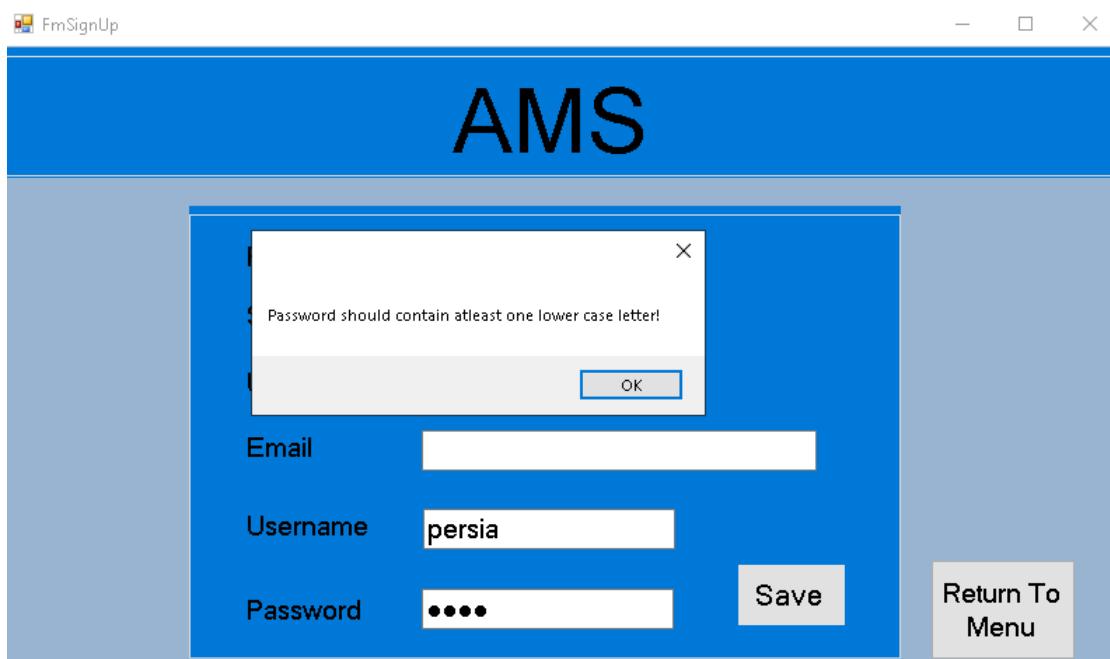
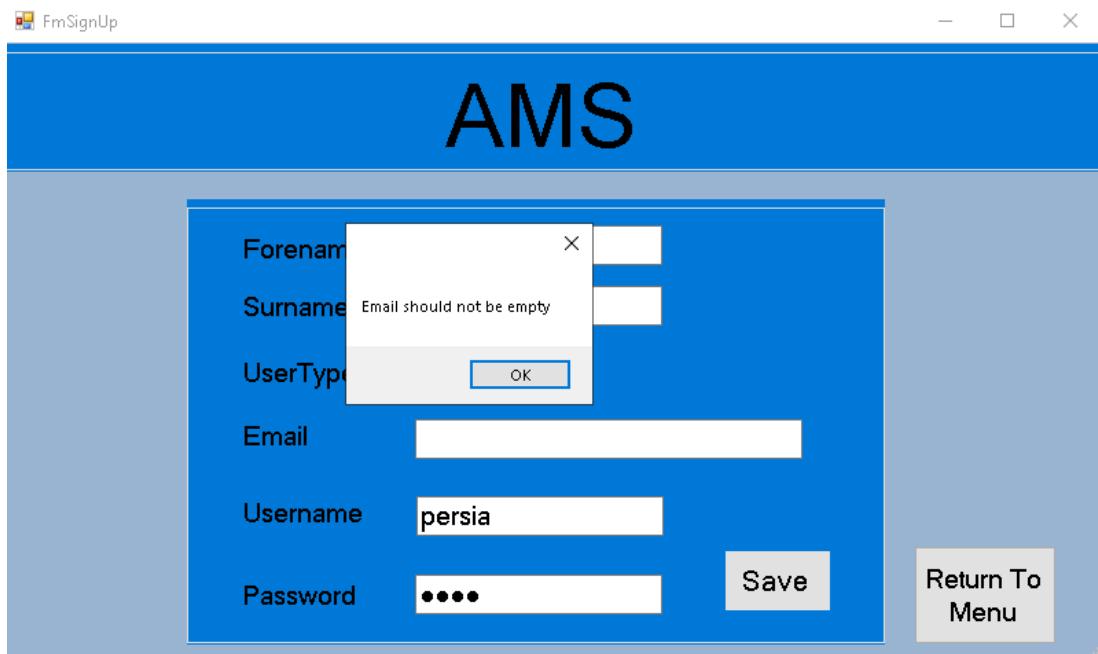
Username	<input type="text"/>
Password	<input type="text"/>

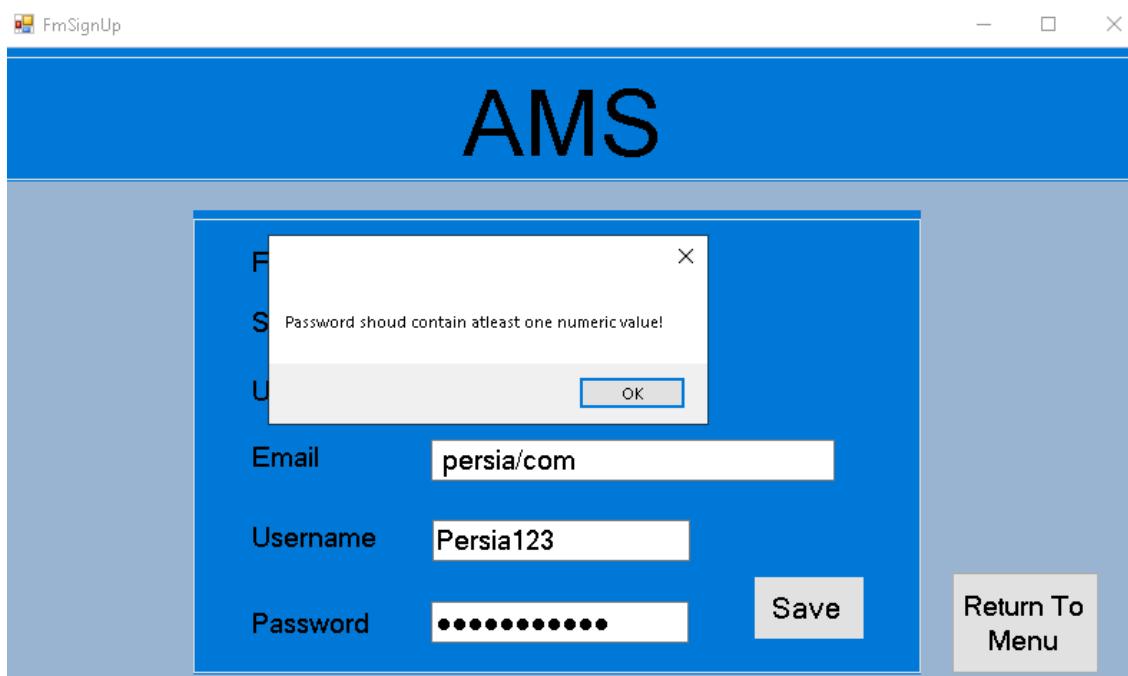
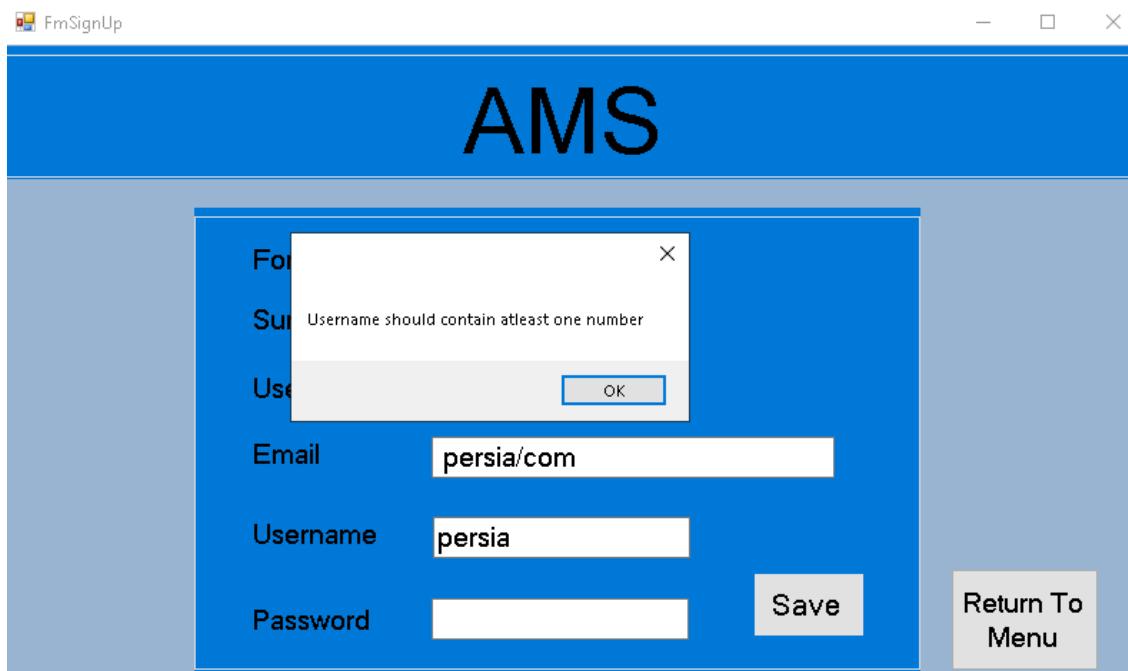
Remember me?

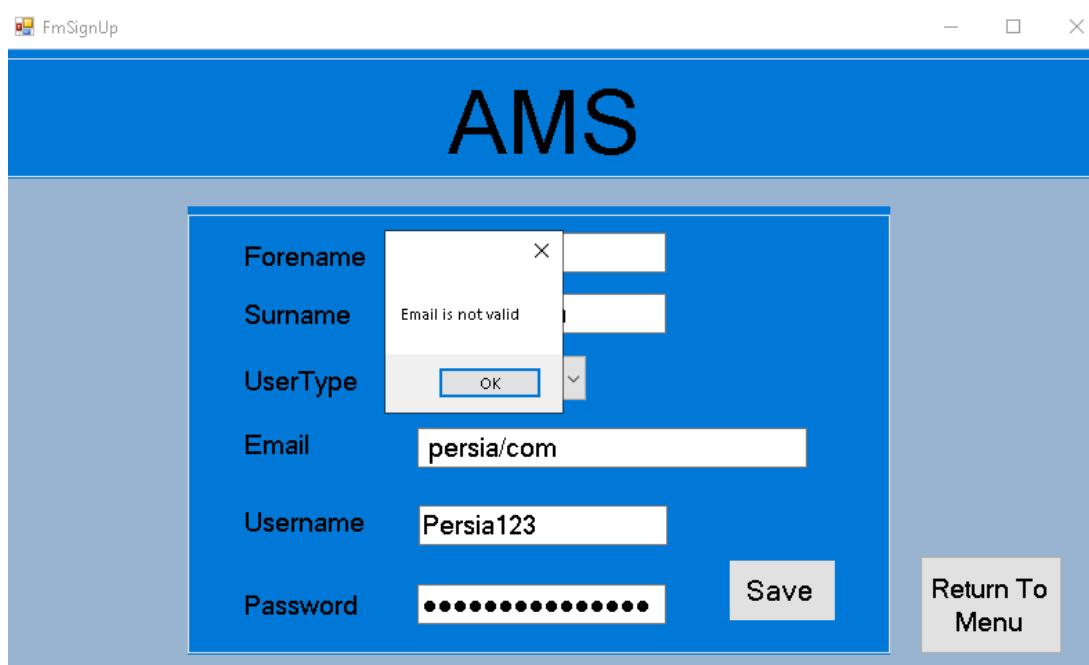
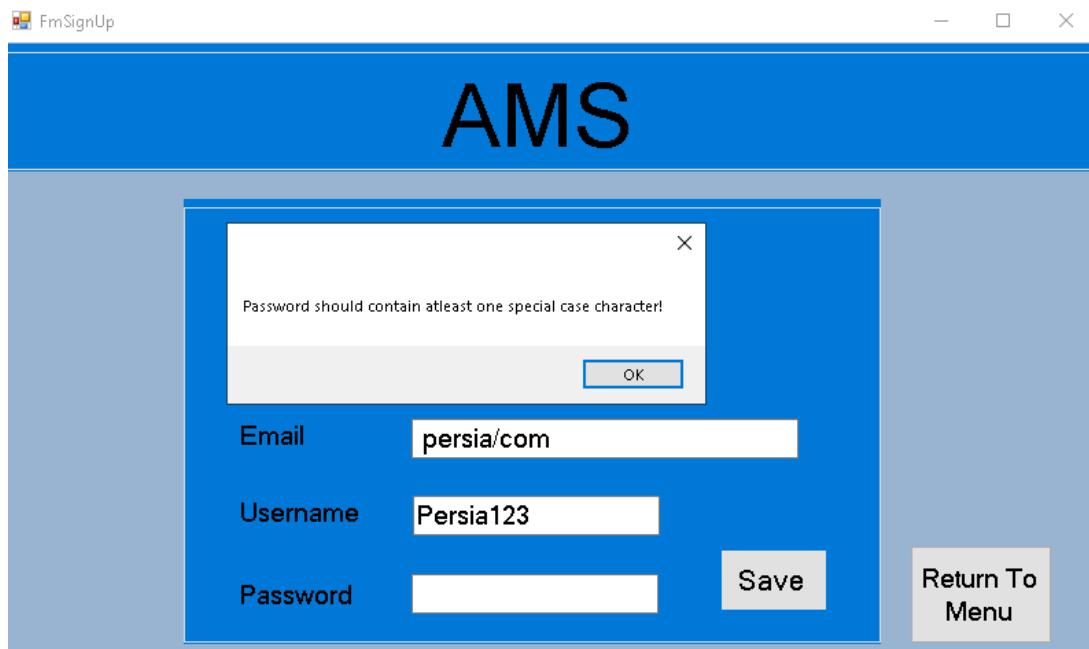
Test #4

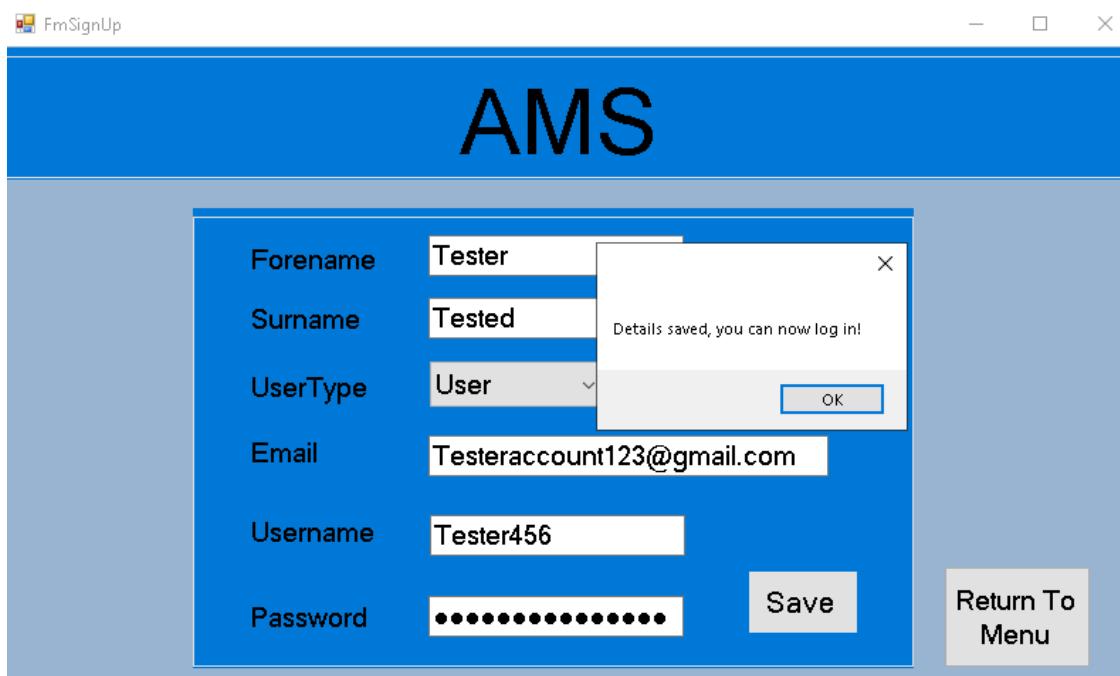
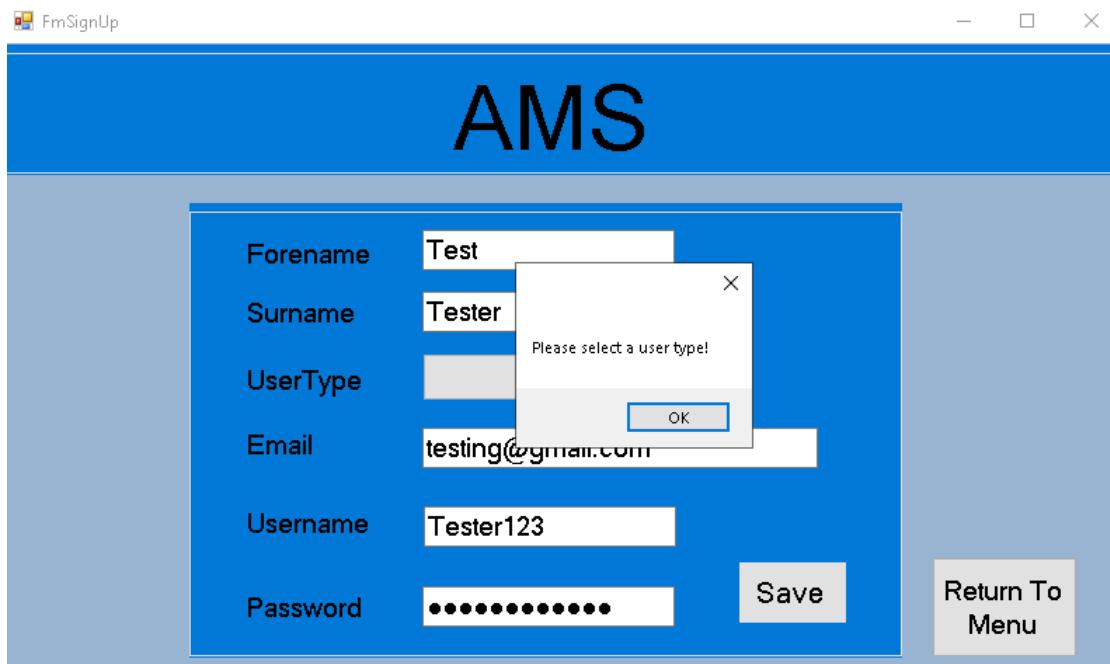












Test #5

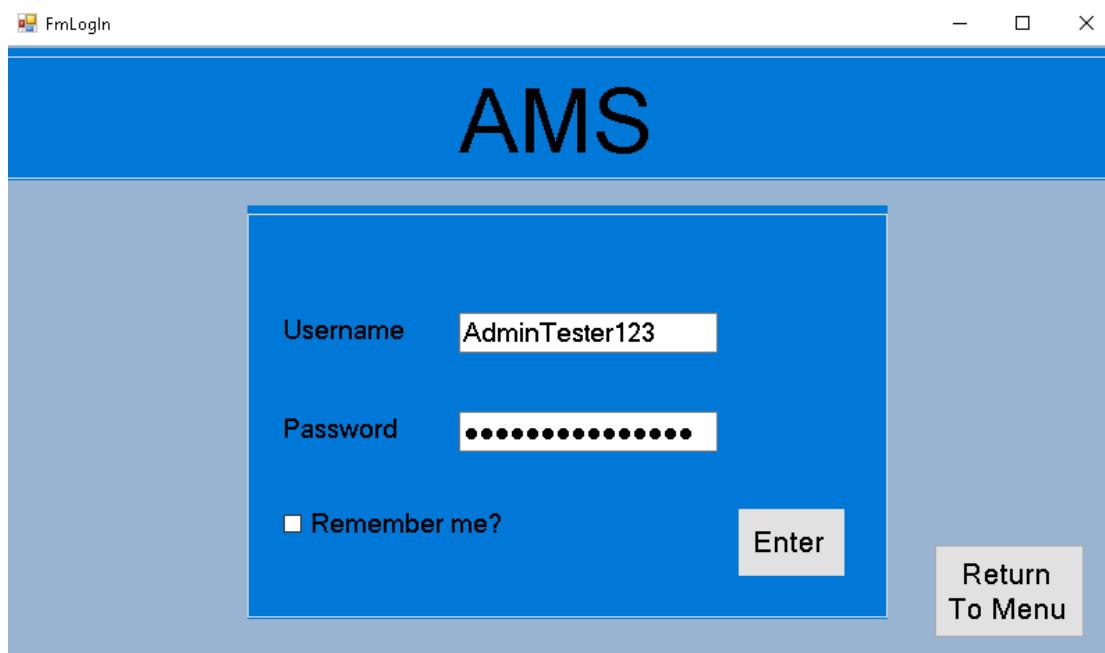
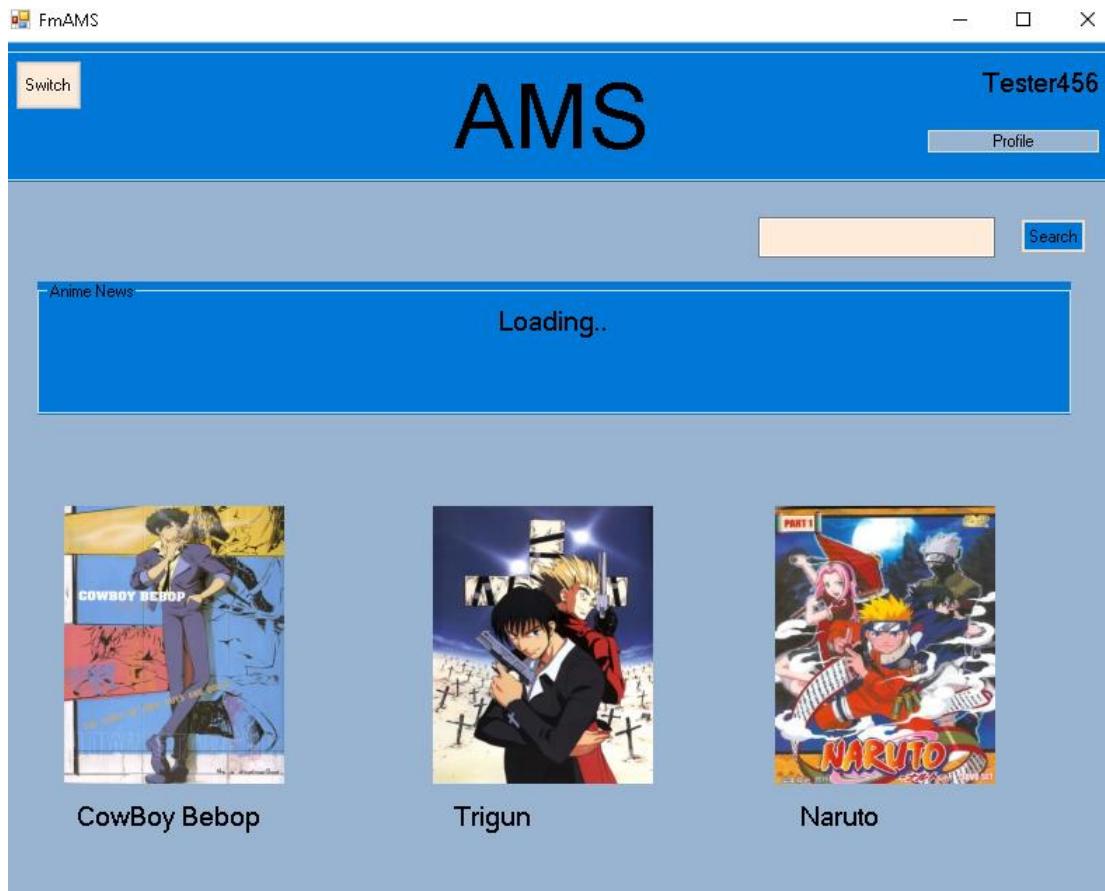
FmLogin

