Anime Recommendation System

PERSONAL PROJECT

Developer: Moller Rodrigues
Email: mar7147@outlook.com
Git: https://github.com/ShuZer0
Stack Overflow: https://stackoverflow.com/users/9073951/moller-rodrigues

Moller Rodrigues mar7147@outlook.com

Table of Contents

Moller Rodrigues mar7147@outlook.com

ANALYSIS

Project Overview and stakeholders

Problem explanation

Comment [MR1]: Provided an outline of what my problem is

I intend to develop a desktop based program, which will generate recommendations of animes for users to watch. The users that will be using my program are people who watch anime and are looking for more animes to watch.

Animes are a style of Japanese film and television animation; they are aimed at all ages but most typically adults and teenagers. The first anime was produced in 1961 and since then there has been over 20,000 Animes produced, with a huge variety of genres from your common 'western' genre's such as action to unique Japanese genre's such as Slice of life and Harem. As a result of the huge number of Animes available, just like with movies it has become difficult for users to browse and find Animes which are similar to an Anime they have previously seen or one that meets their preferences. Therefore, I intend to make an Anime recommendation system which will aid the user's in this process by calculating the similarities between thousands of animes, which of course can be done must faster and more accurately by a computer than any human.

Target audience

Comment [MR2]: Identified all the stakeholders

Unsurprisingly, the audience for Anime is vast ranging from toddlers to the elderly, across all genders. So as to ensure my program is appropriate and appeals to the majority of users who watch anime I will be using a sample of 4 clients who personify the typical anime watcher and will be frequently interviewed in order to gain an insight on their needs and wants and also to test the program throughout development.

Clients

Comment [MR3]: Stated each client and their role

The 4 clients I have chosen:

• **Joshua Santillan** (lead client) – Has been watching anime for 7 years, 25 years old, employed

Role: Joshua will be evaluating the functionality of the program (Efficiency, code quality, robustness, validation …) In addition, as the lead client he will sign off my proposed solutions on behalf of the other clients.

• **Skylar De Souza** – Has been watching anime for less than a month, 16 years old, student

Role: Skylar will handle the form of the program in other words the aesthetics of the user interface.

• **Thomas Wojenski** – Has been watching anime for about 3 months, 18 years old, student

Role: Thomas will provide the underlying requirements for the program as well as feedback throughout development.

• **Hannah Arifi** – Has been watching anime for 2 years, 17 years old, Student

Role: Hannah will be mainly involved in testing the program and in particular evaluating the generated recommendations in order to assess the extent to which they are similar to the desired factors given by the user.

Current observations

Comment [MR4]: Explained the current methods that users are using to find anime recommendations.

Currently, my clients and others who watch anime are manually browsing through huge lists of anime's in order to find one to watch or one that is similar to their tastes. This current method is extremely time consuming, as for each anime the user has to read its description and it is difficult for them to come to a decision solely by doing this as they are unable to view trailers for most Animes, especially if you are outside of Japan.

How my program will solve this problem

Comment [MR5]: Explained and justified how my program will meet the needs of the user

However, the introduction of my recommendation system will greatly benefit the existing users as not only will it save them time to find similar Animes but it will also save them money as it would greatly reduce the chances of a user purchasing or renting an anime that was not suitable or appealing to their preferences. Furthermore, to ensure my program is suitable for users who watch anime, my clients will be able to test out my program by watching the recommended Animes which are generated for them, in order to see whether they are actually similar to their preferences.

Computational methods

Comment [MR6]: Justified why the problem can be solved by computational methods.

The Anime recommendation system is best suited to be solved by a computer program as computers can use algorithms to process and analyse data much faster than a human and so this will be beneficial in analysing a huge variety of Animes and their features in order to generate recommendations. In order to breakdown this problem I will be using computational thinking and methods, the applications of which are covered below.

Thinking abstractly and visualisation

Abstraction is the process of separating details from reality; removing unnecessary details and highlighting relevant details. Abstraction can be used to solve my problem as the application of abstraction would greatly reduce my codes size and complexity by identifying what does and does not matter to solving the problem.

A major feature of abstraction within the process of finding anime recommendations is the removal of the user/human intervention – In reality if a user was trying to find similar animes to ones that they previously watched, then they would have to manually check that animes genres, description and reviews themselves and then compare these with other animes, one at a time. As you can tell this process is extremely time consuming and boring however,

through the use abstraction we can take out the human involvement and replace it with a computer. Using a computer can greatly reduce the time a user would have to spend looking for recommendations as it can process the similarity between thousands of animes in a matter of seconds.

Furthermore, instead of the user having to physically search and find the features of anime in order to find similar ones, through the use of abstraction we can make it so that the user just has to enter the anime they wish to find the recommendation for and let the computer handle the rest. This way we have reduced what was potentially a several step process to merely one and input (what anime the user wants to find similar recommendations from) and one output (the animes similar to that input).

Thinking ahead

Thinking ahead is the process of considering beforehand what the solution of the problem will look like, what conditions it will work under and what processes it will need to function. To give me an insight of the solution I need to consider what the inputs and outputs of my system are.

Data required for recommending an Anime:

- Database for Anime's – contains records like anime ID (INTEGER), anime name (STRING), genre (STRING), episodes (INTEGER) and average rating (INTEGER)…
- Database for Ratings – contains records such as user ID (INTEGER), Anime ID (INTEGER) and Rating (INTEGER).

The database for Animes will be used to provide features of Animes so that they can be used in the content based recommendation system in order to compare with the user's preference to generate the best matches; Whereas, the Ratings database will be used mainly for the collaborative filtering system, where it will take other user's ratings of Animes in order to generate Animes that the user searching for Animes may like as well.

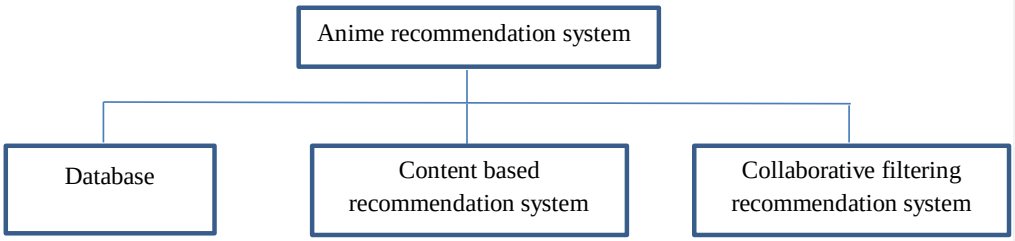
INPUT	PURPOSE	DATA TYPE
User's preferences	Contains a list of features that the user prefers in an Anime. E.g. Action, Romance, Tragedy…	Dictionary
User's Animes	Contains Animes that the user has previously watched and liked. These will be used to find similar Animes by finding users who liked the same Animes as the user and browsing what other Animes that user liked in hopes to find an Anime that will appeal to the user's tastes.	Dictionary

OUTPUT	PURPOSE	DATA TYPE
Recommended Animes	This will be a list of recommended Animes generated by the content based system and the collaborative filtering system.	Array (strings)

Thinking Procedurally

Through the process of thinking procedurally and decomposition I will be able to break down a large problem into smaller manageable sub problems which most often can be broken down into functions, which can then be combined to solve the whole problem.

System decomposition as a whole:



Through thinking procedurally I have broken my system down into 3 main modules database, content based and collaborative filtering system. Breaking down the system in such a way will allow not only allow me to simplify a complex problem into smaller solvable sections but will also allow me to test my program more easily and efficiently.

Thinking logically

The process of thinking logically will allow me to identify decision points for branching or iteration in my program, which will be very beneficial when it comes to writing the code for the program and assessing the complexity of the algorithms I may use. Furthermore, through the application of thinking logically I will be able to identify the points at which a decision is needed, determine the conditions of the decision and determine the outcome depending on the decision made. For example, one decision point in my program would be what kind of Animes does the user like? So the conditions to determining this decision would be, what genre's does the user like and what Animes has the user rated highly in the past. According to the outcome of the decision I will then pass these Animes into the content based and collaborative filtering systems in order to generate recommended Animes for the user to watch.

Thinking Concurrently

Thinking concurrently is the process of tackling different parts of the problem at the same time. In the case of my problem, thinking concurrently can be applied to the task of calculating which animes are the most similar to the user's preferences. In order to calculate the similarity I would have to check each anime in the database against the user's preferences. However, the method of doing would be the same for each anime and so I can write one main function which can be iteratively applied to all the animes in the database at the same time, thus saving time and making my code more efficient. Furthermore, the process of creating a screen can be copied and re=used for each of the screens I intend to make.

Research

Current Observations

Comment [MR7]: Explained current methods in more detail and justified why my program will be better than the current methods in solving this problem.

Currently, my clients and others who watch anime are manually browsing through huge lists of anime's in order to find one to watch or one that is similar to their tastes. This current method is extremely time consuming, as for each anime the user has to read its description and it is difficult for them to come to a decision solely by doing this as they are unable to view trailers for most Animes, especially if you are outside of Japan.

However, the introduction of my recommendation system will greatly benefit the existing users as not only will it save them time to find similar Animes but it will also save them money as it would greatly reduce the chances of a user purchasing or renting an anime that was not suitable or appealing to their preferences. Furthermore, to ensure my program is suitable for users who watch anime, my clients will be able to test out my program by watching the recommended Animes which are generated for them, in order to see whether they are actually similar to their preferences.

Existing recommender algorithms

Comment [MR8]: Research of existing recommender systems with explanations

A recommendation system is a subclass of information filtering system that seeks to predict the rating or preference that a user would give to an item. Recommender systems have become increasingly popular and are utilized in a variety of fields including movies, music and social networking. The two most common types of recommender systems are content based and collaborative filtering systems.

Content based recommendations involve analysing what the users preferences are and the features of the anime. For example, if a user preferred an Action anime where the setting is in a school then Animes which have the most features that match the user's preferences will be generated as a recommendation. On the other hand, collaborative filtering analyses similar users and what Animes they prefer. For example, if the user looking for a recommendation had previously enjoyed watching Anime 'x', and another user who also enjoyed watching anime 'x' liked anime 'y' then anime 'y' may be a suitable recommendation for the user.

Moller Rodrigues mar7147@outlook.com

Some major examples of the use of recommender systems would be Netflix, Facebook, Amazon and Google. Each of these companies uses a recommendation system to predict what a user may want. For example Facebook uses a recommendation system to give user's suggestions on friends they may want to add and Amazon uses a recommendation system to advertise products to users that they may be interested in purchasing. However, the most similar existing solution to my problem would be Netflix's recommendation system. Netflix uses a hybrid of both content based and collaborative systems in order to recommend films and TV shows to users.

Below I have conducted the research of two similar existing apps, Netflix and myanimelist.net:

Netflix research

Comment [MR9]: APP 1 research (Netflix)

Netflix feature analysis

Comment [MR10]: APP 1 (Netflix) analysis with description, and justification of whether the feature was good/bad

FEATURE	DESCRIPTION	PROS	CONS
Interface layout	The layout and form in which the Netflix website is displayed.	Netflix had a clear and simple interface layout this is because there was not an abundance of widgets or colours but only what was essential such as headings, required buttons and images of films in a linear layout. Having a clear and clean interface layout is important as it will make it easier for the user to navigate through the application.	N/A
User Profiles	Feature which allows the user to create their own accounts.	Netflix's use of user profiles was also a good feature as it allows the app to keep track of data about a specific user. In other words, in order to get recommendations the user would not have to re-enter all their preferences every time they launch the app instead by having a user profile all they have to do is login and the app will import their saved preferences from a database.	N/A
Images for movies	Cover images for each movie/TV show.	Netflix also used images to display movies instead of just text of the name of the movie. I found this to be a good feature as it makes the interface less messy and easier to understand rather having a bunch of text on the screen which would actually look pretty dull.	Storing images for each anime in the database can take up a lot of space and it is not essential to the solution of my problem.
Browsing Tool	A tool which allows the user to manually search for entries using text or selecting genres.	The browsing tool was also another good feature as it allows users to find movies by a more specific way in which the user can search by name or genre. This is beneficial as it can also be implemented in my app to	N/A

		allow users to browse for animes instead of getting recommendations.	
Recommender system	The algorithm that Netflix uses to generate their recommendations.	This is the main feature of the Netflix in relation to my app. Netflix's recommender system is good as it takes into account what the user has already watched and based on that it generates recommendations which are similar. This is beneficial as it is the core function of my app.	N/A
Subscription	Netflix require their users to pay a subscription fee in order to use their system.	Raises revenue for the developers, allowing them to re-invest into the app or develop better services.	The recommender system of Netflix is sub product of the main purpose of Netflix which is to stream media. Therefore, users cannot just use Netflix's recommendation system without paying a subscription fee. This is inconvenient as Netflix does not provide a huge variety of animes and so it cost inefficient to use Netflix to get anime recommendations.
Does not have a large database of Animes	Despite Netflix having a lot of movies and TV shows they do not however have a lot of animes and this can restrict the capabilities of their recommender system.	N/A	Netflix is a media streaming site and although it has thousands of films and TV shows it does not in fact have a lot of animes. This means that the anime recommendations provided by Netflix are limited as they can only recommend animes which are in their database.

Based on the analysis of Netflix's features, together with considerations of feature compatibility, I have decided upon the following requirements:

Comment [MR11]: Brief requirement summary of APP 1 (Netflix analysis)

Basic Requirements:

- User profiles
- Browsing tool

- Recommender System

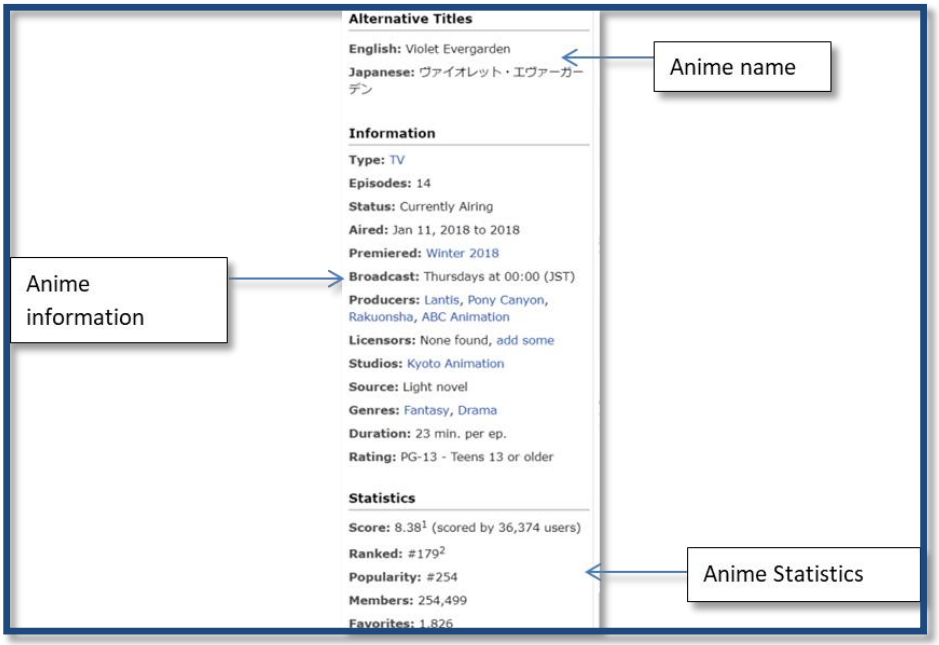
Optional features:

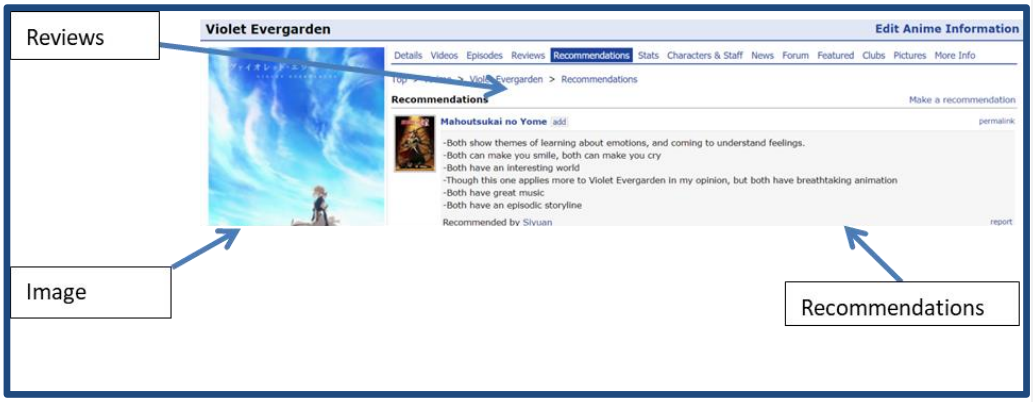
- Images for animes
- Anime database should be updated frequently

Myanimelist.net research

Comment [MR12]: APP 2 RESEARCH (myanimelist.net)

My animalist.net is a social network for anime fans. MyAnimeList provides users a customizable 'listing' feature to show other users what anime or manga they have watched or read.





Myanimelist.net feature analysis

Comment [MR13]: APP 2 (MyAnimeList) analysis with description, and justification of whether the feature was good/bad

FEATURE	DESCRIPTION	PROS	CONS
Anime information	Provides information for each anime. For example, their genre, producers, release date…	Anime information, such as anime name, genres and episodes can be used to determine the similarity between animes. In other words, this information can be used in my recommender systems.	Information such as producers, release data, Japanese name are irrelevant to the solution of my problem,
Images	MyAnimeList provides images for each anime	Images add vibrancy to the website, making to more appealing to users.	Including images for each anime in my app will increase the size of the app and so users will require more memory.
Anime statistics	Statistics on each anime	This feature is extremely beneficial as computing similarities is much easier using integers and stats such as average score will allow us to recommend not only similar animes but similar animes with high ratings as well.	N/A
Anime Reviews	Reviews for animes written by users.	Gives other users, a better insight to an anime rather than the insight they receive just by reading the description of an anime. Reviews can also be parsed through to match reviews from other animes.	Reviews are the opinion of an individual and so it may not represent the general consensus. Parsing through dozens of reviews, which contain thousands of characters, is very inefficient, memory wise.
User recommendations	Anime recommendations given by other users.	Recommendations from users who have watched both animes may provide a better explanation as to why these animes are similar rather than an arbitrary number which represents similarities.	These recommendations are also just the opinions of an individual and so they may differ from person to person. Furthermore, a user looking for recommendations is still going to have to read through all the recommendations, which can be very time consuming. This is also dependent on the fact that a user has actually provided a

			recommendation, in the case of an anime where there are no recommendations from users, this feature may not beneficial.

Based on the analysis of MyAnimeList's features, together with considerations of feature compatibility, I have decided upon the following requirements:

Comment [MR14]: Brief requirement summary of APP 2 (MyAnimeList analysis)

Basic requirements:

- Anime statistics – rating, members
- Anime info – Name, Genre, Episodes, Type
- Recommender system

Optional features:

- Images
- Anime reviews

Interview with Thomas Wojenski (Client)

Comment [MR15]: Interview with my client Thomas Wojenski which is his main role.

Moller Rodrigues (Me) – Hey there Thomas, pleasure to meet you. Today I'm going to be conducting an interview with you in order to gain some insight on what the underlying requirements that this project should have.

So firstly, how do you currently look for animes to watch?

Thomas Wojenski – Hey Moller, glad to be of assistance. So currently, my method of finding animes to watch is very time consuming. Using sites like MyAnimeList, I manually browse through their database of animes trying to find something that may appeal to me. Furthermore, for people like me who have watched most of the popular, 'mainstream' animes it is even more difficult to find animes which are similar to one we have previously seen.

Moller Rodrigues – And so do you think that my program which will find recommendations will help address the current problem you are facing?

Thomas Wojenski – Yes, I think having a program that can generate dozens of recommendations in a matter of seconds compared to manually browsing myself will greatly aid users who are in the same predicament as me.

Moller Rodrigues – I completely agree with that, so moving on, other than the core feature of the recommender system, what other features would you like the app to have?

Comment [MR16]: Free flowing discussion like an ordinary interview, whereby I am asking questions, getting answers from Thomas but also expanding on his answers myself in order to gain more insight.

Thomas Wojenski – I think the app should have a feature which allows the user to save their data, such as preferences and animes they have already seen, so that they wouldn't have to input it into the program every time they want to find a recommendation.

Moller Rodrigues – So, do you mean users should have their own accounts which can be used to store their respective data so that it can be loaded every time the user logs in to the app?

Thomas Wojenski – Yes, that's exactly what I intended to say. This would not only be more convenient for users but can provide the opportunity to add additional functionality to the app.

Moller Rodrigues – What do you mean by additional functionality?

Thomas Wojenski – Erm, I feel like the app could include some sort of extra 'fun' feature, such as anime quotes or trivia which randomly appear throughout the app and perhaps we can you the accounts idea to store user's score just to make it a bit more competitive and appealing to users.

Moller Rodrigues – I think that's a great idea, it will make our app more comprehensive. What's your view on including a live chat for users?

Thomas Wojenski – Well, yeah it's indeed a good feature in theory, but I don't think we should implement a live chat. I say this because; the anime community is much greater and more active on sites like MyAnimeList and Discord and so instead of fighting a losing battle, in trying to get users to switch to our platform to communicate. We should just stick to what we specialise in, which is recommendations. Instead we could provide a link from our app to discussion sites like MyAnimeList and so both our apps can work together in satisfying users.

Moller Rodrigues – Wow, that some detailed feedback on the features side, so let's move on to the User Interface. How would you like to app to function and what would you like the User Interface to look like?

Thomas Wojenski – Firstly, the first screen the user should see when they launch the app is the login screen. The login screen should be very concise and simple. What I mean by this is, it should only include the bare essentials such as the apps logo, username/password entry and login/register buttons. I also think there shouldn't be any extra labels for the entry boxes and instead we should put the label of the entry fields as default text within the fields in order to keep it clean and simple.

As for styling, I think the app should have a consistent colour scheme throughout the entire app and images where possible in order to add vibrancy and appeal to the app. In addition, as the app is anime related it should consist of some images and fonts relating to anime.

Once the user logs in, there should be some sort of option menu enabling them to select whether they want to find a recommendation, change their preferences or browse the database for animes, etc. Continuing with the theme of 'lean and mean', the recommendation screen should provide a tool for the user to select an anime for which they would like to get a

recommendation for similar animes from. The output for the generated recommendations could be displayed on the same screen but I think that may be a bit messy. So in order to make everything look clear, I think the output should be displayed on a pop up window.

The preferences/ profile option should display a screen that provides widgets for the user to select and view their preferences. In addition, it would be beneficial to add help notes/instructions for users in case they are struggling to use the app, or coming across problems.

Moller Rodrigues – Well, that concludes our interview today. Thanks for your time and I will definitely take on board your feedback and convey to my lead client.

Analysis of initial interview

Comment [MR17]: Analysis of the interview with Thomas Wojenski

Overall, the interview with Thomas to my surprise provided some insightful feedback which I had not received from researching similar apps. Below is an extraction of some the keys points that Thomas raised, in my opinion.

- RECOMMENDER RESPONSE TIME – Thomas mentioned that my app would greatly reduce the time for his current method and so I think the response time of my recommender algorithm should take no more than 50 – 60 seconds in the worst case. Therefore, my aim will be to have the response time lower than 40 seconds.
- USER ACCOUNT SYSTEM – Thomas suggested that users should have their own accounts to allow them to store their data.
- EXTRA 'FUN' FEATURE – Some sort of extra mini game or trivia that users can entertain themselves with whilst using the app.
- LINK TO ANIME DISCUSSION SITES – Instead of implementing our own live chat, it was suggested that we include links to anime discussions sites which already have large active communities, instead of trying to compete with them.
- CONSISTENT STYLING – Consistent colour scheme throughout the entire app, formatting, dimensions of widgets.
- ANIME THEME – images, fonts and quotes relating to anime should be included in the development of the interface, in order to increase appeal with users and create a sense of familiarity.
- SCREENS – login screen, recommendation scree, profile screen.
- INSTRUCTIONS – help notes for users who don't know how to use the app or who are encountering problems.

Questionnaire

Comment [MR18]: Questionnaire with 20 random user's who watch anime

Based on my app research and initial interview with one of my clients I have formed a set of questions to ask a sample group of 20 anime fans.

Would you like users to have accounts?

- Yes [17]
- No [3]

What extra 'fun' feature would you like to be implemented into the system?

- Anime trivia questions [4]
- Anime quotes [7]
- Anime Pictionary (Guess anime from images) [9]

Should the app include links to anime discussion sites?

- Yes [15]
- No [5]

If yes, what sites?

- MyAnimeList [14]
- Discord [5]
- Crunchyroll [1]

Which colour scheme would you like to see?

-  [13]
-  [6]
-  [1]

Should we provide descriptions for each anime?

- Yes [4]
- No [16]

Analysis of questionnaire

Comment [MR19]: Analysis of questionnaire

The general consensus of the questionnaire resonates with the analysis of my initial interview with Thomas. However this interview has enabled me to go in more depth and get the opinions of more users rather than just one individual.

Requirements specification

Comment [MR20]: A list of requirements that my app will include along with a description and justification for each feature.

Based on my research on Netflix's recommender system, MyAnimeList, questionnaire and the interview with Thomas, I have taken on-board all the feedback I had received and have come to the conclusions of certain requirements.

Below I have listed each feature along with their description and why I have chosen to include them:

FEATURE	DESCRIPTION	JUSTIFICATION
Content based Recommender system	This is the algorithm that will be used to generate recommendations. It works by calculating the similarities between animes using observations and features.	This feature is compulsory at it is the main solution to my problem which is how to find recommendations for animes. Using a content based recommender system will solve this problem as it will calculate similarities between animes and using these similarities we can recommend animes which are the most similar compared to each other.
Response time	The time taken for the recommender system to compute through the anime dataset and yield recommendations.	From my research I have found that finding recommendations is possible to do manually, however this is time consuming. Therefore, for my app to be effective it must yield a faster response time than the current methods. My aim for the response time is in the worst case 40 seconds, as we computing with thousands of animes.
User accounts	Each user should have their own account which will be used to store their preferences to a database.	Having user accounts will save users from having to re-enter their preferences every time they use our app. (as suggested by Thomas Wojenski)
Profile	Feature that allows users to add animes they watched, their favourite animes and favourite genres.	This will aid in providing recommendations automatically to users rather than having them manually input animes for which they wish to find similar animes to. In addition, they can share their preferences with other users so as to help each other to find animes to watch.
Browsing tool	Tool users can use to find animes from the database	This will allow for a broader search, enabling users who wish

	via anime name, genre or type.	to find animes that meet certain genre criteria.
Instructions	Notes for users who are encountering difficulties/problems	This feature will make our app more user friendly and can help prevent users having misunderstanding as to how to use the app.
Anime Pictionary	Mini game where users have to guess the anime name based of an image of said anime.	Including an extra feature such as a game like this greatly improves the app's appeal to users and can help build a friendlier user environment through the sense of familiarity by implementing a game related to anime. Furthermore, from my research it has been concluded that such a feature is would be highly demanded and appreciated.
Anime discussion sites	Embedding links to anime discussion sites within the app.	The anime community like to converse between each other sharing their opinions and giving advice on recommendations. Therefore, instead of implementing our own discussion feature, it was suggested to me to link more popular anime discussion sites such as MyAnimeList. This way, users can experience a bigger and more active community which will enable them to discuss the recommendations they received from my app and evaluate or perhaps even recommend them to other users.
Colour scheme		Colour scheme that should remain consistent throughout the entire app, as chosen by popular opinion from questionnaire.
Anime theme	Anime related fonts, images and quotes embeded genorously throughout the app.	Having an anime related theme to the app, will create an emotional connection to users, creating a more familiar and friendly environment. Furthermore, it will bring vibrancy and uniqueness to my app.

Requirements Client Review

Comment [MR21]: Client feedback

The clients are pleased with all aspects of my proposed solution apart from using solely content based system. Their reason for this is that using both content based and collaborative based systems will yield a more accurate recommendation. However after much discussion, we have come to the conclusion that we will carry on with content based as our main aim but if and when we are under development and find that we have the time and capability to implement the collaborative filtering system then we shall do so.

Josh Santillan, lead client.

Features I Intend to use

Feature 1 – Account system

Description – The account system will allow the user to create an account, login to an account and store the user's preferences.

Variables	DESCRIPTION	DATA TYPE
Username	Unique identifier set by the user	String
Password	Password set by the user	String
Preferences	User's anime preferences (what animes they like, what genres they like…)	Dictionary

Justification – An account system is needed for the solution of my program as without a method of saving a user's preferences they would have to input them every time they launch the program which is extremely inconvenient.

How does this feature make the problem solvable by a computational approach?

My research has concluded that the implementation of an accounts system is entirely possible using a computational approach. While researching different methods of creating and manipulating databases, I decided to use SQL integrated with Python. One of the main reasons I chose to use SQL in python is because I am familiar with Python's syntax therefore, it will be much easier to develop the accounts system using both languages.