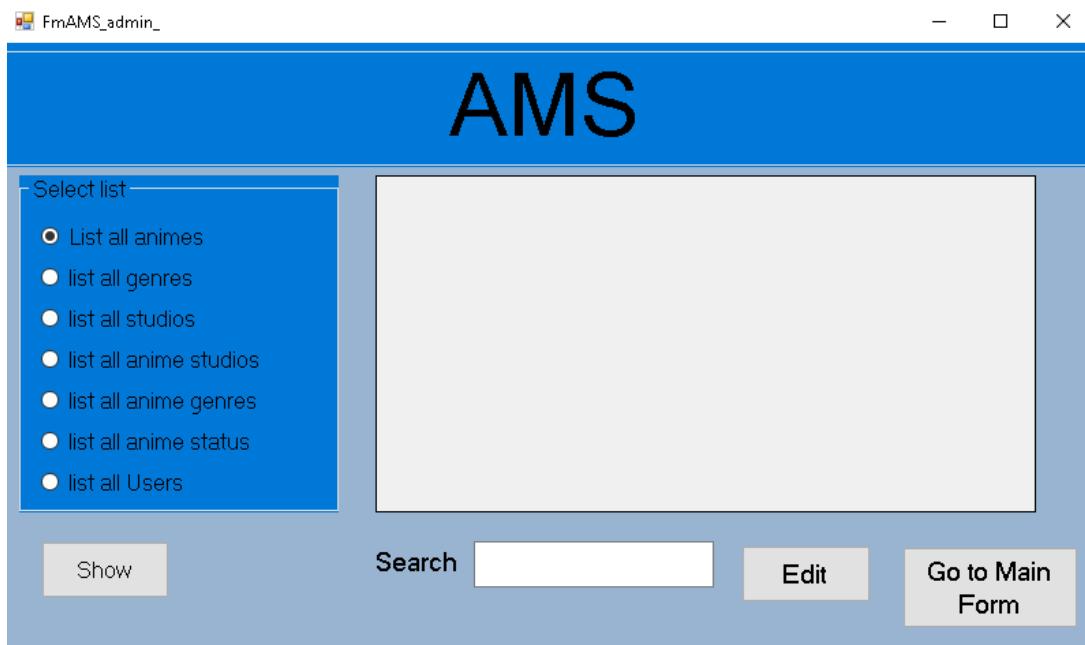
AMS

Username

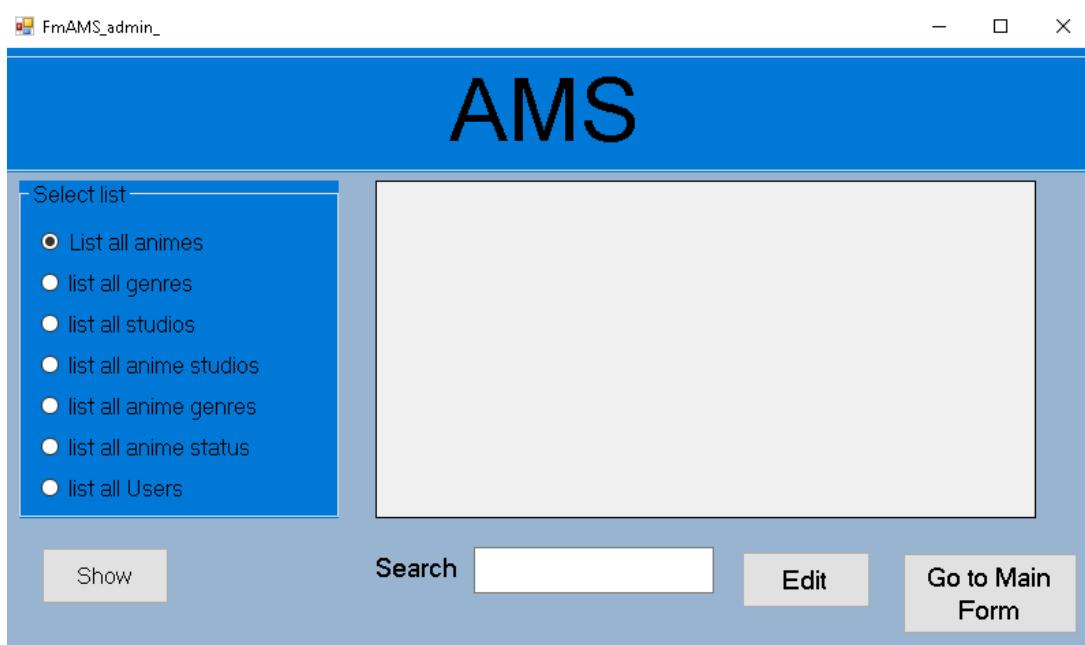
Password

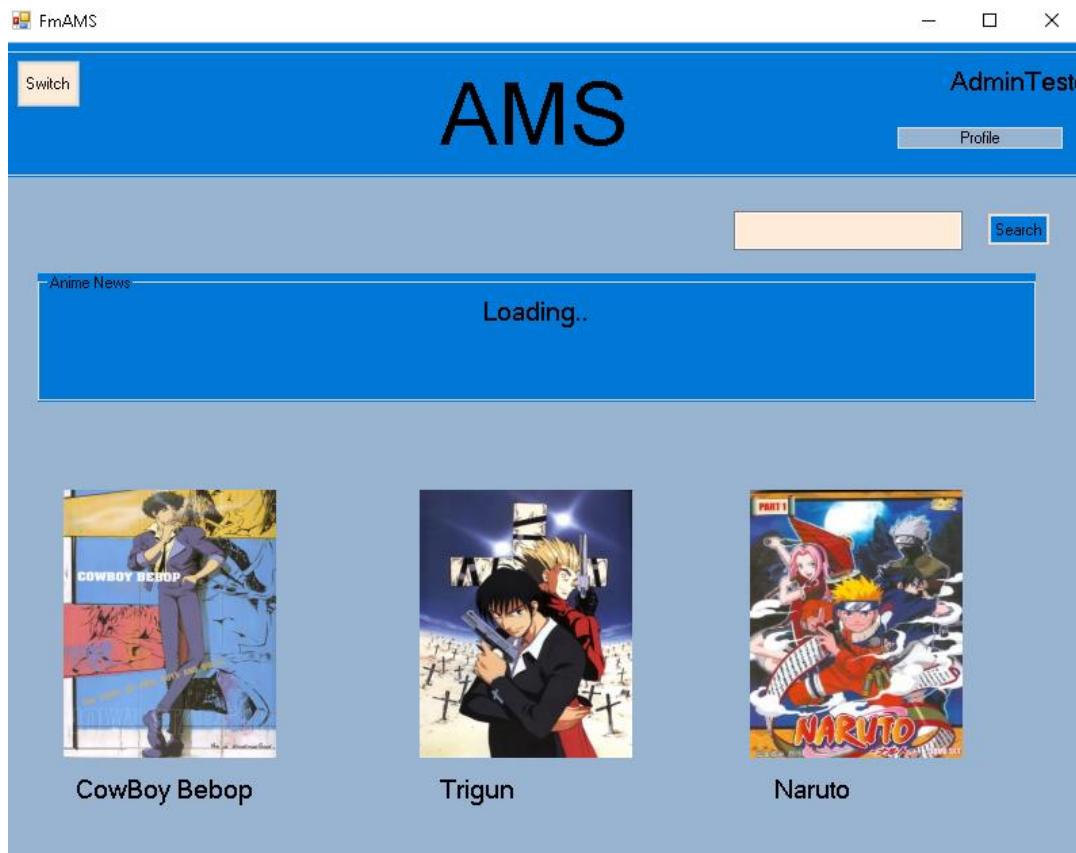
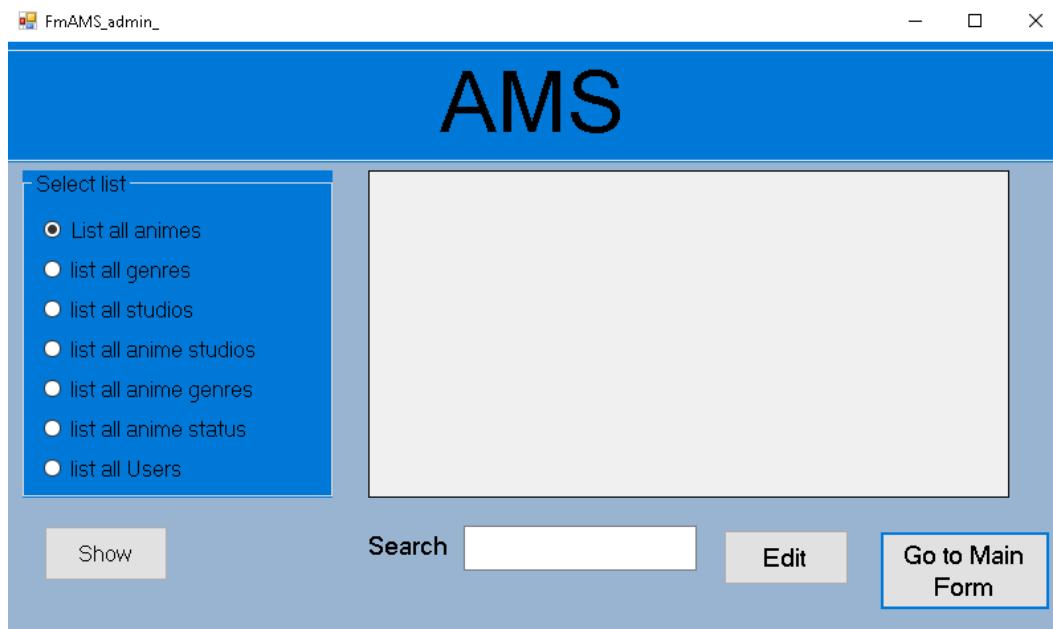
Remember me?





Test #6



Test #7

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

AnimelD	AnimeTitle	Episodes	Ratings	Anil
11013	Inu x Boku SS	12	8	1
2104	Seto no Hanayome	26	8	2
5262	Shugo Chara! D...	51	8	3
721	Princess Tutu	38	8	4
12365	Bakuman. 3rd Se...	25	9	5
6586	Yume-iro Pâtissière	50	8	6
178	Ultra Maniac	26	7	7
2787	Shakugan no Sh...	24	8	8
4477	Nodame Cantabil...	11	8	9
...

Show Search Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

GenreID	Genre
1	Comedy & Super...
2	Comedy & Parody...
3	Comedy & Magic ...
4	Comedy & Drama...
5	Comedy & Drama...
6	Kids & School & ...
7	Magic & Comedy ...
8	Action & Drama &...
9	Music & Slice of ...
10	Comedy & Harem...

Show Search Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

StudioID	Studio
1	David Production
2	Gonzo
3	Sateight
4	Hal Film Maker
5	J.C.Staff
6	Studio Pierrot & S...
7	Production Reed
8	Bones
9	Studio Deen
10	Brain's Base

Show Search Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

StudioID	AnimeID
1	11013
2	2104
3	5262
4	721
5	12365
6	6586
7	178
5	2787
5	4477
8	853

Show Search Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

AnimelD	GenreID
11013	1
2104	2
5262	3
721	4
12365	5
6586	6
178	7
2787	8
4477	9
853	10

Show Search Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

AnimelD	Status
11013	Finished Airing
2104	Finished Airing
5262	Finished Airing
721	Finished Airing
12365	Finished Airing
6586	Finished Airing
178	Finished Airing
2787	Finished Airing
4477	Finished Airing
853	Finished Airing

Show Search Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

	UserID	Username
▶	2	Tester456
	3	AdminTester123
*		

Show Search Edit Go to Main Form

Test #8

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

	AnimelD	AnimeTitle	Episodes	Ratings	Anil
▶	11013	Inu x Boku SS	12	8	1
	2104	Seto no Hanayome	26	8	2
	5262	Shugo Chara! D...	51	8	3
	721	Princess Tutu	38	8	4
	12365	Bakuman. 3rd Se...	25	9	5
	6586	Yume-iro Pâtisserie	50	8	6
	178	Ultra Maniac	26	7	7
	2787	Shakugan no Sh...	24	8	8
	4477	Nodame Cantabil...	11	8	9

Show Search Edit Go to Main Form

FmAMS_admin_

FmEditAnimes

AnimelD	<input type="text"/>
AnimeTitle	<input type="text"/>
Episodes	<input type="text"/>
Ratings	<input type="text"/>
AnimeSynopsis	<input type="text"/>
AnimePictures	<input type="text"/>

Codes Ratings Ani

1	8	1
2	8	2
3	8	3
4	8	4
5	9	5
6	8	6
7	7	7
8	8	8
9	8	9
10	8	10

Show Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

Show Search Edit Go to Main Form

GenreID	Genre
1	Comedy & Super...
2	Comedy & Parody...
3	Comedy & Magic ...
4	Comedy & Drama...
5	Comedy & Drama...
6	Kids & School & ...
7	Magic & Comedy ...
8	Action & Drama & ...
9	Music & Slice of ...
10	Comedy & Harem...

FmAMS_admin_

FmEditGenres

GenreID

Genre

Clear **Insert** Update Delete

Show Search Edit Go to Main Form

GenreID: 1

Genre: Gonzo

GenreID: 2

Genre: Sateight

GenreID: 3

Genre: Hal Film Maker

GenreID: 4

Genre: J.C. Staff

GenreID: 5

Genre: Studio Pierrot & S...

GenreID: 6

Genre: Production Reed

GenreID: 7

Genre: Bones

GenreID: 8

Genre: Studio Deen

GenreID: 9

Genre: Brain's Base

GenreID: 10

Genre: Studio Deen

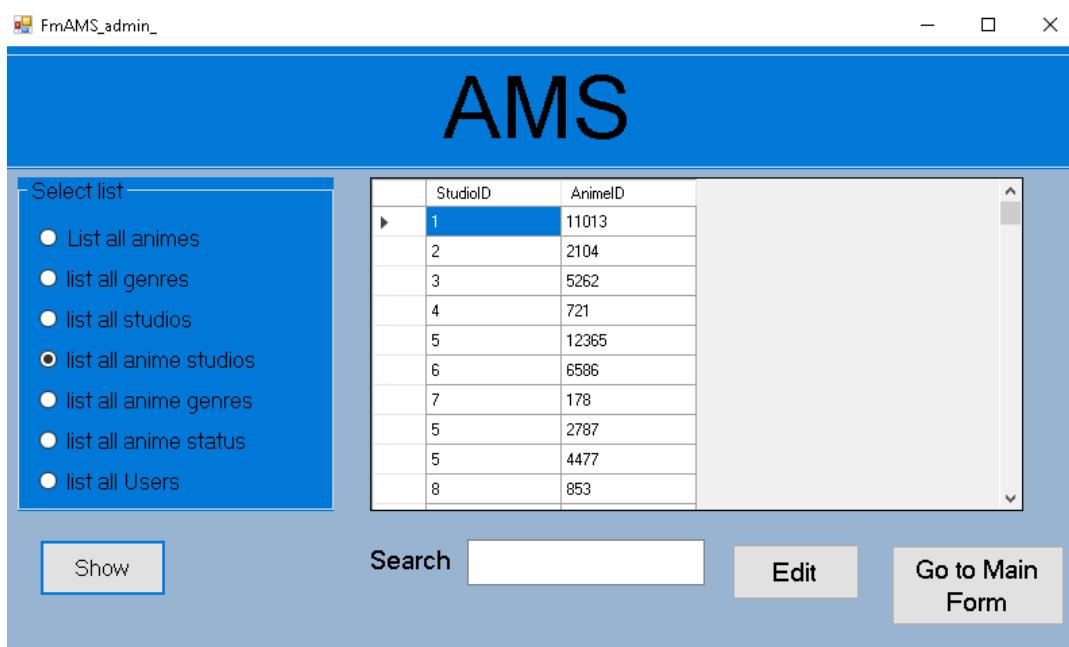
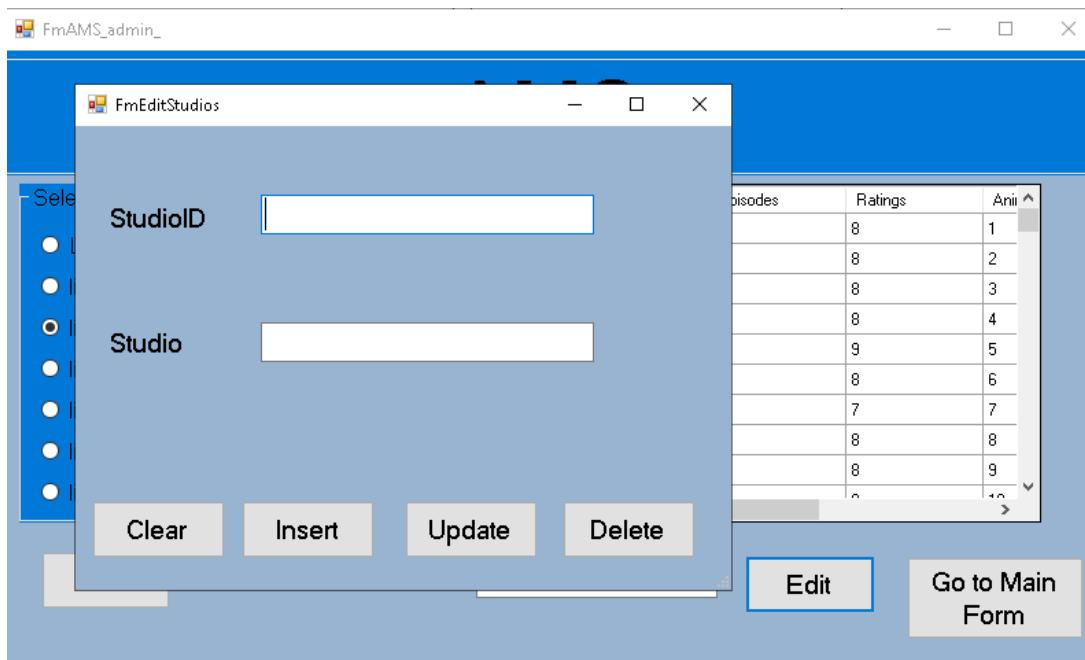
FmAMS_admin_

AMS

StudioID Studio

StudioID	Studio
1	David Production
2	Gonzo
3	Sateight
4	Hal Film Maker
5	J.C. Staff
6	Studio Pierrot & S...
7	Production Reed
8	Bones
9	Studio Deen
10	Brain's Base

Show Search Edit Go to Main Form



FmAMS_admin_

AMS

FmEditAnimeStudios

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

AnimelD

StudioID

FmAMS_admin_

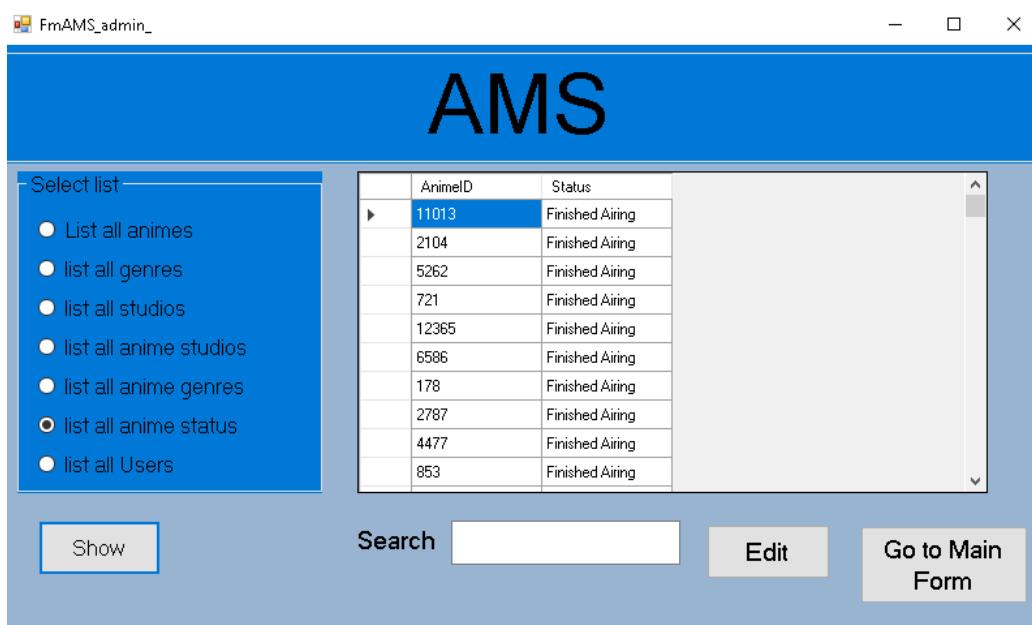
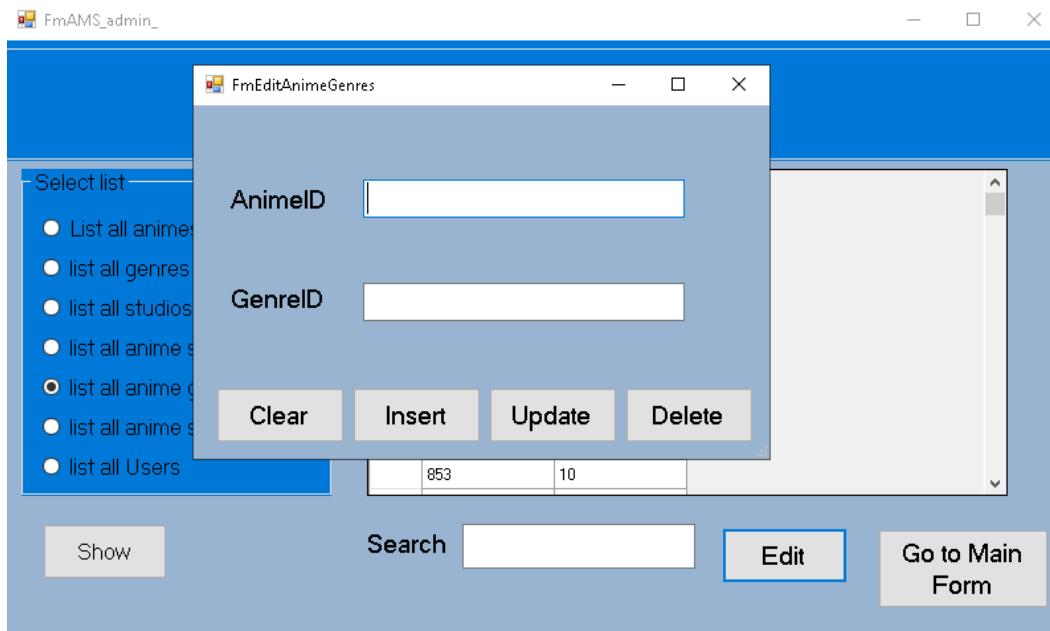
AMS

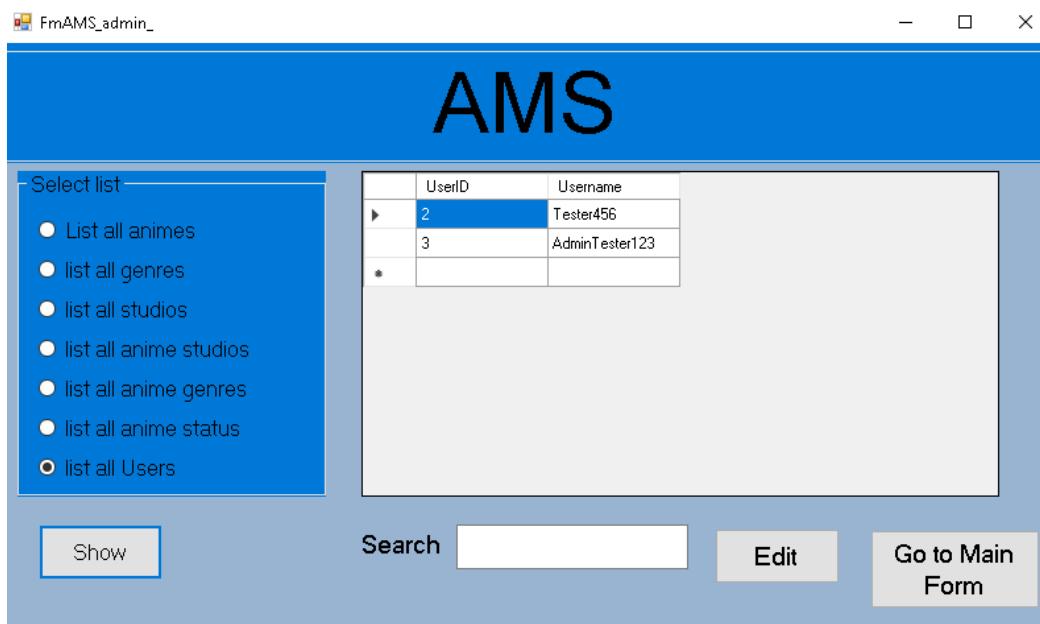
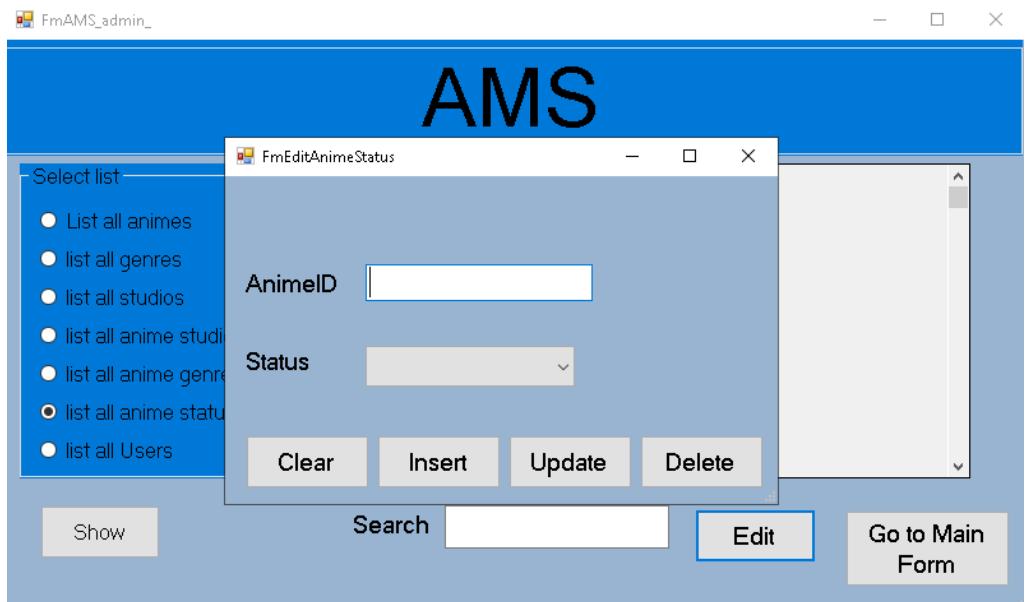
FmEditAnimeStudios

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

	AnimelD	GenreID
▶	11013	1
	2104	2
	5262	3
	721	4
	12365	5
	6586	6
	178	7
	2787	8
	4477	9
	853	10