Research links:

https://www.youtube.com/watch?v=ngynJQ0iVwM

Feature 2 – User preferences

Description – The user preferences feature will allow the user to input their preferences in order to find animes that suit them.

Variables	DESCRIPTION	DATA TYPE
FavAnimes	A list of the user's favourite animes	Array
Genres	A list of the user's favourite genres	Array
Types	A list of what types of anime the user prefers. (TV, Movie or special)	Dictionary

Justification – This feature is essential to the solution of my problem as without knowing what the user likes and wants in anime there is no way to accurately recommend an anime that the user will like.

Comment [MR22]: Identified the features of my proposed solution. Stating each feature with a description, justification and how it can be solved by a computational ap[proach.

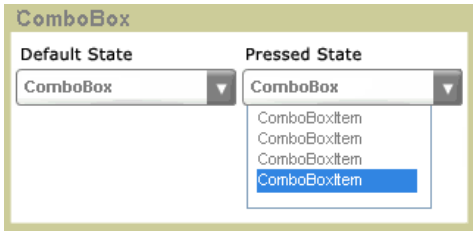
Comment [MR23]: Further justification of how my program can be solved by a computational approach.

Comment [MR24]: I have linked all the sites that I used to research for existing solutions/methods/code.

How does this feature make the problem solvable by a computational approach?

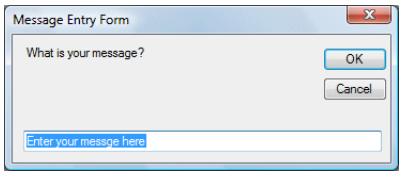
My research has concluded that the implementation of a user preference feature is indeed possible through the use of input widgets. Using the Tkinter library within Python it is possible to use many input widgets such as combo boxes and entry widgets in order to get inputs from the user.

Combo box:



The image to left represents a drop down selection of pre-set values. This would be useful in selecting champions as it would prevent invalid inputs.

Input box:



An input widget is also a viable option, but this would require validation checks as to if the user has inputted the champions name correctly.

Research links:

https://www.tutorialspoint.com/pyqt/index.htm

https://pythonspot.com/en/pyqt5-input-dialog/

Feature 3 – Get Rec (get recommendation)

Description – The get rec feature will generate anime recommendations for a user given their preferences.

Variables	DESCRIPTION	DATA TYPE
UserPreferences	A list of the users preferences(FavAnimes, FavGenres …)	Dictionary
RecAnime	A recommended anime generated from the content based algorithm	String

Justification – This is the most important feature of my program as it is literally solving the problem of recommending animes for a user to watch.

How does this feature make the problem solvable by a computational approach?

As mentioned earlier in my research the two most common recommender algorithms are content based and collaborative filtering. In the case of my project we have decided to go for a content based algorithm which will generate recommendations by checking against the features of an anime that the user likes and other animes.

Research links:

https://www.youtube.com/watch?v=2uxXPzm-7FY

Feature 4 – Browsing Tool

Description – The browsing tool will allow the user to browse through the anime database by anime name and/or genre and/or type.

Variables	DESCRIPTION	DATA TYPE
Genres	List of genres selected by the user	Array
Anime name	Name of anime entered by the user to search for	String
Anime type	Whether the anime is a movie, TV show or special episode.	String

Justification – This is one of my client requirements as this feature will aid users who want to find a specific anime or just want to browse through a huge range of animes of a certain genre combination.

How does this feature make the problem solvable by a computational approach?

My research has concluded that the implementation of a browsing tool is possible through the use of querying using SQL and using Pythons Tkinter radio button widgets for the user to select genres.

Radio buttons:

I am considering the use of radio buttons as again it will avoid any errors in the case of the user entering an invalid response; however they are also efficient for inputting values from a small selection, as there are only three valid responses.

Furthermore, with the integration of Python code for the functionality, it is possible to return the value as string which I can then use in my item calculation algorithm.

Research links:

http://zetcode.com/gui/pyqt5/widgets/

Limitations

Comment [MR25]: Identified limitations that my system may have with descriptions and justifications for each feature.

By nature, my project will have limitations as it is an A level project with limited time and resources. Some features I think my program will not have are:

LIMITATION	DESCRIPTION	JUSTIFICATION
Descriptions for each anime	This feature is where each anime has a description explaining what the anime is about.	Although this is obviously a useful feature the user can just simply find a description by looking up the anime on the internet. However, if I was to implement this feature it may take up a lot of time, although it is not impossible to implement.
Collaborative filtering recommender system	Collaborative filtering analyses similar users and what Animes they prefer. For example, if the user looking for a recommendation had previously enjoyed watching Anime 'x', and another user who also enjoyed watching anime 'x' liked anime 'y' then anime 'y' may be a suitable recommendation for the user.	The implementation of the algorithm will involve using machine learning so as to predict what ratings a user would give an anime and calculate the 'nearest neighbour' or in other words the most similar anime. Therefore, this will involve me having to learn some machine learning syntax whether that will be using the machine learning libraries that python provide such as tenser flow and Scikit learn.

I would just like to reiterate that these limitations are not set in stone and if throughout development there are resources available or the clients decide to prioritise these features then implementation of this features is definitely possible.

Requirements

Comment [MR26]: Software and Hardware requirements with justification

The system will be written using Python 3.6, and so as a guide the hardware and software requirements of the system should be similar to those necessary to run Python 3.6 or greater, together with the hard disk space to store the program itself. I have obtained the minimum requirements needed to run Python 3.6 from the official Python website. The additional hardware requirements of the program will be: A keyboard and a mouse so that the user can input data and interact with the programs interface and a network connection so they can access web content from our app.

Software Requirements	Justification
Windows 7 or later, Mac and Linux OS	I will be using Python 3.6.1 which does not support windows XP or earlier versions of windows
Python version: 3.6.X	I will be using Python as my programming language along with its extensive set of libraries. Therefore, the user will need Python to run my app.

Hardware Requirements	Justification
200 MB of hard drive space	The user will need hard drive space as they would need to store the anime database, accounts database, images and the program executable itself.
Minimum of 32mb RAM	In order to run Python which requires 32 MB + of RAM
VGA or higher resolution monitor; Super VGA recommended	Needed to display the Graphical User Interface
OS compatible Mouse	Needed for user to navigate through interface
OS compatible keyboard	Needed for the user to provide inputs such as their login details and anime preferences.
Network connection	My app will include links to external websites; therefore an internet connection will be required to access these.

Comment [MR27]: Success criteria for proposed solution with justification. Furthermore I have included success criteria which can indeed be demonstrated during testing.

Success Criteria

CRITERIA	JUSTIFICATION
Main Menu	The program should have a menu screen which will allow the user to navigate through to the different parts of the app. The Menu screen should be minimalistic yet consistent with the overall theme of the program.
Efficient and functional recommender system	The core solution to my problem is the recommender system, which will literally generate anime recommendations for users. Therefore for this project to be successful the recommender system that I am to implement must be efficient to a certain degree of accuracy. In order to evaluate the efficiency and accuracy of my algorithm I can make use of a confusion matrix which will test the model's correct and incorrect predictions based on actual recommendations which have been verified as good recommendations from multiple users.
User accounts system	The app must have an account system in place whereby users can create and login to their accounts. In extension to the user accounts, their accounts must be linked to a database to allow users to save their data and retain this data on their next login. I have chosen this as a success criterion as it was not only suggested by multiple clients but also by the sample group of end users.
User preferences	The program will need a feature which will allow the user to input their preferences and these preferences should be stored permanently to the accounts database for that respective user. Also, these preferences should be passed to the recommender system so that it can generate recommendations. This is needed in order for the project to succeed as it will give users more excludability and will allow them to get more recommendations that meet their preferences. Furthermore, this was a requirement that both my clients and end users agreed on implementing.
Browsing tool	From my app research it has come to my attention that it is common for apps such as mine to include a browsing feature which allows the user to browse the database more extensively compared to a name search. Therefore, following the norm and in order to make the app more convenient for users I have decided to implement a browsing tool.
Response time	As discussed with my clients and users, the current methods to finding anime recommendations are extremely time consuming. Therefore, in order for my app to be considered as an effective replacement to the current methods, the response time to generate recommendations should be fairly quick. However, taking into consideration that the algorithm will have to compute through thousands of animes I have come to the decision that the response time should be less than 40 seconds in order for the app to be considered effective and efficient.
Link to anime discussion sites	From my observations and questionnaire, it has been clear that anime fans like to converse about their opinions and aid in giving recommendations to others. And as there are large and active anime fan communities already out there like MyAnimeList I decided that the final app should have embedded links to such discussion sites so that users who use my app can evaluate on the recommendations that were generated for them and maybe even perhaps recommend these animes to other users.
Anime theme with colour scheme set 1	As chosen by popular demand from my research the app should have a consistent colour theme of set 1 throughout the entire app. In addition, users wanted to see images and fonts relating to anime integrated into the app. Therefore, in order to meet user requirements and increase the vibrancy and appeal of my app I have decided that the final project should meet these styling conditions.
Anime Pictionary	All my clients and end users were in the opinion of having an extra 'fun' feature to make the app more unique and interesting. So, in order to meet their needs the final app should have a mini anime Pictionary game, where users have to guess the name of the anime based

	on an image of said anime.

Client review - The clients are on board with all these features and have made it clear that the final product must have achieved these core criteria's anything else is secondary or an addition to the program. However, in addition to these features I have been told by the clients that not all users are native English speakers and so I have been told to include a help feature which will be in English, French, Spanish and German. The help feature will instruct the users on how to use the program.

Comment [MR28]: Client feedback

Josh Santillan, Lead Client

DESIGN

Comment [MR29]: Design section

TOP DOWN MODULAR DESIGN – SYSTEM DIAGRAM

Comment [MR30]: Broken the problem down systematically into a series of smaller problems.

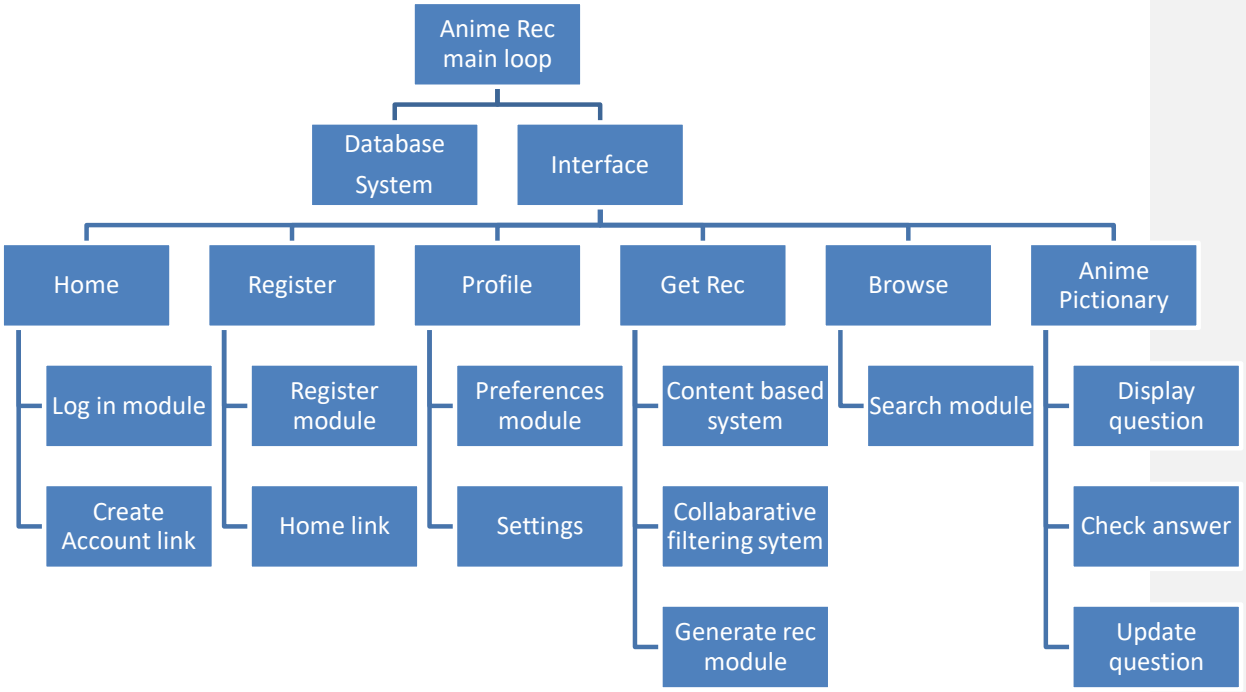


Diagram breakdown

Below I have explained each section of the system diagram above in detail and have shown how each section works together to complete the whole program.

Database System

This module involves the loading of my databases in order for them to be used during other processes. I will be using 3 databases which are as follows:

1. ANIME DATABASE:

SUCCESS CRITERIA MET - The anime Database will contribute to creating the recommender system, browsing tool and the Pictionary game.

FIELDS	DESCRIPTION	DATA TYPE/EXAMPLE DATA
Anime_id	Unique numerical value given to each anime	(INTEGER) e.g. 32281
Name	The name of each anime	(STRING) e.g. 'Gintama'
Genre	The genre of each anime	(STRING) e.g. 'Action, Drama, Shounen'
Type	The type of anime	(STRING) e.g. 'TV, Movie, OVA, Special'
Episodes	Number of episodes	(INTEGER) e.g. 24
Rating	Average rating given by users out of 10	(FLOAT) e.g. 9.37

This database will be used in both the content based algorithm and the collaborative filtering algorithm in order to generate recommendations. It will also be used in the search module which will allow users to look up Animes manually.

2. USER RATING DATABASE:

**LIMITATION POINT ** - This database will be used if we have the capability and resources to implement a collaborative filtering based recommender system.

FIELDS	DESCRIPTION	TEST DATA
User_id	Unique numerical value given to each user	(INTEGER) e.g. 1, 5, 123, 1474
Anime_id	Unique numerical value given to each anime	(INTEGER) e.g. 32281
Rating	Rating given by each individual user for a given anime	(FLOAT) e.g. 4, 5.5,9

This database will be used in the collaborative filtering algorithm as it will provide the ratings for Animes given by multiple users.

Comment [MR31]: Broke down each section of my systems top down diagram, explaining and justifying each section.

Comment [MR32]: Database system section which consists of all the databases that I may need

Comment [MR33]: Referred to the success criteria for each section where I have met a criteria

Comment [MR34]: Test data for the databases

3. ACCOUNTS DATABASE:

SUCCESS CRITERIA MET - The accounts database will contribute to achieving the user accounts system and the user preferences criteria.

FIELDS	DESCRIPTION	TEST DATA
Username	Unique name set by a user for said user	(STRING) e.g. 'ShuZer0'
Password	Password set by a user	(STRING) e.g. 'socks21'
First name	user's first name	(STRING) e.g. 'John'
Surname	user's surname	(STRING) e.g. 'Doe'
Preferences	user's preferences	(STRING) e.g. 'Action, School, teenagers, Sport'

This database will be used to store user accounts which have been created and their preferences in order to generate recommendations.

NOTE - The data for the anime and user rating database will be derived from an anime community website - https://myanimelist.net/modules.php?go=api, using their API. The data provided is open source and will consist of over 20,000 Animes and over 128,000 users.

Object orientated approach

Comment [MR35]: Explained my OOP approach

In order to create my program I will be using an object orientated approach. Object orientated programming involves creating solutions using objects that interact with each other. In order to create objects a template known as a class is used. Classes have attributes (variables associated with an object) and methods (functions that the object carries out). The act of creating an object using a class is known as instantiation.

I chose to use an object orientated approach over a procedural approach due to 3 main features: Inheritance, Encapsulation and Polymorphism. Inheritance allows derived classes to inherit the attributes and methods of super classes as well as having their own attributes and methods; furthermore instances of the same class can share variables between them. Encapsulation prevents attributes of a class from being manipulated by external methods and classes, therefore upholding the integrity of the system; attributes can only be altered using methods of their respective class. Finally, OOP allows objects to be process differently depending on their data type or class through the feature of polymorphism.

Below in the continued diagram breakdown of my system, I have constructed each screen as a class stating their attributes and methods along with their descriptions.

Interface

Comment [MR36]: Interface section with
- Description + Justification
- Class diagram
- Explanation of each attribute and method
- (NO TEST DATA REQUIRED FOR INTERFACE SECTION)

The interface module will act as a control unit which will link all the individual screens together. The interface module is essential as it will allow each screen and their respective functions to interact with other screens (Share variables and methods, change screens). The interface will consist of 6 screens: Home, Register, Profile, Get Rec, Browse and Pictionary. Using OOP the interface module can be modelled as a class.

Class Interface
Container: frame Screens: windows
ChangeScreen ()

Attributes:

- Container – This is a frame inside a window which will be used to hold the screens.
- Screens – These are window objects created by their respective classes (Home class, Profile class…)

Methods:

- ChangeScreen () - This method will be used to show the current screen by placing it at the top of the Container.

(The screen at the top of the container is the screen that will be displayed)

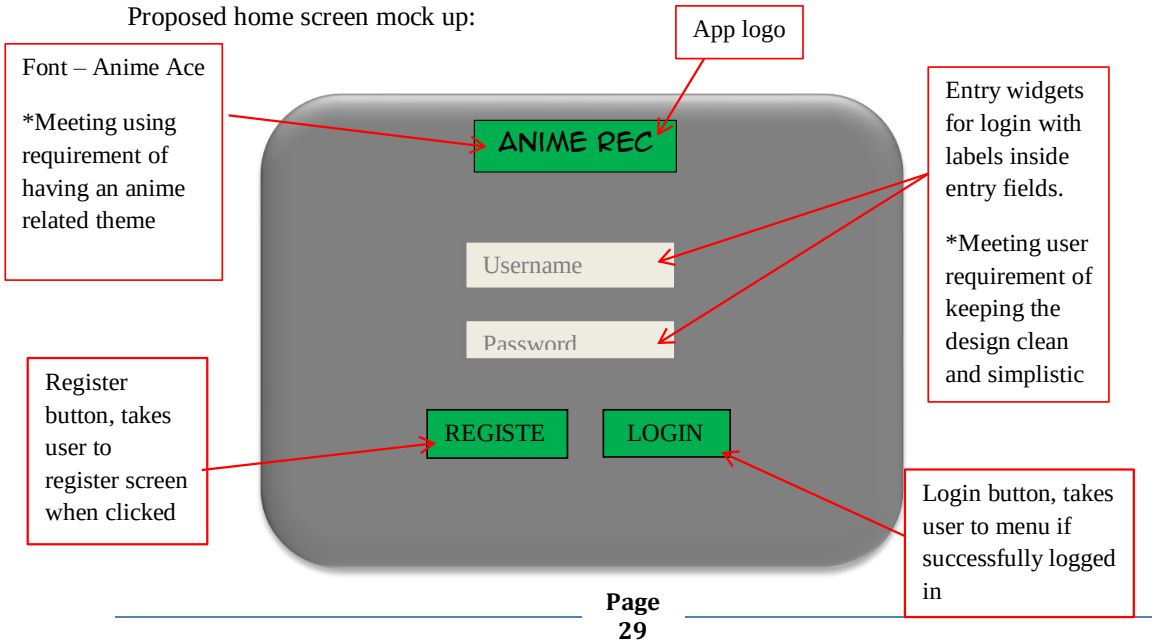
Home

Comment [MR37]: Home section with
- Description + Justification
- Success criteria reference
- Annotated screen mock up
- Class diagram
- Explanation of each attribute and method
- Validation
- Test data

SUCCESS CRITERIA MET - Contributes to accomplishing the user accounts system criterion as it will create the function which will allow users to login.

The home screen will be the first screen displayed to the user, and will consist of a login function and a function to create an account.

Proposed home screen mock up:



DEFAULT STYLE (used for all screens)

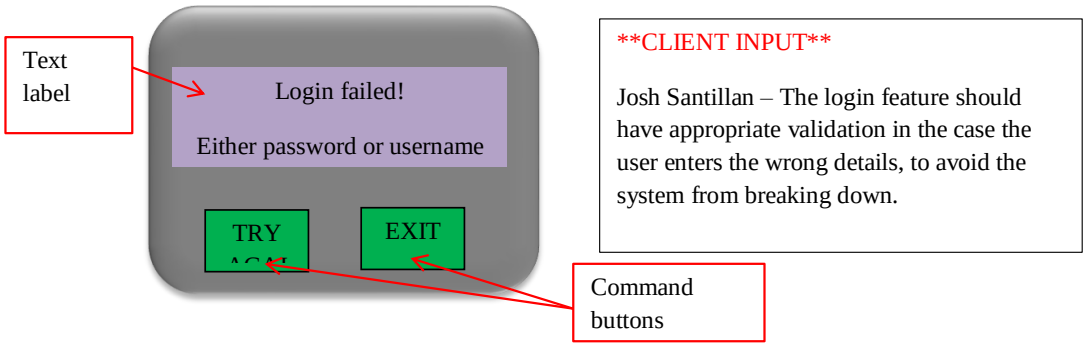
Screen Size: 600x350

Large font: Anime Ace

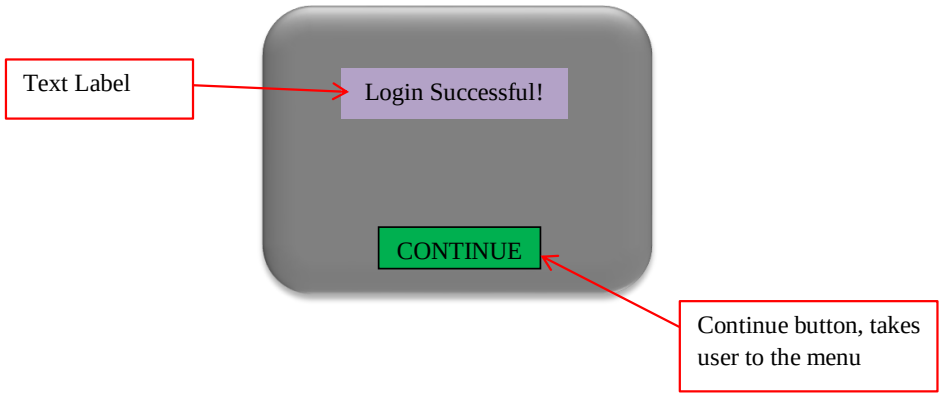
Small font: Calibri (Body)

CLIENT REQUIREMENT - From the interview I conducted the clients wanted a matching colour scheme and an anime themed interface. Therefore I have chosen an anime themed font and gone with a black, grey and green colour

Home login validation failed prompt:



Home login validation success:



Class Home
Text – Label widgets InputFields – Entry widgets LoginButton – Button widget RegisterButton – Button widget
Login ()

Attributes:

- Text – This is all the text on the Home screen and they will be displayed using label widgets. (Title of the screen, username label, password label)
- InputFields – These are entry boxes which will allow the user to enter their login details. (Username entry and password entry)
- LoginButton – This is a button widget which when clicked will execute the login ().
- RegisterButton – This button will display the Register

Methods:

- Login () – This function will take the username and password entered by the user and check the Accounts Database if the details are valid. If login details are valid then the Menu screen will be shown or if not then a prompt will be shown asking the user to try again. Hence validation will be needed here.

HOME SCREEN TEST DATA

TEST DATA	EXAMPLE	TYPE	DATA TYPE
Enter password with 0 characters	“”	INVALID	STRING
Enter username with 0 characters	“”	INVALID	STRING
Enter username with more than 0 characters	“snow24”	VALID	STRING
Enter password with more than 0 characters	“helloWorld!”	VALID	STRING
Select Login button/ Register button (LMC)	LMC	VALID	N/A
Select Login/Register button with any keyboard/mouse input other than LMC	ENTER key	INVALID	N/A

Register

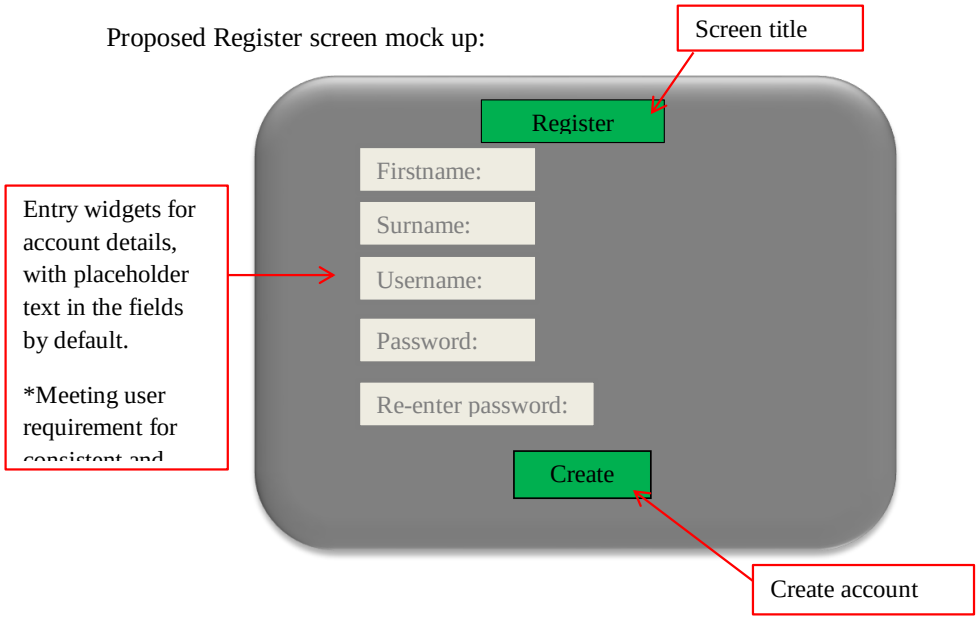
Comment [MR38]: Register section with
- Description + Justification
- Success criteria reference
- Annotated screen mock up
- Class diagram
- Explanation of each attribute and method
- Validation

Test data

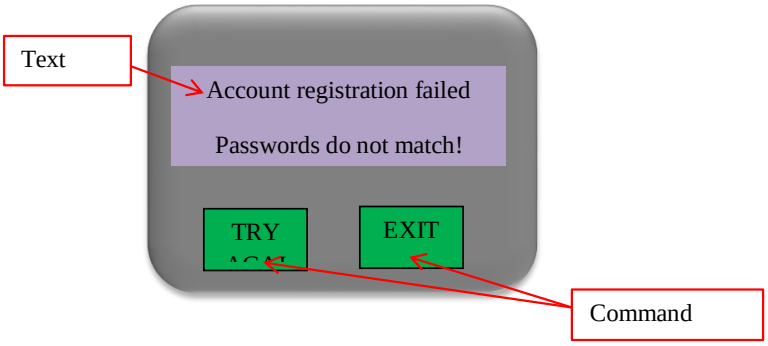
SUCCESS CRITERIA MET - Contributes to accomplishing the user accounts system as it will allow users to create accounts.

The register screen will allow the use to create an account, so that they can login and use the program.

Proposed Register screen mock up:



Register screen validation fail prompt:



Class Register
Text – Label widgets InputFields – Entry widgets CreateButton – Button widget
CreateAcc ()

Attributes:

- Text – This is all the text on the register screen and they will be displayed using label widgets. (Title of the screen, username and password labels)
- InputFields – These are entry boxes which will allow the user to enter their login details.
- CreateButton – This button will return the values entered in the entry boxes and execute the CreateAcc()

Methods:

- CreateAcc () – This function will take the username and password entered by the used and add it to the Accounts Database, thus creating a new account.