FmAMS_admin_

AMS

FmDeleteUser

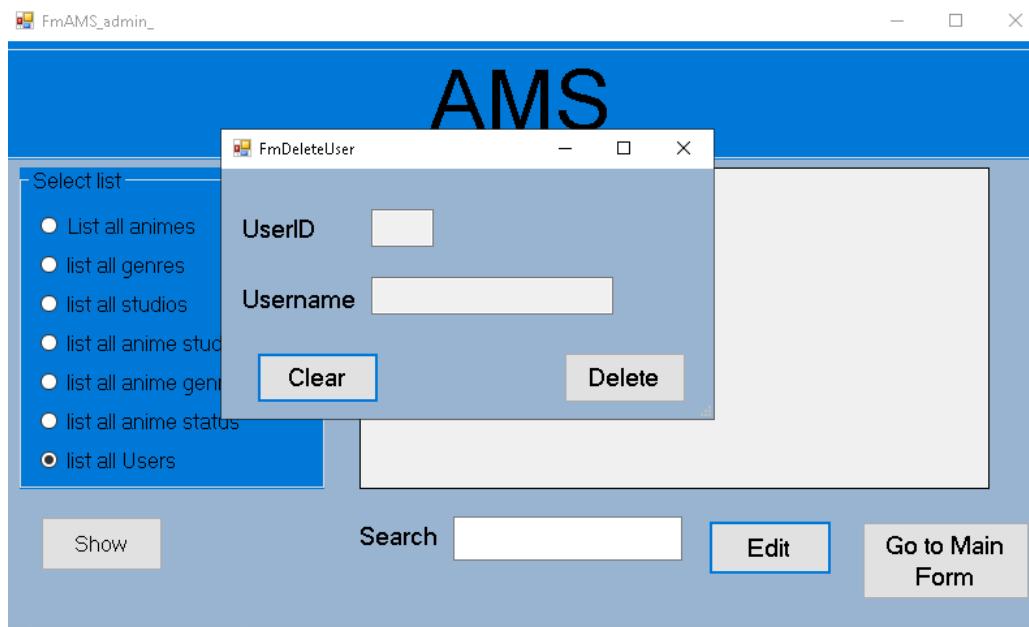
Select list

- List all animes
- list all genres
- list all studios
- list all anime stud
- list all anime gen
- list all anime status
- list all Users

UserID

Username

Show Go to Main Form



Test #9

FmAMS_admin_

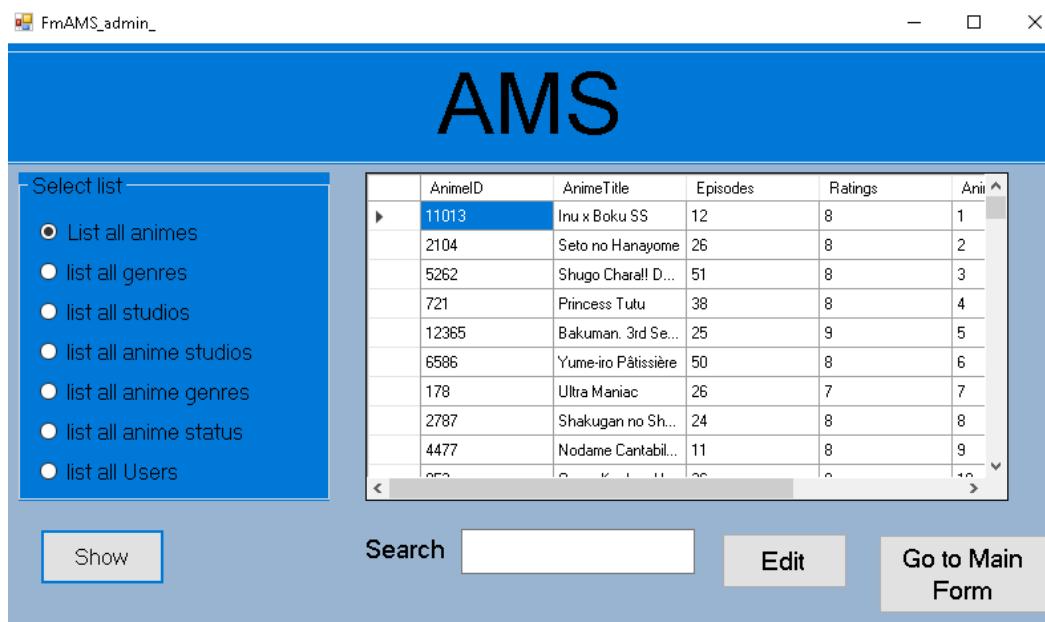
AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

AnimelD	AnimeTitle	Episodes	Ratings	Ani
11013	Inu x Boku SS	12	8	1
2104	Seto no Hanayome	26	8	2
5262	Shugo Chara! D...	51	8	3
721	Princess Tutu	38	8	4
12365	Bakuman. 3rd Se...	25	9	5
6586	Yume-iro Pâtissière	50	8	6
178	Ultra Maniac	26	7	7
2787	Shakugan no Sh...	24	8	8
4477	Nodame Cantabil...	11	8	9
252

Show Go to Main Form



FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

Show

Search Dragon Ball Z

AnimelD	AnimeTitle	Episodes	Ratings	AnimeSynop	AnimePicture
14837	Dragon Bal...	1	8	353	358
903	Dragon Bal...	1	7	1907	1928
898	Dragon Bal...	1	7	2233	2256
895	Dragon Bal...	1	7	2468	2492
905	Dragon Bal...	1	8	3997	4044
894	Dragon Bal...	1	7	4103	4147
906	Dragon Bal...	1	8	4485	4529
25389	Dragon Bal...	1	7	5304	5359
902	Dragon Bal...	1	7	6151	6212
986	Dragon Bal...	1	8	7187	7261

Edit

Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

Show

Search Movie 14:

AnimelD	AnimeTitle	Episodes	Ratings	AnimeSynops	AnimePicture
14837	Dragon Ball...	1	8	353	358
*					

Edit

Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

GenreID	Genre
1	Comedy & Super...
2	Comedy & Parody...
3	Comedy & Magic ...
4	Comedy & Drama...
5	Comedy & Drama...
6	Kids & School & ...
7	Magic & Comedy ...
8	Action & Drama & ...
9	Music & Slice of ...
10	Comedy & Harem...

Show Search Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

GenreID	Genre
324	Horror & Supernatural & Fantasy
390	Horror & School
409	Horror & Mystery & Romance & School & ...
707	Horror & Shoujo & Supernatural
959	Horror & Supernatural
1287	Horror & Mystery & Supernatural
1522	Horror & Mystery & Supernatural & Vampire
1553	Horror & Psychological & Sci-Fi
1609	Horror & Demons & Supernatural & Thriller...
1749	Horror & School & Supernatural

Show Search Horror Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

GenreID	Genre
324	Horror & Supernatural & Fantasy
959	Horror & Supernatural
3464	Horror & Supernatural & School & Seinen
3767	Horror & Supernatural & Vampire
*	

Show Edit [Go to Main Form](#)

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

StudioID	Studio
1	David Production
2	Gonzo
3	Sateight
4	Hal Film Maker
5	J.C.Staff
6	Studio Pierrot & S...
7	Production Reed
8	Bones
9	Studio Deen
10	Brain's Base

Show Edit [Go to Main Form](#)

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

StudioID	Studio
1	David Production
*	

Show Search Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

StudioID	AnimelD
1	11013
2	2104
3	5262
4	721
5	12365
6	6586
7	178
5	2787
5	4477
8	853

Show Search Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

AnimelD	StudioID
11013	1
11019	43
1101	124
11017	22
*	

Show Search Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

AnimelD	GenreID
11013	1
2104	2
5262	3
721	4
12365	5
6586	6
178	7
2787	8
4477	9
853	10

Show Search Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

AnimelD	GenreID
37643	4342
*	

Show Search 4342 | Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

AnimelD	Status
11013	Finished Airing
2104	Finished Airing
5262	Finished Airing
721	Finished Airing
12365	Finished Airing
6586	Finished Airing
178	Finished Airing
2787	Finished Airing
4477	Finished Airing
853	Finished Airing

Show Search | Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

AnimelID	Status
13103	Finished Airing
*	

Show Search 13105 Edit Go to Main Form

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

UserID	Username
2	Tester456
3	AdminTester123
*	

Show Search Edit Go to Main Form

FMAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

	UserID	Username
▶	2	Tester456
*		

Show Search Edit Go to Main Form

Test #10

FMAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

	AnimelD	AnimeTitle	Episodes	Ratings	Anil ^
▶	11013	Inu x Boku SS	12	8	1
	2104	Seto no Hanayome	26	8	2
	5262	Shugo Chara! D...	51	8	3
	721	Princess Tutu	38	8	4
	12365	Bakuman. 3rd Se...	25	9	5
	6586	Yume-iro Pâtissière	50	8	6
	178	Ultra Maniac	26	7	7
	2787	Shakugan no Sh...	24	8	8
	4477	Nodame Cantabil...	11	8	9
<	252	Orange	26	8	10
>					

Show Search Edit Go to Main Form

FmAMS_admin_

FmEditAnimes

AnimelD	2787
AnimeTitle	Shakugan no Shana II (Seco
Episodes	24
Ratings	8
AnimeSynopsis	8
AnimePictures	8

Ratings	AnimeSynopsis	AnimePicture
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
10	10	

Edit **Go to Main Form**

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

GenreID	Genre
1	Comedy & Super...
2	Comedy & Parody...
3	Comedy & Magic ...
4	Comedy & Drama...
5	Comedy & Drama...
6	Kids & School & ...
7	Magic & Comedy ...
8	Action & Drama & ...
9	Music & Slice of ...
10	Comedy & Harem...

Show **Search** **Edit** **Go to Main Form**

FmAMS_admin_

FmEditGenres

GenreID	4
Genre	Comedy & Drama & Magic & ...

Buttons: Clear, Insert (highlighted), Update, Delete, Show, Edit, Go to Main Form

Genre dropdown list:

- Comedy & Supernatural & Romance & Sh...
- Comedy & Parody & Romance & School &...
- Comedy & Magic & School & Shoujo
- Comedy & Drama & Magic & Romance & ...
- Comedy & Drama & Romance & Shounen
- Kids & School & Shoujo
- Magic & Comedy & Romance & School & ...
- Action & Drama & Fantasy & Romance & ...
- Music & Slice of Life & Comedy & Roman...
- Comedy & Harem & Romance & School & ...

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

Show

Search

StudioID	Studio
1	David Production
2	Gonzo
3	Satelight
4	Hal Film Maker
5	J.C.Staff
6	Studio Pierrot & S...
7	Production Reed
8	Bones
9	Studio Deen
10	Brain's Base

Edit

Go to Main Form

FmAMS_admin_

AMS

StudioID: 10

Studio: Brain's Base

Buttons: Clear, Insert, Update, Delete, Edit, Go to Main Form

Studio dropdown list:

- Studio
- David Production
- Gonzo
- Satelight
- Hal Film Maker
- J.C.Staff
- Studio Pierrot & Studio Hibari
- Production Reed
- Bones
- Studio Deen
- Brain's Base

FmAMS_admin_

AMS

Select list:

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

Buttons: Show, Search, Edit, Go to Main Form

StudioID	AnimelD
1	11013
2	2104
3	5262
4	721
5	12365
6	6586
7	178
5	2787
5	4477
8	853

FmAMS_admin_

AMS

FmEditAnimeStudios

AnimelD: 6586

StudioID: 6

Buttons: Clear, Insert, Update, Delete

Search Bar: Show, Search, Test

Buttons: Edit, Go to Main Form

AnimelD List:

AnimelD
11013
2104
5262
721
12365
6586
178
2787
4477
853

FmAMS_admin_

AMS

Select list

- List all animes
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

AnimelD GenrelD

AnimelD	GenrelD
11013	1
2104	2
5262	3
721	4
12365	5
6586	6
178	7
2787	8
4477	9
853	10

Buttons: Show, Search, Edit, Go to Main Form

FmAMS_admin_

AMS

Form: FmEditAnimeGenres

Select list

- List all anime
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

AnimelD:

GenreID:

GenreID

GenreID
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905

FmAMS_admin_

AMS

FmEditAnimeStatus

AnimelD

Status

Search

Select list

- List all anime
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

Status

	Status
1	nished Airing
2	nished Airing
3	nished Airing
4	nished Airing
5	nished Airing
6	nished Airing
7	nished Airing
8	nished Airing
9	nished Airing
10	nished Airing

FmAMS_admin_

AMS

Select list

- List all anime
- list all genres
- list all studios
- list all anime studios
- list all anime genres
- list all anime status
- list all Users

UserID	Username
2	Tester456
3	AdminTester123
*	

Show

FmAMS_admin_

AMS

Select list

List all

list all

list all

list all

list all

list all

list all Users

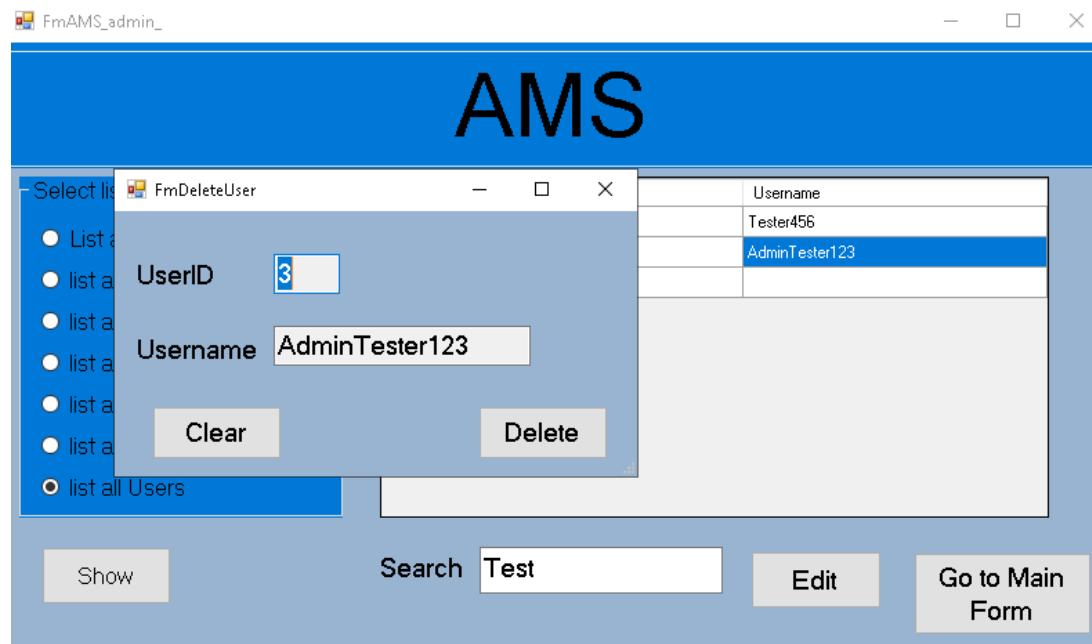
FmDeleteUser

UserID

Username

Username
Tester456
AdminTester123

Show Search Go to Main Form



Test #11

FmEditAnimes

AnimelD

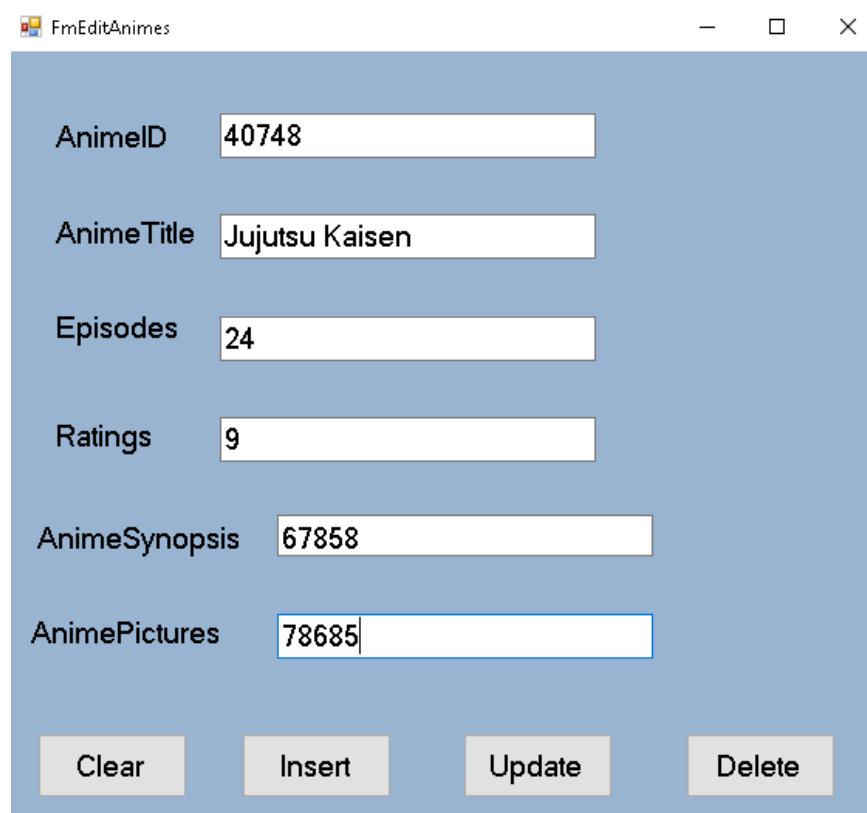
AnimeTitle

Episodes

Ratings

AnimeSynopsis

AnimePictures



FmEditAnimes

AnimelD	40748
AnimeTitle	Jujutsu Kaisen
Episodes	24
Ratings	9
AnimeSynopsis	67858
AnimePictures	78685

Insert is highlighted.

Information successfully added

OK

Clear Update Delete

FmEditAnimes

AnimelD	40748
AnimeTitle	Jujutsu Kaisen
Episodes	24
Ratings	9
AnimeSynopsis	67858
AnimePictures	78685

OK

Could not be added

Clear Insert Update Delete

Test #12

FmEditAnimes

AnimelD	40748
AnimeTitle	Jujutsu Kaisen
Episodes	24
Ratings	5
AnimeSynopsis	67858
AnimePictures	78685

Clear **Insert** **Update** **Delete**

FmEditAnimes

AnimelD	40748
AnimeTitle	Jujutsu Kaisen
Episodes	24
Ratings	5
AnimeSynopsis	678
AnimePictures	78685

Clear **Insert** **Update** **Delete**

Information updated!

OK

FmEditAnimes

AnimelD	<input type="text"/>
AnimeTitle	Jujutsu Kaisen
Episodes	24
Ratings	5
AnimeSynopsis	67858
AnimePictures	78685

Clear **Insert** **Update** **Delete**

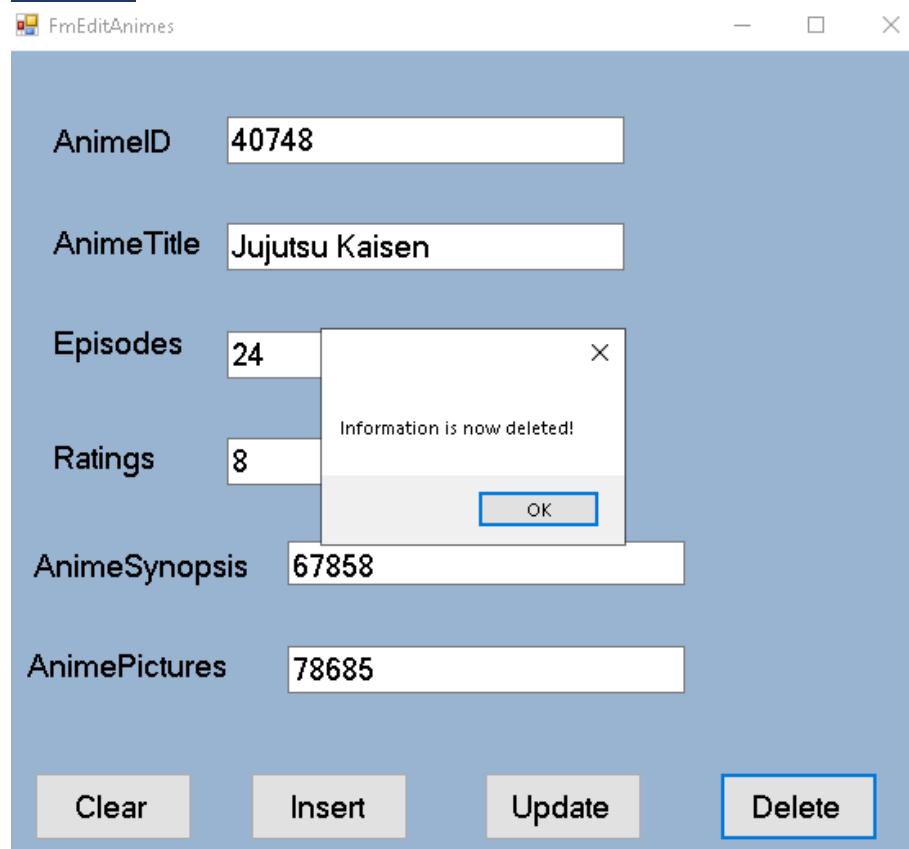
FmEditAnimes

AnimelD	<input type="text"/>
AnimeTitle	Jujutsu Kaisen
Episodes	24
Ratings	5
AnimeSynopsis	67858
AnimePictures	78685

Clear **Insert** **Update** **Delete**

Could not update!

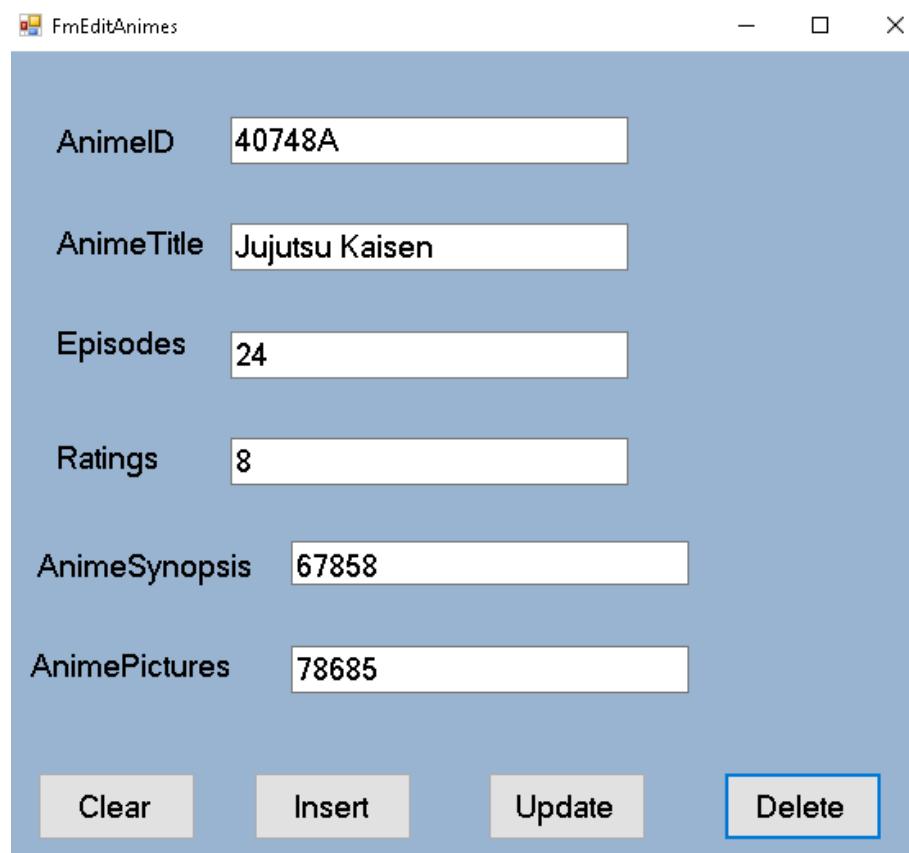
OK

Test #13

The screenshot shows a Windows application window titled "FmEditAnimes". Inside, there are six text input fields for anime data:

- AnimelD: 40748
- AnimeTitle: Jujutsu Kaisen
- Episodes: 24
- Ratings: 8
- AnimeSynopsis: 67858
- AnimePictures: 78685

A modal dialog box is displayed over the "Episodes" field, containing the message "Information is now deleted!" with an "OK" button. Below the input fields are four buttons: "Clear", "Insert", "Update", and "Delete", with "Delete" being highlighted.



The screenshot shows the same Windows application window "FmEditAnimes". The data in the input fields has been modified:

- AnimelD: 40748A
- AnimeTitle: Jujutsu Kaisen
- Episodes: 24
- Ratings: 8
- AnimeSynopsis: 67858
- AnimePictures: 78685

The "Delete" button is still highlighted. The "Episodes" field is empty, indicating the value has been deleted.

FmEditAnimes

AnimelD	40748A
AnimeTitle	Jujutsu Kaisen
Episodes	24
Ratings	8
AnimeSynopsis
AnimePictures	78685

Clear **Insert** **Update** **Delete**

Information could not be deleted!

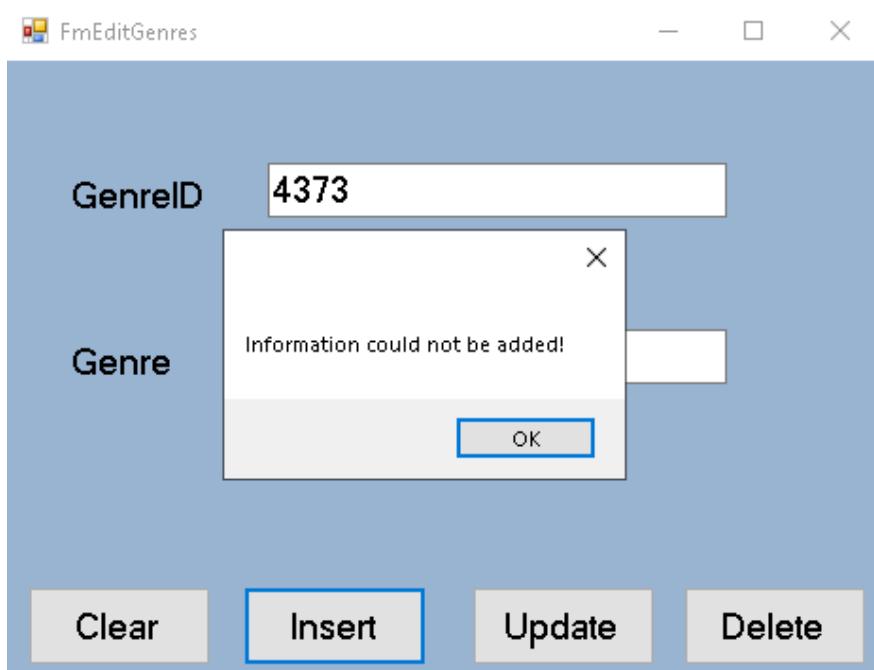
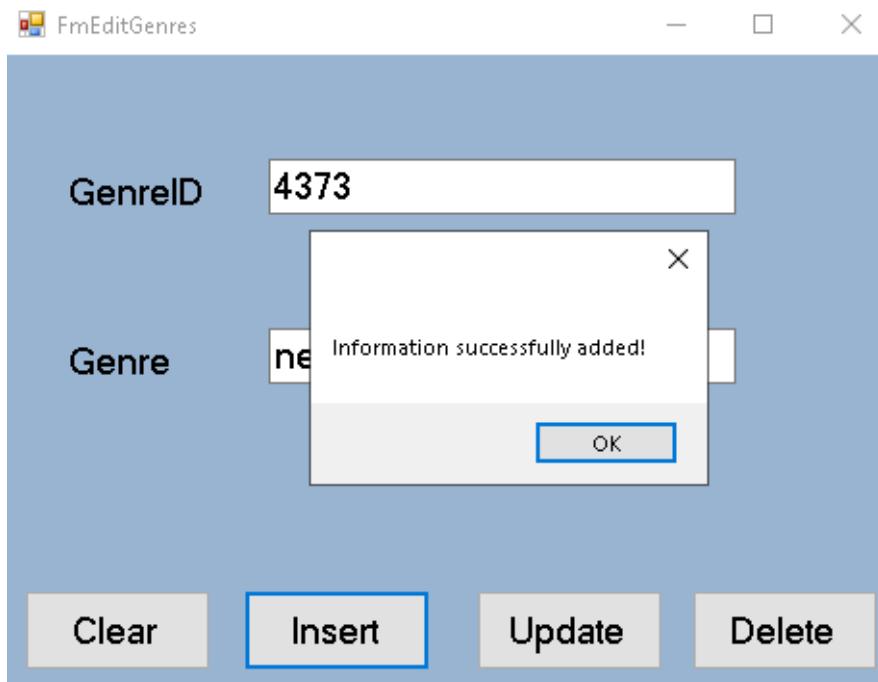
Note: To fully delete an anime you would need to delete the AnimeStudios key, AnimeGenres key and the anime status key first before deleting the anime itself

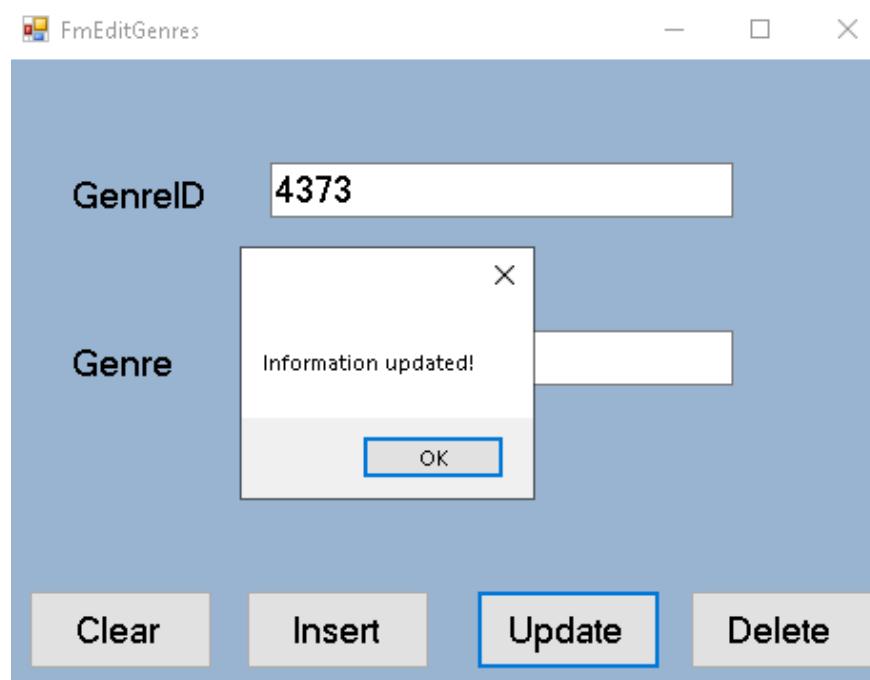
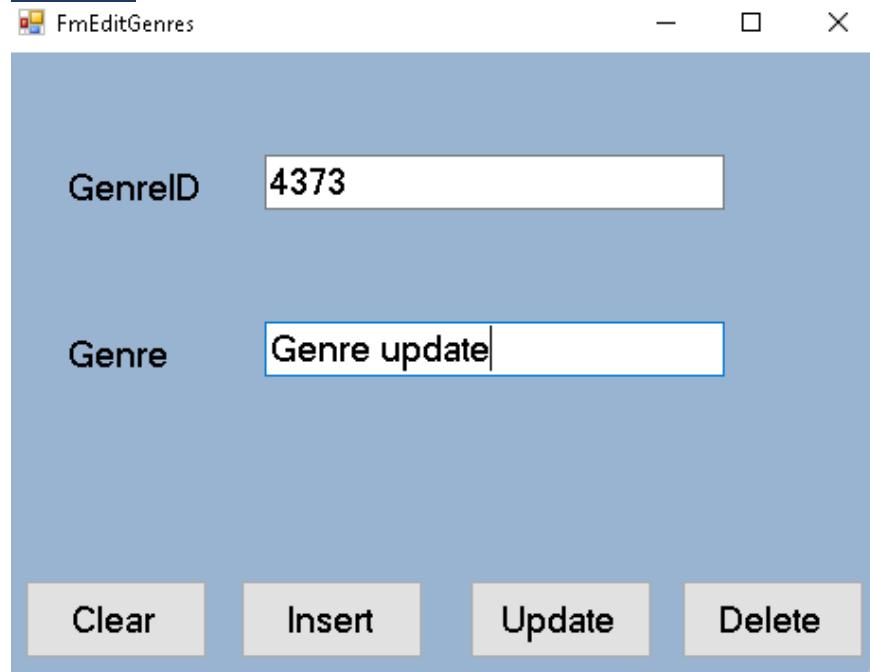
Test #14

FmEditGenres

GenreID	4373
Genre	new genre

Clear **Insert** **Update** **Delete**



Test #15

FmEditGenres

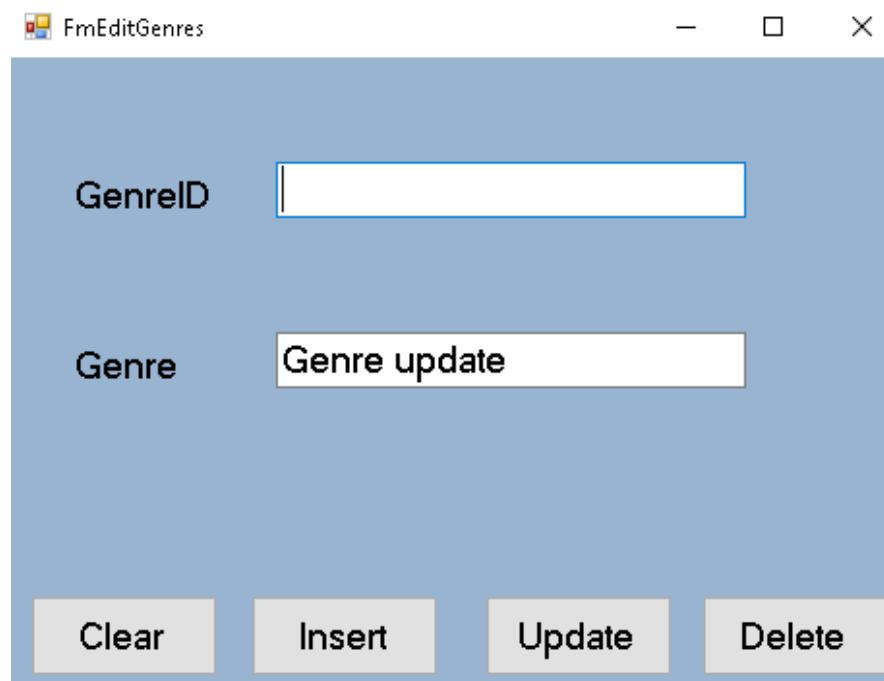
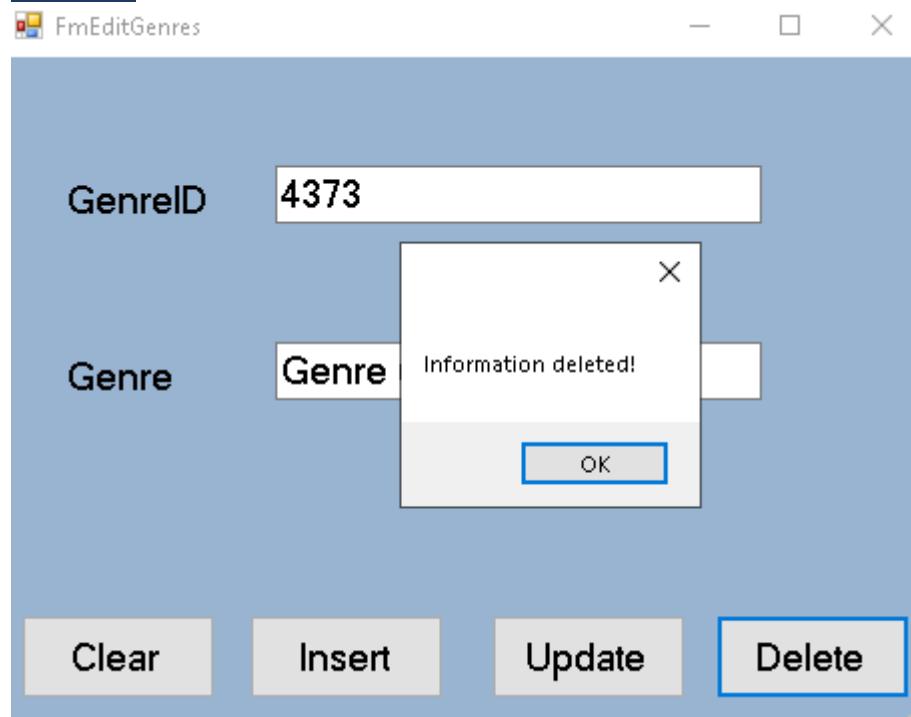
GenreID	<input type="text"/>
Genre	Genre update

Clear **Insert** **Update** **Delete**

FmEditGenres

GenreID	<input type="text"/>
Genre	<div style="border: 1px solid black; padding: 10px; width: fit-content; margin-left: auto; margin-right: auto;"><p>X</p><p>Information could not be updated!</p><p>OK</p></div>

Clear **Insert** **Update** **Delete**

Test #16

FmEditGenres

GenreID	<input type="text"/>
Genre	G <input type="text"/> Information could not be deleted!

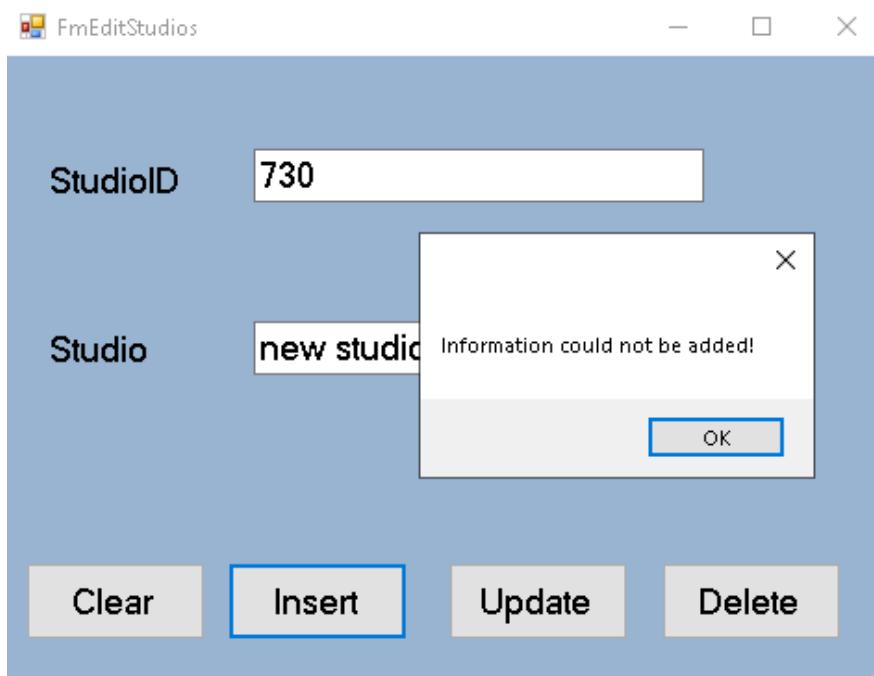
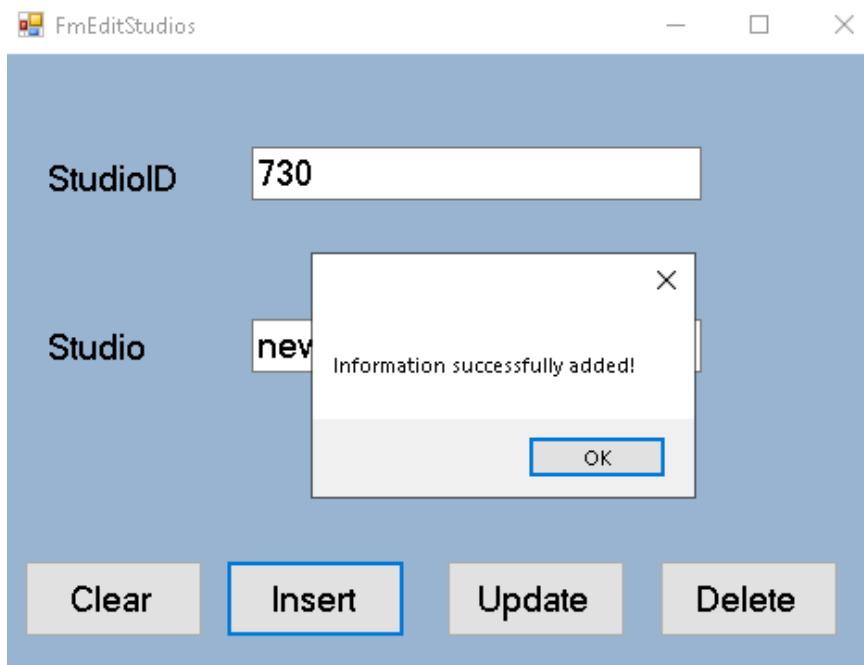
Clear **Insert** **Update** **Delete**

Test #17

FmEditStudios

StudioID	<input type="text"/> 730
Studio	<input type="text"/> new studio

Clear **Insert** **Update** **Delete**



Test #18

FmEditStudios

StudioID	730
Studio	studio update

Clear **Insert** **Update** **Delete**

FmEditStudios

StudioID	730
Studio	Studio up

Information updated!

OK

Clear **Insert** **Update** **Delete**

FmEditStudios

StudioID	<input type="text"/>
Studio	<input type="text" value="Studio update"/>

Clear **Insert** **Update** **Delete**

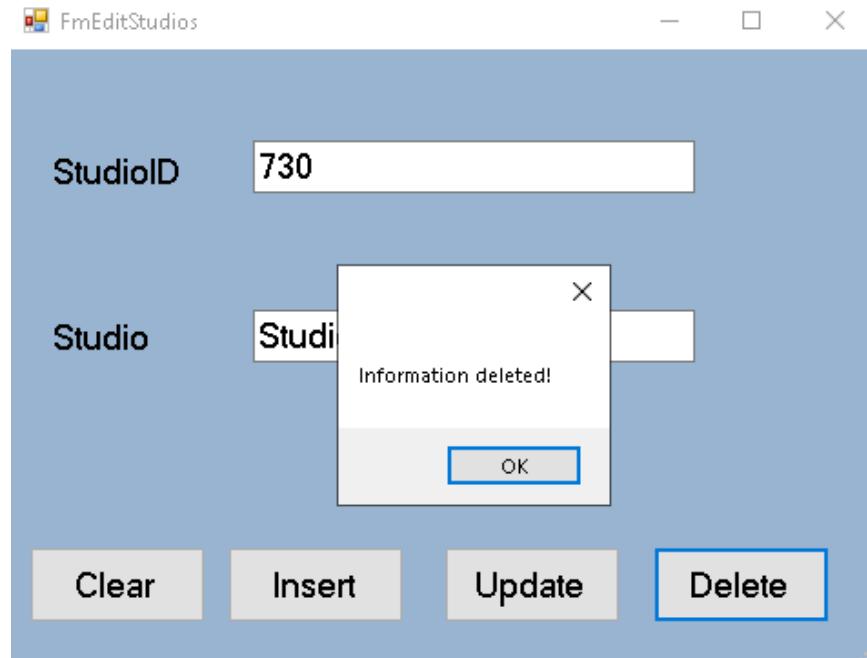
FmEditStudios

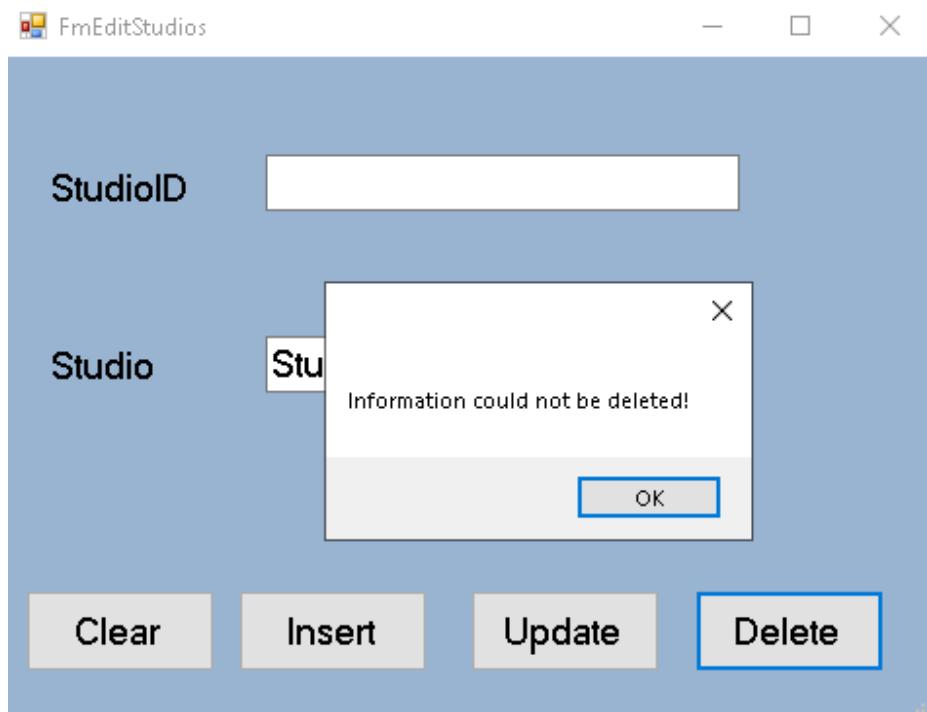
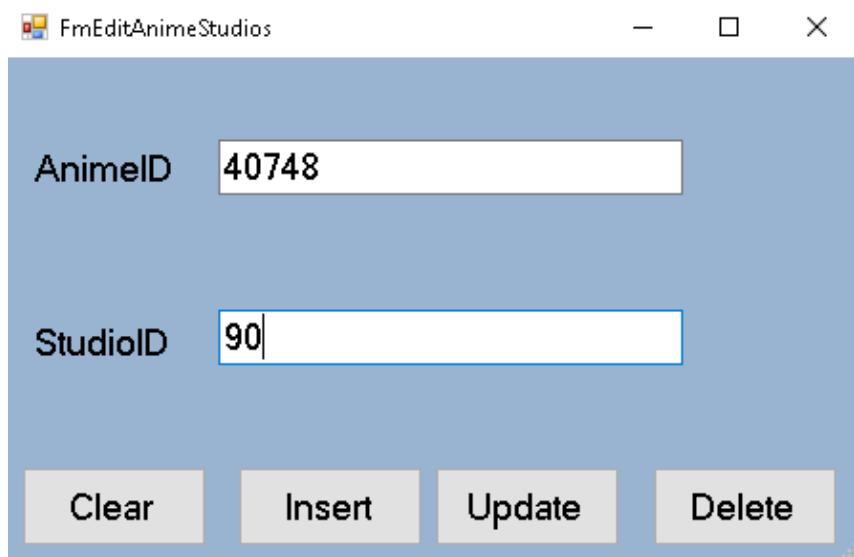
StudioID	<input type="text"/>
Studio	<input type="text" value="Studio update"/>

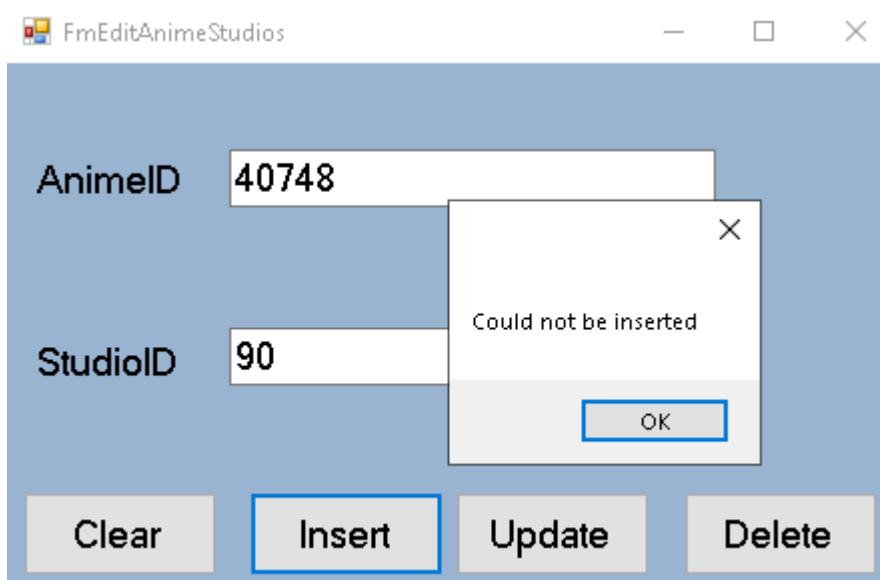
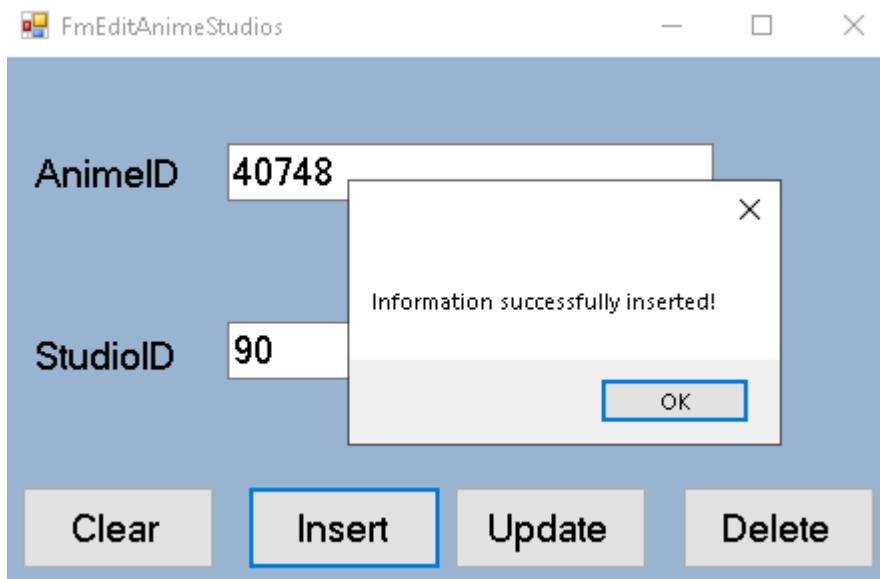
Clear **Insert** **Update** **Delete**

Information could not be updated!