REGISTER SCREEN TEST DATA

DATA	EXAMPLE	TYPE	DATA TYPE
Enter Name with more than 0 characters	"John"	VALID	STRING
Enter Surname with more than 0 characters	"Smith"	VALID	STRING
Enter Password with more than 0 characters	"ruby24"	VALID	STRING
Enter Username with more than 0 characters	"John"	VALID	STRING
Enter Name with 0 characters	""	INVALID	STRING
Enter Surname with 0 characters	""	INVALID	STRING
Enter Username with 0 characters	""	INVALID	STRING
Enter Password with 0 characters	""	INVALID	STRING
Select Create button (LMC)	LMC	VALID	N/A
Select Create button with any keyboard/mouse input other than LMC	ENTER key	INVALID	N/A

Main menu

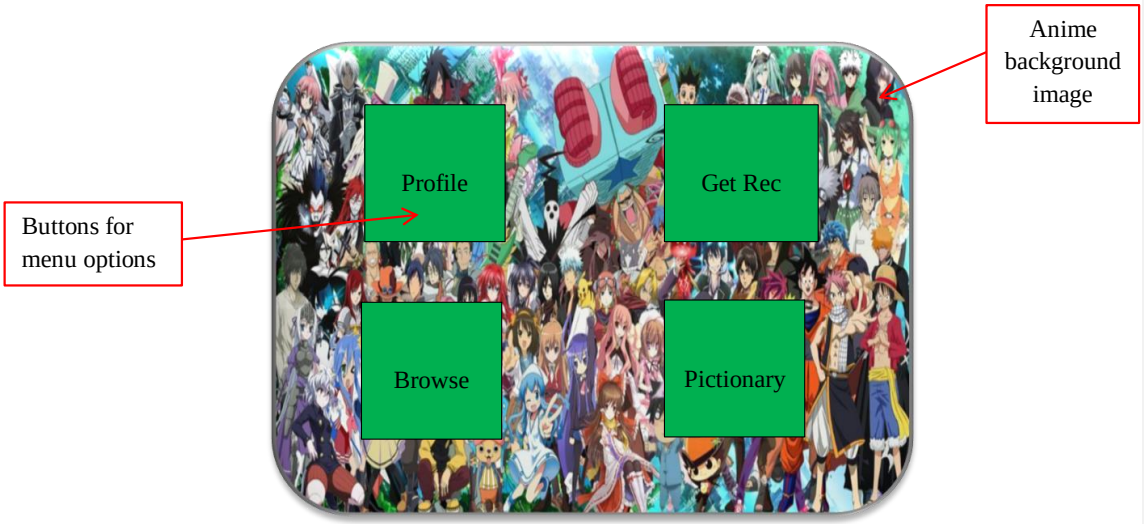
Comment [MR39]: Menu section with
- •Description + Justification
- •Success criteria reference
- •Annotated screen mock up
- •Class diagram
- •Explanation of each attribute and method
- •Validation

Test data

SUCCESS CRITERIA MET - Main menu criterion met

The main menu screen will be displayed once the user has logged in. It will consist of four options, Profile, Get Rec, Browse and Help.

Proposed Main menu screen mock up:



Class Main Menu
Profile: Button widget GetRec: Button widget Browse: Button widget Pictionary: Button widget
ButtonCommand ()

Attributes:

- Profile Button – When clicked this will display the Profile Screen, which will allow the user to enter their preferences.
- GetRec Button – When clicked this will display the GetRec Screen which will generate anime recommendations specific to the user's preferences.
- Browse Button – When clicked this will display the Browse screen which will allow the user to browse through the anime database by genre, name or type.
- Pictionary Button – When clicked it will display the mini anime Pictionary game, where the user will be shown an image of an anime and options of answers to select from.

Methods:

- ButtonCommand () – This function will display a screen respective to the button which was clicked. For instance if the Profile button was clicked then the ButtonCommand () will display the Profile screen.

MENU TEST DATA

TEST DATA	EXAMPLE	DATA TYPE
Has the Profile Button been clicked?	True = show profile screen False = Pass	BOOLEAN
Has the Get Rec Button been clicked?	True = show GetRec screen False = Pass	BOOLEAN
Has the Browse Button been clicked?	True = show Browse screen False = Pass	BOOLEAN
Has the Pictionary Button been clicked?	True = show Pictionary screen False = Pass	BOOLEAN

Get Rec

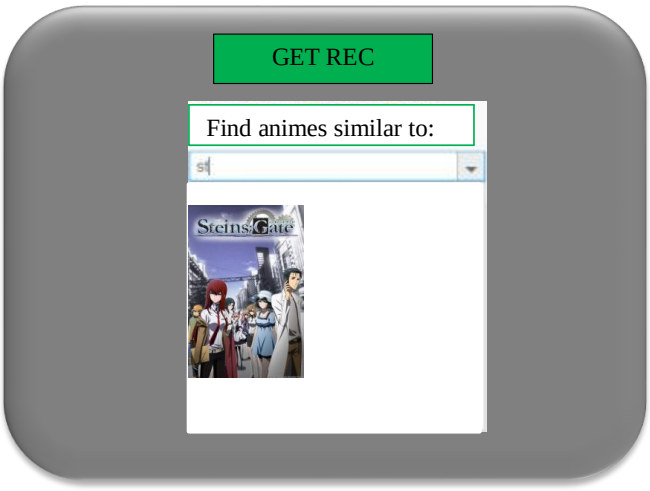
Comment [MR40]: Get Rec section with
- Description + Justification
- Success criteria reference
- Annotated screen mock up
- Class diagram
- Explanation of each attribute and method
- Validation

Test data

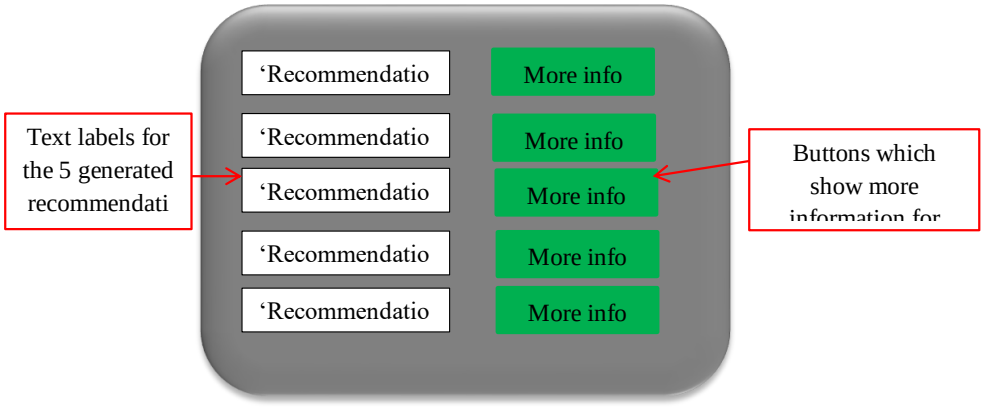
SUCCESS CRITERIA MET - Integrates the recommender system with the GUI, hence achieving the efficient and functional recommendation system criterion point.

The Get Rec screen will be used to generate recommendations and output these to the user.

Proposed Get Rec mock up:



Get rec anime recommendation output screen:



Class Get Rec
SimilarAnime – Combo box widget Recommendation – Pop-up window
CBSystem ()

Attributes:

- SimilarAnime – This combo box will give the user the options for all animes in order for them to select an anime for which they wish to find similar animes to.
- Recommendation – This is a pop-up window that will display the recommended anime that was generated by the recommender system. (as sketched in the diagram above)

Methods:

- CBSystem () – This is the content based recommender system which will use the user's preferences and check against each feature of every anime in the database to generate

GET REC SCREEN TEST DATA

TEST DATA	EXAMPLE	TYPE	DATA TYPE
Enter anime name to find a recommendation for	"Kaze no Stigma"	VALID	STRING
Enter anime name to find a recommendation for	"Kane na stigma"	INVALID	STRING
Select anime to find recommendations for from search results (double LMC)	{Double LMC}	VALID	N/A
Select anime to find recommendations for from search results (any other input other than double LMC)	{Enter}	INVALID	N/A

Moller Rodrigues mar7147@outlook.com

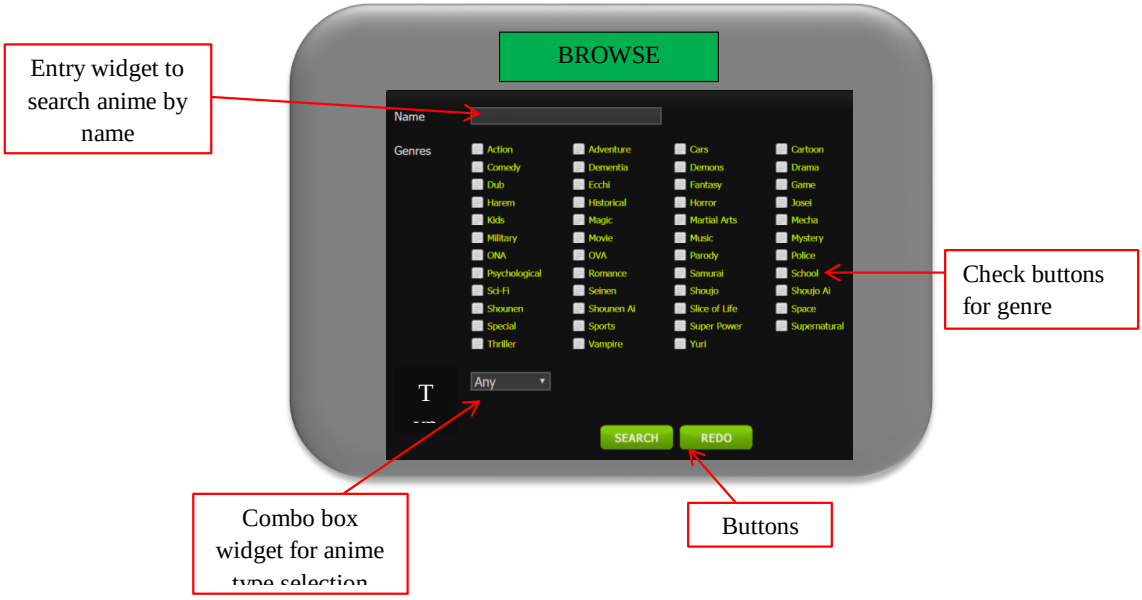
Browse **SUCCESS CRITERIA MET** - Browsing tool

Comment [MR41]: Browse section with
- Description + Justification
- Success criteria reference
- Annotated screen mock up
- Class diagram
- Explanation of each attribute and method
- Validation

Test data

The browse screen will allow the user to browse through the database of animes.

Proposed Browse screen mock up:



Class Browse
Genres – check button widgets Name – Entry widget Type – Combo box widget
SearchDatabase ()

Attributes:

- Genres – These will be a collection of check buttons for each genre available. This will allow the user to search animes by genres.
- Name – this is an entry widget which will allow the user to search animes by name.
- Type – This is a combo box which will allow the user to select what type of anime they wish to search for.

Methods:

- SearchDatabase ()- This function will take the data entered by the user and query the anime database based on the user's input and return the suitable output.

BROWSE SCREEN TEST DATA

DATA	EXAMPLE	DATA TYPE
Genres	["Action", "Comedy", "…"]	ARRAY containing STRING elements
Name	"Attack on Titan"	STRING
Type	"TV", "Movie", "OVA"	STRING

Pictionary

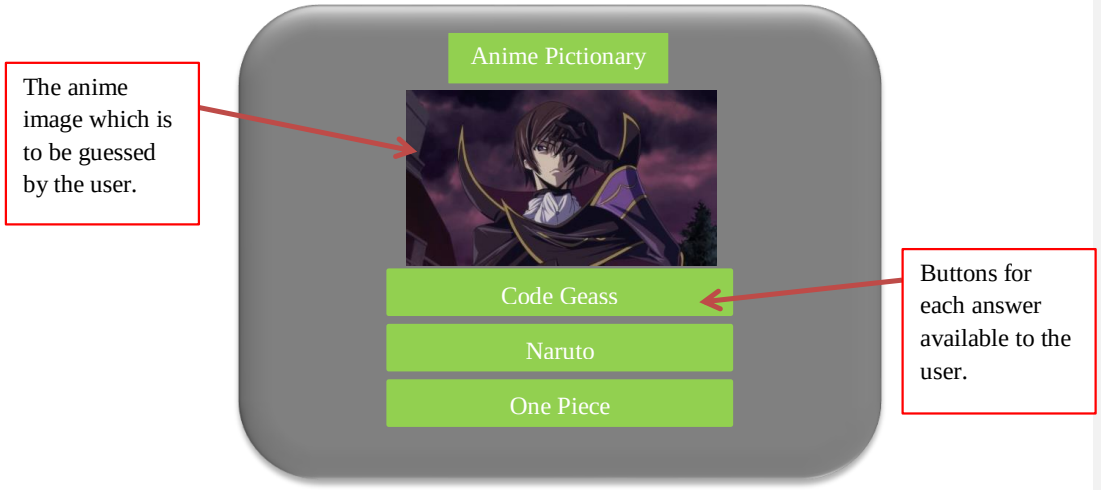
Comment [MR42]: Pictionary section with
- Description + Justification
- Success criteria reference
- Annotated screen mock up
- Class diagram
- Explanation of each attribute and method
- Validation

Test data

***SUCCESS CRITERIA MET** - Anime Pictionary*

The Pictionary screen will display an anime Pictionary game where the user has to guess the anime name from the image shown.

Proposed Pictionary screen mock up:



Class Pictionary
Answers – Button widgets AnimeImage – Image file (PNG)
CheckAnswer ()

Attributes:

- Answers – A set of buttons for each answer available for the user to choose from.
- AnimeImage - Image of the anime to be guessed.

Methods:

- CheckAnswer () - Takes the answer that the user clicks on and checks whether it is correct. If it is correct then score is incremented and next question is shown, else if answer is incorrect then game over.

Moller Rodrigues mar7147@outlook.com

ANIME PICTIONARY SCREEN TEST DATA

TEST DATA	EXAMPLE	DATA TYPE
correctAnswer	"Code Geass"	STRING
wrongAnswer	"Naruto"	STRING

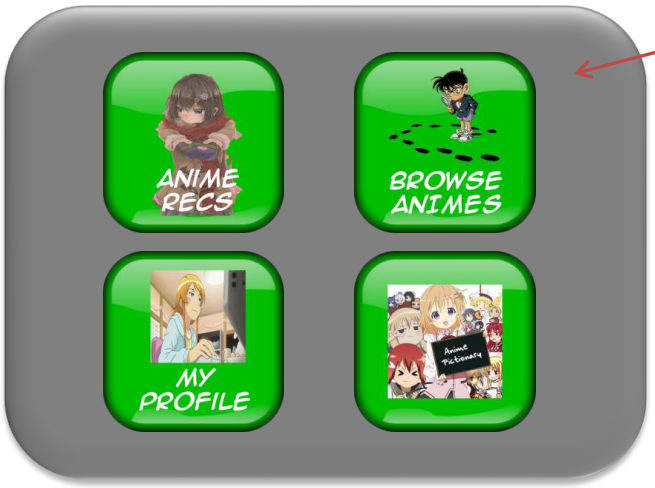
Design mock ups Client feedback

Comment [MR43]: Client feedback of screen mock ups

My clients were pleased with all the proposed screen mock ups apart from the menu screen. The initial menu screen mock-up was said to be too much and that it was not consistent with the rest of the app. Therefore I have I created another design for the menu screen below which has been approved by my clients.

Revised Menu screen

Comment [MR44]: Improvement based on client's feedback



New custom made

Image buttons

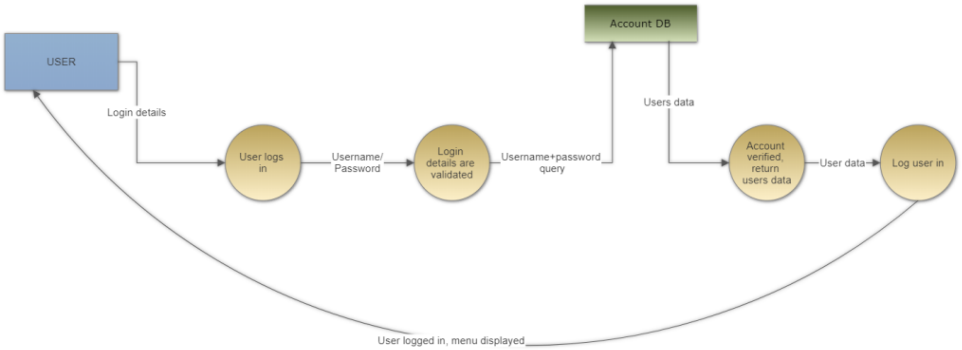
Comment [MR45]: Data flow diagrams to represent the flow of data in my program.

Data Flow Diagram (DFD)

A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. DFD focuses on the flow of information, where data comes from, where it goes and how it gets stored.

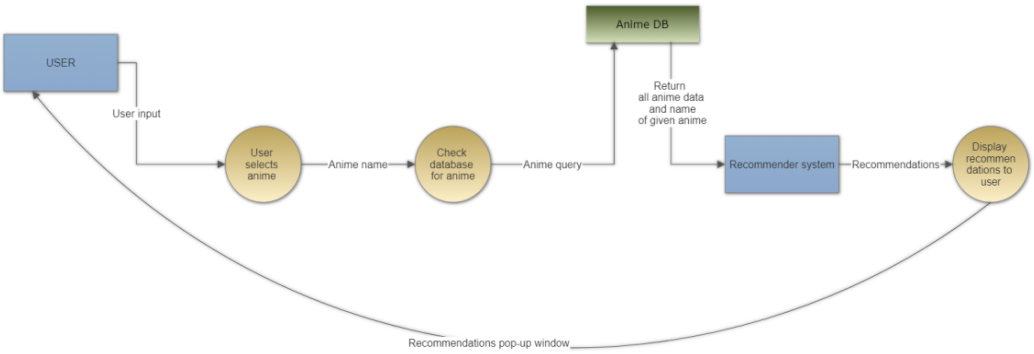
User login data flow

External Entities	Processes	Data stores
• **User** • **Account system**	• User enters login details • Login details are verified using the accounts database • Login verified, return users data • Load data into app • Log user in and display menu.	• Accounts Database



Get recommendation dataflow

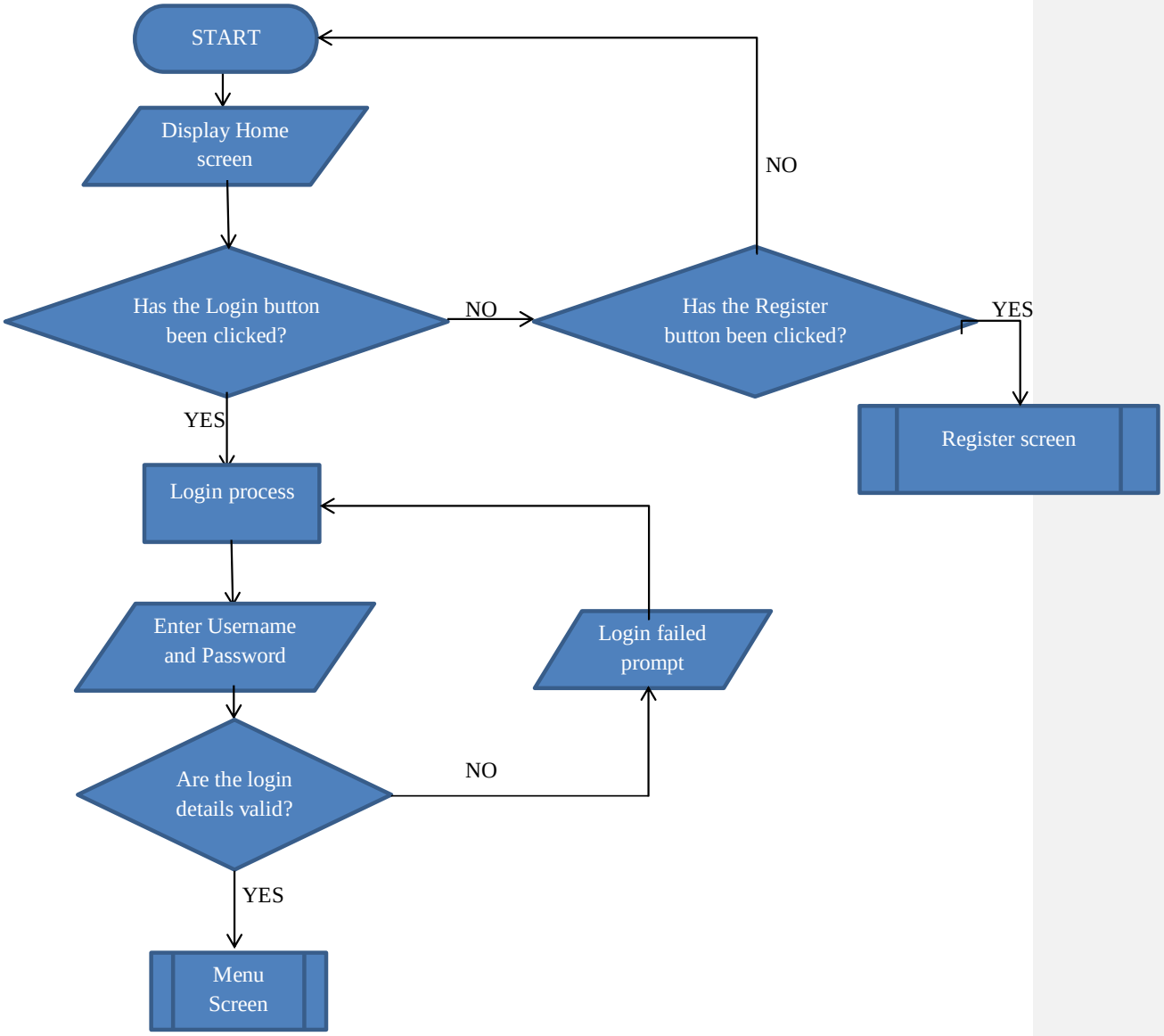
External Entities	Processes	Data stores
• **User** • **Recommender system**	• User selects anime to find a recommendation for • Anime database queried for anime • Pass all anime data to recommender system • Recommender system is executed using given anime • Results outputted to screen	• Anime Database



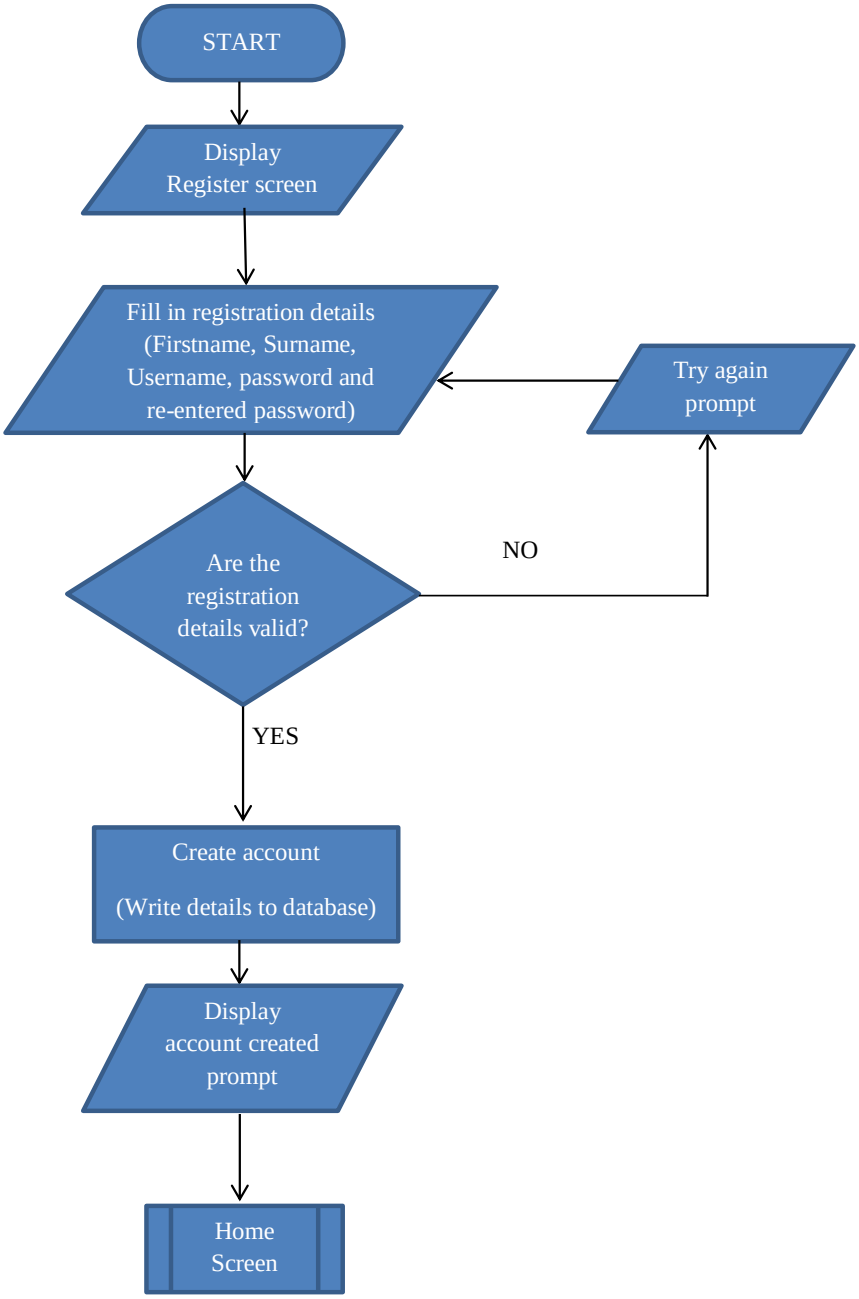
Comment [MR46]: Flow charts for each main section from my systems diagram

FLOW CHARTS

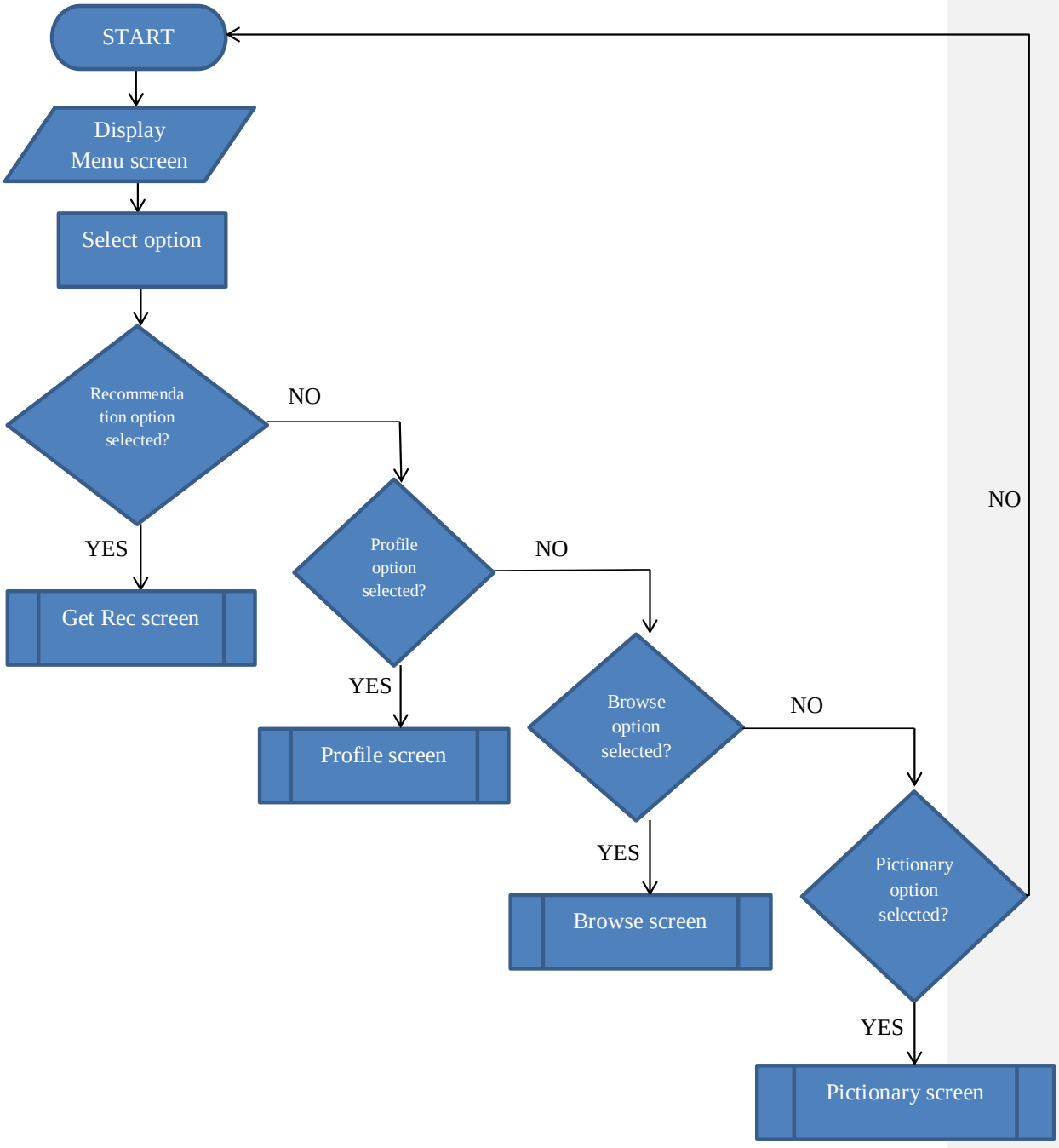
Home Screen



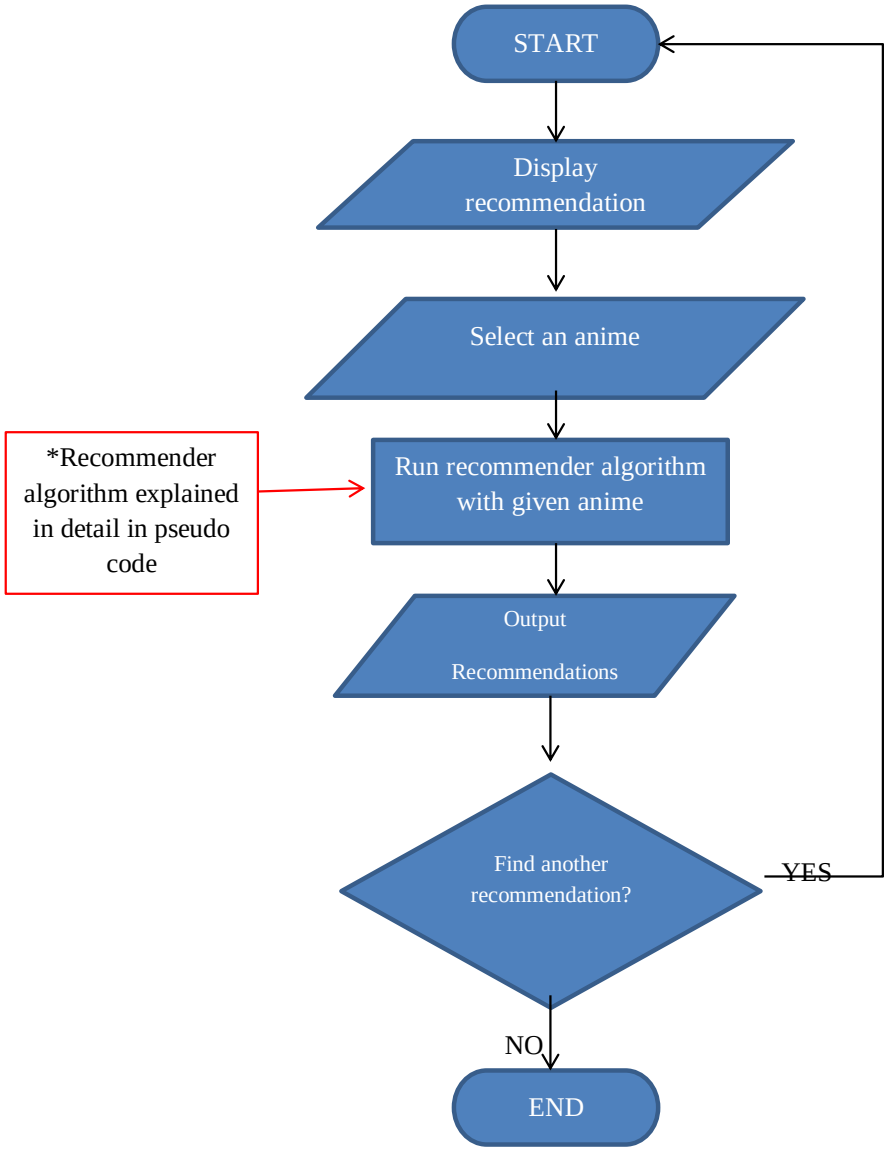
Register Screen



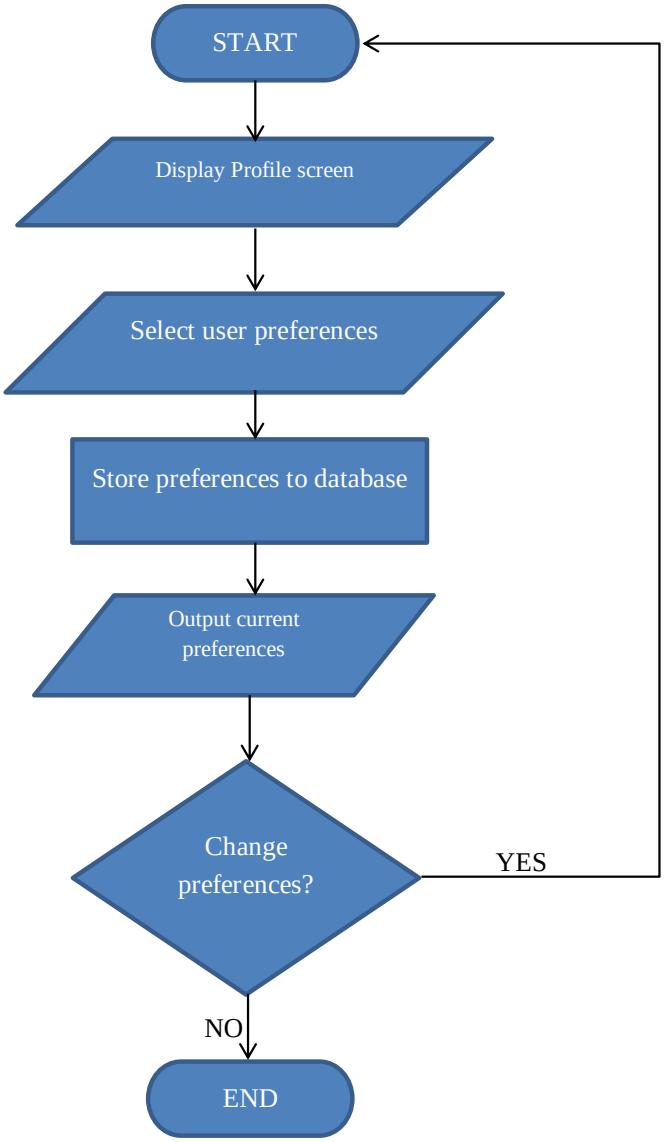
Menu Screen


START
Display Menu screen
Select option
Recommendation option selected?
NO
YES
Get Rec screen
Profile option selected?
NO
YES
Profile screen
Browse option selected?
NO
YES
Browse screen
Pictionary option selected?
NO
YES
Pictionary screen

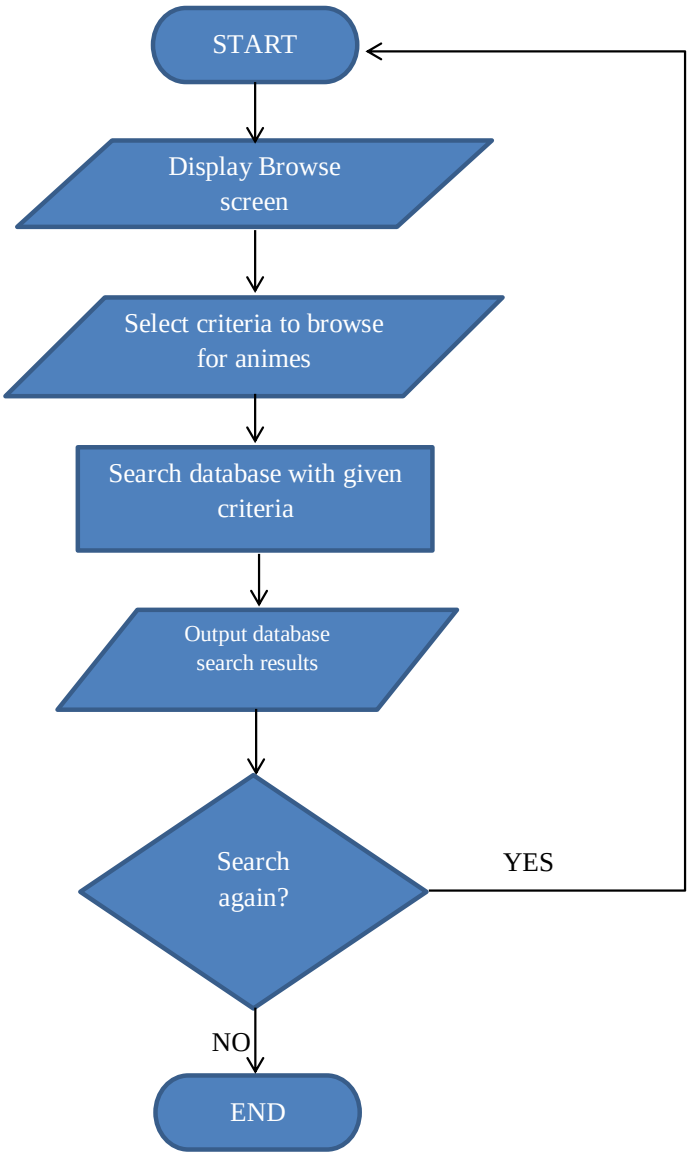
Get Rec screen



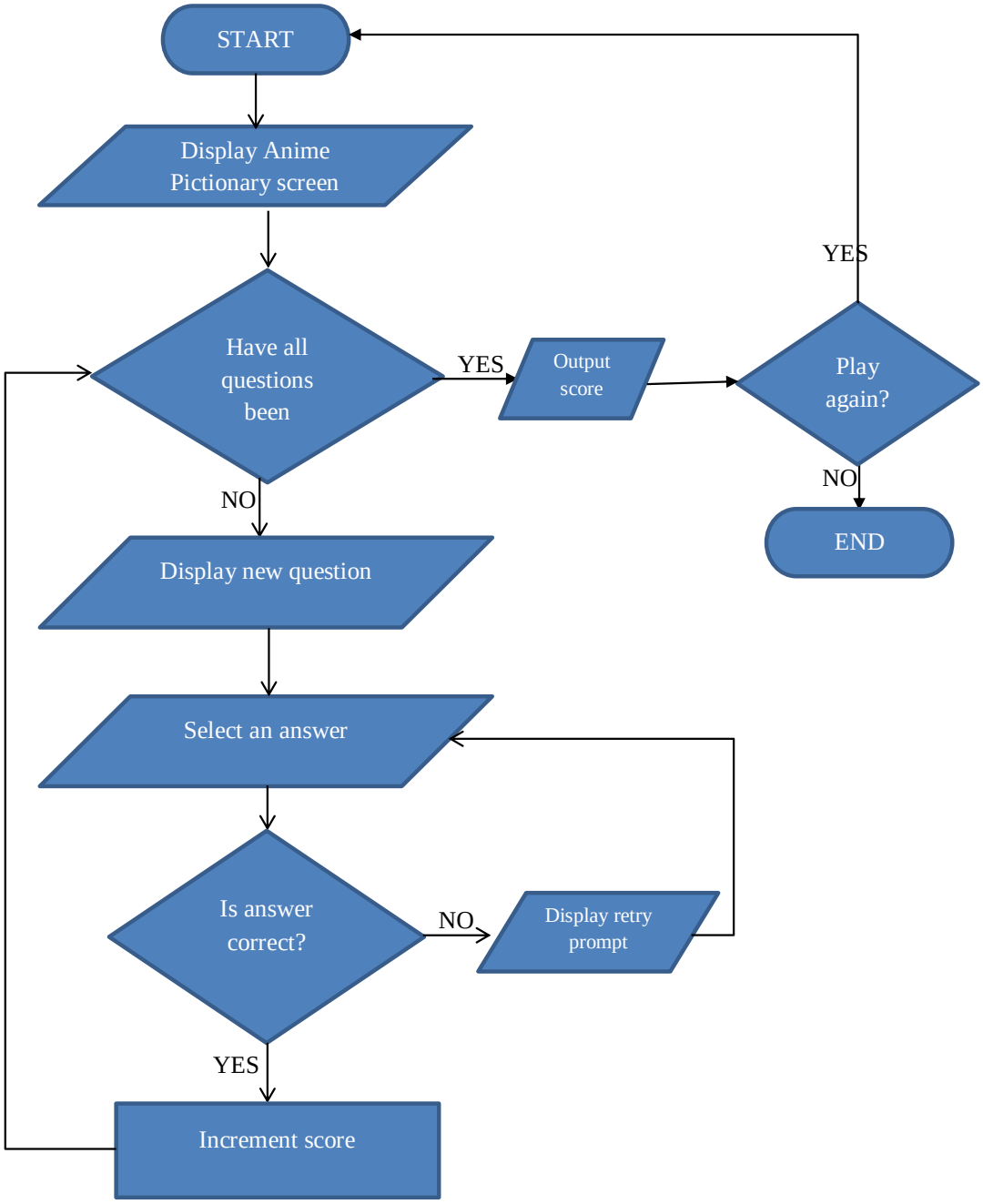
Profile screen



Browse Screen



Anime Pictionary



Comment [MR47]: Pseudo code for each section of my systems diagram

ALGORITHMS

NOTE the pseudo code below for each class is following the class designs in the screen mock ups above. Therefore every attribute and method in the pseudo code is explained in the class diagrams in the screen mock ups above.

Accounts system

The accounts system consists of the accounts database, the login function and the register function.

Creating the accounts database:

```
Procedure create_database ()

    Create database "Accounts.db"

    Create table if not exists "user"

    Add fields (userID, username, firstname, surname,
    password, favAnimes) to user table

    Save changes to Accounts.db

    Close database

End procedure
```

Data structure for Accounts system

Comment [MR48]: Included data structures for each section along with justification.

Variable name	Description	Data type	Sample value	Justification
userID	Unique numerical value to identify a user	INTEGER	1	UserID is of integer type as it is easier to check whether they exist or not user integers rather than referencing their usernames.
Username	Unique display name chosen by a user	STRING	"bill5"	Username, firstname, password and surname are of STRING type in order to make later functions easier to execute as all of these variables will be in the same format.
Password	Password chose by user	STRING	"Snow24"	
firstname	User's firstname	STRING	"Moller"	
surname	User's surname	STRINF	"Rodrigues"	

Interface

```
Class Interface inherits tkinter.Tk

    Procedure new (self)

        Create a window

        Within that window create a container that fills the
        whole window

        FRAMES = [Home, Menu, GetRec, Browse, Profile,
        register, Pictionary]

        For frame in FRAMES:

            Place frame in container

        Set Home frame to the top of the container (display
        home screen by default)

        AnimeNamesList = All names from anime.DB

    End procedure


    Procedure changeScreen (self, givenFrame)

        Raise givenFrame to the top of the container
        (display givenFrame)

    End procedure


End class
```

Data structure for interface

Variable name	Description	Data type	Justification
Container	An invisible frame within a window used to hold widgets	Tkinter Container Widget	Used to raise and lower frames in order to display and change screens
FRAMES	list of frame classes	ARRAY	1D ARRAY used in order to loop through, in order to place each frame in a container
AnimeNamesList	List of all anime names from the anime.DB	ARRAY	Stored as an array in order to make it more efficient and quick to iterate through all the anime names within the database when we need to display them or use them in a function.

Home screen

```
Class Home inherits Interface

    Procedure new (self)

        Create and display the following Tkinter WIDGETS
        onto the   Home frame:

            Title LABEL (text = "Anime Rec")

            Username LABEL (text= "Username :")

            Password LABEL (text= "Password :")

            Username ENTRY

            Password ENTRY (show = "*")

            Login BUTTON (text ="Login", command= procedure
            Login ())

            Register BUTTON (text="Register", command =
            procedure SUPER.changeScreen (Register))


    End procedure

    Procedure Login (self)

        Get username and password from Username and Password
        ENTRY widgets

        Connect to Accounts.db

        Check if there is a user in the database with the
        same username that was retrieved from the ENTRY
        Widget

        If TRUE:

            If the passwords match for that user:

                Execute procedure LoginSucess ()

            End if

        Else if the username or password is not in the
        database:

            Execute procedure LoginFail ()
```

```
        End if
    End procedure

    Procedure LoginSucess (self)
        Display pop - up WIND0W with the text "Login
        Successful"
        If pop-up WIND0W is closed:
            Execute procedure SUPERchangeScreen (Menu)
        Else:
            Pass
        End if
    End procedure

    Procedure LoginFail (self)
        Create a pop - up WIND0W with the text "Login
        Failed"
        Create a BUTT0N with the text "try again" on the pop
        - up WIND0W
        If try again BUTT0N is clicked:
            Execute Procedure SUPER.changeScreen (Home)
        Else:
            Pass
        End if
    End procedure
End class
```

Data structure for Home screen

Variable name	Description	Data type	Sample Value	Justification
Title	Text label containing a STRING representing the title of the screen.	Tkinter LABEL WIDGET	"Home"	In order to display text using Tkinter, you must place the text in a LABEL WIDGET
Username	ENTRY widget used to input the username	Tkinter ENTRY WIDGET	"Username:"	ENTRY widgets are used to input text in Tkinter.
Password	ENTRY widget used to input the password	Tkinter ENTRY WIDGET	"Password:"	ENTRY widgets are used to input text in Tkinter.
Login	BUTTON widget used to execute the login function	Tkinter BUTTON WIDGET	"Login"	BUTTON widgets are used to carry out commands upon click in Tkinter.
Register	BUTTON widget used to display the Register screen	Tkinter BUTTON WIDGET	"Register"	BUTTON widgets are used to carry out commands upon click in Tkinter

Register screen

```
Class Register inherits from Interface

    Procedure new (self)

        Create and display the following WIDGETS on the
        register FRAME:

            Firstname ENTRY (text="firstname")

            Surname ENTRY (text="surname")

            Username ENTRY (text="Username")

            Password ENTRY (text="Password")

            Re-enter password ENTRY (text="Re-enter
            password")

            CreateAccount BUTTON (text="Create", command =
            procedure Register ())

    End procedure
```

```
    Procedure Register (self)

        Get username, password, re-entered password, first
        name and surname from their respective ENTRY
        widgets.

        Connect to Accounts.db

        Check if user already exists in the database

        If the user exists:

            Output a prompt saying "User taken" and make
            them try again

        Else:

            Pass

        End if

        If password does not match re-entered password:

            Display pop-up WINDOW with the text "Passwords
            do not match"

            Prompt user to try again

        Else:

            Insert data (username, password, first name,
            and surname) into database

        End if

        Save changes to Accounts.db

        Close Accounts.db

    End procedure

End class
```

Data structure for Register screen

Variable name	Description	Data type	Sample Value	Justification
Firstname	ENTRY widget used to input the users firstname	Tkinter ENTRY WIDGET	"Moller"	ENTRY widgets are used to input text in Tkinter.
Surname	ENTRY widget used to input the users surname	Tkinter ENTRY WIDGET	"Rodrigues"	ENTRY widgets are used to input text in Tkinter.
Username	ENTRY widget used to input the username	Tkinter ENTRY WIDGET	"Username:"	ENTRY widgets are used to input text in Tkinter.
Password	ENTRY widget used to input the password	Tkinter ENTRY WIDGET	"Password:"	ENTRY widgets are used to input text in Tkinter.
Create	BUTTON widget used to execute the Register procedure	Tkinter BUTTON WIDGET	"Register"	BUTTON widgets are used to carry out commands upon click in Tkinter

Menu screen

```
Class Menu inherits from Interface

    Procedure new (self)

        Create and display the following WIDGETS onto the
        Home FRAME:

            GetRecommendation BUTTON
            (image="AnimeRecs.png"), command = Procedure
            SUPER.changeScreen (GetRec))

            Profile BUTTON (image="MyProfile.png", command
            = Procedure SUPER.changeScreen (Profile))

            Browse BUTTON (image="BrowseAnimes.png",
            command= Procedure SUPER.changeScreen (Browse))

            Pictionary BUTTON (image="Pictionary.png",
            command = Procedure SUPER.changeScreen
            (Pictionary))

    End procedure

End class
```

Data structure for Menu screen

Variable name	Description	Data type	Sample Value	Justification
GetRecommendation	BUTTON widget used to display the GetRec screen	Tkinter BUTTON WIDGET		BUTTON widgets are used to carry out commands upon click in Tkinter
Profile	BUTTON widget used to display the Profile screen	Tkinter BUTTON WIDGET		BUTTON widgets are used to carry out commands upon click in Tkinter
Browse	BUTTON widget used to display the Browse screen	Tkinter BUTTON WIDGET		BUTTON widgets are used to carry out commands upon click in Tkinter
Pictionary	BUTTON widget used to display the Pictionary screen	Tkinter BUTTON WIDGET		BUTTON widgets are used to carry out commands upon click in Tkinter

Profile screen

```
Class Profile inherits Interface

    Procedure new (self)

        Create and display the following Tkinter WIDGETS on
        the profile FRAME:

            FavAnime COMBOBOX (text ="Choose your favourite
            animes", combobox items =AnimeNamesList)

            AddToFavourites BUTTON (text="Add", command =
            Procedure AddTo ())

            CurrentFavourites LABEL (text =
            listOfFavAnimes)

    End Procedure

    Procedure AddTo (self)

        Get selected anime from FavAnime COMBOBOX

        X = selected anime

        ListOfFavAnimes = []

        Append X to ListOfFavAnimes

    End Procedure

End class
```

Data structure for Profile screen

Variable name	Description	Data type	Sample Value	Justification
FavAnime	COMBOBOX for the user to select their favourite animes	Tkinter COMBOBOX WIDGET	COMBOBOX ITEMS = ListOfAnimes	COMBOBOX widgets are used when there are lists of multiple values but the user can only choose one
AddToFavourites	BUTTON which executes the AddTo procedure when clicked	Tkinter BUTTON WIDGET	"Add"	BUTTON widgets are used to carry out commands upon click in Tkinter
CurrentFavourites	LABEL used to display the users current favourite animes	Tkinter LABEL WIDGET	"Code Geass Naruto One Piece Fairy Tail"	In order to display text using Tkinter, you must place the text in a LABEL WIDGET
X	Temporary variable used to hold the selected anime	STRING	"Code Geass"	In order to append the users selected anime to ListOfFavAnimes

	from the COMBOBOX			
ListOfFavAnimes	List used to hold all the users favourite animes	ARRAY	[Code Geass, Naruto, One Piece, Fairy Tail]	It is necessary to store all the users' favourite animes in a list so they can easily be displayed in one LABEL WIDGET.

Get recommendation (Get Rec) screen

```
Class GetRec inherits from Interface

    Procedure new (self)

        Create and display the following Tkinter WIDGETs on
        the GetRec FRAME:

            SimilarAnime COMBOBOX (text = "Find anime
            similar to:", combobox items = AnimeNamesList)

            GetRec BUTTON (text="Get Rec!", command =
            Procedure CBSystem ())

    End Procedure

    Procedure CBSystem (self)

        Get value from SimilarAnime COMBOBOX and store as
        SelectedAnime

        A1 = SelectedAnime

        BestScore = 0

        For each other anime (A2) in AnimeNamesList:

            MatchScore =0

            For each feature:

                Compare A1 and A2 on that feature

                If they match:

                    MatchScore += 1

                End if

            If MatchScore > BestScore:

                BestMatch = A2

                BestScore = MatchScore
```

```
            End if
        Return top 5 bestMatches
        Results = []
        Results. Append (top 5 bestMatches)
        Create pop-up WINDOW displaying the contents of
        Results in a Tkinter LABEL widget
    End Procedure

    Procedure CFSystem (self):
        Given a user x, recommend an anime that x might like
        For all animes A:
            Set score [A] = 0
        For each other user y:
            If y's preferences match x's preferences:
                Increment score [A] for all A that y likes
            End if
        Find the top 5 animes with the highest score and
return them
    End Procedure
End class
```

Data structure for Get Rec Screen

Variable name	Description	Data type	Sample Value	Justification
SimilarAnime	COMBOBOX for the user to select which anime they would like to find a recommendation for	Tkinter COMBOBOX WIDGET	COMBOBOX ITEMS = ListOfAnimes	COMBOBOX widgets are used when there are lists of multiple values but the user can only choose one
GetRec	BUTTON which executes the CBSystem procedure when clicked	Tkinter BUTTON WIDGET	"Get Rec!"	BUTTON widgets are used to carry out commands upon click in Tkinter
A1	The selected anime from SimilarAnime COMBOBOX WIDGET	STRING	"Naruto"	The CBSystem takes the given anime as a STRING.
BestScore	A baseline score which an anime must exceed in order to be considered as the best match.	INTEGER	START = 0 END = 987	For each other anime that the selected anime is compared against the BestScore will be incremented if it is higher than the previous BestScore thus returning the BestMatch at the end of the loop.
MatchScore	An indicator of how similar the selected anime and the anime being compared are.	INTEGER	START of loop = 0 END of loop = 0-55	Given a stage in the loop, if the current MatchScore is greater than the current BestScore then The BestScore will take the value of MatchScore
BestMatch	The anime with the highest BestScore	STRING	"Steins Gate"	BestMatch is returned as a STRING so it can be appended to the Results ARRAY.
Results	List of the 5 most similar animes	ARRAY	[Attack on Titan, SAO, Detective Conan, Naruto, Code Geass]	Stored as an array in order to make it easier to display these recommendations on a FRAME using a Tkinter LABEL WIDGET.

Browse Screen

```
Class Browse inherits Interface
    Procedure new (self)
        AnimeGenres = list of anime genres from anime.DB
        Create and display the following Tkinter WIDGETS on
        the Browse FRAME:
            Genres CHECKBUTTONS   (text = AnimeGenres)
            Name ENTRY ()
            Type COMBOBOX (text = "TV", "Movie", "OVA")
            Search BUTTON (text ="search", command =
            Procedure Search ())
    End procedure

    Procedure Search (self)
        Get values from Genres, Name and Type WIDGETS and
        store as → values
        Query Anime.DB with values
        Display query results in a pop-up WINDOW
    End Procedure
End class
```

Data structure for Browse Screen

Variable name	Description	Data type	Sample Value	Justification
AnimeGenres	list of anime genres from anime.DB	ARRAY	[(list of genres)]	All the genres are in an ARRAY so that they can easily be added as text to each CHECKBUTTON using a simple for loop
Genres	CHECKBUTTONs for the user to select what genres they would like the anime they wish to search for to have	Tkinter CHECKBUTTONs WIDGET	CHECKBUTTON1 = TRUE (selected) CHECKBUTTON2 = FALSE (not selected) …	CHECKBUTTONs are used when there multiple options can be selected.
Name	ENTRY WIDGET which allows the user to search animes by name	Tkinter ENTRY WIDGET	"Kaze"	ENTRY widgets are used to input text in Tkinter.
Type	COMBOBOX WIDGET which allows the user to select the anime type they wish to search for	Tkinter COMBOBOX WIDGET	"Movie" "TV" "OVA"	COMBOBOX widgets are used when there are lists of multiple values but the user can only choose one
Search	BUTTON which executes the Search Procedure	Tkinter BUTTON WIDGET	"Search"	BUTTON widgets are used to carry out commands upon click in Tkinter
values	List of the browsing options selected by the user from the genres. Name and type WIDGETS.	ARRAY	["Action", "Romance", "TV", "Naruto"]	Using Arrays is an efficient method of storing multiple values, which can be easily manipulated as required.

Pictionary Screen

```
Class Pictionary inherits Interface

    Procedure new (self)

        Score = 0

        AnimeImage = cover images of animes for Pictionary
        game

        Answer1, Answer2, Answer3, Answer4 = 3 random anime
        name options with 1 correct anime name for the image
        displayed/

        Create and display the following Tkinter WIDGETS on
        the Pictionary FRAME:

            AnimeImage LABEL (image = AnimeImage)

            Answer1 BUTTON (text = answer1, command =
            Procedure checkAnswer ())

            Answer2 BUTTON (text = answer2, command =
            Procedure checkAnswer ())

            Answer3 BUTTON (text = answer3, command =
            Procedure checkAnswer ())

            Answer4 BUTTON (text = answer4, command =
            Procedure checkAnswer ())

    End procedure

    Procedure checkAnswer (self)

        Get text from the answer BUTTON which was clicked

        If text == to correct answer:

            Increment score

            Display next question

        Else:


            Prompt user to try again

        End if

    End procedure

End class
```

Data structure for Pictionary Screen

Variable name	Description	Data type	Sample Value	Justification
AnimeImage	Image of the anime that is to be guessed by the user	PNG	animePic1.PNG	PNG utilizes lossless compression, therefore no image data is lost when saving or viewing the image.
Answers	List of answers for each question, containing 3 randomly generated answers and 1 correct answer	ARRAY (containing STRING elements)	["Code Geass", "One Piece", "Naruto", "Fairy Tail"]	So that I can iteratively create buttons to display the available answers to the user.
text	The value of the button that was clicked	STRING	"Code Geass"	In order to check whether the value of the button which was clicked matches the correct answer
Score	Users score	INTEGER	3	In order to keep a track of how the user is doing so that we can display the end result at the end of the game

Comment [MR49]: Explained my development strategies

TRADITIONAL SOFTWARE DEVELOPMENT LIFE CYCLE

Waterfall Development

For the first part of my project I will be using the waterfall methodology which will cover the areas of feasibility and problem definition, requirements and analysis and design. Using the feedback and requirements I accumulate from the requirements section, I will use them to forge my analysis and design sections for my project; by doing so I will be able to ensure my project fits and appeals to my target audience.

Agile methodology

During the development phase of my project, I have decided to switch from the waterfall methodology to an agile approach. Using an agile approach in the development phase will allow me to perform multiple iterations of testing and evaluating on a piece of code/ prototype in the hopes of improving my program to follow and meet the requirements that were stated in the analysis section. It will also give me the chance to check in with my clients to see if they would like to make any modifications/improvements before I move on to the next prototype.

Approaches to testing

Comment [MR50]: Improved testing strategies relating to my context

Throughout the development of my app I will be continuously implementing Black box, white box and alpha testing after I complete each sub section of my app. Testing is essential throughout development as testing in blocks is much more maintainable and makes it easier to root down errors and solve them.