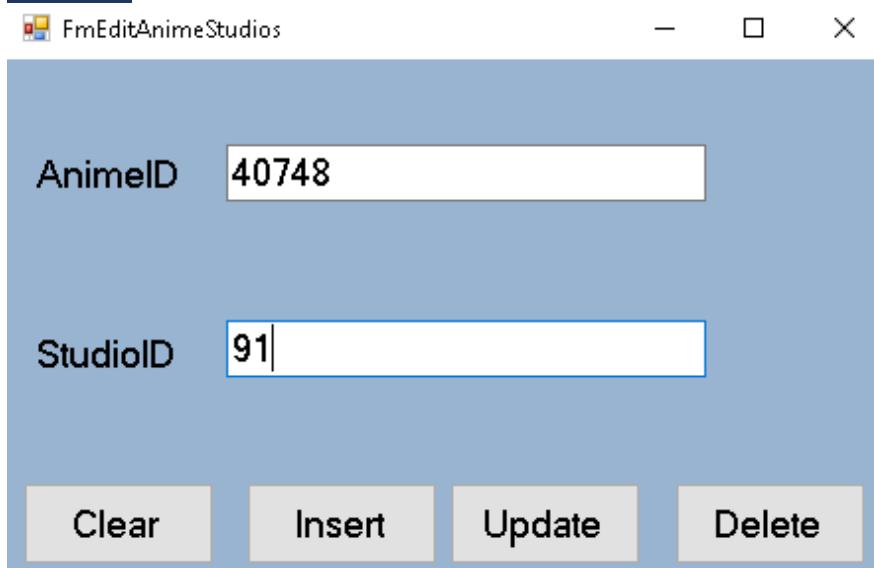
OK

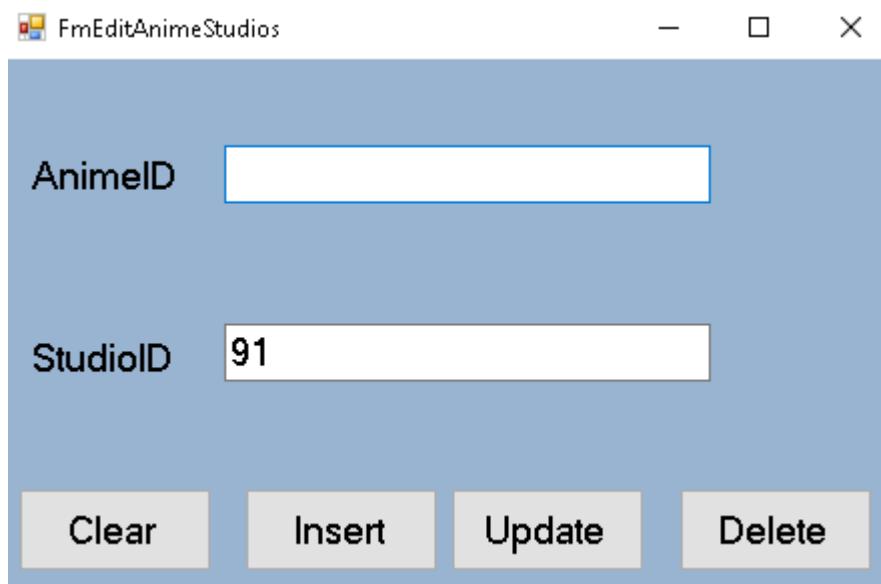
Test #19

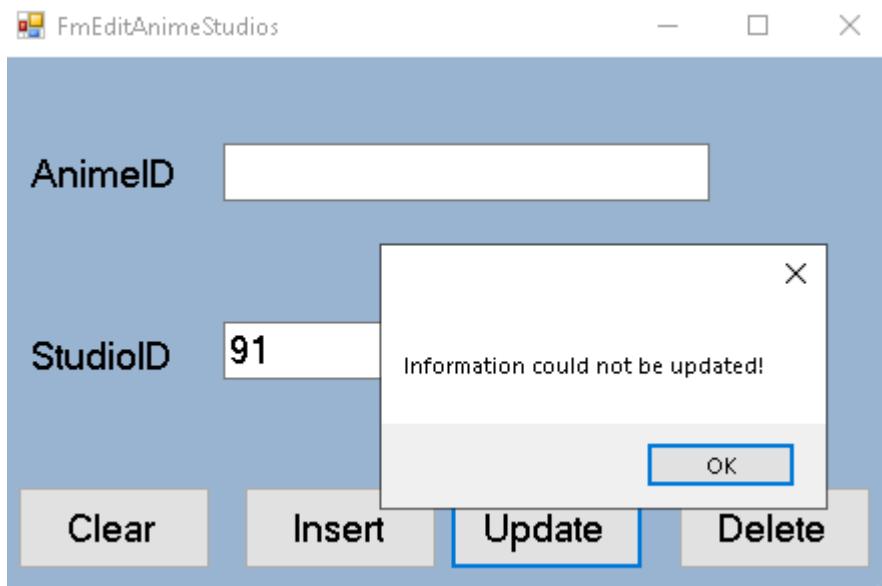
Test #20



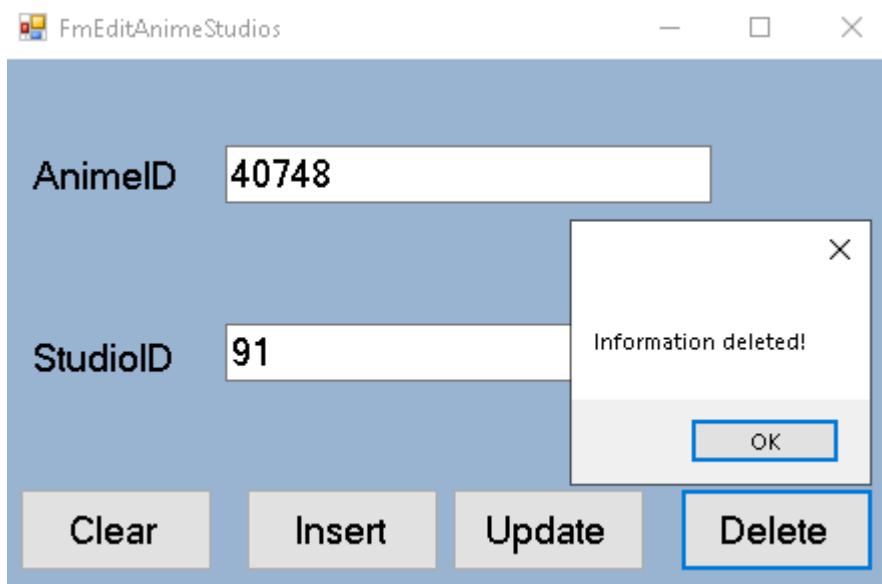
Test #21







Test #22



FmEditAnimeStudios

AnimelD

StudiolD

Clear **Insert** **Update** **Delete**

FmEditAnimeStudios

AnimelD

StudiolD

Clear **Insert** **Update** **Delete**

Information Could not be deleted!

OK

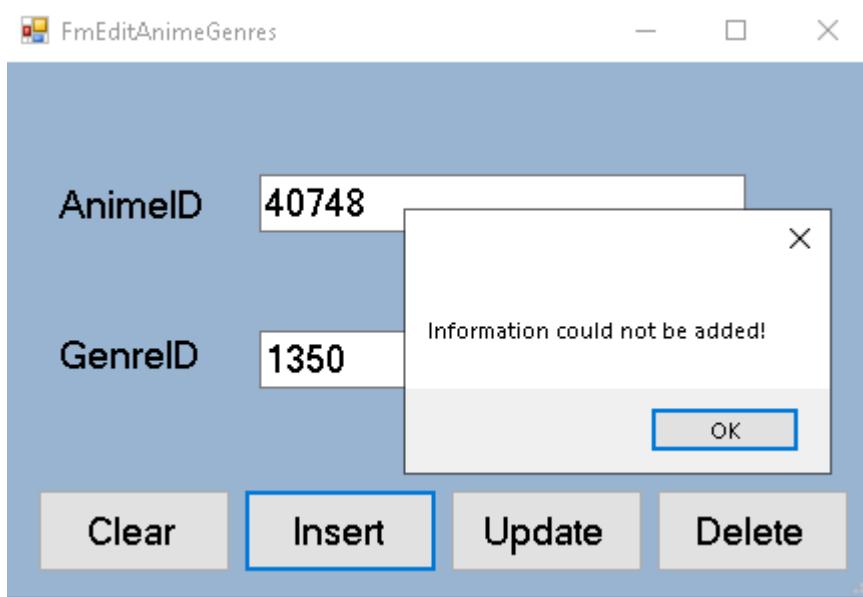
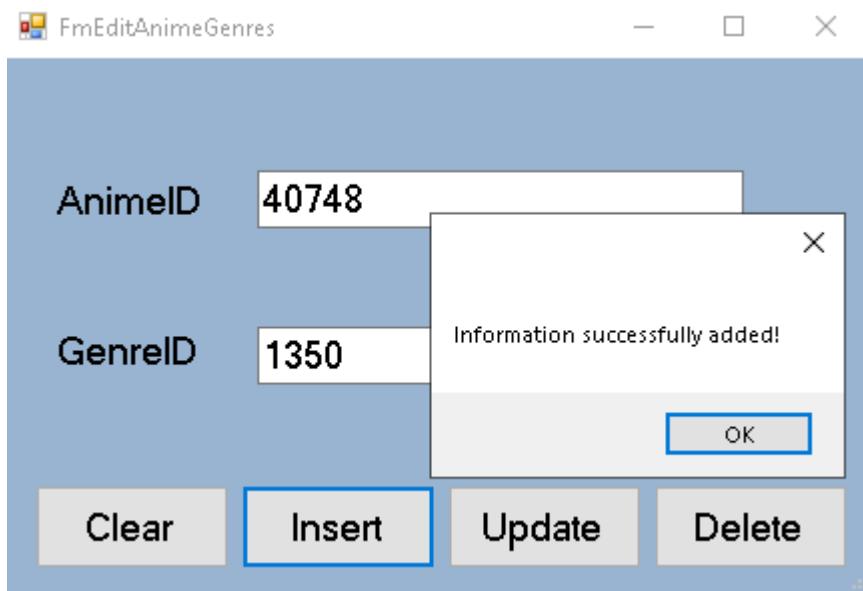
Test #23

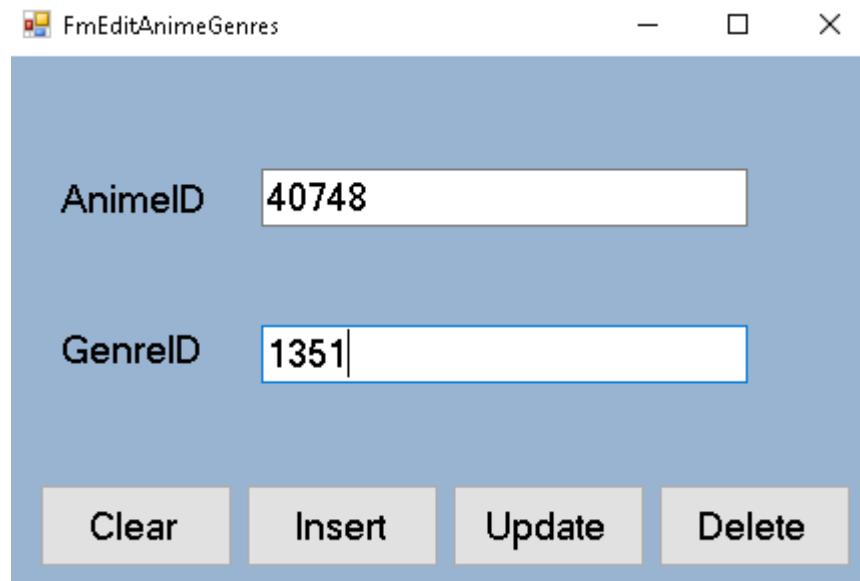
FmEditAnimeGenres

AnimelD

GenreID

Clear **Insert** **Update** **Delete**

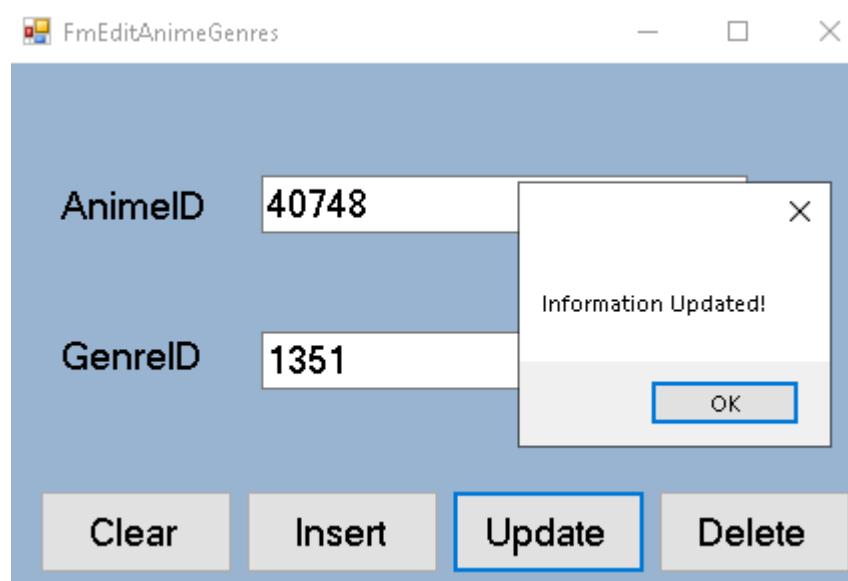


Test #24

AnimelD

GenreID

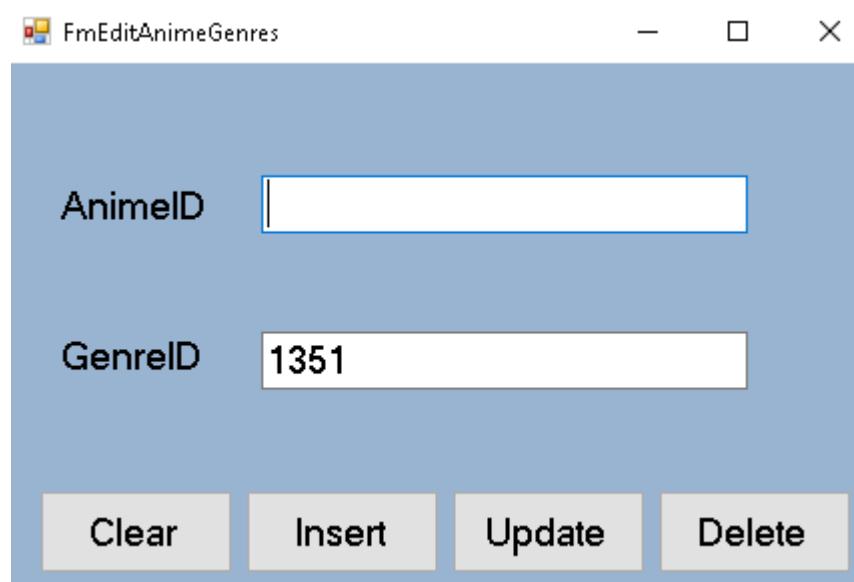
Clear **Insert** **Update** **Delete**



AnimelD

GenreID

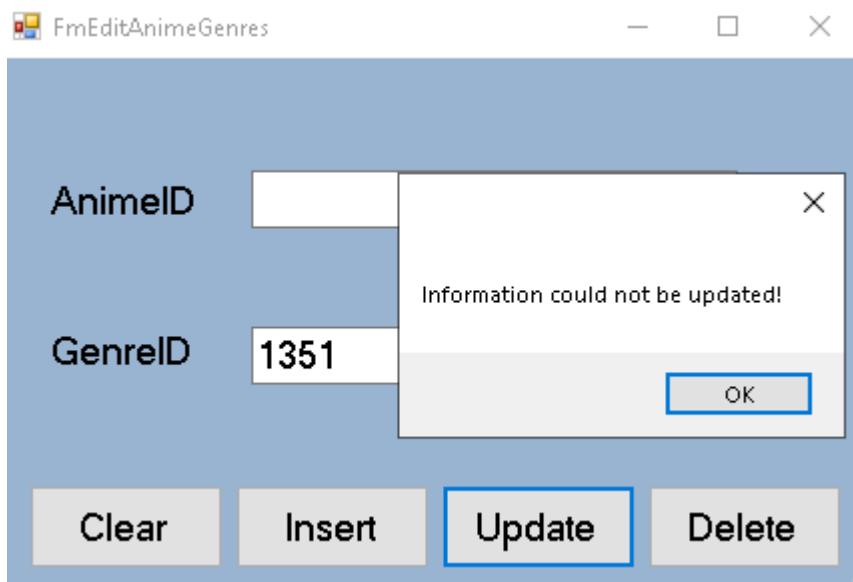
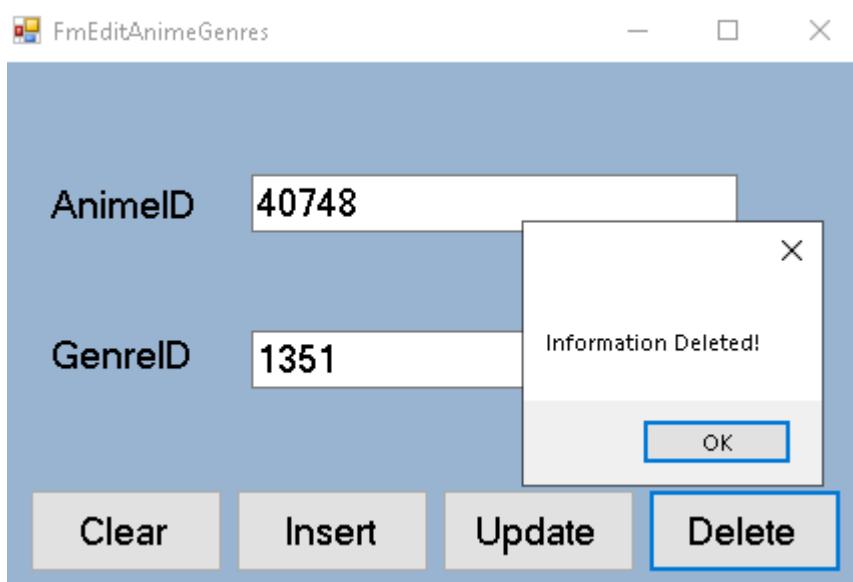
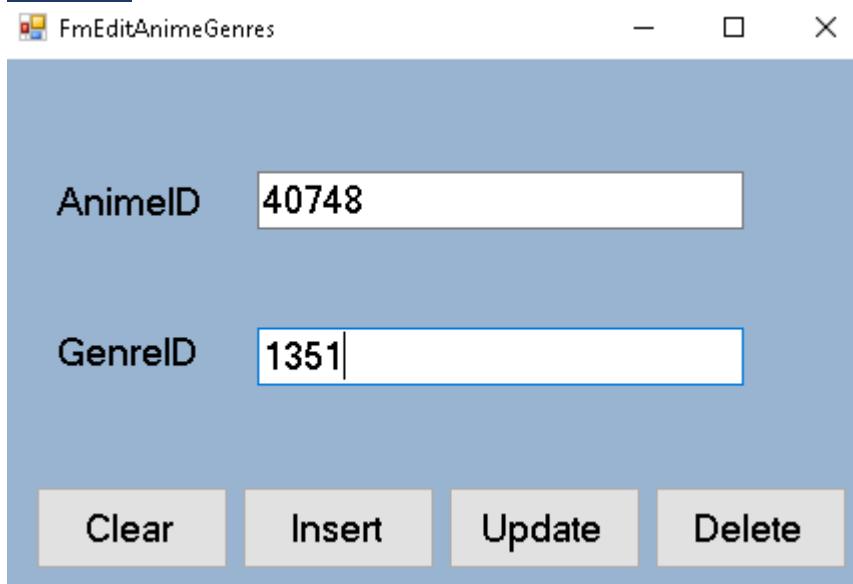
Clear **Insert** **Update** **Delete**



AnimelD

GenreID

Clear **Insert** **Update** **Delete**

Test #25

FmEditAnimeGenres

AnimelD	<input type="text"/>
GenreID	<input type="text" value="1351"/>
Clear Insert Update Delete	

FmEditAnimeGenres

AnimelD	<input type="text"/>
GenreID	<input type="text" value="1351"/>
Clear Insert Update Delete	

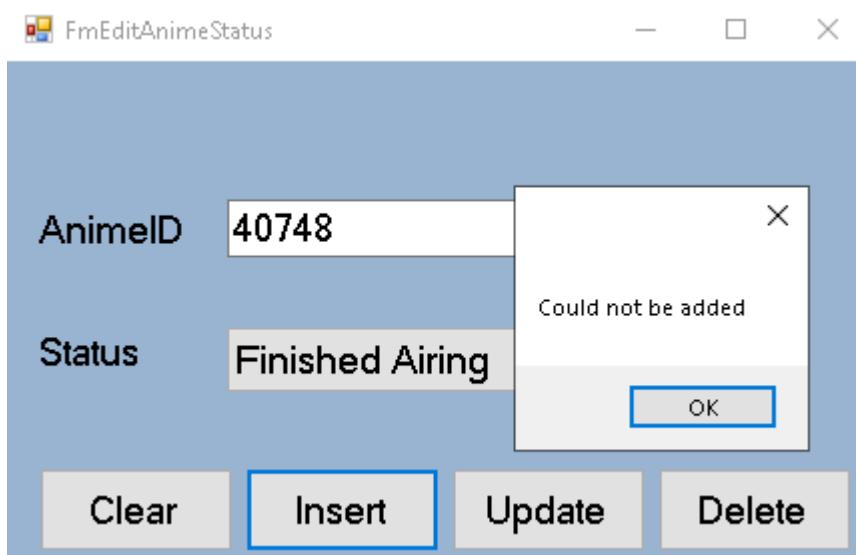
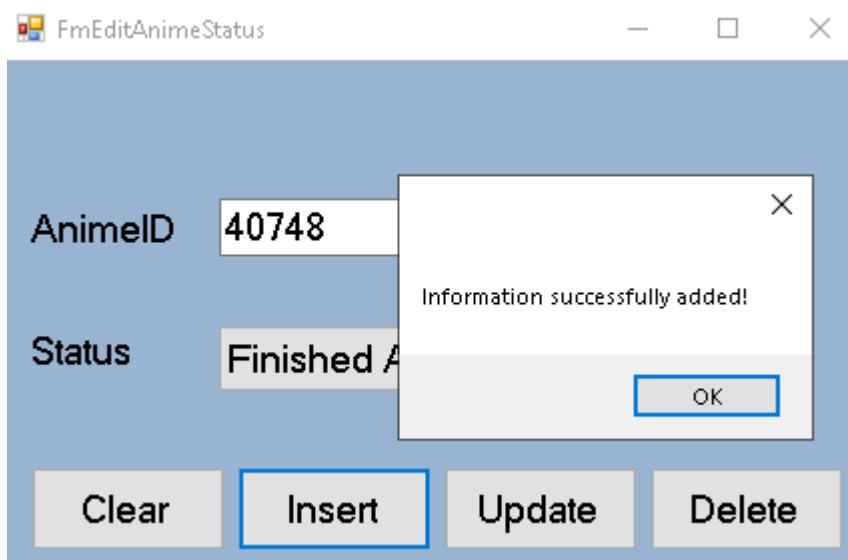
Information could not be deleted!

OK

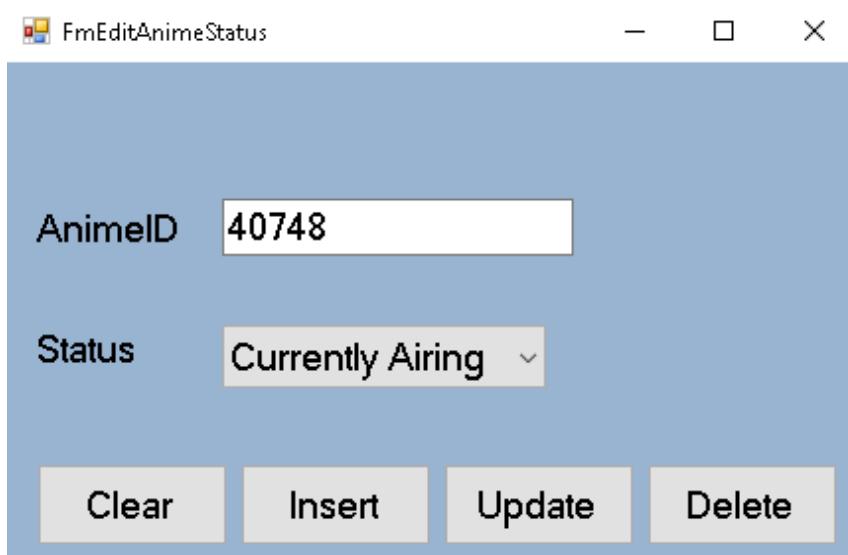
Test #26

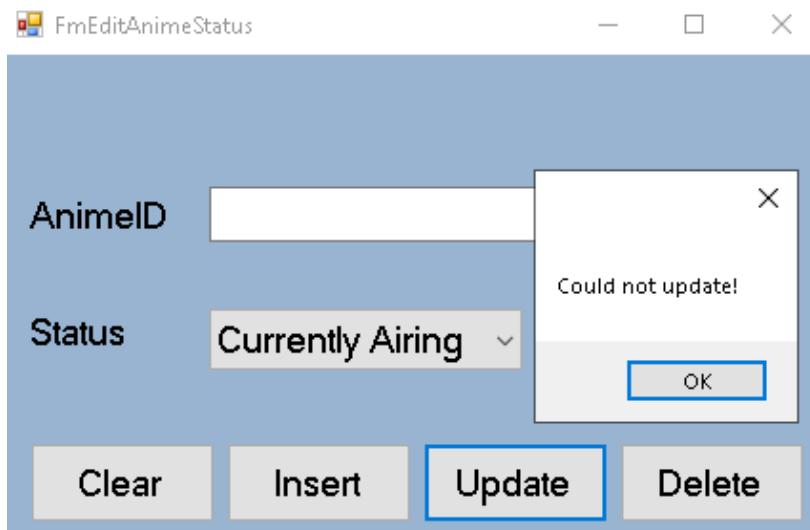
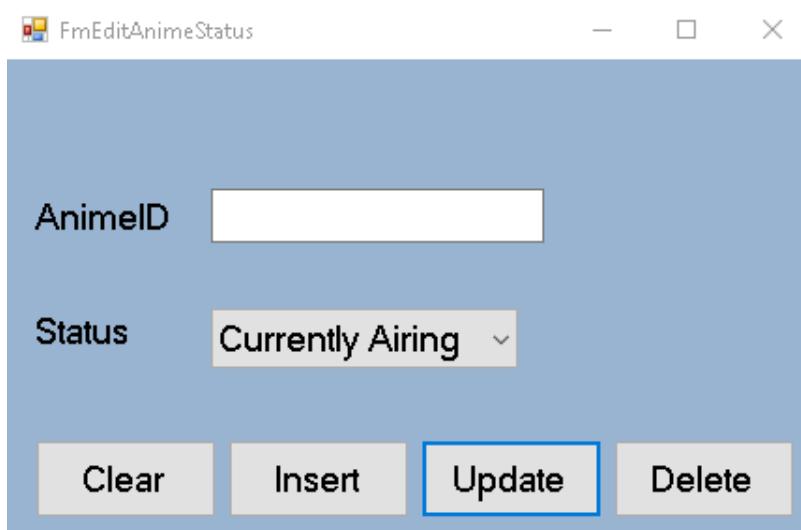
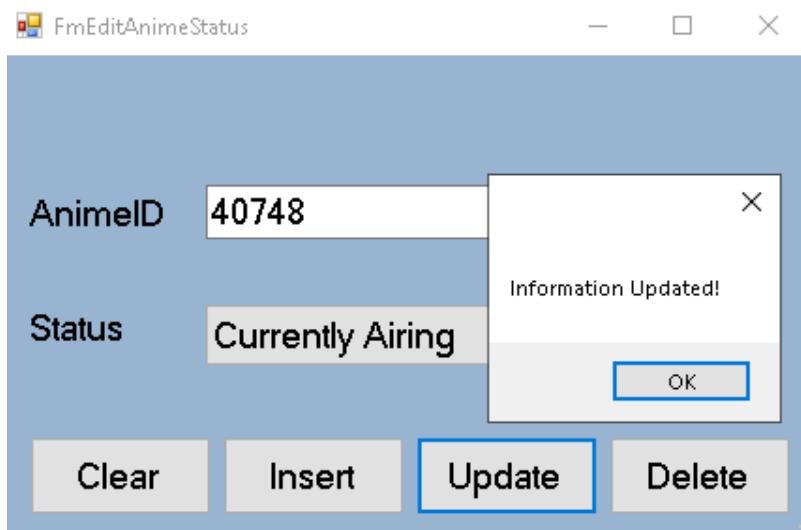
FmEditAnimeStatus