The first type of testing I will be doing in black box testing. Black box testing is where you put an input in your system and expect a certain output for that given input. For example if you had a function that took a number and outputted the square of that number than for the input 2 you would expect the output 4. Black box testing disregards the quality and efficiency of the code but rather focuses on the end result of the program. I will use black box testing on all of the functions in my program, for example one function in my program will be a login function. Using black box testing on the login function, testing in the case of inputting valid data into the function I would expect the output to be that the user has successfully logged in and on the flipside using invalid data as the input the expected outcome should be that the user's login attempt has failed. For each function I will be stating, given an input, what the expected outcome will be and then carry out black box testing to see the actual result and record these in a table using screen shots as evidence.

The second type of testing that I will be carrying out is white box testing. White box testing is where you look at the actual contents of the code and test if you can improve its efficiency, robustness or validations. White box testing will allow me to make my app respond faster and improve its time complexity and space complexity. An example of where I can use white box testing in my program is on the recommender system. The recommender system is the algorithm that will calculate the similarity between each anime in order to aid me in deciding what a good recommendation for a given anime will be. White box testing is ideal in this

case as computing the similarities between thousands of animes can be a huge burden on computing resources and so it can have a negative effect on the performance of the application. But through the use of white box testing I can improve the quality of my recommender system's code in the aim to reduce its run time and the amount of resources it needs. When I use white box testing in development I will take a screen shot of the code before and after and identify the changes that were made and the effect of these changes on factors such as efficiency, robustness and validation.

Finally, alpha testing is a preliminary software field test which I will carry out myself in order to find bugs that were not found previously through other tests. The main purpose of alpha testing is to refine the software product by finding (and fixing) the bugs that were not discovered through previous tests. I will use alpha testing on my application to root out minor bugs with the GUI and usability features such as entry fields which have not been cleared when the user returns to a screen with entry widgets that they have previously typed in.

Just to summarise black box and white box testing will be done by me and reviewed by my clients. Furthermore, alpha testing will be carried out by me and Hannah Arifi (Client) and user acceptance testing will be carried out by a random end user from the test sample for questionnaire we conducted in the analysis stage.

Comment [MR51]: Test case for my algorithms

Test Case

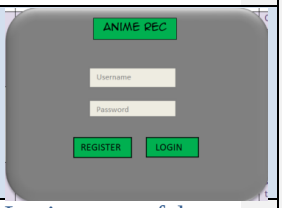
Below I have created a test case for each section of my algorithms.

Comment [MR52]: Tested each section /modular by modular

Account system

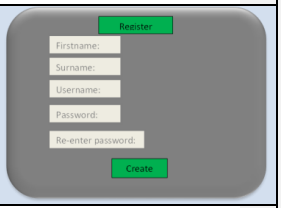
Test No.	Test case	Test Steps	Test Data	Expected result
1.0	Check if a database is created if it does not already exist	Run the code that creates a database before the program launches.	No Accounts.db in file directory before code is executed.	Creation of a database file called Accounts.db in the directory of the program.
1.1	Check to see that there is no action taken if the database already exits.	Run the code that creates a database before the program launches.	Accounts.db should be present in the directory of the program before it is launched.	No creation of an additional database file or modification to the existing database file.
1.2	Check to see whether the specified fields are actually added to the user table that is created in the database file	Run the code that creates a database before the program launches.	Accounts.db	Accounts.db should have a table called user with the following fields: UserID, username, password, firstname and surname.

Home screen

Test No.	Test case	Test Steps	Test Data	Expected result
2.0	Check whether the home screen is displayed correctly	1.Launch application 2.Home screen should be displayed on launch	N/A	
2.1	Check login functionality in the case of valid username and password	1.Launch application 2.Enter username 3.Enter password 4.Click LOGIN button	Username: m4 Password: snow24	Login successful prompt should be displayed and app should switch to Menu screen

2.2	Check login functionality in the case of invalid username and password	1.Launch application 2.Enter username 3.Enter password 4.Click LOGIN button	Username: g4 Password: rain42	Login failed prompt should be displayed with the option for the user to re-attempt to login
2.3	Check REGISTER button functionality	1.Launch application 2.Click REGISTER button	'Register button clicked'	Switch to Register screen

Register screen

Test No.	Test case	Test Steps	Test Data	Expected result
3.0	Check whether the register screen is displayed correctly	1.Launch application 2.Click REGISTER button		
3.1	Check CREATE button functionality in the case of valid details	1.Launch application 2.Click REGISTER button 3.Enter all details correctly 4.Click CREATE button	**Firstname:** Moller **Surname:** Rodrigues **Username:** mr2 **Password:** snow24 **Re-enter password:** snow24	Account successfully created and added to the database Accounts.db. Return to Home screen
3.2	Check CREATE button functionality in the case of blank entry fields	1.Launch application 2.Click REGISTER button 3.keep an entry field blank 4.click CREATE button	**Firstname:** **Surname:** Rodrigues **Username:** mr2 **Password:** snow24 **Re-enter password:** snow24	Display register fail prompt with error – 'Blank fields'
3.3	Check CREATE button functionality in the case of mismatching passwords	1.Launch application 2.Click REGISTER button 3. Input a different value in the **re-enter password** field compared to the	**Firstname:** Moller **Surname:** Rodrigues **Username:** mr2 **Password:** snow24 **Re-enter password:** rain64	Display register fail prompt with error – 'Passwords do not match'

		password field. 4.click CREATE button		

Menu screen

Test No.	Test case	Test Steps	Test Data	Expected result
4	Check whether the menu screen is displayed correctly	1.Launch application 2.Login successfully	N/A	
4.1	Check functionality of The Get Recommendation button	(Currently on the menu screen) 1.Click on Anime Rec Button	'user clicks on anime rec button'	Switch to Get Rec screen
4.2	Check functionality of The profile button	(Currently on the menu screen) 1.Click on My Profile Button	'user clicks on My Profile button'	Switch to Profile screen
4.3	Check functionality of The Browse button	(Currently on the menu screen) 1.Click on Browse Animes Button	'user clicks on Browse Animes button'	Switch to Browse screen
4.4	Check functionality of The Pictionary button	(Currently on the menu screen) 1.Click on Pictionary Button	'user clicks on Pictionary button'	Switch to Pictionary screen

Profile screen

Test No.	Test case	Test Steps	Test Data	Expected result
5	Check whether the profile screen is displayed correctly	1.Launch application 2.Login successfully 3. Click on Profile button on the menu screen.	N/A	

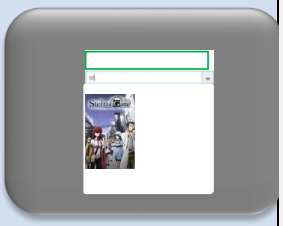
5.1	Check functionality of selecting a favourite anime	(currently on profile screen) 1.Enter anime name 2.Click the find Button	**Input 1** "Steins Gate" **Input 2** "steins gate" **Input 3** "steins" **Input 4** "gate"	**Input 1** – display all animes containing "Steins Gate" in their name in the scrollbar widget **Input 2** – display all animes containing "steins gate" in their name in the scrollbar widget **Input 3** – display all animes containing "steins" in their name in the scrollbar widget **Input 4** – display all animes containing "gate" in their name in the scrollbar widget
5.2	Check that the scrollbar results from a previous search clear when the user searches again.	(currently on profile screen) 1. Enter first anime into entry widget and click the find button 2. Enter second anime into entry widget and click the find button.	**Input 1 –** "Naruto" **Input 2 –** "One piece"	**Input 1** – displays all the animes containing "Naruto" in their name in the scrollbar widget. **Input 2** – clears all the results of input 1from the scrollbar and populates it with the search results for Input 2 (animes containing "One piece" in their names)

Browse screen

Test No.	Test case	Test Steps	Test Data	Expected result
6	Check whether the Browse screen is displayed correctly	(Currently on Menu Screen) 1.Click on Browse button (LMC)	N/A	
6.1	Check that browsing by genre works	(currently on browse screen) 1. Select Action and Comedy. 2.Click Search	[Action,Comedey]	A new window with a list of animes that have both Action and Comedy as their genres.
6.2	Check that browsing by name works	(currently on browse screen) 1.Type in "Steins" 2.Click Search	"Steins"	A new window should appear with a list of animes that contains 'steins' in their name.

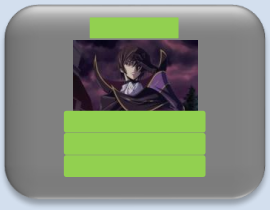
6.3	Check that browsing by type works	(currently on browse screen) 1. Select 'TV' from the type combo box 2. Click search	['TV']	A new window should appear with animes that are of the TV type only.

Get Rec screen

Test No.	Test case	Test Steps	Test Data	Expected result
7	Check whether the Get Rec screen is displayed correctly	(Currently on Menu Screen) 1.Click on Browse button (LMC)	N/A	
7.1	Check whether the search entry widget is functional	(Currently on the get rec screen) 1.Type in 'Steins' 2.Click on search	'Steins'	The Listbox should be populated with animes that contain 'Steins' in their name.
7.2	Check whether recommendation is generated when an anime is selected.	(List box is currently populated with search results of 'Steins') 1. Double click on the anime called 'Steins Gate'	'Steins Gate'	A new pop up window shgould be displayed with 5 animes that are similar to 'Steins Gate'. Also at the side of each recommendation their should be a button that says clcik fro more info.
7.3	Check whether the click for more info button is functional	(Currently on the recommendation pop up window) 1.Click on the more info button for 'anime x'	N/A	The my anime List website for that anime s hould open up in the users default browser.

7.4	Evaluate the recommender systems accuracy	Check each feature of the generated recommendations and compare them to the anime that the user entered to find recommendations for.	N/A	The generated recommendations should have a lot of the same features as the annime thatr was searched for.

Pictionary screen

Test No.	Test case	Test Steps	Test Data	Expected result
8	Check whether the Pictionary screen is displayed correctly	(Currently on the menu screen) 1.Click the Pictionary button	N/A	
8.1	Check result in the case of correct answer	1.Click on the correct answer	N/A	1.Display correct prompt 2.Increment score 3.Display next question
8.2	Check result in the case of incorrect answer	1.Click on an incorrect answer	N/A	1.Display game over screen 2.Prompt user to exit or try again

Comment [MR53]: Design review with client

Design proposal sign off

Home screen proposal summary

What was good?	Improvements/modifications that can be made
• Clean and consistent aesthetics • Minimalistic design • User friendly • Button to login or create an account • Validation for login process	• Add anime related image to the home screen • Instructions link on home page

Register screen proposal summary

What was good?	Improvements/modifications that can be made
• Consistent with the theme and colour scheme of the program • Entry widgets for account registration and a button to create an account • Minimalistic design (e.g. placeholder text instead of label for entry widgets) • Validation for register process	• Possibly add a feature which would allow users to change their username and password although this is not completely necessary as security is not a concern regarding this program.

Menu screen proposal summary

What was good?	Improvements/modifications that can be made
• The image buttons for the menu screen make it conform to the anime theme.	N/A

Profile screen proposal summary

What was good?	Improvements/modifications that can be made
• Easy and quick method to add animes to favourites using list box widget.	• A feature that links the user's favourite animes from this app to their myanimelist.net profile.

Get Rec screen proposal summary

What was good?	Improvements/modifications that can be made
• Consistent with the theme of the app. • Easy and simple to use – Takes one input and a button click to generate recommendations. • Buttons to find out more info about the generated recommendations	• Provide descriptions for each anime on the app. This however is not vital as the more info button will take the user to the myanimlist.net website for that anime, although this mean the user must have an internet connection.

Browse screen proposal summary

What was good?	Improvements/modifications that can be made
• Entry widget to search by name • Check button to search by genre • Combo box to search by type	N/A

Pictionary screen proposal summary

What was good?	Improvements/modifications that can be made
• The game is clear and easy to follow • Layout of widgets is aesthetically pleasing • Using buttons to display and get the users answer.	• Display the current level and score at the top of the screen

Josh Santillan, Lead Client

Comment [MR54]: Client approval

DEVELOPMENT

Comment [MR55]: For each section of development I have:
- Stated what I am developing
- The choices I made
- Screen shot of code with annotations
- Black box testing/white box testing/ GUI testing/ Robustness and validation
- Client review of initial prototype
- Review of final prototype

NOTE for the first 3 sections (interface, account database, anime database there is only one iteration reason why is explained in the review part for each)

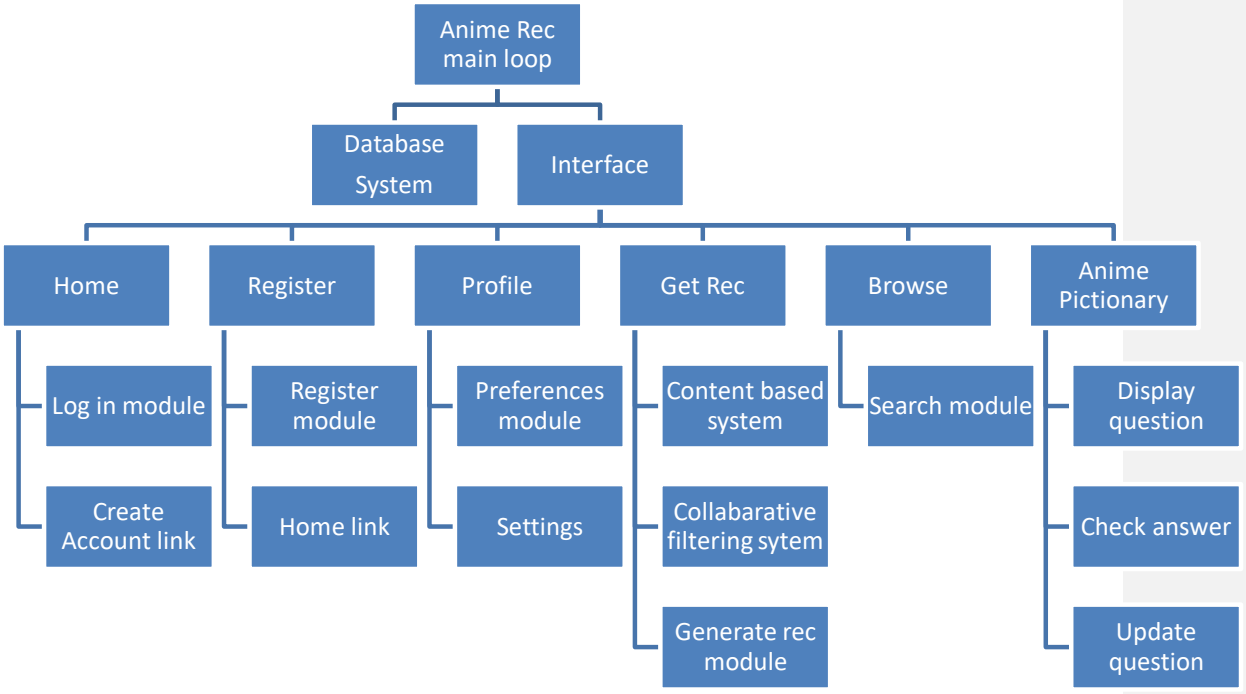
I will developing the program using Python as the main programming language. I chose Python as not only am I familiar with the language but Python alos provides many external libraries which will aid me in developing this app but in particular Python has really useful machine learning libraries such as Scikit Learn which will aid me in generating recommendations by calculating the similiarities between animes.

Iterative development

Comment [MR56]: Explained my iterative approach to development and testing

For the development of my program I will be following an iterative approach whereby after producing each module, I will provide a prototype to my lead client, Josh Santillan. Josh will evaluate this prototype so that I can incorporate his feedback into making improvements in the next iteration where I will develop a new prototype.

I will be developing modules in the order shown in my systems top-level diagram produced in the design section, in order to retain a consistent and organised flow to the development of my program.



Moller Rodrigues mar7147@outlook.com

26/03/2018 Interface – Creating the template for the GUI of my app

In order to create the GUI of my app I have chosen to use Tkinter. Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI and is included with the standard Microsoft Windows and Mac OS X install of Python.
I chose Tkinter over other Python GUI bindings like PyQt due to its layered approach and superior accessibility. However, a downside to using Tkinter is that it does not have a software which helps in the building of GUI's like PyQt does, therefore I have to hard code everything.

Comment [MR57]: Creating the interface section of my system.

Comment [MR58]: Annotated code of creating the interface class along with an explanation of my choices at the top.

OOP and GUI building

```
import tkinter as tk # Imports the Tkinter library used to create GUI's
from tkinter import ttk # Imports the ttk module from Tkinter which is used to style Tkinter widgets
import sqlite3 # Imports the sqlite3 library which is an embedded relational database management system

## FONT CONFIGURATION ##
large_font= ("Anime Ace Bold", 12)
norm_font =("Anime Ace Bold", 10)
small_font =("Anime Ace Bold", 8)

##
''' (STEP 1) CREATING THE INTERFACE CLASS - Instead of making a window for each of our screens, i decided to create just one window
and instead make multiple Tkinter Frame widgets which will represent the screens. I then plan to create an invisible Frame which will
act as a container to hold all the screens. So then if we want to change to a different screen we can do so by raising the screen
that we want to see to the top of the container.

I chose to do this in this way as creating multiple windows can be very inefficient on memory and cause cause delays and freezes in
runtime of the application.Whereas, the use of frames is much more generous on memory consumption.
'''
##

# Declaring the Interface class which inherits from Tkinter's Tk class
class interface(tk.Tk):

    # Declaring interface's constructor method and following convention by stating the default parameters
    def __init__(self, *args, **kwargs):

        # Initialising Tkinter (Creates the window)
        tk.Tk.__init__(self, *args, **kwargs)

        # Setting the title of the window
        tk.Tk.wm_title(self, "Anime Rec")

        # Creating a Tkinter Frame widget which will act as a container
        container = tk.Frame(self)

        # Displaying/putting the container onto the window and making it fill the entire window
        container.pack(side="top", fill="both", expand=True)

        # Giving the container priority over all other widgets
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        # Declaring a dictionary which will contain all the frames that we will make
        self.frames = {}

        # For loop which puts each frame in to the container and appends each container item to the self.frames dictionary
        for F in {home, register, menu, profile, get_rec, browse, pictionary}: # These are classes that are yet to be created
            frame = F(container, self)
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky="nsew")

        # Sets the Home screen to the top of the container by defualt (displays the home screen by default)
        self.show_frame(home)

        # This function will be called whenever we want to change screens, it does this by taking the screen we want to change to
        # as a parameter and using Tkinters tkraise() we can raise that screen to the top of the container
        def show_frame(self, cont):
            frame = self.frames[cont]
            frame.tkraise()

# Instantiates the Interface as the object app
app = interface()

# Declaring the desired windows dimensions
width_of_window = 605
height_of_window = 538

# Configuring the windows dimensions to the desired values
app.geometry("%dx%d" % (width_of_window, height_of_window))

# Makes the window un-resizable
app.resizable(False, False)

# The entire systems main loop
app.mainloop()
```

Interface testing

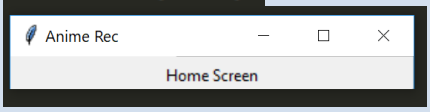
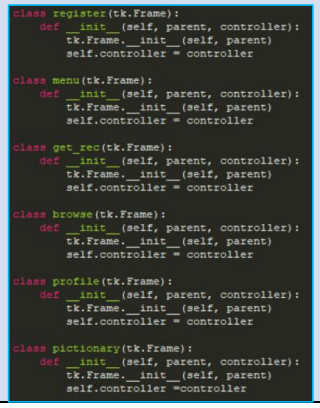
In order to test that the code for the Interface class is functional I created a screen with text 'home screen' to see if it displays on the window.

Creating blank Home screen:

```
class home(tk.Frame): # Class Home inherits from Tkinter's Frame class (Creating a frame for the Home screen)
    def __init__(self, parent, controller): # Constructor method for home class
        tk.Frame.__init__(self, parent) # Initialises the Frame class

        test_label = tk.Label(text='Home Screen') # Creates a text label for testing purposes
        test_label.pack()

app = interface() # Instantiates the Interface as the object app
app.mainloop() # apps mainloop
```

TEST CASE	**Check whether the code for the interface class is functional.**
EXPECTED RESULT	Display a screen with the text 'Home screen' and the title 'Anime Rec'

ACTUAL RESULT	`for F in {home, menu, get_rec, browse`
	`se are classes that are yet to be created`
	`NameError: name 'menu' is not defined`

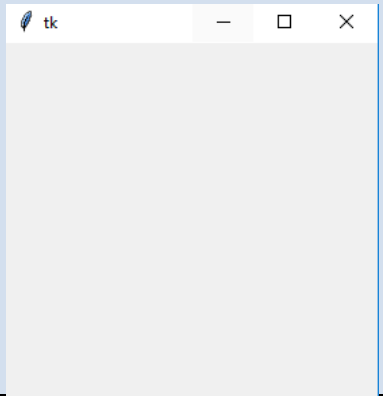
FIX/ MODIFICATION	Remove the frames that are yet to be created from the self.frames dictionary:
	BEFORE
	`for F in {home, menu, get_rec, browse, pictionary, profile, register}:`
	
	AFTER FIX
	`for F in {home}:`
	 
	
	ADDITIONAL MODIFICATION:
	I decided to create classes for all the screens beforehand to avoid any errors when I reference them whilst developing earlier sections of the program.
	 

```
class register(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

class menu(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

class get_rec(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

class browse(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

class profile(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

class pictionary(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller =controller
```

Comment [MR59]: Black box testing for the functionality of the Interface class

Comment [MR60]: Error with fix below

26/03/2018 – Interface prototype interview with client, Josh Santillan

Success criteria / requirement	Achieved / Not Achieved with evidence
GUI	Achieved The Interface class creates a window (GUI) which will be used by the user to interact with my application. Screen shot of window created by Interface class: 

ME: The purpose of the interface class was to create the main window of our app which will store all our different screens and act as a control unit in switching between these screens. Given this purpose I think that the outcome from the testing I have conducted indeed demonstrate that this purpose has been achieved.

Josh Santillan: I have reviewed the testing for the interface class and concur with Moller that the interface class achieves its intended purpose of creating the GUI of our app and switching between screens.

Furthermore, given the interface section has a very specific and limited purpose I believe we have achieved an optimal solution in our first iteration and we do not require another iteration of development for the Interface section.

Josh Santillan, Lead client

Comment [MR61]: Interface feedback and sign off **NOTE** the interface and the databases system are the only two sections which do not require second iterations of development as their purpose is very limited and whether I have achieved these purposes is as easy as distinguishing between white and black.

Comment [MR62]: I've done this thorough documentation for each section so I won't comment each section as it will take ages but instead ill comment things I haven't done in a section before

27/03/2018 – Creating the Accounts database

In order for users to create account and login we need to create permanent data storage to store these details. Therefore, to do this I will be using SQL within Python to create a database which will store users' data.

Comment [MR63]: Creating the accounts database using SQL to create a database.

```
import sqlite3 # Imports the sqlite3 library which is an embedded relational database management system

with sqlite3.connect("Accounts.db") as db: # Connects to the database, 'Accounts.db', if it doesn't exist then it is created
    cursor = db.cursor() # Creates a cursor object which is used to traverse the database

## Executes a sql query on the database which creates a table called 'user' with the fields: 'userID','username','firstname','surname
## if they do not already exist.
cursor.execute( '''
CREATE TABLE IF NOT EXISTS user(
userID INTEGER PRIMARY KEY,
username VARCHAR(20) NOT NULL,
firstname VARCHAR(20) NOT NULL,
surname VARCHAR(20) NOT NULL,
password VARCHAR(20) NOT NULL);
''')

db.commit() # Makes changes made to the database permanent

cursor.execute("SELECT * FROM user") # Selects all the data from the table 'user'

db.close() # Closes the connection to 'Accounts.db' so that it can be accesed by other processes; avoiding locking of the database.
```

Accounts system testing

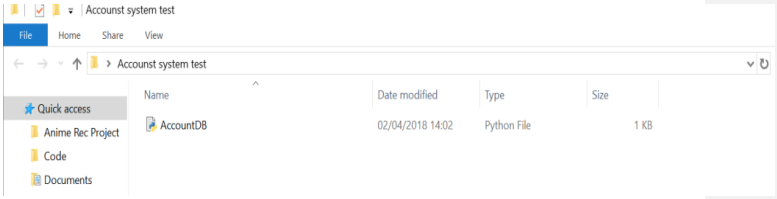
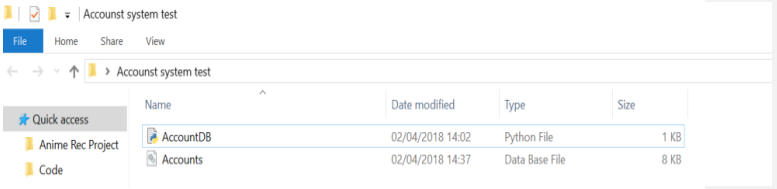
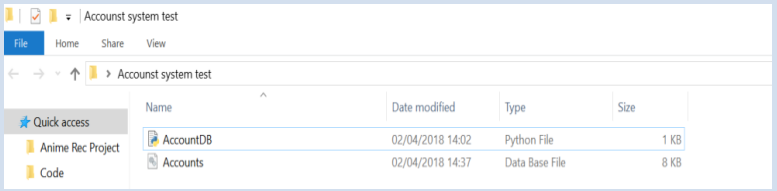
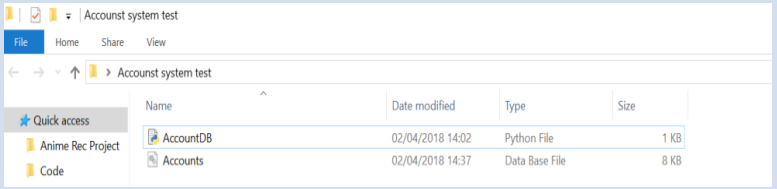
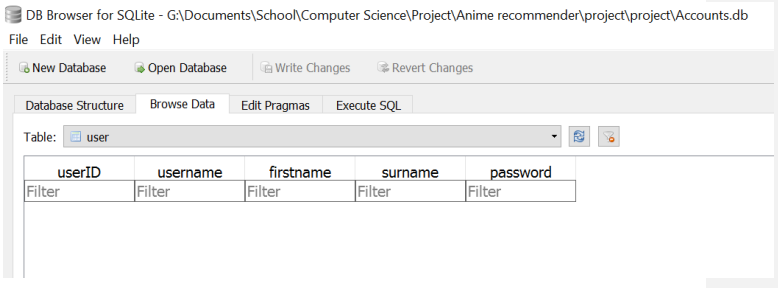
Comment [MR64]: Black box testing

Comment [MR65]: Testing different scenarios VALID and INVLAID for robustness

Test case	Test Steps	Test Data	Expected result	Actual Results	Fix
Check if a database is created if it does not already exist	Run the code that creates a database before the program launches.	No Accounts.db in file directory before code is executed.	Creation of a database file called Accounts.db in the directory of the program.	Database called 'Accounts.db' is created in file directory as expected. SCREENSHOT: 1	N/A
Check to see that there is no action taken if the database already exits.	Run the code that creates a database before the program launches.	Accounts.db should be present in the directory of the program before it is launched.	No creation of an additional database file or modification to the existing database file.	No extra file created and existing file is opened without any errors. SCREENSHOT: 2	N/A
Check to see whether the specified fields are actually added to the user table that is created in the database file	Run the code that creates a database before the program launches.	Accounts.db	Accounts.db should have a table called user with the following fields: UserID, username, password, firstname and surname.	A table called 'User' has been successfully created along with the declared fields (userID, username, firstname, surname, password) SCREENSHOT: 3	N/A

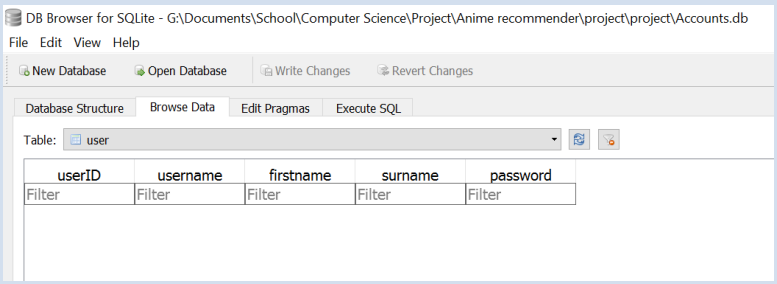
Accounts system screenshots -

Comment [MR66]: Evidence for the testing done above

SCREEN SHOT 1	**File Directory before the script is executed:**  **File Directory after the script is executed:** 
SCREEN SHOT 2	File Directory **before** the script is executed:  File Directory **after** the script is executed: 
SCREEN SHOT 3	

27/03/2018 –Accounts database interview with client, Josh Santillan

Comment [MR67]: Accounts database feedback and sign off

Success criteria / requirement	Achieved / Not Achieved with evidence
User accounts system – Must have a database to store user details such as their login details and favourite animes.	Achieved 

ME: The purpose of the Accounts database is to store the user's data which has been achieved as demonstrated by the testing above.