AnimelD	<input type="text" value="40748"/>
Status	<input type="text" value="Finished Airing"/>
Clear Insert Update Delete	



Test #27





Test #28

FmEditAnimeStatus

AnimID	40748
Status	Currently Airing

Clear **Insert** **Update** **Delete**

FmEditAnimeStatus

AnimID	40748
Status	Currently Airing

Clear **Insert** **Update** **Delete**

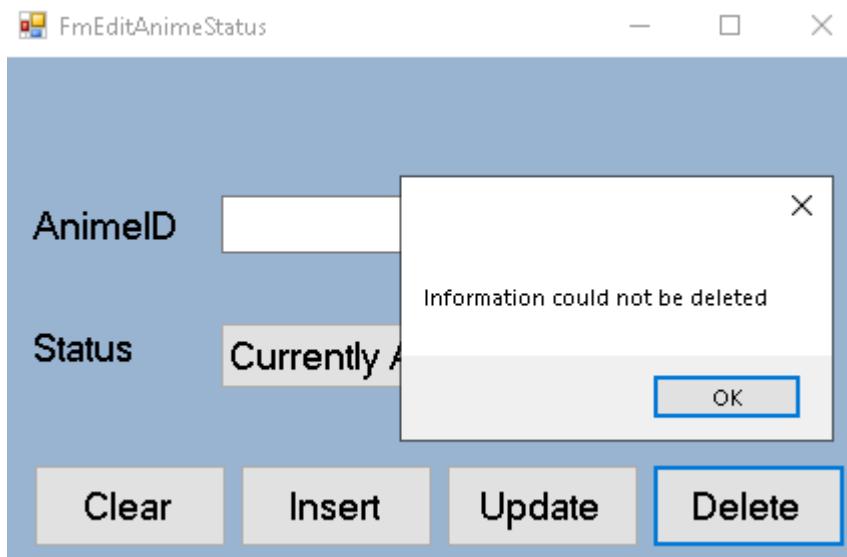
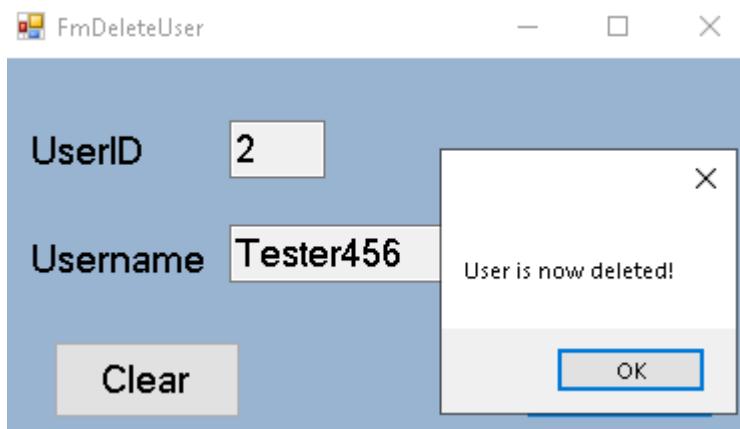
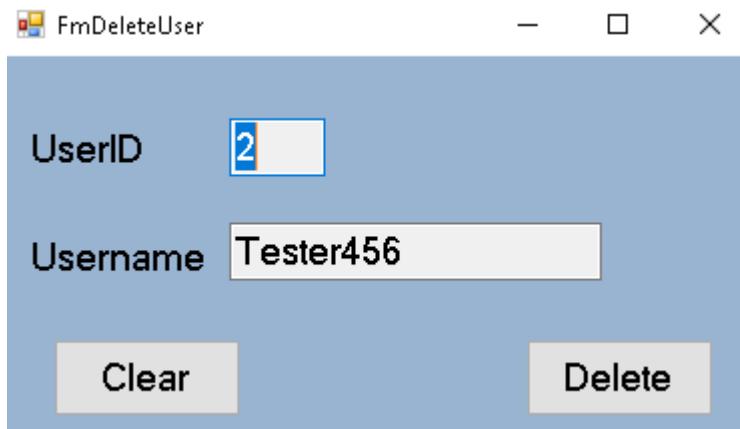
Information Deleted!

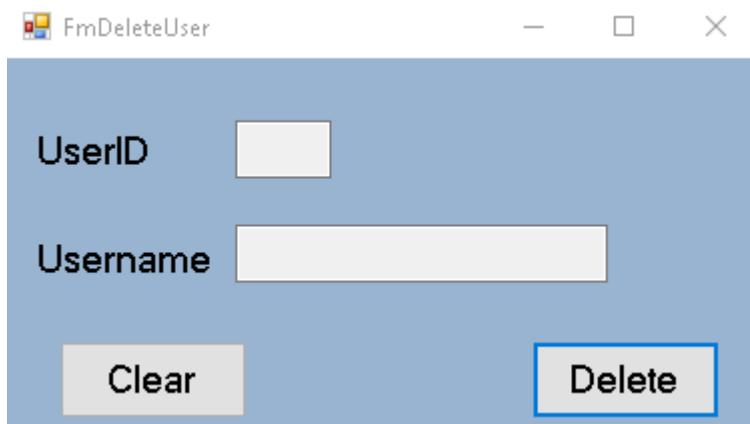
OK

FmEditAnimeStatus

AnimID	
Status	Currently Airing

Clear **Insert** **Update** **Delete**

Test #29



Test #30



20220206_184305.mp4

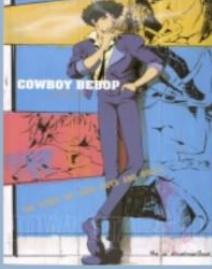
Test #31

FmAMS

Switch AdminTest Profile

AMS

Anime News Loading..

CowBoy Bebop

Trigun

Naruto

Dragon ba

- Dragon Ball
- Dragon Ball GT
- Dragon Ball GT: Gokuu Gaid
- Dragon Ball Heroes
- Dragon Ball Kai
- Dragon Ball Kai (2014)
- Dragon Ball Kai: Mirai ni He
- Dragon Ball Movie 1: Shen L
- Dragon Ball Movie 2: Majinj
- Dragon Ball Movie 3: Makaf
- Dragon Ball Movie 4: Saikyo
- Dragon Ball Specials
- Dragon Ball Super
- Dragon Ball Super Movie
- Dragon Ball Z
- Dragon Ball Z Movie 01: Ora r...

Search

FmAMS

Switch AdminTest Profile

AMS

Dragon Ball

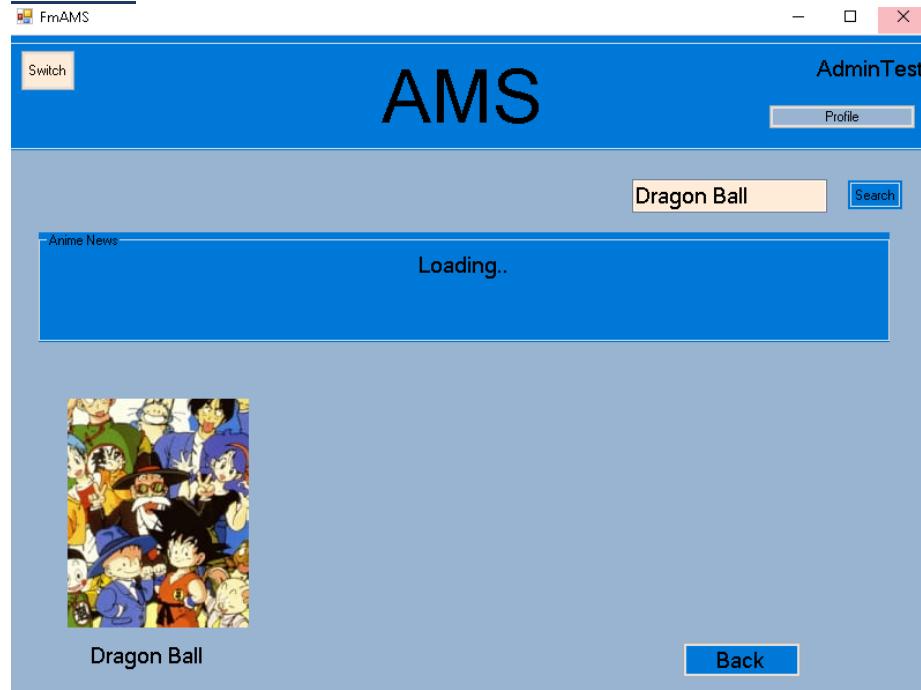
Search

Anime News Loading..

Dragon Ball

Back

Test #32



Test #33

FmAnimeInformation

AMS



Dragon Ball

[Add to favourites](#)

[Add to WatchList](#)

AnimID = 223

Status = Finished Airing

Ratings = 8

Gokuu Son is a young boy who lives in the woods all alone—that is, until a girl named Bulma runs into him in her search for a set of magical objects called the "Dragon Balls." Since the artifacts are said to grant one wish to whoever collects all seven, Bulma hopes to gather them and wish for a perfect boyfriend. Gokuu happens to be in possession of a dragon ball, but unfortunately for Bulma, he refuses to part ways with it, so she makes him a deal: he can tag along on her journey if he lets her borrow the dragon ball's power. With that, the two set off on the journey of a lifetime. They don't go on the journey alone. On the way, they

Rate 6 Episodes Watched: 0

Genres = Adventure & Comedy & Fantasy &

Episodes 153

Studios = Toei Animation

Watch Status:

FmAnimeInformation

AMS



Dragon Ball

[Add to favourites](#)

[Add to WatchList](#)

AnimID = 223

Status = Finished Airing

Ratings = 8

Gokuu Son is a young boy who lives in the woods all alone—that is, until a girl named Bulma runs into him in her search for a set of magical objects called the "Dragon Balls." Since the artifacts are said to grant one wish to whoever collects all seven, Bulma hopes to gather them and wish for a perfect boyfriend. Gokuu happens to be in possession of a dragon ball, but unfortunately for Bulma, he refuses to part ways with it, so she makes him a deal: he can tag along on her journey if he lets her borrow the dragon ball's power. With that, the two set off on the journey of a lifetime. They don't go on the journey alone. On the way, they

Added to favourites!

OK

Rate 6 Episodes Watched: 0

Genres = Adventure & Comedy & Fantasy &

Episodes 153

Studios = Toei Animation

Watch Status:

FmAnimeInformation

AMS



Dragon Ball

Add to favourites

Add to WatchList

AnimID = 223

Status = Finished Airing

Ratings = 8

Gokuu Son is a young boy who lives in the woods all alone—that is, until a girl named Bulma runs into him in her search for a set of magical objects called the "Dragon Balls." Since the artifacts are said to grant one wish to whoever collects all seven, Bulma hopes to gather them and wish for a perfect boyfriend. Gokuu happens to be the last person to find a Dragon Ball, but unfortunately, he can't keep it for long. He part ways with it, so he can tag along on her quest to borrow the dragon balls. The two set off on the journey of a lifetime. They don't go on the journey alone. On the way, they meet a lot of interesting people, including the Saiyans, who are the most powerful warriors in the universe. They also meet the Z-fighters, who are a group of heroes who protect the Earth from various threats. The journey is full of adventure, excitement, and danger, but the characters are determined to succeed in their quest to find the Dragon Balls. They face many challenges along the way, including battles with powerful enemies like Cell and Frieza, and obstacles like the Namekian dragon and the界王神. The story is filled with action, humor, and emotional moments, making it a classic anime that has won the hearts of many fans around the world.

Genres = Adventure & Comedy & Fantasy & Sci-Fi

Episodes = 153

Studios = Toei Animation

Watch Status:

OK

Anime is already favourited

FmAnimeInformation

AMS



Gol D. Roger was known as the "Pirate King," the strongest and most infamous being to have sailed the Grand Line. The capture and execution of Roger by the World Government brought a change throughout the world. His last words before his death revealed the existence of the greatest treasure in the world, One Piece. It was this revelation that brought about the Grand Age of Pirates, men who dreamed of finding One Piece—which promises an unlimited amount of riches and fame—and quite possibly the pinnacle of glory and the title of the Pirate King. Enter Monkey D. Luffy, a 17-year-old boy who defies your standard definition of a

One Piece

Add to favourites

Add to WatchList

AnimelD = 21

Status = Currently Airing

Ratings = 9

Rate 5 Episodes Watched: 203

Genres = Action & Adventure & Comedy &

Episodes 1008

Studios = Toei Animation

Watch Status: Currently Watch

FmAnimeInformation

AMS



Gol D. Roger was known as the "Pirate King," the strongest and most infamous being to have sailed the Grand Line. The capture and execution of Roger by the World Government brought a change throughout the world. His last words before his death revealed the existence of the greatest treasure in the world, One Piece. It was this revelation that brought about the Grand Age of Piracy. x

One Piece

Add to favourites

Add to WatchList

AnimID = 21

Status = Currently Airing

Ratings = 9

Genres = Action & Adventure & Comedy &

Episodes = 1008

Studios = Toei Animation

Rate 5 ▼

Episodes Watched: 203

Watch Status: Currently Watch ▼

Added to watchlist! OK

FmAnimeInformation

AMS



Gol D. Roger was known as the "Pirate King," the strongest and most infamous being to have sailed the Grand Line. The capture and execution of Roger by the World Government brought a change throughout the world. His last words before his death revealed the existence of the greatest treasure in the world, One Piece. It was this revelation that brought about the Grand Age of Piracy, a time where pirates could find One Piece with unlimited amounts of treasure, possibly the pinnacle of the Grand Line. Entering the world of the Pirate King. Enter the world of a boy who defies your standard definition of a hero.

One Piece

Add to favourites

Add to WatchList

AnimID = 21

Status = Currently Airing

Ratings = 9

Rate: 5

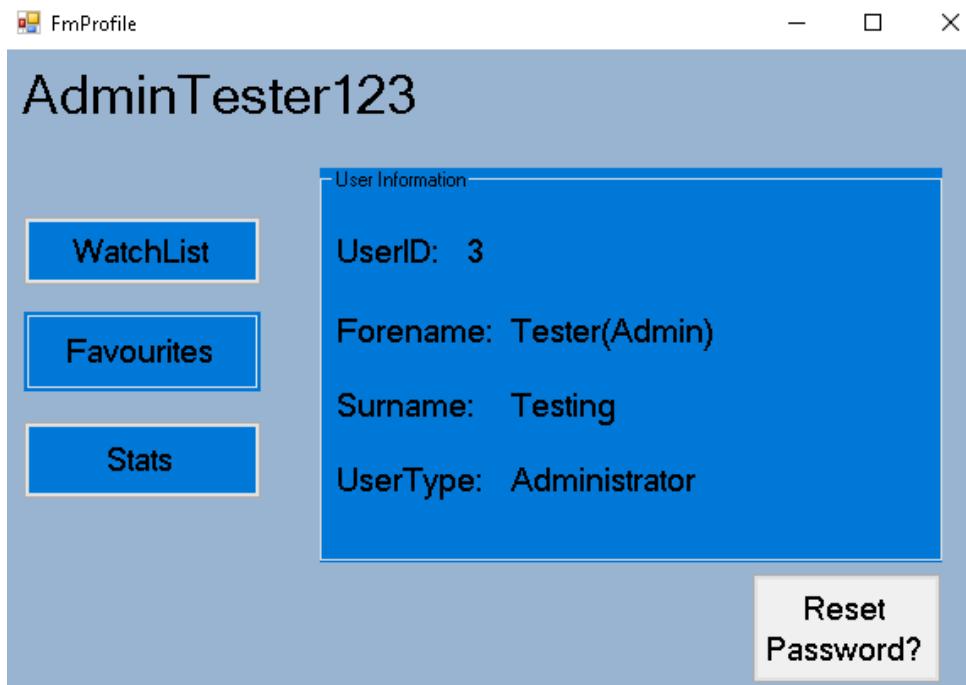
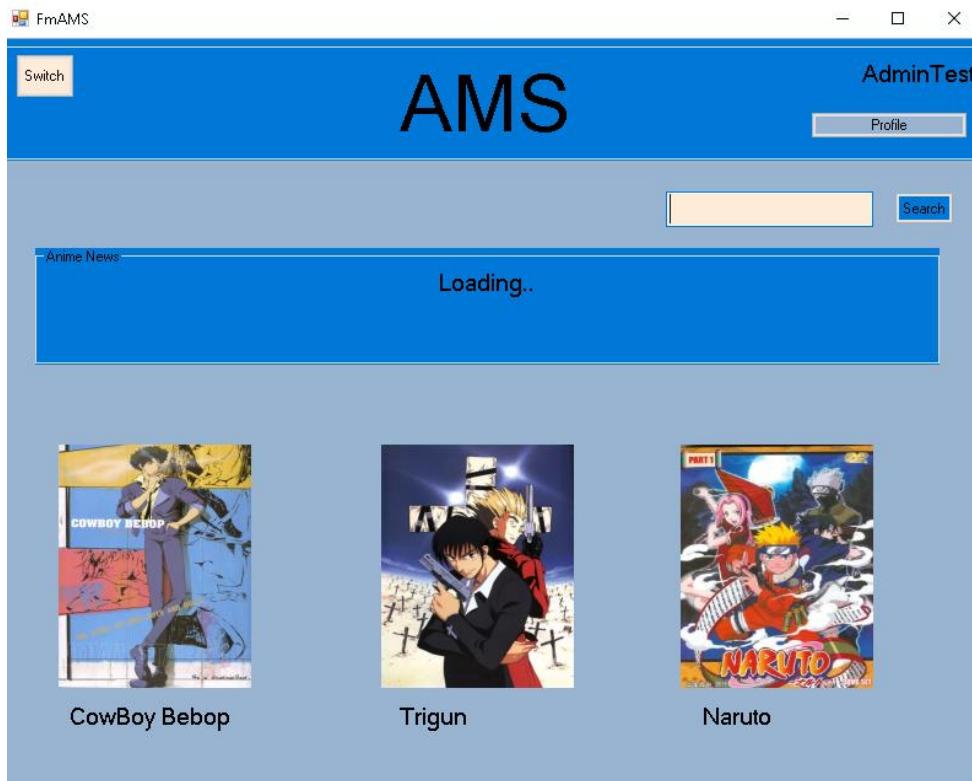
Episodes Watched: 203

Genres = Action & Adventure & Comedy & ...

Episodes = 1008

Studios = Toei Animation

Watch Status: Currently Watch

Test #34

Test #35

FmProfile

- □ ×

AdminTester123

WatchList**Favourites****Stats****User Information****UserID:** 3**Forename:** Tester(Admin)**Surname:** Testing**UserType:** Administrator**Reset
Password?**

FmReset

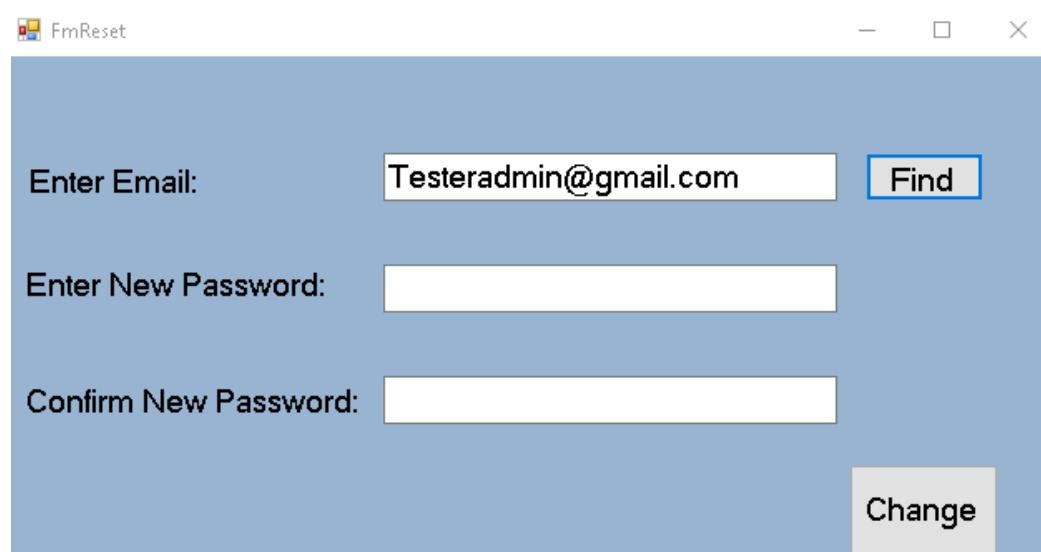
- □ ×

Enter Email:**Find****Change**

Test #36

Enter Email: Find

Change

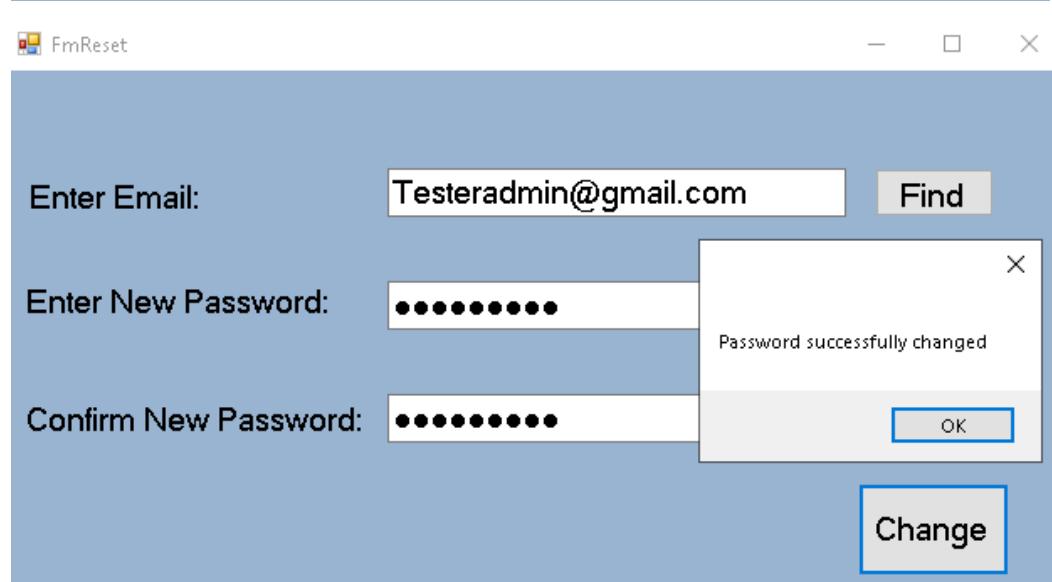


Enter Email: Find

Enter New Password:

Confirm New Password:

Change



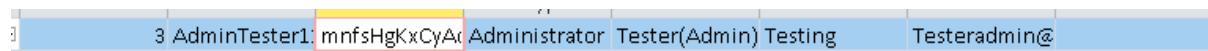
Enter Email: Find

Enter New Password:

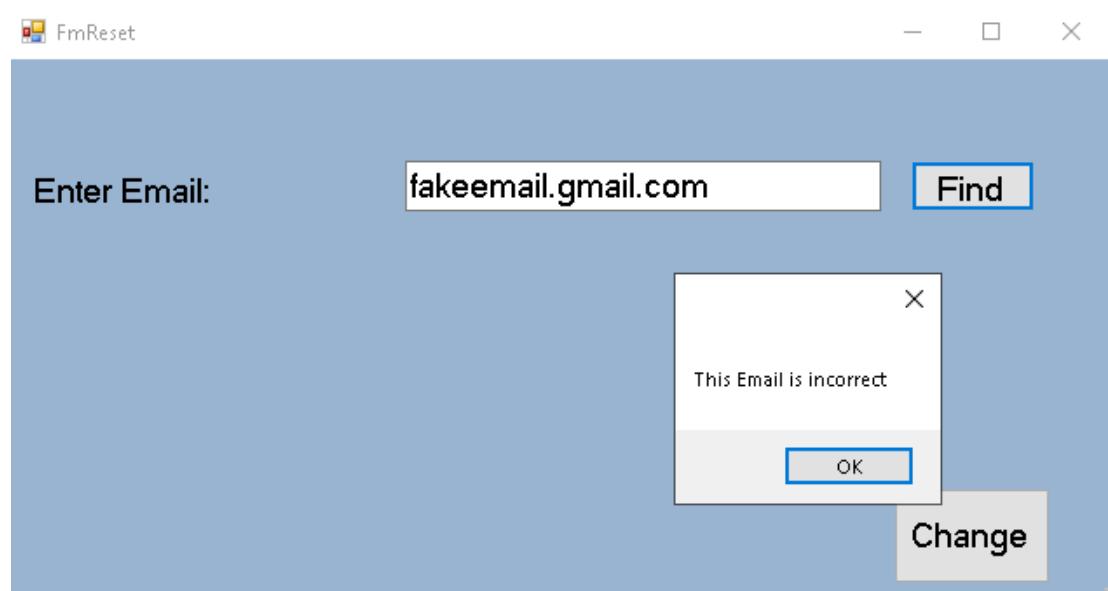
Confirm New Password:

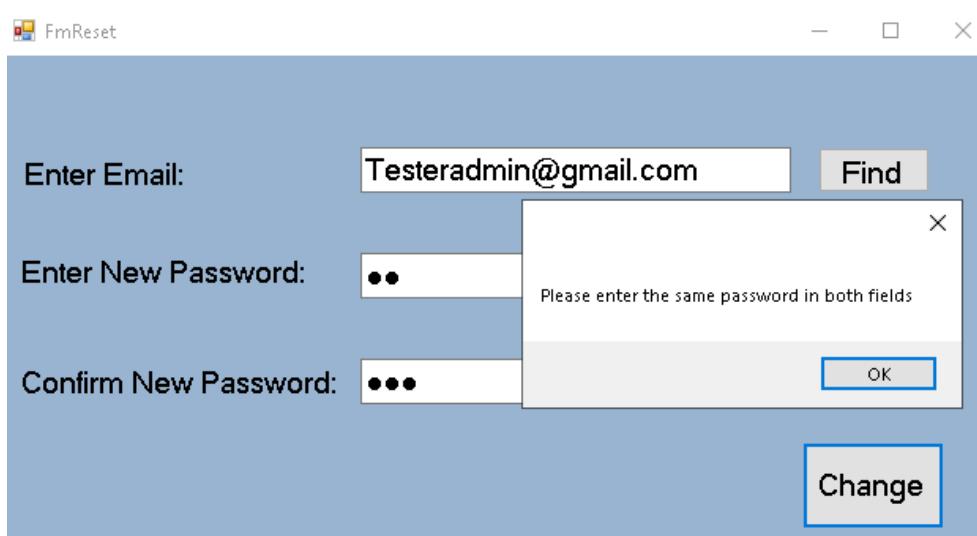
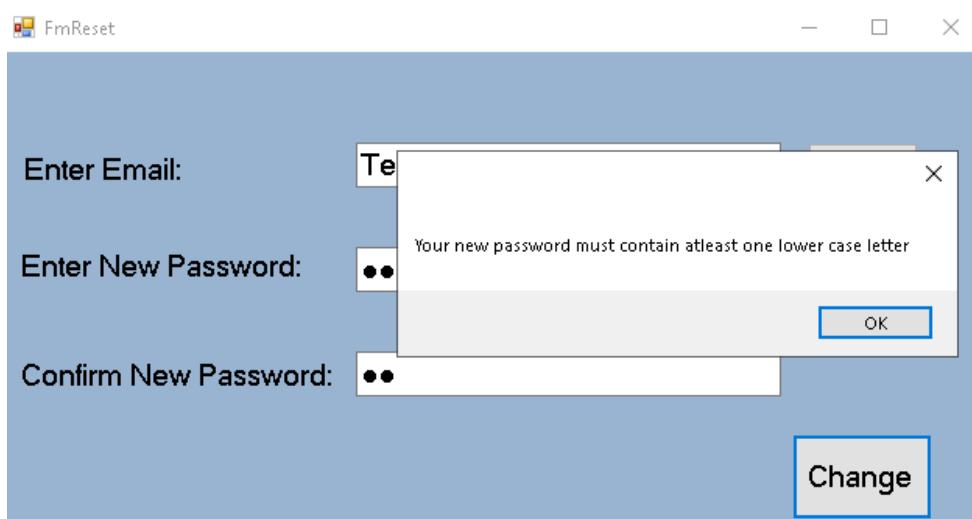
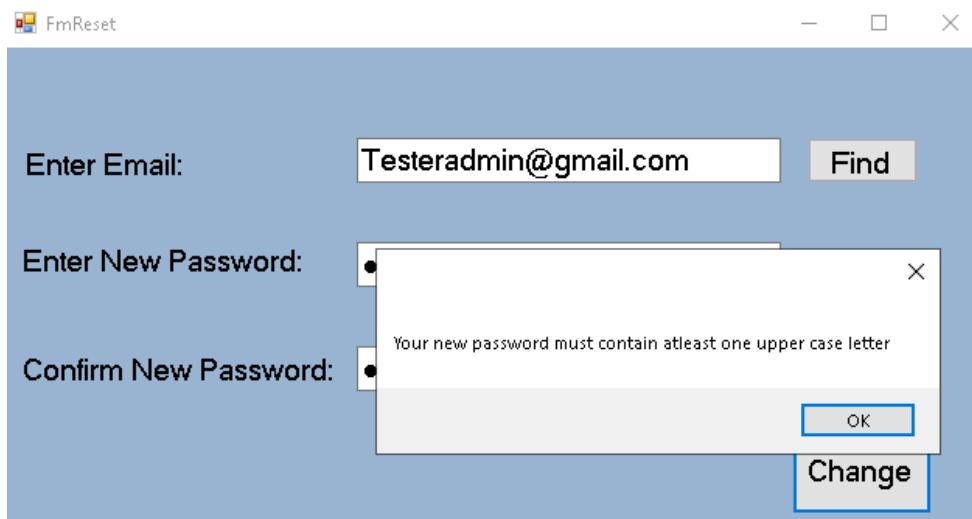
OK

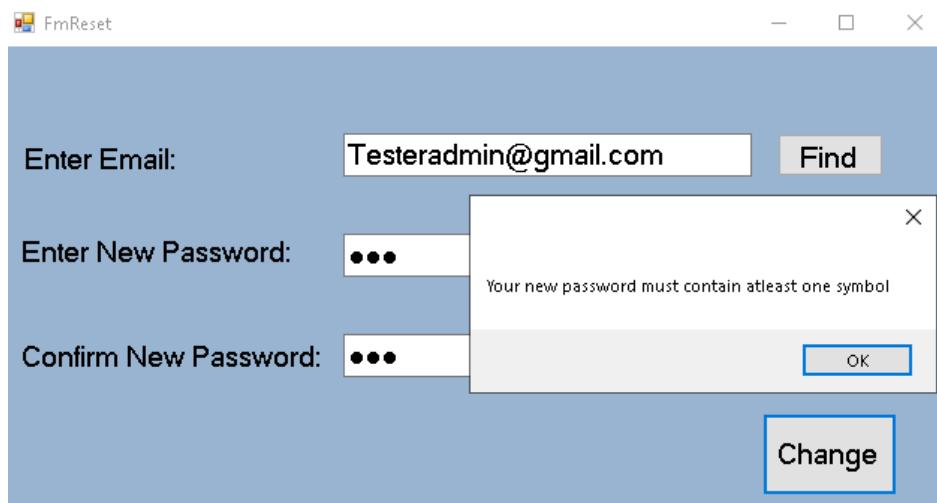
Change

New Password

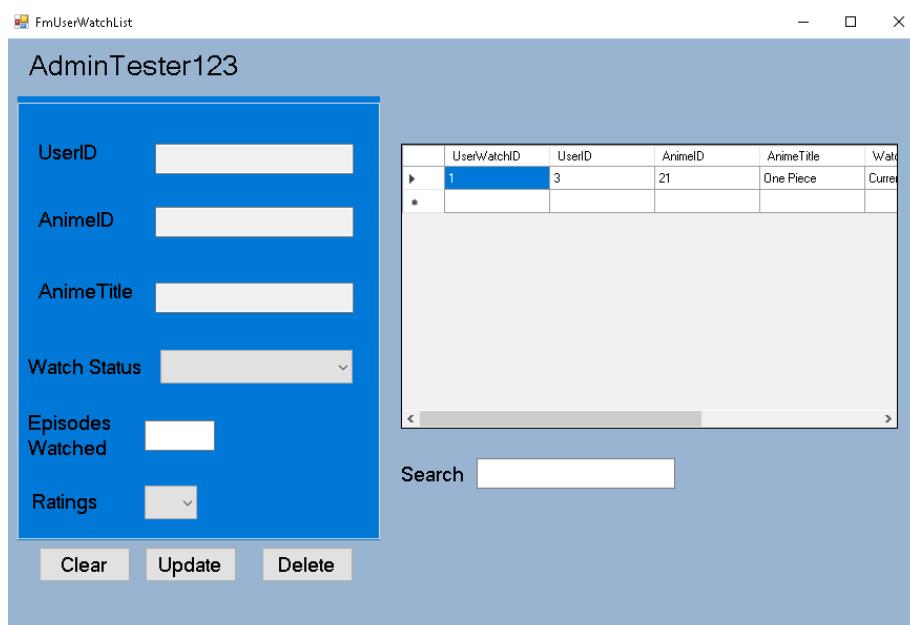
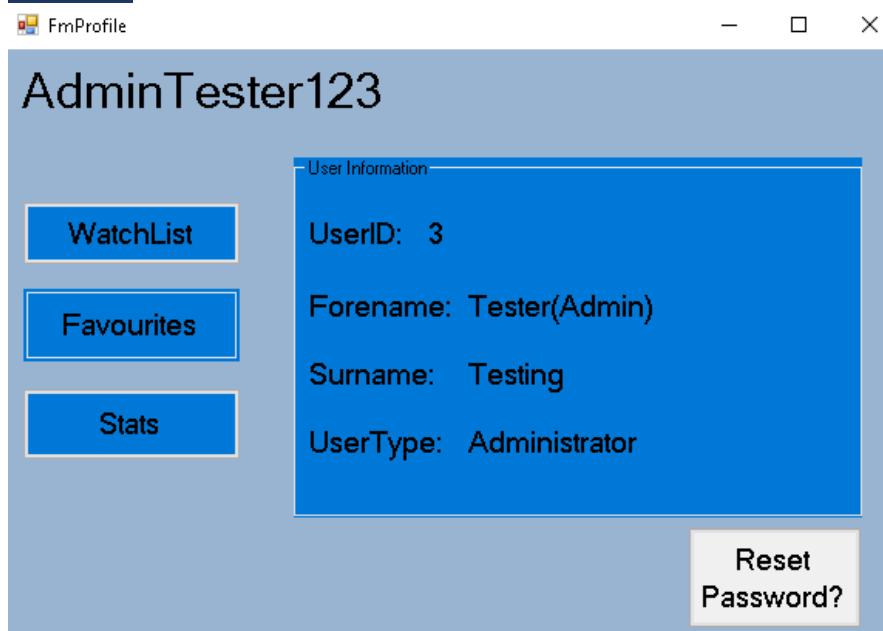
(See on Test #5(page 236) for the old password)







Test #37



FmUserFavourites

AdminTester123

UserID	<input type="text"/>
AnimelD	<input type="text"/>
AnimeTitle	<input type="text"/>
Ratings	<input type="button" value="▼"/>

Search

UserFavID	UserID	AnimelD	AnimeTitle	Ratir
1	3	223	Dragon Ball	6
*				

Test #38

FmUserFavourites

AdminTester123

UserID	<input type="text"/>
AnimelD	<input type="text"/>
AnimeTitle	<input type="text"/>
Ratings	<input type="button" value="▼"/>

Search

UserFavID	UserID	AnimelD	AnimeTitle	Ratir
1	3	223	Dragon Ball	6
*				

FmUserFavourites

AdminTester123

UserID	3
AnimelD	223
AnimeTitle	Dragon Ball
Ratings	6

Clear **Update** **Delete**

UserFavID	UserID	AnimelD	AnimeTitle	Ratings
1	3	223	Dragon Ball	6
*				

Search



Test #39

FmUserFavourites

AdminTester123

UserID	3
AnimelD	223
AnimeTitle	Dragon Ball
Ratings	8

Clear **Update** **Delete**

UserFavID	UserID	AnimelD	AnimeTitle	Ratings
1	3	223	Dragon Ball	6
*				

Search



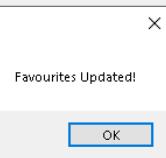
FmUserFavourites

AdminTester123

UserID	3
AnimelD	223
AnimeTitle	Dragon Ball
Ratings	8

Search

UserFavID	UserID	AnimelD	AnimeTitle	Ratings
1	3	223	Dragon Ball	6
*				

 Favourites Updated!



Test #40

FmUserFavourites

AdminTester123

UserID	3
AnimelD	223
AnimeTitle	Dragon Ball
Ratings	8

Search

UserFavID	UserID	AnimelD	AnimeTitle	Ratings
1	3	223	Dragon Ball	8
*				



FmUserFavourites

AdminTester123

UserID	3
AnimelD	223
AnimeTitle	Dragon Ball
Ratings	8

Search

Deleted from Favourites!

OK



Test #41

FmUserWatchList

AdminTester123

UserID	
AnimelD	
AnimeTitle	
Watch Status	
Episodes Watched	
Ratings	

Search

FmUserWatchList

AdminTester123

UserID	3
AnimelD	21
AnimeTitle	One Piece
Watch Status	Currently Watching
Episodes Watched	203
Ratings	5

UserWatchID	UserID	AnimelD	AnimeTitle	Watch Status
1	3	21	One Piece	Currently Watching
*				

Search



Total Episodes: 1008

Test #42

FmUserWatchList

AdminTester123

UserID	3
AnimelD	21
AnimeTitle	One Piece
Watch Status	Dropped
Episodes Watched	300
Ratings	2

UserWatchID	UserID	AnimelD	AnimeTitle	Watch Status
1	3	21	One Piece	Currently Watching
*				

Search



Total Episodes: 1008

FrmUserWatchList

AdminTester123

UserID	3
AnimelD	21
AnimeTitle	One Piece
Watch Status	Dropped
Episodes Watched	300
Ratings	2

Clear **Update** **Delete**

WatchList Updated!

OK

Search



Total Episodes: 1008

Test #43

FrmUserWatchList

AdminTester123

UserID	3
AnimelD	21
AnimeTitle	One Piece
Watch Status	Dropped
Episodes Watched	300
Ratings	2

Clear **Update** **Delete**



Total Episodes: 1008

FrmUserWatchList

AdminTester123

UserID	3
AnimelD	21
AnimeTitle	One Piece
Watch Status	Dropped
Episodes Watched	300
Ratings	2

Clear **Update** **Delete**

UserWatchID	UserID	AnimelD	AnimeTitle	Watch Status
1	3	21	One Piece	Dropped
*				

Deleted from WatchList!

OK

Search



Total Episodes: 1008

Test #44

FrmUserFavourites

AdminTester123

UserID	
AnimelD	
AnimeTitle	
Ratings	

Clear **Update** **Delete**

UserFavID	UserID	AnimelD	AnimeTitle	Ratings
2	3	20	Naruto	7
3	3	813	Dragon ball z	10
4	3	22319	Tokyo Ghoul	7
5	3	30831	Kono Subarashii Seishun	8
*				

Search

FmUserFavourites

AdminTester123

UserID	<input type="text"/>
AnimelD	<input type="text"/>
AnimeTitle	<input type="text"/>
Ratings	<input type="text"/>

Clear Update Delete

UserFavID	UserID	AnimelD	AnimeTitle	Ratings
4	3	22319	Tokyo Ghoul	7
*				

Search T

Tokyo Ghoul

FmUserWatchList

AdminTester123

UserID	<input type="text"/>
AnimelD	<input type="text"/>
AnimeTitle	<input type="text"/>
Watch Status	<input type="text"/>
Episodes Watched	<input type="text"/>
Ratings	<input type="text"/>

Clear Update Delete

UserWatchID	UserID	AnimelD	AnimeTitle	Watch Status
2	3	20	Naruto	Completed
3	3	813	Dragon ball z	Completed
4	3	22043	Fairy Tail (2014)	Completed
5	3	22319	Tokyo Ghoul	Completed
6	3	30831	Kono Subarashii ...	Completed
*				

Search

FmUserWatchList

AdminTester123

UserID	<input type="text" value=""/>
AnimelD	<input type="text" value=""/>
AnimeTitle	<input type="text" value=""/>
Watch Status	<input type="text" value=""/>
Episodes Watched	<input type="text" value=""/>
Ratings	<input type="text" value=""/>

Clear Update Delete

UserWatch	UserID	AnimelD	AnimeTitle	WatchStat	EpisodesW	Ratings
4	3	22043	Fairy Tail (2014)	Completed	102	5
*						

Search

Fairy Tail (2014)

Test #45

FmProfile

AdminTester123

WatchList

Favourites

Stats

User Information

UserID: 3
Forename: Tester(Admin)
Surname: Testing
UserType: Administrator

Reset
Password?

FmUserStats

AdminTester123

Total Ratings: 9

Average Ratings: 7.7

Animes Completed: 5

Animes Currently Watching: 0

Animes Yet To Watch: 0

Animes Dropped: 0

Total Episodes Watched: 635

Hours Spent Watching: 254

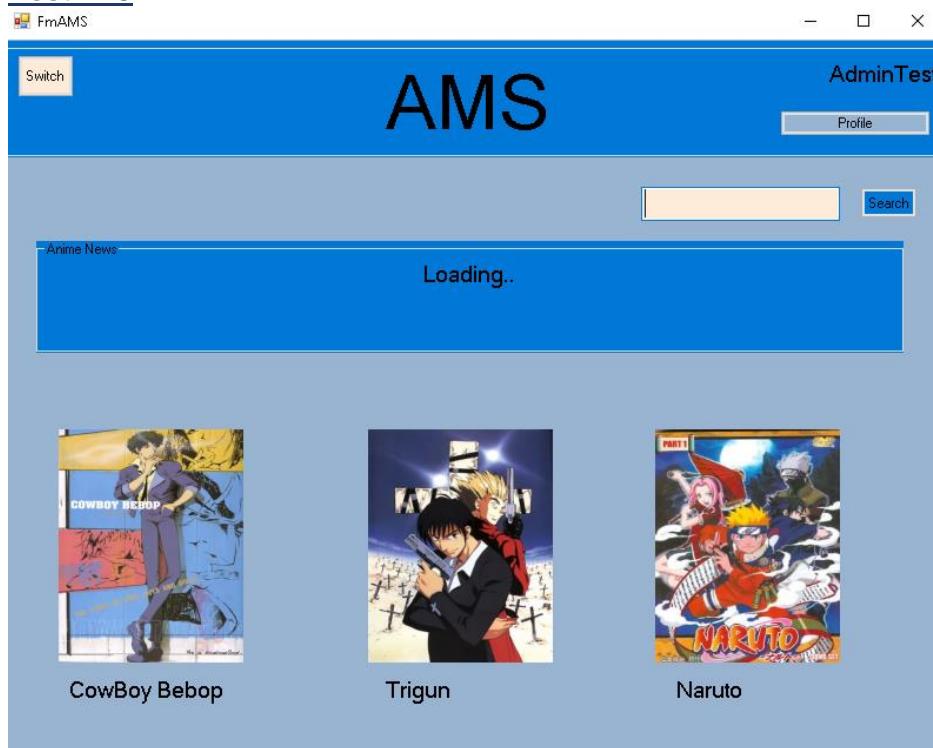
Test #46

FmAMS

Switch AdminTester123 Profile

Anime News

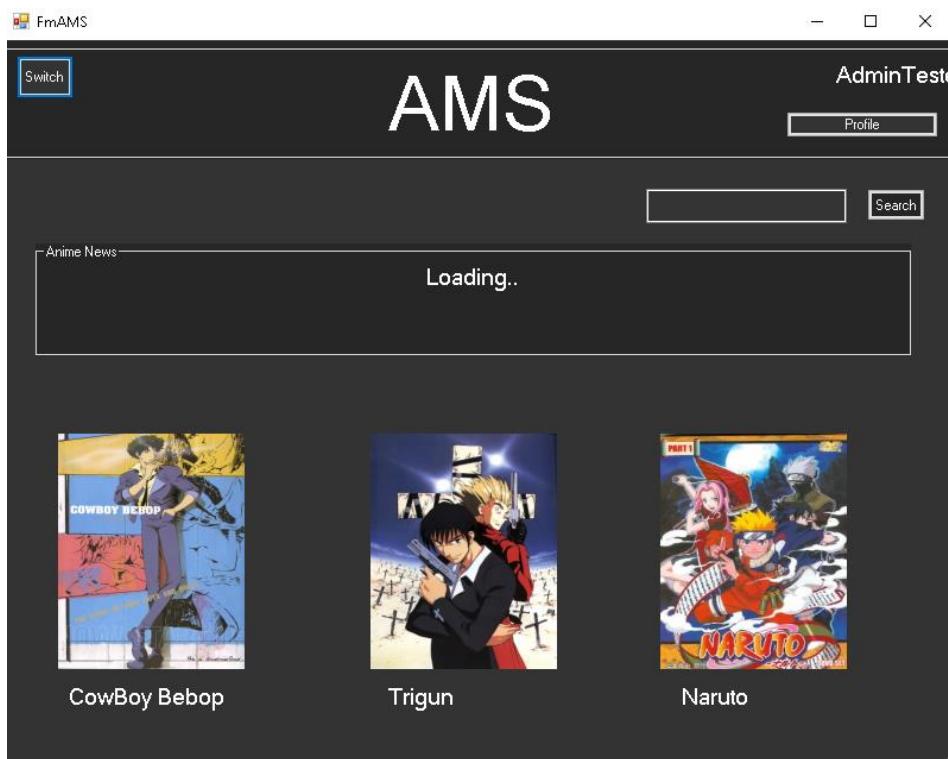
Loading..



CowBoy Bebop

Trigun

Naruto



NEA VIDEO

<https://youtu.be/160foY5t5j0>

Evaluation

Evaluation against objectives

Objectives	How the objectives was met	Possible improvements or enhancements
<p>1. My program should check if the database already exists – if it doesn't exist is should create all the tables within it</p> <p>1.1 There must be an 'Animes' table to store all Animes down to their titles, IDs for every Anime, Episode amount of all Animes, Pictures allocated to all Anime, Synopsis allocated to all Animes. A csv file will be used to mass insert all the values required for this table.</p> <p>1.2 There must be a 'Studios' table to store all Studios and their respective IDs</p>	<p>Objectives met. The program uses the <code>FileExists</code> function to check if the database exist. If it does not exist, then the database and the tables are created using DDL commands. The program also 'inserts' animes, genres, studios,</p>	None

<p>alongside it. A csv file will be used to mass insert all the values required for this table</p> <p>1.3 There must be a 'Genres' table to store all Genres and their respective IDs alongside it. . A csv file will be used to mass insert all the values required for this table.</p> <p>1.4 There must be an 'AnimeStudios' table to split the many to many relationship between the table Animes and Studios. This table must contain all AnimeIDs from the tables 'Animes' and StudioIDs from the table 'Studio' to form a composite key. A csv file will be used to mass insert all the values required for this table</p> <p>1.5 There must be an 'AnimeGenres' table to split the many to many relationship between the table Animes and Genres. This table must contain all AnimeIDs from the tables 'Animes' and GenreIDs from the table 'Genres' to form a composite key. A csv file will be used to mass insert all the values required for this table.</p> <p>1.6 There must be an 'AnimeStatus' table to reduce repeating data in the main table 'Animes'. This table must contain all AnimeIDs from the table Animes and Status that was originally from the table 'Animes' to form a composite key. A csv file will be used to mass insert all the values required for this table.</p> <p>1.7 There must be a 'Users' table to store all Users information to that table. The table must contain the usernames the users decide to give themselves, UserIDs as a primary key, Forename, Surname and Emails (Emails will be used to reset password via entering the email to find the user's account in this program) and the User type of the user. All passwords must be hashed for security reasons</p> <p>1.8 There must be a 'UserFavourites' table that contains all UserIDs as a foreign key to link back to the table 'Users' and a unique primary key called 'UserFavID' to be assigned with the UserID, the table should also contain AnimeIDs, AnimeTitles and the user's respective rating of an Anime they have favoured</p> <p>1.9 There must be a 'UserWatchList' table that contains all the UserIDs as a foreign key to link back to the table 'Users' and a unique</p>	<p>animestudios, animegenres, animestatus through from their respective csv files and by using multiple foreach loops to bulk insert into the corresponding tables</p>	
--	--	--

<p>Primary key called 'UserWatchID' to be assigned with the UserID the table should also contain AnimeIDs, AnimeTitles, the number of episodes watched by the user and the user's respective rating of an Anime they have watched.</p>		
<p>2. If the database already exists the application must access it</p> <p>2.1 All the tables filled with values from their respective csv file must be in the database.</p>	<p>Objectives met. The program accesses everything after running and the tables are filled with everything from the csv files</p>	<p>None</p>
<p>3.1 There should be three buttons labelled 'Sign up', 'Log in' and 'Exit'.</p> <p>3.2 Once a button is pressed, the corresponding form should be displayed</p> <p>3.3 The menu form 'FmMenu' should close when another Form is displayed.</p>	<p>Objectives met. The form FmMenu contains 3 buttons labelled 'Sign up', 'Log in' and 'Exit'. Clicking these buttons shows the forms 'FmSignIn' and 'FmSignUp' and closes the form 'FmMenu'.</p>	<p>None</p>
<p>4. The Sign-up Form 'FmSignUp' must include a 'Return to Menu' button and a 'Save' button.</p> <p>4.1 The Sign-Up form must include all the required fields the user needs to enter to create an account through textboxes and the user must select the user type through a combo box with the options 'Administrator' and 'User'</p> <p>4.2 The User information that is entered by the user must be inserted into the table 'Users'</p> <p>4.2.1 The passwords entered by the user must be hashed before the insert</p> <p>4.2.2. User information must be inserted via a Save Button</p> <p>4.2.3 A message box must be shown to let the user know that they can Log in and it should also let the user know whether they need to change something within the fields.</p> <p>4.3 RegEx validation must be used for all the textbox in the Sign-up form to properly validate information</p> <p>4.3.1 It should let the user know what must be include in their Sign-up form E.g. "Your username must contain numbers" etc. through error messages when trying to save information</p>	<p>Objectives met. The forms 'FmSignUp' and 'FmSignIn' both contain buttons that will return the user to 'FmMenu'. 'FmSignUp' contains all required fields for the user to create an account, passwords are hashed upon insert and RegEx validation is used successfully and informs the user about what needs to be done. The creation of the accounts go into the table Users via insert statements in the program.</p>	<p>None</p>