Josh Santillan: I have reviewed the testing for the accounts database and concur that it fulfils its intended purpose and also meets part of the user accounts system success criteria.

Furthermore, given the Accounts database has a very specific and limited purpose I believe we have achieved an optimal solution in our first iteration and we do not require another iteration of development for the Interface section.

Josh Santillan, Lead client

28/03/2018 – Creating the Anime database

In order to create the anime database, I will be using the open source library, mal-scraper. Mal scraper is a library which scrapes data of the website, myanimelist.net.

I will first be creating a csv file of all the anime data because the csv file will make it easier for me to format the data into the right format for the KNN algorithm which will be used to calculate the similarities between animes.

I will then convert the csv file to a database file, as querying database files is much easier and quicker than querying csv files; which means it will be more efficient to use the database file in features such as browsing for animes, rather than the csv file.

Creating the csv file of anime data

```
import mal_scraper # Imports the mal_scraper library which is used to scrape anime data from myanimelist.net
import csv # Imports the csv library which is used to read and write csv files

with open('anime.csv','w',newline='') as f: # creates csv file called 'anime.csv'
    thewriter = csv.writer(f) # Creates write object for anime.csv used to write data to he file

    # adds the follwoing column headers to the csv file
    thewriter.writerow(['anime_id,','name','genre','type','episodes','rating','members'])

next_anime = 1

# Using methods from mal_scraper, i am iteratively scraping data from each anime and
# writing it to the csv file under their respective headers
for i in range(0,12652):
    try:
        meta, data = mal_scraper.get_anime(next_anime)
    except mal_scraper.ParseError as err:
        logger.error('Investigate page %s with error %d', err.url, err.code)
    except requests.exceptions.HTTPError as err: # Excepts an network/requets errors
        print(err)
        pass

    else: # adds data to csv file
        print(data)
        thewriter.writerow([next_anime,data['name'],data['genre'],data['type'],
                            data['episdoes'],data['rating'],data['members']])

    next_anime = meta['id_ref'] + 1
```

SCREENSHOT OF THE CREATED CSV FILE:

	A	B	C	D	E	F	G
1	anime_id	name	genre	type	episodes	rating	members
2	32281	Kimi no Na	Drama, Rc	Movie	1	9.37	200630
3	5114	Fullmetal /	Action, Ad	TV	64	9.26	793665
4	28977	GintamaÂ	Action, Co	TV	51	9.25	114262
5	9253	Steins;Gat	Sci-Fi, Thri	TV	24	9.17	673572
6	9969	Gintama&	Action, Co	TV	51	9.16	151266
7	32935	Haikyuu!!:	Comedy, [	TV	10	9.15	93351
8	11061	Hunter x H	Action, Ad	TV	148	9.13	425855
9	820	Ginga Eiyu	Drama, M	OVA	110	9.11	80679
10	15335	Gintama M	Action, Co	Movie	1	9.1	72534

Comment [MR68]: Creating the anime database (explained in design) contains animes and their features.

Comment [MR69]: Use of an external library for web scraping

Comment [MR70]: Creating a csv file and reading to it

Converting the csv file to a database file

Comment [MR71]: Converting CSV to SQL

```
import csv # Imports the csv library which is used to read and write csv files
import sqlite3 # Imports the sqlite3 library which is an embedded relational database management system

con = sqlite3.connect("animeData.db") # Creates a database file called, 'animeData.db'
cur = con.cursor() # creates a cursor object used to traverse the database
# Creates the table 't' with the given fields.
cur.execute("CREATE TABLE IF NOT EXISTS t (anime_id, name, genre, type, episodes, rating, members);")

with open('anime.csv','rt', encoding="utf8") as f: # opens the csv file, 'anime.csv' as f
    # csv.DictReader uses first line in file for column headings by default
    dr = csv.DictReader(f) # comma is default delimiter
    to_db = [(i['anime_id'], i['name'], i['genre'], i['type'], i['episodes'], i['rating'], i['members']) for i in dr]

#writes the data from each line in the csv file to the database file
cur.executemany("INSERT INTO t (anime_id, name, genre, type, episodes, rating, members) VALUES (?, ?, ?, ?, ?, ?, ?);", to_db)
con.commit() # Makes changes made to the database file permanent
con.close() # closes connection to he database file
```

SCREENSHOT OF THE CREATED DATABASE FILE:

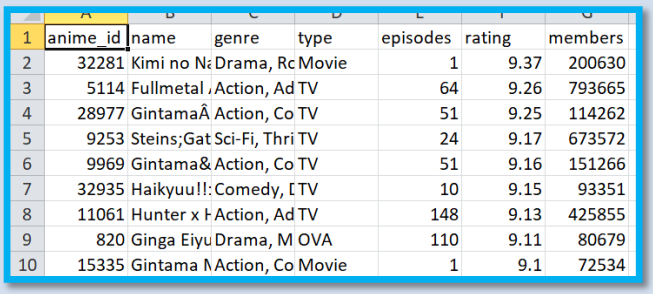
Table: t — New Record

	anime_id	name	genre	type	episodes	rating	members
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	32281	Kimi no Na wa.	Drama, Roma...	Movie	1	9.37	200630
2	5114	Fullmetal Alch...	Action, Adven...	TV	64	9.26	793665
3	28977	Gintama°	Action, Come...	TV	51	9.25	114262
4	9253	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17	673572
5	9969	Gintama'	Action, Come...	TV	51	9.16	151266
6	32935	Haikyuu!!: Ka...	Comedy, Dra...	TV	10	9.15	93351
7	11061	Hunter x Hunt...	Action, Adven...	TV	148	9.13	425855
8	820	Ginga Eiyuu D...	Drama, Milita...	OVA	110	9.11	80679
9	15335	Gintama Movi...	Action, Come...	Movie	1	9.10	72534
10	15417	Gintama&#03...	Action, Come...	TV	13	9.11	81109
11	4181	Clannad: Afte...	Drama, Fanta...	TV	24	9.06	456749
12	28851	Koe no Katachi	Drama, Schoo...	Movie	1	9.05	102733
13	918	Gintama	Action, Come...	TV	201	9.04	336376
14	2904	Code Geass: ...	Action, Dram...	TV	25	8.98	572888
15	28891	Haikyuu!! Sec...	Comedy, Dra...	TV	25	8.93	179342
16	199	Sen to Chihiro...	Adventure, Dr...	Movie	1	8.93	466254
17	23273	Shigatsu wa K...	Drama, Music...	TV	22	8.92	416397
18	24701	Mushishi Zoku...	Adventure, Fa...	TV	10	8.88	75894
19	12355	Ookami Kodo...	Fantasy, Slice...	Movie	1	8.84	226193

1 - 19 of 12294 — Go to: 1

Comment [MR72]: Anime database feedback and sign off

28/03/2018 –Anime database interview with client, Josh Santillan

Success criteria / requirement	Achieved / Not Achieved with evidence
Efficient and functional recommender system – Must have a database of animes with their features which will be used by the recommender system (KNN algorithm) to calculate the similarity between animes by comparing their features.	Achieved 

	anime_id	name	genre	type	episodes	rating	members
2	32281	Kimi no Na	Drama, Ro	Movie	1	9.37	200630
3	5114	Fullmetal A	Action, Ad	TV	64	9.26	793665
4	28977	GintamaÂ	Action, Co	TV	51	9.25	114262
5	9253	Steins;Gat	Sci-Fi, Thri	TV	24	9.17	673572
6	9969	Gintama&	Action, Co	TV	51	9.16	151266
7	32935	Haikyuu!!:	Comedy, D	TV	10	9.15	93351
8	11061	Hunter x H	Action, Ad	TV	148	9.13	425855
9	820	Ginga Eiyu	Drama, M	OVA	110	9.11	80679
10	15335	Gintama N	Action, Co	Movie	1	9.1	72534

ME: The purpose of the Anime database is to store the features of thousands of animes and as the testing and screenshots above demonstrate; this purpose has been achieved.

Josh Santillan: I have reviewed the testing for the anime database and concur that it fulfils its intended purpose and also meets part of the Efficient and functional recommender system success criteria.

Furthermore, given that the Anime database has a very specific and limited purpose I believe we have achieved an optimal solution in our first iteration and we do not require another iteration of development for the Interface section.

Josh Santillan, Lead client

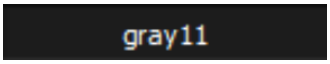
29/03/2018 – Creating the Home screen

Comment [MR73]: Creating the home screen of my system

Following the home screen mock up from the design section I have created the following images using 3d paint for the buttons for the home screen, in order to meet the clients' requirements for the aesthetics of the program.

Comment [MR74]: Creating images for my buttons which are consistent with the chosen colour scheme and anime theme.

Home background colour:



Home title Image:

Login Button:

Create Account button:

I decided to fix my window size at 605x538 pixels, thus avoiding any widget scaling problems and providing the optimal size to use my app by default.

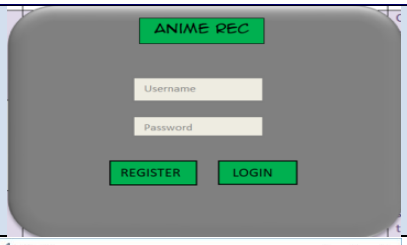
```
app = interface() # Instantiates the Interface as the object app
width_of_window = 605 #  declaring windows width
height_of_window = 538 #  declaring windows height
app.geometry("%dx%d" % (width_of_window, height_of_window)) # Setting the windows dimensions to the declared values
app.resizable(False, False) # Making the window a fixed size (Not resizable)
app.mainloop() # apps mainloop
```

Creating the front-end of the home screen

Comment [MR75]: Hard coding the GUI for the home screen

```
## CREATING AND DISPLAYING THE HOME TITLE ##
self.title_img = tk.PhotoImage(file="Home_Title.png") # Loads image
    # Creates Label widget containing the image
self.title_label = ttk.Label(self, width=392, image=self.title_img, relief='flat', background =  'grey11')
self.title_label.place(relx=0.21, rely=0.02, height=108, width=392) # Places widget onto window

## CREATING AND DISPLAYING THE LOGIN BUTTON ##
login_img = tk.PhotoImage(file = "Login_button.png") # Loads image
    # Creates Button widget containing the image
self.login_butt = tk.Button(self,image=login_img,borderwidth=0,background ='grey11', activebackground='grey11')
self.login_butt.image=login_img # keeps a reference of the widgets image
self.login_butt.place(relx=0.4, rely=0.46, height=35, width=120) # Places widget onto window

## CREATING AND DISPLAYING THE REGISTER BUTTON ##
reg_img = tk.PhotoImage(file = "Reg_Button.png")
self.register_butt = tk.Button(self, image=reg_img , borderwidth=0,background ='grey11', activebackground='grey11')
self.register_butt.image = reg_img
self.register_butt.place(relx=0.31, rely=0.74, height=45, width=250)

## CREATING AND DISPLAYING USERNAME ENTRY WIDGET ##
self.username_entry = ttk.Entry(self) # Creates usenrmae entry widget
self.username_entry.place(relx=0.35, rely=0.3, relheight=0.06, relwidth=0.34) # Places widget onto window
self.username_entry.insert(0, 'Enter Username') # Sets placeholder text

## CREATING AND DISPLAYING PASSWORD ENTRY WIDGET ##
self.password_entry = ttk.Entry(self)
self.password_entry.place(relx=0.35, rely=0.37, relheight=0.06, relwidth=0.34)
self.password_entry.insert(0, 'Enter Password')
```

Comment [MR76]: Black box and GUI testing for the Home screen

Test Case	**Check whether the home screen is displayed correctly**
Expected result	
Actual result	
Fix	N/A

Creating the Back-end for the Home screen

Adding the Login Button Functionality

1. *Creating a login function (temporarily does nothing, in order to run the script):*

```
def login(self):
    pass
```

2. *Bind login function to login button:*

```
self.login_butt = tk.Button(self,image=login_img,command = self.login,
```

3. *Coding the login function:*

Comment [MR77]: SQL – querying a database

```
def Login(self): # Login function which is executed when the login button is clicked
    username = self.username_entry.get() # Gets the value from the username entry widget
    password = self.password_entry.get() # Gets the value from the password entry widget

    with sqlite3.connect("Accounts.db") as  db:# Connects to the Accounts database and checks if the login credentials are valid
        cursor = db.cursor() # Creates a cursor object used to traverse the database
    find_user = ("SELECT * FROM user WHERE username = ? AND password =?") # runs a sql query on the database to check login validity
    cursor.execute(find_user,[(username),(password)])
    results = cursor.fetchall() # Stores the query results in variable called 'results'
    if results:
        for i in results:
            self.name=i[2]
        self.LoginSuccess() # if credentials are valid then the LoginSuccess() is executed
    else:
        self.LoginFail() # if credentials are invalid then the LoginFail() is executed
    db.commit() # Makes changes made to the database permanent
```

The function above involves interacting with the Accounts database, whereby I am running an SQL query to check whether the entered account details by the user are valid.

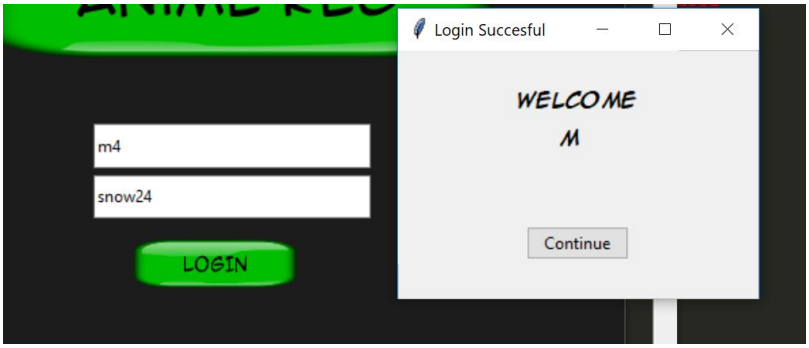
4. *Creating Login Success pop-up window:*

```
def LoginSuccess(self): # executed if Login is succesful
    self.popup = tk.Tk() # Creates a pop-up window
    self.popup.wm_title("Login Succesful")

    # Creates and displays a welcome message on the pop up window
    label = ttk.Label(self.popup,text=("Welcome\n\n      "+self.name), font =NORM_FONT)
    label.pack(side='top', expand=True)

    # Creates and displays a continue button, which is binded to the LSMenu function
    button = ttk.Button(self.popup, text="Continue", command=self.LSMenu)
    button.pack(side='bottom', expand=True)

    # Sets the dimensions of the pop-up window
    width_of_window = 200
    height_of_window = 150
    self.popup.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window))
    self.popup.mainloop() # ends the loop for the pop up window

def LSMenu(self):
    self.popup.destroy() # Closes the login sucessful pop up window
    self.controller.show_frame(Menu) # Displays the Menu screen
```

5. *Creating Login Fail pop-up window:*

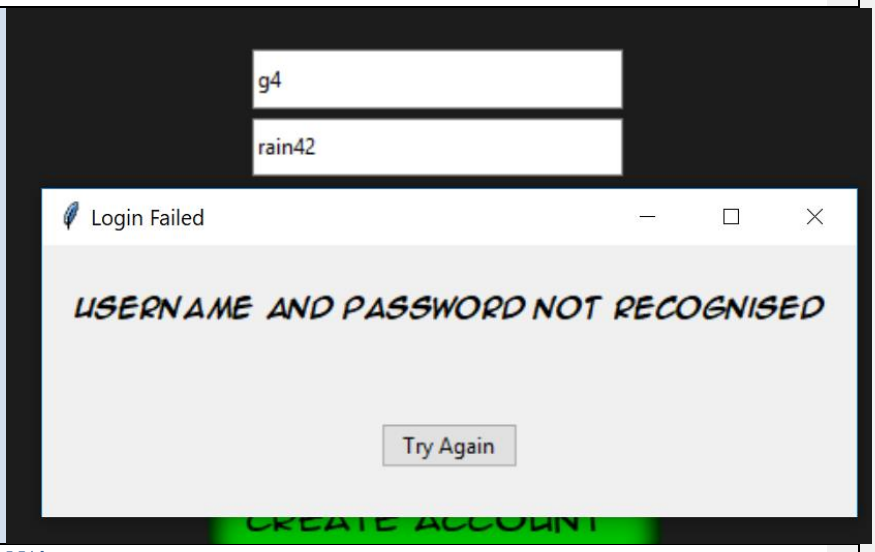
```
def login_fail(self): #  Executed if user login fails
    self.popup = tk.Tk() # Creates a pop-up window
    self.popup.wm_title("Login Failed") # Sets windows title

    label = ttk.Label(self.popup,text=("Username and password not recognised"), font =NORM_FONT)# Error message (Label widget)
    label.pack(side='top', expand=True)

    button = ttk.Button(self.popup, text="Try Again", command=self.lf_home) # Try again button
    button.pack(side='bottom', expand=True)

    # Sets the dimensions of the pop-up window
    width_of_window = 450
    height_of_window = 150
    self.popup.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window))
    self.popup.mainloop()# ends the loop for the pop up window

def lf_home(self):
    self.popup.destroy() # Closes the login failed pop up window
    self.controller.show_frame(home) # Displays the Menu screen
```

Login button testing

Test Case	**Check login functionality in the case of valid username and password**
Test Data	Username: m4 Password: snow24 (ADMIN ACCOUNT ALREADY CREATED)
Expected Result	Login successful prompt should be displayed.
Actual Result	`NameError: name 'sqlite3' is not defined`
Fix/Modification	Fix: `import sqlite3` After fix: 

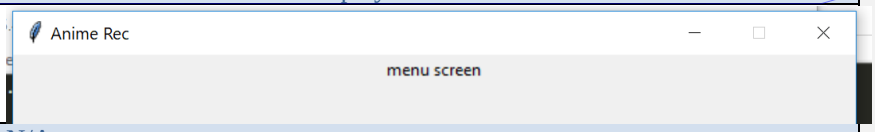
Comment [MR78]: Black box/Usability/robustness testing for the login function

Comment [MR79]: Error with fix below

Test Case	**Check login functionality in the case of invalid username and password**
Test Data	Username: g4 Password: rain42
Expected Result	Login failed prompt should be displayed.
Actual Result	
Fix/Modification	N/A

Comment [MR80]: Black box/Usability/robustness testing for the login function

Pop-up window testing

Test case	**Check functionality of the Continue button on the login successful pop up window**
Expected result	The Menu screen should be displayed after the Continue button is clicked
Actual result	
Fix/Modification	N/A

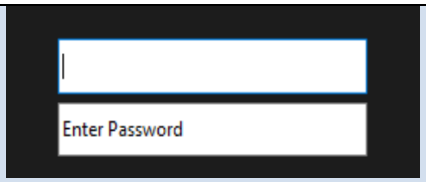
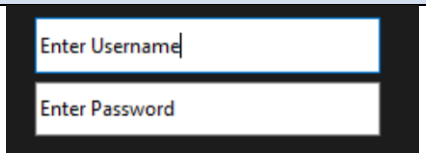
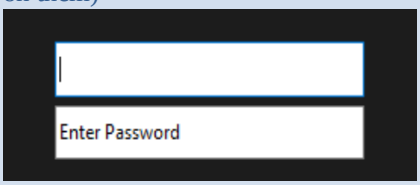
Comment [MR81]: Black box/Usability/robustness testing for the continue button

Test case	**Check functionality of the Try again button on the login failed pop up window**
Expected result	The Home screen should be displayed
Actual result	
Fix/Modification	N/A

Comment [MR82]: Black box/Usability/robustness testing for the try again button

Entry field testing/modification

Both the username and password entry fields are fully functional in the sense that they take and return the user's inputs correctly. However, I have come across a few minor technical problems.

Test Case	**Check to see if the placeholder text in the entry widgets clears when you click into them to type.**
Expected result	
Actual result	
Fix/Modification	Modification: `self.username_entry.bind("<FocusIn>", lambda args: self.username_entry.delete('0', 'end'))` ** Clears all the placeholder text in the entry widgets when you click into them (focus on them) ** 

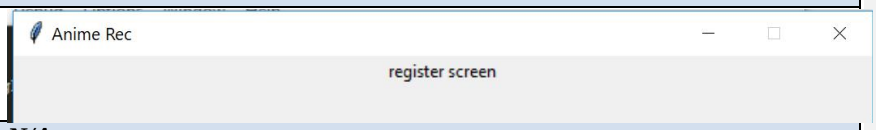
Comment [MR83]: Black box/Usability/robustness testing

Adding functionality to the Create Account Button

- *Binding a function to the Create Account button*

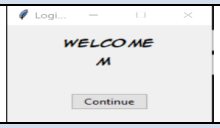
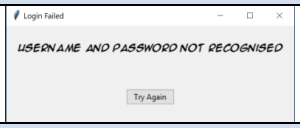
```
self.register_butt = tk.Button(self, image=reg_img,command = lambda:controller.show_frame(register))
```

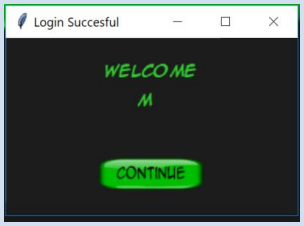
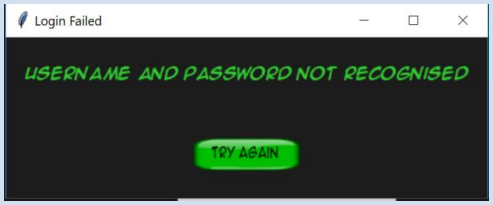
Create Account Button testing

Test Case	***Check to see that the Register Screen is displayed when the Create Account Button is clicked***
Expected Result	***The Register screen is displayed***
Actual Result	
Modification	***N/A***

Comment [MR84]: Black box/Usability/robustness testing

Home Screen form testing

Test Case	**Check to see if the form of the home screen and the attributes associated with the home screen conform with the programs aesthetic requirements**
Expected result	All widgets/frames/pop-up windows should match with the home screens colour scheme
Actual result	 

Modification	• I created the following buttons:
	 
	• Changed all the background colours to 'grey11'
	`tk.Frame.__init__(self, parent, bg = 'grey11')`
	ERROR WHILE TRYING TO CHANGE BUTTON IMAGE:
	`_tkinter.TclError: image "pyimage4" doesn't exist`
	Research: https://stackoverflow.com/questions/20251161/tkinter-tclerror-image-pyimage3-doesnt-exist?noredirect=1&lq=1
	Fix:
	AFTER `self.popup = tk.Toplevel()` MODIFICATION:
	  

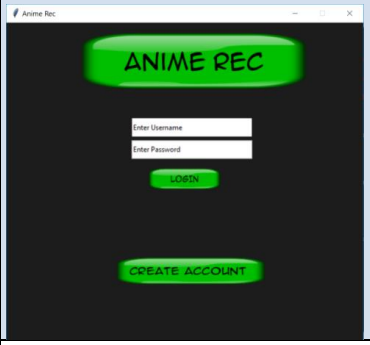
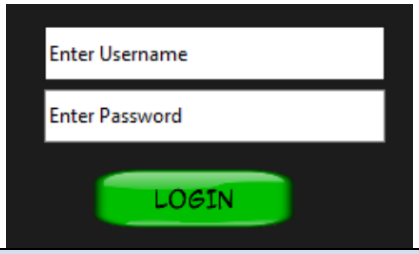
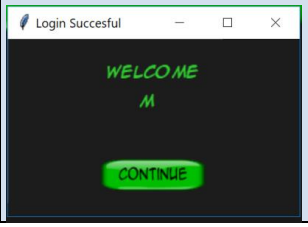
Comment [MR85]: Black box/Usability/robustness testing

Comment [MR86]: Fix/Modification

30/03/2018 – Home Screen prototype interview with client, Josh Santillan

Comment [MR87]: Home screen client review/feedback and success criteria reference

NOTE Referring to the requirements/success criteria created in the analysis section

Success criteria / requirement	Achieved / Not Achieved with evidence
Home screen – A screen that allows the user to login into the program and/or create an account.	Achieved 
User accounts system- - **Function which allows users to login.**	Achieved 
Consistent colour scheme and anime theme 	Achieved  Font – ANIME ACE Consistent colour scheme – Black/green

Me: I have managed to cover all the requirements that are related to the home screen, as I have created the home screen and its pop-up prompts to follow a consistent colour scheme and have integrated the front end GUI of the home screen with its respective back end functions which allows the user to login to their account by querying the accounts database. Furthermore, I have also include validation for the login process by including pop-up prompts allowing the user to re-try in the case of a failed login attempt.

Josh Santillan: I am extremely happy with the home screen prototype as it is follows are pre-determined requirements and designs very closely. As for any improvements I would say that the icon for the program should be updated to match the anime theme and also perhaps an image of an anime character on the home screen.

FINAL HOME SCREEN PROTOTYPE

Comment [MR88]: Final prototype for home screen with the modifications that were made

Modifications from first prototype:

- Entry widgets – Place holder text clears when user focuses on entry widget, Entered text in the entry widgets is cleared when the user logs out and returns to home screen
- Consistent colour scheme throughout the entire program so far
- Consistent themed images for all buttons
- Anime ace font for all large text
- Added anime character image on the home screen

Josh Santillan, Lead client

Comment [MR89]: Client sign off for final Home screen prototype

31/03/2018 – Creating the Register screen

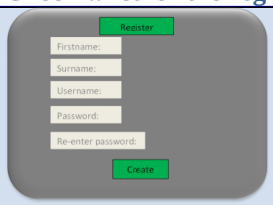
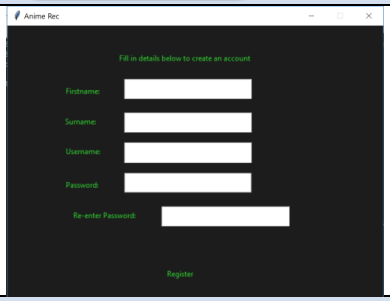
Comment [MR90]: Creating the Register screen of my app

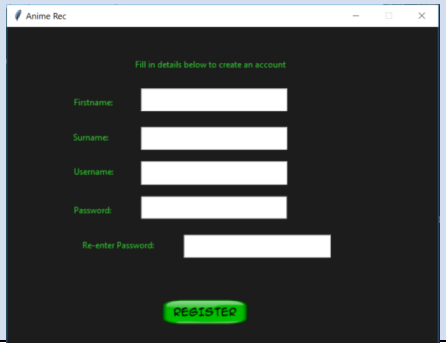
Creating the front-end for the Register screen

Comment [MR91]: Hard coding the GUI for the register screen (OOP and Tkinter GUI building)

```
class register(tk.Frame): # Declaring register class which inherits from Tkinters Frame class (Creating Register screen)
    def __init__(self, parent, controller):# Constructor method for register class
        tk.Frame.__init__(self, parent, bg='grey11') # Initialising Tkinters Frame class
        self.controller = controller # Assigning controller as a variable of the register class
        ## CREATING AND DISPLAYING REGISTER TITLE LABEL WIDET ##
        reg_title = tk.Label(self,text='Fill in details below to create an account', background='grey11', foreground='limegreen')
        reg_title.place(relx=0.21, rely=0.07, height=29, width=318)
        ## CREATING AND DISPLAYING THE USERNAME LABEL WIDGET ##
        reg_username = tk.Label(self, text='Username:' , background='grey11', foreground='limegreen')
        reg_username.place(relx=0.13, rely=0.35, height=29, width=87)
        ## CREATING AND DISPLAYING THE FIRSTNAME LABEL WIDGET ##
        reg_firstname = tk.Label(self, text='Firstname:', background='grey11', foreground='limegreen')
        reg_firstname.place(relx=0.13, rely=0.17, height=29, width=85)
        ## CREATING AND DISPLAYING THE SURNAME LABEL WIDGET ##
        reg_surname = tk.Label(self, text='Surname:' , background='grey11', foreground='limegreen')
        reg_surname.place(relx=0.13, rely=0.26, height=29, width=78)
        ## CREATING AND DISPLAYING THE PASSWORD LABEL WIDGET ##
        reg_pass = tk.Label(self,text='Password:' , background='grey11', foreground='limegreen')
        reg_pass.place(relx=0.13, rely=0.45, height=29, width=84)
        ## CREATING AND DISPLAYING THE RE-ENTER PASSWORD LABEL WIDGET ##
        reg_pass2 = tk.Label(self, text='Re-enter Password:' , background='grey11', foreground='limegreen')
        reg_pass2.place(relx=0.13, rely=0.54, height=29, width=156)
        ## CREATING AND DISPLAYING THE FIRSTNAME ENTRY WIDGET ##
        self.reg_firstname_entry = tk.Entry(self,cursor="ibeam")
        self.reg_firstname_entry.place(relx=0.31, rely=0.16, relheight=0.06, relwidth=0.34)
        ## CREATING AND DISPLAYING THE SURNAME ENTRY WIDGET ##
        self.reg_surname_entry = tk.Entry(self,cursor="ibeam")
        self.reg_surname_entry.place(relx=0.31, rely=0.26, relheight=0.06, relwidth=0.34)
        ## CREATING AND DISPLAYING THE USERNAME ENTRY WIDGET ##
        self.reg_username_entry = tk.Entry(self,cursor="ibeam")
        self.reg_username_entry.place(relx=0.31, rely=0.35, relheight=0.06, relwidth=0.34)
        ## CREATING AND DISPLAYING THE PASSWORD ENTRY WIDGET ##
        self.reg_pass_entry = ttk.Entry(self,show="*" ,cursor="ibeam")
        self.reg_pass_entry.place(relx=0.31, rely=0.44, relheight=0.06, relwidth=0.34)
        ## CREATING AND DISPLAYING THE RE-ENTER PASSWORD ENTRY WIDGET ##
        self.reg_pass2_entry = tk.Entry(self,show="*" ,cursor="ibeam")
        self.reg_pass2_entry.place(relx=0.41, rely=0.54, relheight=0.06, relwidth=0.34)
        ## CREATING AND DISPLAYING THE REGISTER BUTTON ##
        self.reg_butt = tk.Button(self, text='Register', borderwidth=0,background ='grey11', activebackground='grey11', foreground='li
        self.reg_butt.place(relx=0.36, rely=0.71, height=35, width=120)
```

Register screen front-end testing

Test Case	Check whether the register screen is displayed correctly
Expected Result	
Actual Result	

Modification	Created an image for the register button:
	 
	`reg_img = tk.PhotoImage(file='register_button.png') # Adding image to register button`
	`self.reg_butt = tk.Button(self, image=reg_img, borderwidth=0,background ='grey11', act`
	`self.reg_butt.image=reg_img # Keeping a reference of the image`
	`self.reg_butt.place(relx=0.36, rely=0.71, height=35, width=120)`
	Register screen after modification:
	 

Comment [MR92]: Black box and GUI testing for the register screen front end

Comment [MR93]: Modifying register button to match the anime theme

Adding functionality to the Register button

1. Bind a function to create an account to the Register button:

```
self.reg_butt = tk.Button(self, image=reg_img, command = self.create,
```

Code
the function to create an account. The function must validate the entered registration details and add them to the database if they are valid: thus creating an account.