<p>4.3.2 An error message must show up when the required fields are empty, or anything related to RegEx validation</p> <p>4.3.3 An error message must show up when the username that is entered by the user already exists in the database</p> <p>4.4 If the user selects 'Administrator' in the combo box User type it must show a textbox telling the user to enter an admin key</p> <p>4.4.1 An error message must be shown if the user entered the wrong key for Admin via clicking the save button</p> <p>4.5 Once everything is validated and meets the requirements from the validation the user should be able to save their information.</p>		
<p>5. The log in form 'FmLogIn' must include an 'Enter' button and a 'Return to Menu' button</p> <p>5.1 The Log in form must include all the required fields for the user to enter their Username and password</p> <p>5.2 The details entered by the user must be checked if the details exist in the table 'Users' through a data reader</p> <p>5.2.1 The password entered by the user must be hashed to match hashed password in the table 'Users'</p> <p>5.2.2 An error message must be shown if the password entered by the user does not match the password in the table 'Users'.</p> <p>5.2.3. If the data reader reads that the User's UserType is 'Administrator' then the Enter button must navigate the user to the Admin Form 'FmAMS (Admin)'</p> <p>5.2.4 If the data reader reads that the User's UserType is just 'User' then the Enter button must navigate the user to the Main Form 'FmAMS'</p>	<p>Objectives met. The form FmLogIn contains an 'Enter' button and a 'Return to Menu' button that allows the user to log in by matching input with a data reader and select queries to match the input of the user to that of the account in the table users. The data reader can also send the user to different forms based of their user type.</p>	<p>Create a function that can limit the amount of log in attempts.</p>
<p>6. The Admin Form must include a 'Go to MainForm' button to navigate the user to FmAMS</p> <p>6.1 The Admin Form must contain radio buttons that the admin can check to view all the tables in the database via a DataGridView Viewer</p> <p>6.2 The Admin Form must contain a textbox for searching stuff in the DataGridView Viewer.</p>	<p>Objectives Met. The admin form contains a 'Go to MainForm' button that navigates the user to 'FmAMS' The admin form also contains radio buttons that are name after the tables from the database. The form contains a search bar and a data grid viewer to show the tables</p>	<p>None</p>

7. The DataGrid viewer must be able to show the table the user wants by clicking on the radio button	Objectives Met. The datagridviewer can show all the tables by checking one of the radio buttons and clicking the 'show' button. This is done by using select queries.	None
8. The Admin Form must contain an 'Edit' button which allow Admin Users to navigate to the edit forms that correspond with the table they are trying to make changes to 8.1 The Edit Form must be used in conjunction with the radio buttons to view the tables in the database. Example: if the admin user selected the radio button "List all Animes" and then pressed the edit button the Edit Form for Animes will show up with empty fields.	Objectives met. The admin form 'FmAMS(admin)' contains an edit button that links to all the other edit forms by checking on one of the radio buttons and clicking the 'Edit' button.	Make it so that the edit forms are all in one forms through panels to reduce the amount of forms in the program
9. The Search bar must be able to refresh the data grid viewer and narrow down the tables in the DataGrid Viewer 9.1 The Whenever an Admin inputs a letter into the search bar the DataGrid Viewer should refresh with values that begin with that letter E.g. If the Admin clicked the radio button "List all Animes" and the grid shows the table 'Animes' the admin should be able type in a single letter and the grid will refresh with everything starting with that letter.	Objectives Met. The search bar refreshes the datagridviewer by using Select Like queries alongside a data adapter that ensures it refreshes the datagridviewer for every letter entered	None
10. The DataGrid Viewer cells should be clickable by the Admin. 10.1. If the Admin clicks on the cell, the Edit form should load up with all information gathered on the row where the cell has put them into text boxes.	Objectives met. The cells are clickable in the datagridviewer and the editform corresponding to the table loads up with all the values from the cell's row. This is done by making the columns global variables and using a for loop to loop the entire cell row and then reference it into the edit forms through the load function in all the edit forms	None
11. Admin Users should be able to add Animes through the edit Form that corresponds to Animes	Objectives met. The admin users can add Animes to the	Make it so that when the admin user enters an

<p>11.1 Adding Animes will be done through an SQL insert query</p> <p>11.2 The admin user would need to get the ID of the anime through the site called MyAnimeList. For Example:</p> <p>https://myanimelist.net/anime/48583/Shingeki_no_Kyojin_The_Final_Season_Part_2</p> <p>The admin user would insert the ID from the URL of the anime they want to insert</p> <p>11.3 The admin user will need to use that ID to make connections to different tables such as AnimeStudios, AnimeStatus, and AnimeGenres</p> <p>11.4 The Anime Synopsis and the Anime Picture for that anime will have placeholder values to replace them with the actual synopsis and picture for that anime using functions in the code it is however optional to manually input the synopsis and picture link for that anime.</p> <p>11.5 The admin user should be prompted with a message box saying that the anime has been added.</p> <p>11.6 The admin user should be prompted with a message box saying that the anime could not be added due to an error instead of crashing</p>	<p>table 'Animes' through the edit forms. This is done by using an insert query. The anime can be fully added in by entering the ID onto the edit form onto the edit form from the site myanimelist. The edit form also prompts the user with a message box informing the user that the Anime has been added to the table 'Animes' or if it hasn't.</p>	<p>AnimeTitle onto the edit form, the program checks the site my anime list and finds the ID and automatically fill the ID onto the ID text box. This is so that it is easier to add animes onto the database.</p> <p>Automatically add numbers to the AnimeSynopsis and AnimePictures textbox so that the admin user does not need to add the link and synopsis by themselves but rather let the program do it by accessing the anime through the Main form, or automatically fill the textboxes with the synopsis and anime picture link via entering the AnimeTitle so that adding an Anime is a lot easier</p>
<p>12 The admin User should be able to update Animes through the edit form that correspond to Animes</p> <p>12.1 Updating Animes will be done through an SQL Update Query</p> <p>12.2 Any part of the information that was loaded from the DataGridView can be updated BESIDES the ID for that anime as updating the ID will mess up the link with the other tables</p> <p>12.2.1 The update query will update everything besides the ID so whatever the admin enters the ID text box will be rendered useless through the update query</p>	<p>Objectives Met. The admin user can update anime information through the edit form. This is done by using update query. The edit form also prompts the user with a message box letting them know that the Anime has been updated or if it hasn't.</p>	<p>None</p>
<p>13. The admin user should be able to delete Animes through the edit form that correspond to the table Animes</p> <p>13.1 Deleting Animes will be done through an SQL Delete query</p>	<p>Objectives Met. The admin user can delete an anime through the edit form. This is done by using</p>	<p>Make it so that everything is deleted from the form 'FmEditAnimes'</p>

<p>13.1.1 To properly delete an Anime the admin user will need delete the links to that Anime E.g., the Admin must delete the AnimeIDs that match the anime in AnimeStudios, AnimeGenres and AnimeStatus</p> <p>13.2 The admin user must be prompted by a message box confirming that they have deleted an Anime</p> <p>13.3 The admin user must be prompted if the anime could not be deleted due to an error instead of crashing</p>	<p>a delete query. But in order to delete the Anime you would have to delete the following composite keys first, AnimeStudio, AnimeGenres and Anime Status.</p>	<p>By using multiple delete queries to delete all the composite keys and the anime.</p>
<p>14. The admin User should be able to add Genres through the edit Form that correspond to the table Genres</p> <p>14.1 The Admin User Should be able to assign a new Unique ID for GenrelID and input genres into the genre textbox</p> <p>14.2 The Admin User Should be prompted with a message box saying that the Genre has be added</p> <p>14.3 The Admin User Should be prompted with a message box saying that the Genre could not be added due to an error instead of crashing</p>	<p>Objectives met. The admin users can add genres to the table 'Genres' through the edit forms. This is done by using an insert query. The edit form also prompts the user with a message box informing the user that the Genre has been added to the table 'Genres' or if it hasn't.</p>	<p>Make it so that that the GenrelID is automatically assigned onto the genre via typing in the genre.</p>
<p>15. The admin User Should be able to update Genres through the edit form that correspond to the table Genres</p> <p>15.1 Updating Genres will be done through an SQL Update Query</p> <p>15.2 The admin user should be prompted with a message box saying that the genre has been updated</p> <p>15.3 The admin user should be prompted with a message box saying that the genre could not be updated due to an error</p>	<p>Objectives Met. The admin user can update genre information through the edit form. This is done by using an update query. The edit form also prompts the user with a message box letting them know that the Genre has been updated or if it hasn't.</p>	<p>None</p>
<p>16.1 Deleting Genres will be done through an SQL delete query</p> <p>16.2 The admin user Should be prompted with a message box saying that the Genre has been deleted</p> <p>16.3 The admin User Should be prompted with a message box saying that the Genres could not be deleted due to an error instead of crashing</p>	<p>Objectives Met. The admin user can delete genres through the edit form. This is done by using a delete query. The edit form also prompts the user with a message box letting them know that the Genre has</p>	<p>None</p>

	been deleted or if it hasn't.	
17 The admin user should be able to Add Studios through the edit form that correspond to the table 'Studios'. 17.1 Adding Studios will be done through an SQL insert query 17.2 The admin user should be prompted with a message box saying that the studio has been added 17.3 The admin user should be prompted with a message box saying that the studio could not be added because of an error instead of crashing	Objectives met. The admin users can add Studios to the table 'Studios' through the edit form. This is done by using an insert query. The edit form also prompts the user with a message box informing the user that the Studio has been added to the table 'Studios' or if it hasn't.	Make it so that the StudioID is automatically assigned onto the studio via typing in the studio.
18.1. Updating Studios Will be done through an SQL update query 18.2. The admin user should be prompted with a message box saying that the studio has been updated 18.3 The admin user should be prompted with a message box saying that the studio could not be updated because of an error instead of crashing	Objectives Met. The admin user can update studio information through the edit form. This is done by using an update query. The edit form also prompts the user with a message box letting them know that the Studio has been updated or if it hasn't	None
19. The admin users should be able to delete studios through the edit form that correspond to the table 'Studios' 19.1 Deleting Studios Will be done through an SQL delete query 19.2. The admin user should be prompted with a message box saying that the studio has been deleted	Objectives met. The admin user can delete studios through the edit form. This is done by using a delete query. The edit form also prompts the user with a message box letting them know that the Studio has been deleted or if it hasn't.	None
20 The admin users should be able to add AnimeStudios through the edit form that correspond to the table 'AnimeStudios' 20.1 Adding AnimeStudios will be done through an SQL insert query 20.1.1 The Admin user must use the AnimeID that they used to add a new anime, they must also use the site MytAnimeList to see the studio behind that anime and check the database if the studio exists if the studio	Objectives met. The admin users can add AnimeStudio composite keys to the table 'AnimeStudios' through the edit form. This is done by using an insert query. The edit form also	None

<p>does not exist then then the admin user would need to add the studio and use the StudioID from it to combine with the AnimeID</p> <p>20.2 The admin user should be prompted with a message box saying that the AnimeStudio has been added</p> <p>20.3 The admin user should be prompted with a message box saying that the AnimeStudio has not been added due to an error instead of crashing</p>	<p>prompts the user with a message box informing the user that the key has been added to the table 'AnimeStudios' or if it hasn't.</p>	
<p>21. The admin users should be able to update AnimeStudios through the edit form that correspond to the table 'AnimeStudios'</p> <p>21.1. Updating AnimeStudios will be done through an SQL update query</p> <p>21.2 The admin user should be prompted with a message box saying that the AnimeStudio has been updated</p> <p>21.3 The admin user should be prompted with a message box saying that the AnimeStudio has not been updated due to an error instead of crashing</p>	<p>Objectives Met. The admin user can update AnimeStudio composite keys through the edit form. This is done by using an update query. The edit form also prompts the user with a message box letting them know that the composite key has been updated or if it hasn't</p>	<p>None</p>
<p>22. The admin users should be able to delete AnimeStudios through the edit form that correspond to the table 'AnimeStudios'</p> <p>22.1 Deleting AnimeStudios Will be done through an SQL delete query</p> <p>22.2. The admin user should be prompted with a message box saying that the AnimeStudio has been deleted</p> <p>22.3 The admin user should be prompted with a message box saying that the AnimeStudio has not been deleted due to an error instead of crashing</p>	<p>Objectives met. The admin user can delete AnimeStudio composite keys through the edit form. This is done by using a delete query. The edit form also prompts the user with a message box letting them know that composite key has been deleted or if it hasn't.</p>	<p>None</p>
<p>23. The admin users should be able to add AnimeGenres through the edit form that correspond to the table 'AnimeGenres'</p> <p>23.1 Adding AnimeGenres will be done through an SQL insert query</p> <p>23.1.1 The Admin user must use the AnimeID that they used to add a new anime, they must also use the site MytAnimeList to see the Genres for that anime and check the database though the search bar in the Form if the Genre exists. If the genre does not exist, then then the admin user would need to add the Genre and use the GenreID from it to combine with the AnimeID</p>	<p>Objectives met. The admin users can add AnimeGenre composite keys to the table 'AnimeGenres' through the edit form. This is done by using an insert query. The edit form also prompts the user with a message box informing the user that the key has been added to the</p>	<p>None</p>

<p>23.2 The admin user should be prompted with a message box saying that the AnimeGenre has been added</p> <p>23.3 The admin user should be prompted with a message box saying that the AnimeGenres has not been added due to an error instead of crashing</p>	<p>table 'AnimeGenres' or if it hasn't.</p>	
<p>24. The admin users should be able to update AnimeGenres through the edit form that correspond to the table 'AnimeGenres'</p> <p>24.1 Updating AnimeGenres will be done through an SQL update query</p> <p>24.2 The admin user should be prompted with a message box saying that the AnimeGenres has been updated</p> <p>24.3 The admin user should be prompted with a message box saying that the AnimeGenres has not been updated due to an error instead of crashing</p>	<p>Objectives Met. The admin user can update AnimeGenre composite keys through the edit form. This is done by using an update query. The edit form also prompts the user with a message box letting them know that the composite key has been updated or if it hasn't</p>	<p>None</p>
<p>25. The admin users should be able to delete AnimeGenres through the edit form that correspond to the table 'AnimeGenres'</p> <p>25.1 Deleting AnimeGenres Will be done through an SQL delete query</p> <p>25.2 The admin user should be prompted with a message box saying that the AnimeGenres has been deleted</p> <p>25.3 The admin user should be prompted with a message box saying that the AnimeGenres has not been deleted due to an error instead of crashing</p>	<p>Objectives met. The admin user can delete AnimeGenre composite keys through the edit form. This is done by using a delete query. The edit form also prompts the user with a message box letting them know that the composite key has been deleted or if it hasn't.</p>	<p>None</p>
<p>26. The admin users should be able to add AnimeStatus through the edit form that correspond to the table 'AnimeStatus'</p> <p>26.1 Adding AnimeStatus will be done through an SQL insert query</p> <p>26.1.1 The Admin user must use the AnimelD that they used to add a new anime, they must also use the site MytAnimeList to see the Status for that anime they can then add the status to the textbox with the AnimelD to successfully insert</p> <p>26.2 The admin user should be prompted with a message box saying that the AnimeStatus has been added</p> <p>26.3 The admin user should be prompted with a message box saying that the AnimeStatus has not been added due to an error instead of crashing</p>	<p>Objectives met. The admin users can add AnimeStatus composite keys to the table 'AnimeStatus' through the edit form. This is done by using an insert query. The edit form also prompts the user with a message box informing the user that the key has been added to the table 'AnimeStatus' or if it hasn't.</p>	<p>None</p>

<p>27. The admin users should be able to update AnimeStatus through the edit form that correspond to the table 'AnimeStatus'</p> <p>27.1 Updating AnimeStatus will be done through an SQL update query</p> <p>27.2 The admin user should be prompted with a message box saying that the AnimeStatus has been updated</p> <p>27.3 The admin user should be prompted with a message box saying that the AnimeStatus has not been updated due to an error instead of crashing</p>	<p>Objectives Met.</p> <p>The admin user can update AnimeStatus composite keys through the edit form. This is done by using an update query. The edit form also prompts the user with a message box letting them know that the composite key has been updated or if it hasn't</p>	<p>None</p>
<p>28. The admin users should be able to delete AnimeStatus through the edit form that correspond to the table 'AnimeStatus'</p> <p>28.1 Deleting AnimeStatus will be done through an SQL delete query</p> <p>28.2 The admin user should be prompted with a message box saying that the AnimeStatus has been deleted</p> <p>28.3 The admin user should be prompted with a message box saying that the AnimeStatus has not been deleted due to an error</p>	<p>Objectives met. The admin user can delete AnimeStatus composite keys through the edit form. This is done by using a delete query. The edit form also prompts the user with a message box letting them know that the composite key has been deleted or if it hasn't</p>	<p>None</p>
<p>29. The admin users should be able to delete through the edit form that correspond to the table 'Users'</p> <p>29.1 Deleting User will be done through multiple SQL delete queries because the user will have data in the table 'UserFavourites' and 'UserWatchList'</p> <p>29.2 The admin user should be prompted with a message box saying that the user has been deleted</p> <p>29.3 The admin user should be prompted with a message box saying that the User has not been deleted due to an error instead of crashing</p>	<p>Objectives met. The admin user can delete Users through the delete form. This is done by using delete queries. The edit form also prompts the user with a message box letting them know that the User has been deleted or if it hasn't</p>	<p>None</p>
<p>30. The Main Form 'FmAMS' must contain a 'search bar', a 'search button', a 'profile button', clickable Anime Pictures and a group box containing Anime News</p> <p>30.1 If the profile button is clicked the then it will navigate the user to the profile form.</p> <p>30.2 The profile form must contain the user's information such as UserID, Username, Forename, Surname and their UserType</p>	<p>Objectives Met.</p> <p>The main Form 'FmAMS' contains a 'search button', a 'profile button', clickable Anime Pictures and a group box containing Anime News. The profile button</p>	<p>Create a function that can limit the amount of times the user can input their emails in the reset form.</p>

<p>30.3 The profile form must also contain a reset password button</p> <p>30.4 If the Reset password button is clicked then the button should navigate the user to the Reset Password Form</p> <p>30.5 In the Reset Password Form the user must be prompted to enter their Emails</p> <p>30.6 If the Email is entered then the email will be checked to see if it matches the email associated to the user in the Table 'Users'</p> <p>30.7 If the Email matches the user will see 2 textboxes appeared for them to type their new password and confirm their new password. To change passwords, they must press the 'change' button. When pressed the password will be hashed and added to the table the user can then close the form</p>	<p>navigates the user to the forms 'FmProfile' and. The form 'FmProfile' contains 'Reset Password' button which will navigate the user to the form FmReset. Where the can enter their emails to reset their passwords. A data reader is used to confirm that the email that the user entered matches the email in their account details in the table 'Users'. Changing passwords is done by using an update query</p>	
<p>31 In the Profile Form there should be a 'Favourites' button and a 'Watchlist' button</p> <p>31.1 if the user clicks on either of those buttons it should navigate the user to the Favourites form and Watchlist Form where they can update, delete their animes in their favourites table and watchlist table.</p> <p>31.1.1 This will be done using SQL Queries</p>	<p>Objectives met. The form 'FmProfile' contains a 'Favourites' button and a 'Watchlist' button which navigates the user to forms 'FmUserFavourites' and 'FmUserWatchList' In those forms the user can update the ratings of the anime in their favourites list as well as delete animes of their favourites list. The user can also update the ratings, the number of episodes watched and the watch status of the animes in their watch list and delete animes from their watch list. This is all done by using Update and Delete queries.</p>	<p>None</p>

<p>32. In the main Form the user should be able to see Anime news that refreshes in the group box</p> <p>32.1 Fetching AnimeNews will be done using a webscraper on the site animenewsnetwork</p> <p>32.2 It should be able to refresh and run forever.</p>	<p>Objectives met.</p> <p>The form 'FmAMS' shows news on the groupbox</p> <p>gbAnimeNews via webscraping news from the site animenewsnetwork.</p> <p>The news is refreshed every 4 seconds by using a delay function in the program</p>	<p>Make it so that the news is parsed from the website animenewsnetwork instead of webscraping the news so that the form can load up faster.</p>
<p>33. In the Main Form the user should be able to type in Animes and see a list of animes below the search bar</p> <p>33.1 This will be done using a function that contains SQL</p> <p>33.2 When the user clicks on the search button the Anime Picture and the title of that Anime Should be shown</p> <p>33.3 The anime picture will be retrieved using an API and then inserted into the animes database so that it could use the database to get the pictures instead.</p> <p>33.4 A panel will be used to show the searched Anime and a 'back' button will also be on the panel to take the user back to original state of the main form</p>	<p>Objectives met.</p> <p>The user can type in animes and see a drop down list that shows a list of animes based on the user input. This is done by an AutoFill function in the program which makes use of the Autocompletestringcollection function.</p> <p>The auto fill function also uses a select query to select all the anime titles from the table 'Animes' to then be added to the list. The search button shows the user the anime picture correspondent to the anime name they have typed in. This is done by using json parsing to get the anime picture from an API and manipulating the visibility of the panel to give the impression that the search bar is in use.</p> <p>The anime picture link is then added into the database via an update query where it updates the</p>	<p>None</p>

	place holder value in the table with the actual anime picture link.	
<p>34. In the Main Form the User should be able to click on the anime pictures and navigate them to the anime information form where all the information about the anime is on it</p> <p>34.1 The synopsis of an anime will be retrieved via an API and then inserted into the animes database and replace the place holder value. If the User were to click on that anime again after that then it must pull the synopsis out of the database instead of using an API.</p> <p>34.2 In the Anime Information form the user should be able to rate the anime, add to watchlist, add to favourites and type in the number of episodes they have watched for that specific anime</p> <p>34.3 An aggregate SQL query will be used to validate the add to favourites button and add to watchlist button and make sure that it is only inserted once into the tables 'User Favourites' and 'UserWatchList'.</p>	Objectives met. Clicking an anime picture will navigate the user to the form FmAnimeInformation where all the information about the anime is retrieved from the database and is then outputted via the load function. The synopsis is fetched by using json parsing that gets the synopsis of the anime from an API and then loads the synopsis into the textbox in FmAnimeInformation. The synopsis is then added into the table 'Animes' via an update query where it updates the place holder value in the table with the actual synopsis.	None
35. Add a statistics button in the profile form that will navigate the user to the statistics form that will show them their average ratings across their favourites list and watchlist, their total ratings, the amount of animes they're currently watching, the amount of animes yet to watch, animes dropped, total episodes watched, and hours spent watching	Objectives met. The form 'FmProfile' contains a 'stats' button which will navigate the user to the form 'FmUserStats' which will then show the user the following stats:average ratings across their favourites list and watchlist, their total ratings, the amount of animes they're currently watching, the amount of animes yet to watch, animes dropped, total episodes watched, and hours spent watching. This	None

	is done by retrieving information from the tables User, 'UserFavourites' and 'UserWatchList'. This is done by using multiple aggregate queries.	
36. Add DarkMode to the main Form	Objectives met. The form 'FmAMS' has button labelled as 'Switch' which allows the user to change the form into dark colours by a click.	Make it so that clicking the button will affect the colours of all forms.
<u>EXTENSION</u> 1. Add a recommendation feature that recommends the user animes based on the genres of the animes they have favourited and watched	Objective not met. Time Constraint	
<u>EXTENSION</u> 2. Add categories for animes based on genres and studios	Objective not met. Time Constraint	

Feedback

“As a person who’ watched a few animes, I found this NEA program to be very cool to use for me as it provides a table showing animes that are currently available within the database, it shows in information about animes in detail and I was surprised to see how well managed this program is in terms of security and useability, everything is organised in a pleasing way and generally its very responsive and pleasing to use. I love the addition of the dark mode in the main form as it visually looks so good on the eyes at night as well as that, I love how I can search up any anime and see its picture. I also like how I can rate animes and favourite them too, I’ve tried many ways to break this program, but the developer knows what he’s doing and has implemented exception handling on in almost every form I tried to break. For people who are into animes more than I, I can see how insightful this program will be for them.”

→ Faheem Saleem – A level Computer Science student

Analysis of User Feedback

I am glad that the program has been useful for finding animes and adding them to their favourites. The interface was liked by the user especially the dark mode function due to how useful it is. Faheem has tried to break my program to reveal flaws, but due to my exception handling and various other validations Faheem had very little success. However, despite the positive feedback I still think that there are lots of space for improvement such as adding animes into the database, I could make this simpler by getting the ID of the anime upon the

user inputting the anime title and the layout of the form 'FmAnimeInformation' could look better.

Overall Assessment of the project

Overall, I am satisfied with the quality of this application- I met the essential criteria and targets. I had initially set for myself in the objectives list. The user requirements from the analysis section were also met; this is evident from the user feedback. This application has been useful in finding out new animes to watch by putting them in my watchlist and it is also useful knowing how much hours I have spent watching animes. This application is proven to be practical and can help others find out new animes.

Completing the project in this given time frame was challenging; normalising the database and coding the SynopsisApi function, Search functions and the news function were the most difficult parts of my programming as I had to learn and understand how to webscrape and how to parse json files. It took a few attempts before getting everything right and I had to broaden my knowledge on normalisation as my previous attempts at normalising this database were incorrect and the sheer size of the database also made it harder to normalise as I had to find out which parts of the table had repeating data and which part had partial key dependencies in order to make new tables. I also had to broaden my knowledge on SQL throughout this project as I faced multiple errors from my SQL statements which were hard to pinpoint but over time it became easier to correct.

If I had more time, I would enhance parts of my codes based on what I listed in the possible 'improvements and enhancement objectives' in order to make my program more user friendly. The extension objectives were ambitious and would've required a lot longer given the time frame. This project has made me a better programmer overall due to how hard this project was to me.