Comment [MR94]: SQL query

```
def create(self):
    username=self.reg_username_entry.get() # Gets the entered username from the username entry widget
    with sqlite3.connect("Accounts.db") as  db: # Connects to the Accounts database
        cursor = db.cursor() #  Creates cursor object to traverse through the database
        findUser = ("SELECT * FROM user WHERE username = ?") # SQL query to check if that username already exists
        cursor.execute(findUser,[(username)])
    # If username exists execute userTaken() (pop-up window that prompts that the username is taken/ already exists)
    if cursor.fetchall():
        self.userTaken()
    # Else if username is not taken then gets all the values from the other entry widgets
    else:
        firstname=self.reg_firstname_entry.get()
        surname=self.reg_surname_entry.get()
        password=self.reg_pass_entry.get()
        password1=self.reg_pass2_entry.get()
        if password != password1: # VALIDATION - checks to see whether the initial password matches the re-entered password
            self.passMissMatch() # If they dont match then it executes passMissMatch function
            self.controller.show_frame(Register)
        ## SQL query to Write users account details to the Accounts database
        insertData = '''INSERT INTO user (username,firstname,surname,password)
        VALUES(?,?,?,?)'''
        cursor.execute(insertData,[(username),(firstname), (surname), (password)])
        db.commit() # Makes changes made to the database permanent
        self.accCreated() # Executes the accCreated() function
```

2. **VALIDATION** - Create the pop-up window for the case when the username is taken/account already exists. (**self.userTaken()**)

Comment [MR95]: Validation

```
def userTaken(self): # userTaken function, executed if the username already exists in the database
        self.popup = tk.Toplevel() #Creates user taken pop-up window
        self.popup.wm_title("!USER TAKEN!") # Window title
        ## CREATING AND DISPLAYING THE USER TAKEN MESSAGE LABEL WIDGET ##
        label = ttk.Label(self.popup,text="User already exists!", font =NORM_FONT)
        label.pack(side='top', expand=True)
        ## CREATING AND DISPLAYING THE TRY AGAIN BUTTON WIDGET##
        button = ttk.Button(self.popup, text="Try Again", command=self.r2Register)
        button.pack(side='bottom', expand=True)
        ## CREATING AND DISPLAYING THE HOME BUTTON WIDGTET ##
        button1 = ttk.Button(self.popup, text="Home", command=self.r2StartPage)
        button1.pack(side='bottom', expand=True)

        ## CENTERING THE POP-UP WINDOW ##
        width_of_window = 250
        height_of_window = 150
        screen_width = self.popup.winfo_screenwidth()
        screen_height = self.popup.winfo_screenheight()
        x_coordinate = (screen_width//2) - (width_of_window//2)
        y_coordinate = (screen_height//2) - (height_of_window//2)
        self.popup.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window, x_coordinate, y_coordinate))
        self.popup.mainloop() ## User taken pop-up window loop
```

3. **VALIDATION** – Create the pop up window for the case when the password and the re-entered password do not match. (**self.passMissMatch()**)

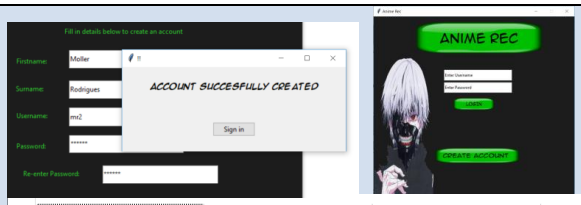
Comment [MR96]: Validation

```
def passMissMatch(self): # passMissMatch function, executed if the two entered passwords by the the user do not match
        self.popup = tk.Toplevel() #  Creates password mismatch pop-up window
        self.popup.wm_title("! PASSWORD ERROR !") # Window title

        ## CREATING AND DISPLAYING THE ERROR MESSAGE LABEL WIDGET ##
        label = ttk.Label(self.popup,text="Passwords do not match!", font =NORM_FONT)
        label.pack(side='top', expand=True)
        ## CREATING AND DISPLAYING THE TRY AGIAN BUTTON WIDGET ##
        button = ttk.Button(self.popup, text="Try Again", command=self.r2Register)
        button.pack(side='bottom', expand=True)
        ## CREATING AND DISPLAYING THE HOME BUTTON WIDGET ##
        button1 = ttk.Button(self.popup, text="Home", command=self.r2StartPage)
        button1.pack(side='bottom', expand=True)

        ##  CENTERING THE POP-UP WINDOW ##
        width_of_window = 400
        height_of_window = 150
        screen_width = self.popup.winfo_screenwidth()
        screen_height = self.popup.winfo_screenheight()
        x_coordinate = (screen_width//2) - (width_of_window//2)
        y_coordinate = (screen_height//2) - (height_of_window//2)
        self.popup.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window, x_coordinate, y_coordinate))
        self.popup.mainloop() # Password mismatch pop-up window main loop
```

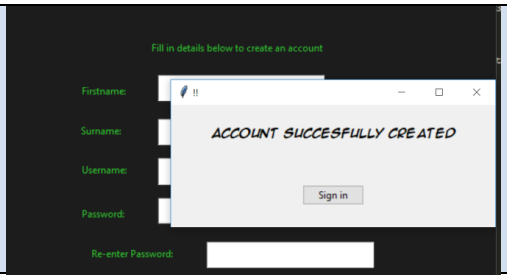
4. Create pop up window for the case when the account is successfully created. (self.accCreated())

```
def accCreated(self): # accCreated function, executed in the case when the account is succesfully created
        self.popup = tk.Toplevel() # Creates the account created pop up window
        self.popup.wm_title("!!") # Setting the windows title

        ## CREATING AND DISPLAYING THE SUCCESS MESSAGE, LABEL WIDGET ##
        label = ttk.Label(self.popup,text="Account succesfully created", font =NORM_FONT)
        label.pack(side='top', expand=True)
        ## CREATING AND DISPLAYING THE SIGN IN BUTTON WIDGET ##
        button = ttk.Button(self.popup, text="Sign in", command=self.r2StartPage)
        button.pack(side='bottom', expand=True)

        ## CENTERING THE POP-UP WINDOW ##
        width_of_window = 400
        height_of_window = 150
        screen_width = self.popup.winfo_screenwidth()
        screen_height = self.popup.winfo_screenheight()
        x_coordinate = (screen_width//2) - (width_of_window//2)
        y_coordinate = (screen_height//2) - (height_of_window//2)
        self.popup.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window, x_coordinate, y_coordinate))
        self.popup.mainloop() # Account created pop up window main loop
```

Register button testing

Test case	**Check Register button functionality in the case of valid details**
Test Data	Firstname: Moller **Surname:** Rodrigues Username: mr2 Password: snow24 Re-enter password: snow24
Expected Result	• Account successfully created and added to the database Accounts.db. • Return to Home screen
Actual Result	 1 1 admin Moller Rodrigues m 2 2 mr2 Moller Rodrigues snow24
Modification	N/A

Comment [MR97]: Black box and robustness testing

Test case	**Check Register button functionality in the case of blank entry fields**
Test Data	Firstname: Surname: Username: Password: Re-enter password:
Expected Result	• Display register fail prompt with error – 'Blank fields'
Actual Result	 PROBLEM – creates an account with each value as literally nothing ('').
Modification	• Add validation to check to see whether the entry fields are empty updated create account function

Comment [MR98]: Black box and robustness testing

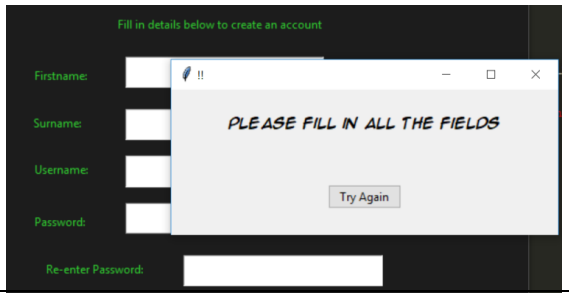
Comment [MR99]: Error with fix below

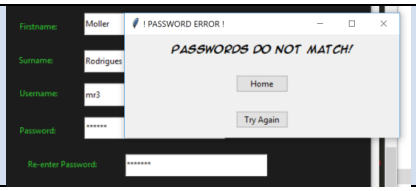
```
    def create(self):
        # Gets the entered values for all entery widgets
        username=self.reg_username_entry.get()
        firstname=self.reg_firstname_entry.get()
        surname=self.reg_surname_entry.get()
        password=self.reg_pass_entry.get()
        password1=self.reg_pass2_entry.get()
        values = [username, firstname, surname, password, password1]

        # Check to see if a entry widget is empty
        empty = True
        count = 0

        while empty == True and count != 5: # While loop that sets empty to True if a value is blank for an entry widget
            if len(values[count]) > 0:
                empty = False
            else:
                empty = True
                break
            count+=1

        if empty == True: # If one or more entry widgets are blank the appropriate prompt message is displayed
            self.fill_in_all_details()
        else: # Else if no entry widget is blank continue
            with sqlite3.connect("Accounts.db") as  db: # Connects to the Accounts database
                cursor = db.cursor() #  Creates cursor object to traverse through the database
                findUser = ("SELECT * FROM user WHERE username = ?") # SQL query to check if that username already exists
                cursor.execute(findUser,[(username)])
            # If username exists execute userTaken() (pop-up window that prompts that the username is taken/ already exists)
            if cursor.fetchall():
                self.user_taken()
        else: # Else if no entry widget is blank continue
            with sqlite3.connect("Accounts.db") as  db: # Connects to the Accounts database
                cursor = db.cursor() #  Creates cursor object to traverse through the database
                findUser = ("SELECT * FROM user WHERE username = ?") # SQL query to check if that username already exists
                cursor.execute(findUser,[(username)])
            # If username exists execute userTaken() (pop-up window that prompts that the username is taken/ already exists)
            if cursor.fetchall():
                self.user_taken()
            # Else if username is not taken then gets all the values from the other entry widgets
            else:
                if password != password1: # VALIDATION - checks to see whether the initial password matches the re-entered password
                    self.passs_miss_match() # If they dont match then it executes passMissMatch function
                    self.controller.show_frame(register)
                ## SQL query to Write users account details to the Accounts database
                insertData = '''INSERT INTO user (username,firstname,surname,password)
                VALUES(?,?,?,?)'''
                cursor.execute(insertData,[(username),(firstname), (surname), (password)])
                db.commit() # Makes changes made to the database permanent
                self.acc_created() # Executes the accCreated() function
```

- VALIDATION – Creating prompt for empty entry fields

```
def fill_in_all_details(self): # function that is executed in the case of one or more empty entry fields
        self.popup = tk.Toplevel() # Creates the empty entry  pop up window
        self.popup.wm_title("!!") # Setting the windows title

        ## CREATING AND DISPLAYING THE ERROR MESSAGE, LABEL WIDGET ##
        label = ttk.Label(self.popup,text="Please fill in all the fields", font =NORM_FONT)
        label.pack(side='top', expand=True)
        ## CREATING AND DISPLAYING THE SIGN IN BUTTON WIDGET ##
        button = ttk.Button(self.popup, text="Try Again", command=self.r2_register)
        button.pack(side='bottom', expand=True)

        ## CENTERING THE POP-UP WINDOW ##
        width_of_window = 400
        height_of_window = 150
        screen_width = self.popup.winfo_screenwidth()
        screen_height = self.popup.winfo_screenheight()
        x_coordinate = (screen_width//2) - (width_of_window//2)
        y_coordinate = (screen_height//2) - (height_of_window//2)
        self.popup.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window, x_coordinate, y_coordinate))
        self.popup.mainloop() # fill_in_all_deatils pop up window main loop
```

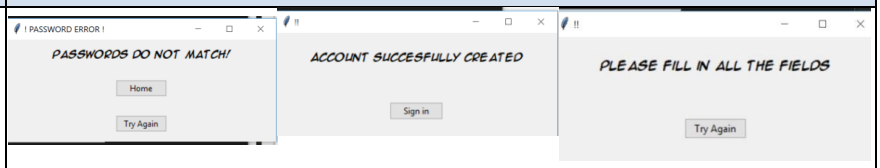
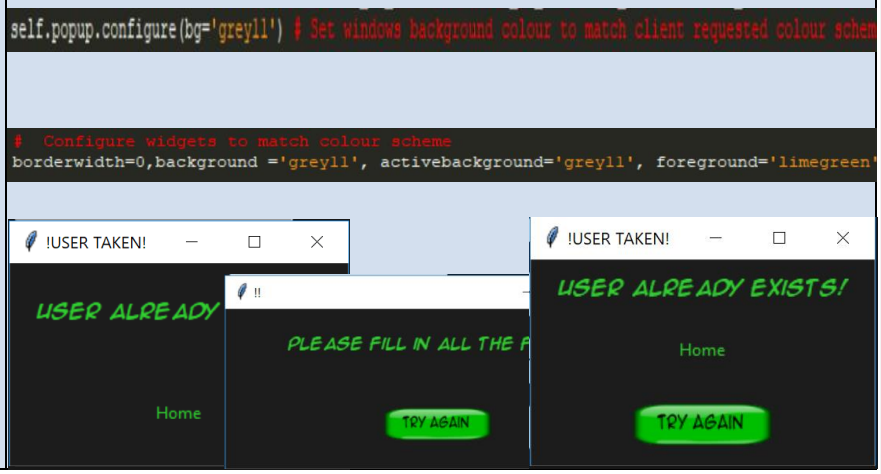
After fix:



Test Case	**Check Register button functionality in the case of mismatching passwords**
Test Data	Firstname: Moller **Surname:** Rodrigues Username: mr3 Password: snow24 Re-enter password: rain64
Expected result	Display register fail prompt with error – 'Passwords do not match'
Actual result	
Modification	N/A

Comment [MR100]: Black box and robustness testing

Register screen front – end form testing

Test Case	**Check to see if the form of the register screen and the attributes associated with the register screen conform with the programs aesthetic requirements**
Expected result	All widgets/frames/pop-up windows should match with the register screens colour scheme
Actual result	
Modification	Modification: 

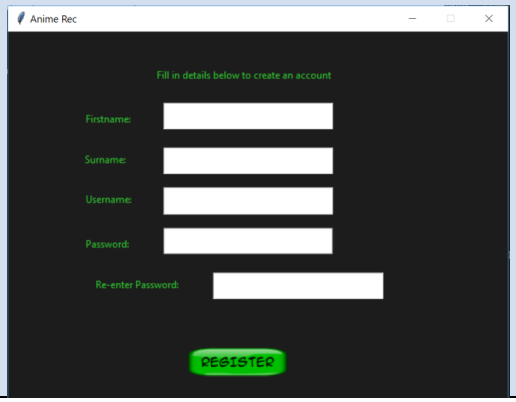
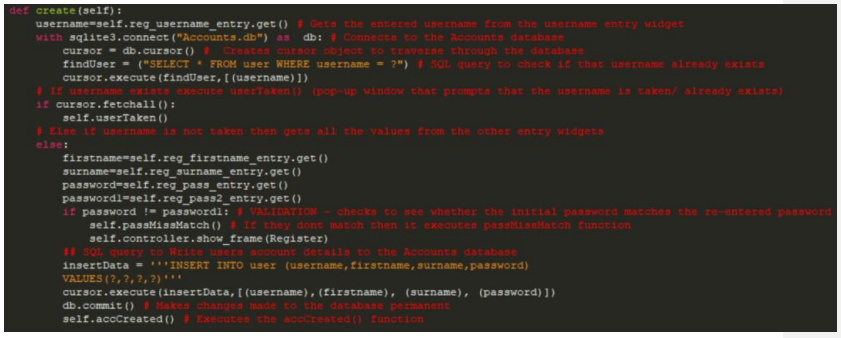
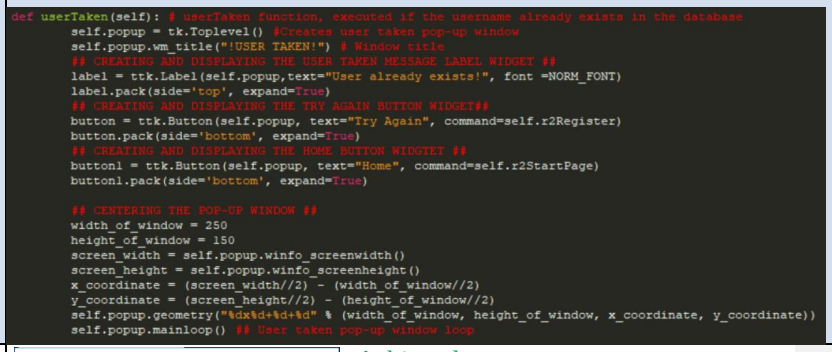
Comment [MR101]: GUI and usability testing

Comment [MR102]: Error with fix below

01/04/2018 – Register Screen prototype interview with client, Josh Santillan

Comment [MR103]: Client and success criteria review for register screen

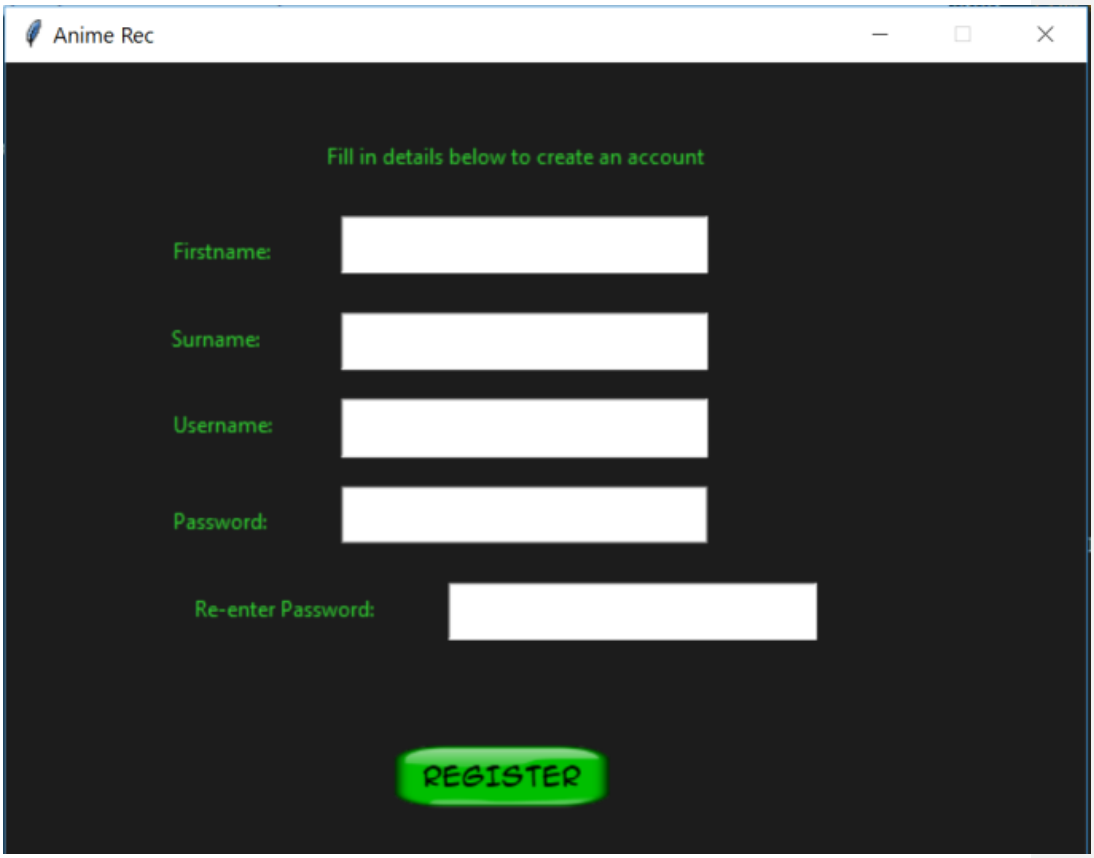
NOTE Referring to the requirements/success criteria created in the analysis section

Success criteria / requirement	Achieved / Not Achieved (with evidence)
Register Screen – A screen that allows the user to create an account	Achieved 
User accounts system- - Function which allows users to create an account and store these to permanent data storage.	Achieved 
VALIDATION	Achieved 
Consistent colour scheme and anime theme 	 Achieved Font – ANIME ACE Consistent colour scheme – Black/green

FINAL REGISTER SCREEN PROTOTYPE

Modifications from first prototype:

- Created register image for register button
- Created validation to check if any entry widgets are blank



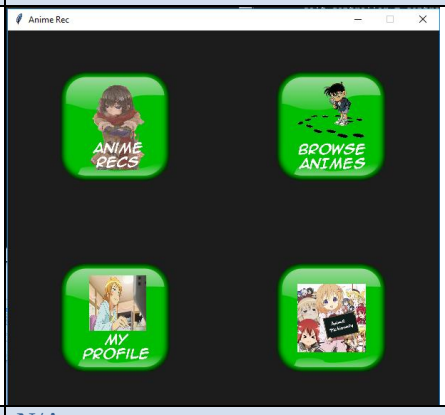
Josh Santillan, Lead client

02/04/2018 – Creating the Menu screen

Creating the front end of the menu screen

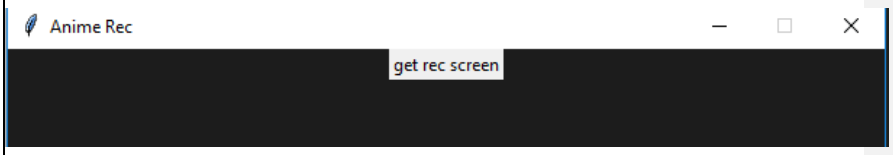
```
# Declaring the menu class whioch inherits from Tkinter's Frame class
class menu(tk.Frame):

    # Declaring menu's constructor method which is called when the class in instantiated
    def __init__(self, parent, controller):

        # Initialises Tkinter's Frame class
        tk.Frame.__init__(self, parent, bg='grey11') # Setting the background of the frame to 'grey11'

        # Storking controller as a varioable of the menu class
        self.controller = controller

        # Loading the image for the Get rec button
        get_rec_img = tk.PhotoImage(file="get_rec_button.png")
        # Shrinks the image
        get_rec_img = get_rec_img.subsample(3)
        # Crerating the get_rec_button with get_rec_img as its image.
        get_rec_button = tk.Button(self,image=get_rec_img, borderwidth = 0, background = "grey11")
        # Keeping a reference of get_rec_button's image
        get_rec_button.image=get_rec_img
        # Displaying the Button widget on the window
        get_rec_button.grid(row=0,sticky="nsew")


        # Loading the image for the browse button
        browse_button_img = tk.PhotoImage(file="browse_button.png")
        # Shrinks the image
        browse_button_img = browse_button_img.subsample(3)
        # Crerating the browse_button with browse_button_img as its image.
        browse_button = tk.Button(self, image= browse_button_img, borderwidth = 0, background = "grey11")
        # Keeping a reference of browse_button's image
        browse_button.image=browse_button_img
        # Displaying the Button widget on the window
        browse_button.grid(row=0,column=1, sticky="nsew")

        # Loading the image for the profile button
        profile_button_img = tk.PhotoImage(file="profile_button.png")
        # Shrinks the image
        profile_button_img = profile_button_img.subsample(3)
        # Crerating the profile_button with profile_button_img as its image.
        profile_button = tk.Button(self,image=profile_button_img, borderwidth = 0, background = "grey11")
        # Keeping a reference of profile_button's image
        profile_button.image=profile_button_img
        # Displaying the Button widget on the window
        profile_button.grid(row=1,sticky="nsew")

        # Loading the image for the pictionary button
        pictionary_button_img = tk.PhotoImage(file="pictionary_button.png")
        # Shrinks the image
        pictionary_button_img = pictionary_button_img.subsample(3)
        # Crerating the pictionary_button with pictionary_button_img as its image.
        pictionary_button = tk.Button(self,image=pictionary_button_img, borderwidth = 0, background = "grey11")
        # Keeping a reference of pictionary_button's image
        pictionary_button.image=pictionary_button_img
        # Displaying the Button widget on the window
        pictionary_button.grid(row=1,column=1, sticky="nsew")

        # Configuring the priority of the widgets
        self.columnconfigure(0, weight=1)
        self.columnconfigure(1, weight=1)
        self.rowconfigure(0, weight=1)
        self.rowconfigure(1, weight=1)
```

Menu screen front end testing

Test case	Check whether the menu screen is displayed correctly
Expected Result	
Actual Result	
Modification	N/A

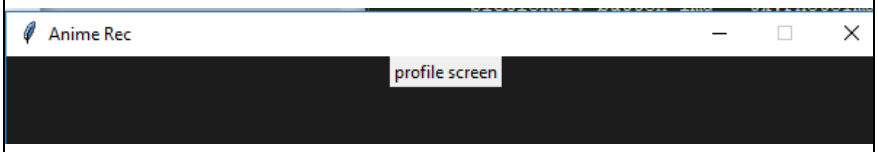
Adding functionality to the get rec button (get recommendation button)

```
# Crerating the get_rec_button with get_rec_img as its image and binding it with the function to display the get rec screen
get_rec_button = tk.Button(self,image=get_rec_img, borderwidth = 0,command=lambda: controller.show_frame(get_rec), background = "grey11")
```

Test case	Check functionality of The Get Recommendation button
Expected Result	Switch to Get Rec screen
Actual Result	
Modification	N/A

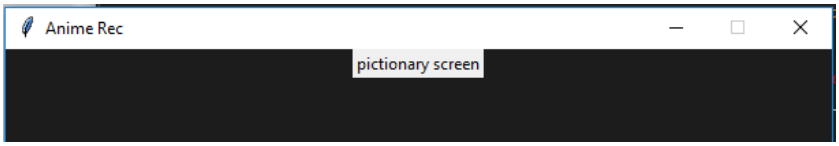
Adding functionality to the profile button

```
# Crerating the profile_button with profile_button_img as its image and binding it with the function to display the profile screen
profile_button = tk.Button(self,image=profile_button_img, borderwidth = 0,command=lambda: controller.show_frame(profile), background = "grey11")
```

Test case	Check functionality of The profile button
Expected Result	Switch to Profile screen
Actual Result	
Modification	N/A

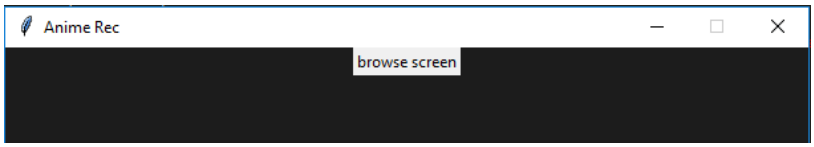
Adding functionality to the Pictionary button

```
# Crerating the pictionary_button with pictionary_button_img as its image and binding it with the function to display the pictionary screen
pictionary_button = tk.Button(self,image=pictionary_button_img, borderwidth = 0,command=lambda: controller.show_frame(pictionary), background = "
```

Test case	Check functionality of The Pictionary button
Expected Result	Switch to Pictionary screen
Actual Result	
Modification	N/A

Adding functionality browse button

```
# Crerating the browse_button with browse_button_img as its image and binding it with the function to display the browse screen
browse_button = tk.Button(self, image= browse_button_img, borderwidth = 0,command=lambda: controller.show_frame(browse), background = "grey11")
```

Test case	Check functionality of The Browse button
Expected Result	Switch to Browse screen
Actual Result	
Modification	N/A

03/04/2018 – Menu Screen prototype interview with client, Josh Santillan

Comment [MR104]: Client and success criteria review for the menu screen

Success criteria / requirement	Achieved / Not Achieved (with evidence)
Main Menu – A screen which will allow the user to navigate through to the different sections of the app.	Achieved As you can see from the menu screen I have indeed created a screen which will allow the user to navigate through to the different sections of the app through the use of buttons.
Anime theme and consistent colour theme 	Achieved To conform to the animes based theme I have re modelled the buttons as images which are relevant to what the button represents, for example the browse button has an image of a popular anime character, Detective Conan who in the image is looking for something thus appropriate for the browse button. 

Client Review

Josh Santillan – I am pleased with this prototype for the Menu screen and I am extremely happy with the image buttons which are consistent with the theme of our app. The only suggestion I can make is to

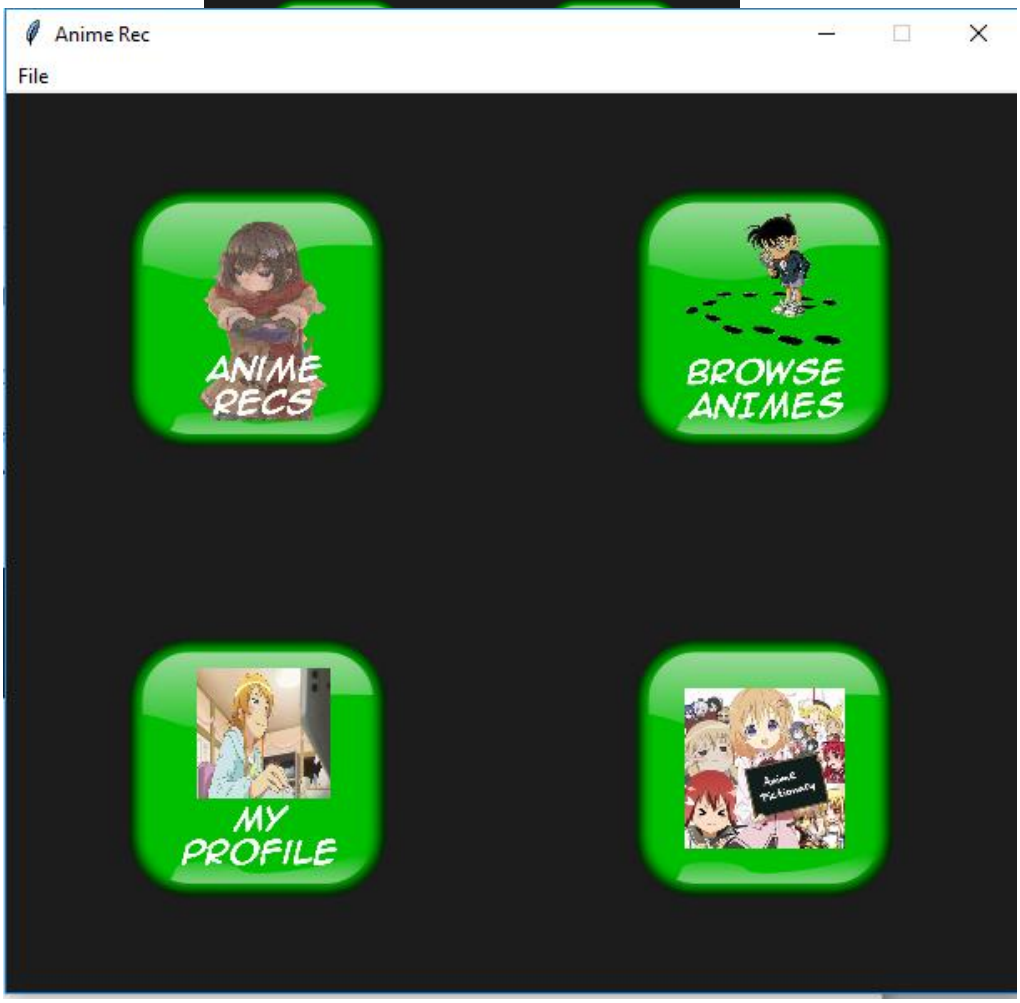
Add a navigation bar at the top of the window which will allow the user to log out o#r return to the Menu screen.

FINAL MENU SCREEN PROTOTYPE

Modifications from first prototype:

- Navigation bar

```
# Creating and displaying a navigation bar which will allow the user to logout and exit the app
menubar = tk.Menu(container)
filemenu = tk.Menu(menubar, tearoff = 0)
filemenu.add_command(label="Log out",command=lambda: self.show_frame(home) )
filemenu.add_separator()
filemenu.add_command(label="Exit",command=lambda: self.show_frame(menu))
menubar.add_cascade(label="File", menu=filemenu)
tk.Tk.config(self, menu=menubar)
```



Josh Santillan, Lead client

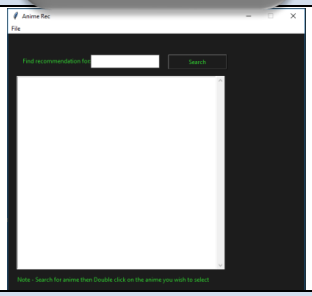
04/04/2018 – Creating the Get Recommendation screen

The Get rec (Get recommendation) screen is essentially the main section to our app as it is literally the core solution to the problem of recommending animes to user's given what they have previously watched and liked.

Creating the frontend of the Get Rec Screen

```
# Declaring the class for the Get Recommendation screen as get_rec, which inherits from Tkinter's Frame class.
class get_rec(tk.Frame):

    # Declaring the constructor method for the get_rec class, which runs w henb the class is instantiated
    def __init__(self, parent, controller):
        # Intialising Tkinter's Frame  class
        tk.Frame.__init__(self, parent, bg='greyll') #  Setyting backgroung to 'greyll'
        # Storing controller as a variable of the get_rec class
        self.controller = controller

        # Creating and displaying the label for the entry widget  which will be used so that the user can enter which anime they wish to find recommendations for
        self.getRec_label = tk.Label(self, text='Find recommendation for:', width=210, relief='flat', bg = 'greyll', fg='lime green', borderwidth=2,highlightbackground="lime green" )
        self.getRec_label.place(relx=0.00, rely=0.07, height=35, width=200)

        # Creating and displaying the Entry widget which will allow the user to input an anime
        self.search_entry = ttk.Entry(self)
        self.search_entry.place(relx=0.28, rely=0.08, relheight=0.05, relwidth=0.23)

        # Creating and displaying a button, which when clicked will output the search reesults of the user's input
        self.search_butt = tk.Button(self, text='Search', bg = 'greyll', fg='lime green', borderwidth=2,highlightbackground="lime green" )
        self.search_butt.place(relx=0.54, rely=0.08, height=30, width=120)



        # Creating and  displaying a frame to contain the list box
        frm = tk.Frame(self)
        frm.place(relx=0.03, rely=0.16, relheight=0.73, relwidth=0.7)

        # Creating and displaying a scroll bar  used to scroll up and down  the list box
        scrollbar = tk.Scrollbar(frm, orient="vertical", highlightcolor="#d9d9d9", highlightbackground="#d9d9d9")
        scrollbar.pack(side='right', fill='y')


        # Creating a displaying a List box widget within the same frame as the scroll bar
        self.listNodes = tk.Listbox(frm, width=100, yscrollcommand=scrollbar.set, font=("Helvetica", 12))
        self.listNodes.pack(expand=True, fill='y')
        scrollbar.config(command=self.listNodes.yview)

        # Creating and displaying a label to give the user some instructions as to how to use this feature.
        self.getRec_note = tk.Label(self, text='Note - Search for anime then Double click on the anime you wish to select', bg = 'greyll', fg='lime green', borderwidth=0)
        self.getRec_note.place(relx=0.02, rely=0.9, relheight=0.06, relwidth=0.67)
```

Get rec screen front end testing

Test Case	Check whether the Get Rec screen is displayed correctly
Expected result	
Actual result	
Modification	N/A

Adding functionality to the search button

In order to find a recommendation for an anime we require the user to select which anime they wish to find a recommendation for. However, for our recommender system to work we require the exact name of the anime as it is stored in our database. This means that it is inconvenient for the user as they may not know how to spell the names of certain animes.

In order to overcome this problem I decided to create a function which would output all the animes that contain the input of the user and output the results in a list box so that the user can select the anime they meant to search for.

```
# Creating and displaying a button, which when clicked will output the search reesults of the user's input
self.search_butt = tk.Button(self, text='Search', command= self.populate, bg = 'greyll', fg='lime green', b
self.search_butt.place(relx=0.54, rely=0.08, height=30, width=120)
```

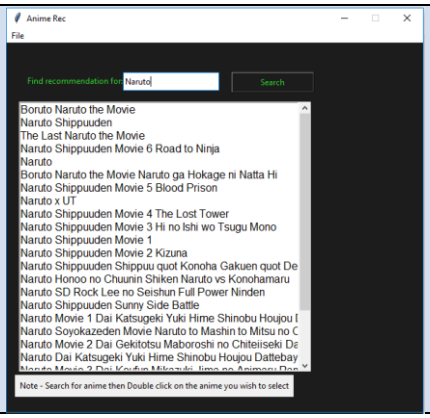
self.populate():

```
# function that is executed when the search button is clicked; which populates the list box with the search results
def populate(self):
    self.listNodes.delete(0, 'end')
    partial=string.capwords(self.search_entry.get())
    possibleSearch= get_possible_searches(partial)
    for i in range(0, len(possibleSearch)):
        self.listNodes.insert(i, possibleSearch[i])
```

get_possible_searches (partial): #LOCATED IN ANOTHER LOCAL SCRIPT

```
# function which is called to return all the animes whioch are contain the string that is passed to this function
def get_possible_searches(partial):
    possibleSearch=[]
    for name in all_anime_names:
        if partial in name:
            possibleSearch.append(name)
    return possibleSearch
```

Get recommendation screen search button testing

Test Case	**Check whether the search button widget is functional**
Test Steps	1.Type in 'Naruto' 2.Click on search
Expected result	The Listbox should be populated with animes that contain 'Naruto' in their name.
Actual Result	
Modification	N/A

Comment [MR105]: Usability testing

CREATING THE RECOMMENDER SYSTEM

Comment [MR106]: KNN algorithm

As covered in the analysis section there are two main ways of generating recommendations, these using content based recommendation system and/or a collaborative filtering based system.

Given my limited time frame I have chosen to go with a content based recommender system. To reiterate, a content based recommender system works by comparing each feature of an item to the features of another item and calculating a value representing how similar these two items are.

After some thorough research I have to the decision to use the k nearest neighbours (KNN) algorithm.

What is KNN and how does it work?

A k-nearest-neighbour algorithm, often abbreviated KNN, is an approach to data classification that estimates how likely a data point is to be a member of one group or the other depending on what group the data points nearest to it are in.

KNN is an example of a "lazy learner" algorithm, meaning that it does not build a model using the training set until a query of the data set is performed.

KNN is a data classification algorithm that attempts to determine what group a data point is in by looking at the data points around it.

An algorithm, looking at one point on a grid, trying to determine if a point is in group A or B, looks at the states of the points that are near it. The range is arbitrarily determined, but the point is to take a sample of the data. If the majority of the points are in group A, then it is likely that the data point in question will be A rather than B, and vice versa.

KNN is an example of a "lazy learner" algorithm because it does not generate a model of the data set beforehand. The only calculations it makes are when it is asked to poll the data point's neighbours. This makes KNN very easy to implement for data mining.

Recommender system

Comment [MR107]: Recommender system MAIN SOLUTION TO MY PROBLEM:
- Anime csv file formatting (reading/writing/updating/modifying)
- Normalising data
- Using KNN algorithm on my formatted data
- Creating functions to get the results of the KNN in an appropriate way

The code for the recommender system below is located in another external python script which is imported into the main script. I chose to do this to keep my code efficient and maintainable.

```
from sklearn.preprocessing import MaxAbsScaler # Imports sklearn's MaxAbsScaler module which Scales each feature by its maximum absolute value.
from sklearn.neighbors import NearestNeighbors # Imports sklearn's NearestNeighbors module which provides functionality for unsupervised and supervised neighbors-based learning methods
import numpy as np # Imports the Numpy library which adds support for large, multi-dimensional arrays and matrices
import pandas as pd # Imports the pandas library which is used for data manipulation and analysis
import re #  Imports the re library which is used to read/write and match regular expressions



## STEP 1 - Formatting the csv file which contains all the anime data  in a formaat that the K-Nearest-Neighbour (KNN) algorithm can understand ##

# Opening the anime.csv file and storing it in a pandas dataframe
anime = pd.read_csv("anime.csv")

# Returns the first row  of the anime dataframe, i.e the headers
anime.head()

# Manually setting the number of episodes for certain animes which do not have a value for number of episodes
anime.loc[(anime["genre"]=="Hentai") & (anime["episodes"]=="Unknown"),"episodes"] = "1"
anime.loc[(anime["type"]=="OVA") & (anime["episodes"]=="Unknown"),"episodes"] = "1"
anime.loc[(anime["type"] == "Movie") & (anime["episodes"] == "Unknown")] = "1"
known_animes = {"Naruto Shippuuden":500, "One Piece":784,"Detective Conan":854, "Dragon Ball Super":86,
                "Crayon Shin chan":942, "Yu Gi Oh Arc V":148,"Shingeki no Kyojin Season 2":25,
                "Boku no Hero Academia 2nd Season":25,"Little Witch Academia TV":25}
for k,v in known_animes.items():
    anime.loc[anime["name"]==k,"episodes"] = v

# Replacing the remaining animes which do not have a value for number of episodes with the median number of episodes of the entire dataset
anime["episodes"] = anime["episodes"].map(lambda x:np.nan if x=="Unknown" else x)
anime["episodes"].fillna(anime["episodes"].median(),inplace = True)

# Converting all the rating values to floats and replacing all animes which do not have a rating with the median rating of the enitre data set
anime["rating"] = anime["rating"].astype(float)
anime["rating"].fillna(anime["rating"].median(),inplace = True)

# Converting categorical variable of type to an indicator variable
pd.get_dummies(anime[["type"]]).head()

# Converting all the values of members to the float data type
anime["members"] = anime["members"].astype(float)

# Creating a dataframe with all the anime features as the headings
anime_features = pd.concat([anime["genre"].str.get_dummies(sep=","),pd.get_dummies(anime[["type"]]),anime[["rating"]],anime[["members"]],anime["episodes"]],axis=1)
anime["name"] = anime["name"].map(lambda name:re.sub('[^A-Za-z0-9]+', " ", name))
anime_features.head()
anime_features.columns

## STEP 2 - Executing the KNN algorithm on the anime features dataframe

# Instantiating sklearn's MaxAbsScaler which scales each feature by its maximum absolute value
max_abs_scaler = MaxAbsScaler()

# Fit to data and then transforms it
anime_features = max_abs_scaler.fit_transform(anime_features)

# Runs KNN's ball tree algorithm on the anime_features dataframe and returns the the 6 most similar animes to every single anime in the data frame
nbrs = NearestNeighbors(n_neighbors=6, algorithm='ball_tree').fit(anime_features)
distances, indices = nbrs.kneighbors(anime_features)



# Function which returns the index of an anime
def get_index_from_name(name):
    return anime[anime["name"]==name].index.tolist()[0]

all_anime_names = list(anime.name.values)

# Function which gets the full anime name from a partial input
def get_id_from_partial_name(partial):
    for name in all_anime_names:
        if partial in name:
            print(name,all_anime_names.index(name))

# Function which return the 5 most  similar animes to a given anime using the results from the KNN algorithm
def get_rec(query=None,id=None):
    if id:
        for id in indices[id][1:]:
            print(anime.ix[id]["name"])
    if query:
        found_id = get_index_from_name(query)
        for id in indices[found_id][1:]:
            print(anime.ix[id]["name"])
```

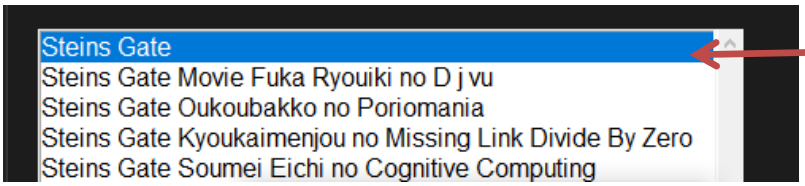
Integrating the recommender system with the front end of the get rec screen

Adding a function to the List Box Nodes

Comment [MR108]: Creating and populating a list box with on click commands

```
# When a node from the List box is double clicked then the OnDouble function is executed
self.listNodes.bind("<Double-Button-1>", self.OnDouble)
```

(A list box node is a value in the list box)



So when a list node is double clicked the **OnDouble** function is executed.

OnDouble function

```
# Function which displays pop-up window which displays the generated recommendations to the user
def OnDouble(self, event):
    # Creating a pop up window
    self.popup = tk.Tk()
    self.popup.wm_title("Anime Recommendations!")

    # Get the anime that was selected from the List Box widget
    widget = event.widget
    selection=widget.curselection()
    value = widget.get(selection[0])

    # Storing the results of the recommender system in the list anime
    anime = get_rec(value)
    t_text = "Recommendations for "+value

    # Creating and displaying a label widget which tells the user th anime for which the recommendations are for
    title = ttk.Label(self.popup, text = t_text )
    title.grid(row=0, sticky = "ne")


    # Displays each recommendation with their respective link for more information
    a = ttk.Label(self.popup,text=anime[0])
    a.grid(row=1, column=0)
    b = ttk.Button(self.popup,text='Click for more info', command=lambda:self.openLink(anime[0]))
    b.grid(row=1, column=1)

    a1 = ttk.Label(self.popup,text=anime[1])
    a1.grid(row=2, column=0)
    b = ttk.Button(self.popup,text='Click for more info',command=lambda:self.openLink(anime[1]))
    b.grid(row=2, column=1)

    a2 = ttk.Label(self.popup,text=anime[2])
    a2.grid(row=3, column=0)
    b = ttk.Button(self.popup,text='Click for more info',command=lambda:self.openLink(anime[2]))
    b.grid(row=3, column=1)

    a3 = ttk.Label(self.popup,text=anime[3])
    a3.grid(row=4, column=0)
    b = ttk.Button(self.popup,text='Click for more info',command=lambda:self.openLink(anime[3]))
    b.grid(row=4, column=1)

    a4 = ttk.Label(self.popup,text=anime[4])
    a4.grid(row=5, column=0)
    b = ttk.Button(self.popup,text='Click for more info',command=lambda:self.openLink(anime[4]))
    b.grid(row=5, column=1)

    # Pop up window main loop
    self.popup.mainloop()
```

Get_rec function from the recommender system script above

OpenLink Function defined below

openLink function

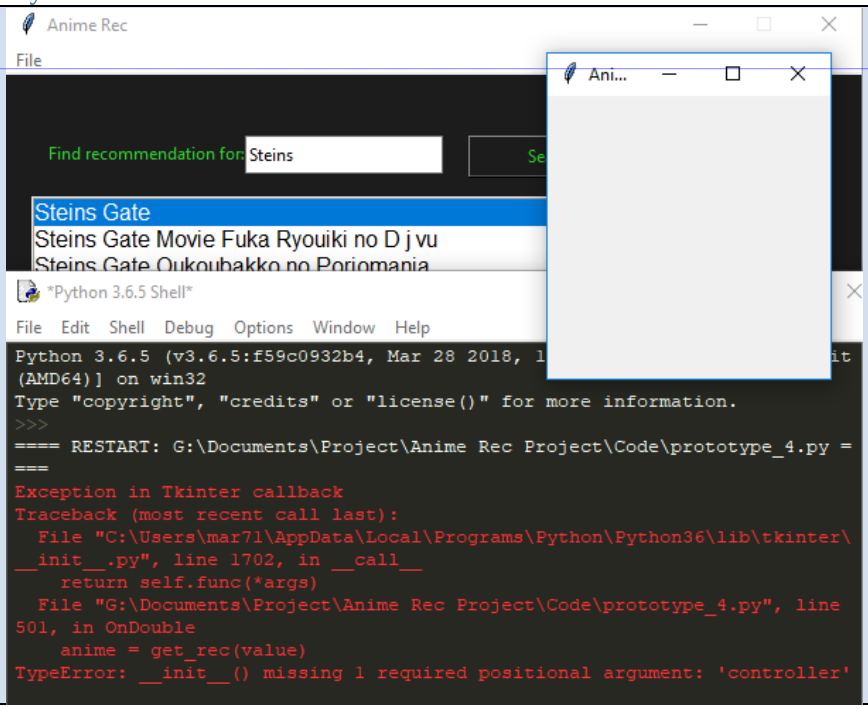
```
# function that open the myanimelist.net website for the given anime
def openLink(self, anime):
    try:
        a_id=get_ID(anime)
        url = "https://myanimelist.net/anime/"+a_id[0][0]
        webbrowser.open(url)
    except IndexError:
        print("Link broken")
```

This function is executed when the 'Click for more info' button next to the generated recommendations is clciked.

It takes the the name of the anime linked to the 'click for more info' button and creates a myanimelist.net url link specific for that anime.

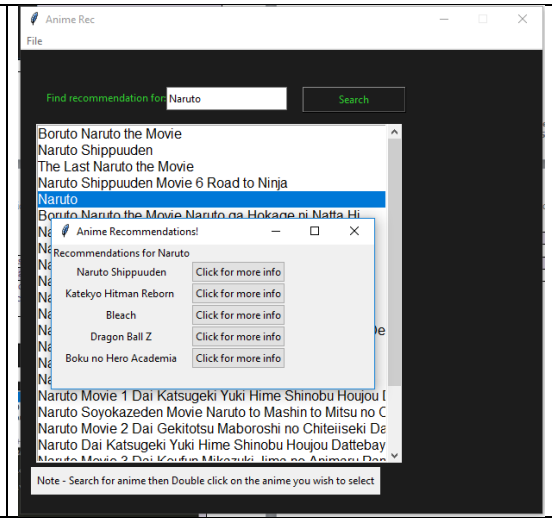
Then using Python's webbrowser module I open the link in the user's internet browser.

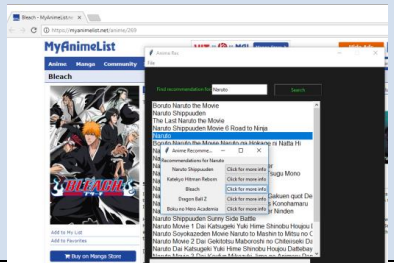
Recommender system testing

Test Case	**Check whether recommendation is generated when an anime is selected.**
Test steps	Double click on the anime called 'Naruto
Expected result	A new pop up window shgould be displayed with 5 animes that are similar to 'Naruto. Also at the side of each recommendation their should be a button that says clcik fro more info.
Actual result	

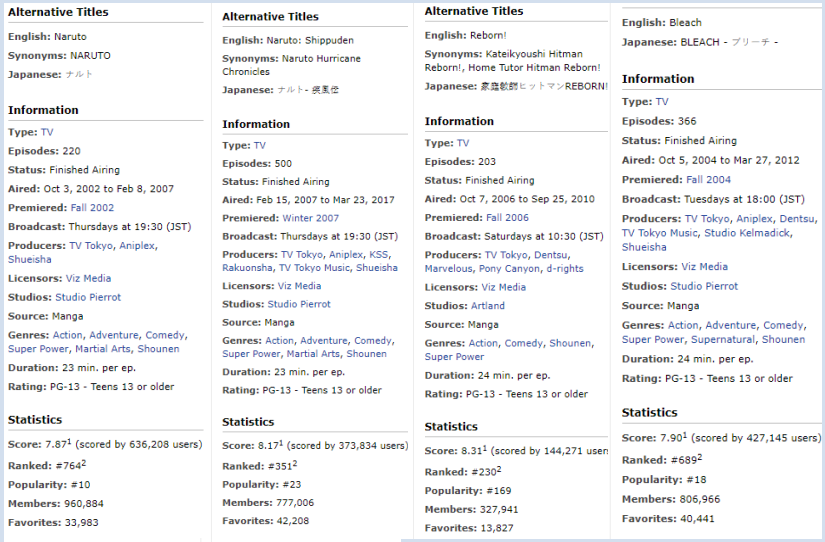
Modification	Problem – used the same name for the function and class (get_rec)
	Fix:
	`from c2 import get_ID, get_possible_searches,get_similar_animes,get_index`
	`anime = get_similar_animes(value)`

Comment [MR109]: Error with fix below



Test Case	**Check whether the click for more info button is functional**
Test Steps	.Click on the more info button for 'Bleach'
Expected result	The my anime List website for Bleach should open up in the users default browser.
Actual results	
Modifications	N/A

Test Case	Evaluate the recommender systems accuracy
Test Steps	Check each feature of the generated recommendations and compare them to the anime that the user entered to find recommendations for.
Expected result	The generated recommendations should have a lot of the same features as the annime thatr was searched for
Actual results	The recommender system results for the anime Naruto: • Naruto Shippuuden • Katekyo Hitman Reborn • Bleach • Dragon Ball z • Boku no Hero Academia
Modifications	N/A



Alternative Titles
English: Naruto
Synonyms: NARUTO
Japanese: ナルト

Information
Type: TV
Episodes: 220
Status: Finished Airing
Aired: Oct 3, 2002 to Feb 8, 2007
Premiered: Fall 2002
Broadcast: Thursdays at 19:30 (JST)
Producers: TV Tokyo, Aniplex, Shueisha
Licensors: Viz Media
Studios: Studio Pierrot
Source: Manga
Genres: Action, Adventure, Comedy, Super Power, Martial Arts, Shounen
Duration: 23 min. per ep.
Rating: PG-13 - Teens 13 or older

Statistics
Score: 7.87[1] (scored by 636,208 users)
Ranked: #764[2]
Popularity: #10
Members: 960,884
Favorites: 33,983

Alternative Titles
English: Naruto: Shippuden
Synonyms: Naruto Hurricane Chronicles
Japanese: ナルト- 疾風伝

Information
Type: TV
Episodes: 500
Status: Finished Airing
Aired: Feb 15, 2007 to Mar 23, 2017
Premiered: Winter 2007
Broadcast: Thursdays at 19:30 (JST)
Producers: TV Tokyo, Aniplex, KSS, Rakuonsha, TV Tokyo Music, Shueisha
Licensors: Viz Media
Studios: Studio Pierrot
Source: Manga
Genres: Action, Adventure, Comedy, Super Power, Martial Arts, Shounen
Duration: 23 min. per ep.
Rating: PG-13 - Teens 13 or older

Statistics
Score: 8.17[1] (scored by 373,834 users)
Ranked: #351[2]
Popularity: #23
Members: 777,006
Favorites: 42,208

Alternative Titles
English: Reborn!
Synonyms: Kateikyoushi Hitman Reborn!, Home Tutor Hitman Reborn!
Japanese: 家庭教師ヒットマンREBORN!

Information
Type: TV
Episodes: 203
Status: Finished Airing
Aired: Oct 7, 2006 to Sep 25, 2010
Premiered: Fall 2006
Broadcast: Saturdays at 10:30 (JST)
Producers: TV Tokyo, Dentsu, Marvelous, Pony Canyon, d-rights
Licensors: Viz Media
Studios: Artland
Source: Manga
Genres: Action, Comedy, Shounen, Super Power
Duration: 24 min. per ep.
Rating: PG-13 - Teens 13 or older

Statistics
Score: 8.31[1] (scored by 144,271 users)
Ranked: #230[2]
Popularity: #169
Members: 327,941
Favorites: 13,827

English: Bleach
Japanese: BLEACH - ブリーチ -

Information
Type: TV
Episodes: 366
Status: Finished Airing
Aired: Oct 5, 2004 to Mar 27, 2012
Premiered: Fall 2004
Broadcast: Tuesdays at 18:00 (JST)
Producers: TV Tokyo, Aniplex, Dentsu, TV Tokyo Music, Studio Kelmadick, Shueisha
Licensors: Viz Media
Studios: Studio Pierrot
Source: Manga
Genres: Action, Adventure, Comedy, Super Power, Supernatural, Shounen
Duration: 24 min. per ep.
Rating: PG-13 - Teens 13 or older

Statistics
Score: 7.90[1] (scored by 427,145 users)
Ranked: #689[2]
Popularity: #18
Members: 806,966
Favorites: 40,441

Alternative Titles
English: Dragon Ball Z
Synonyms: DBZ, Dragonball Z
Japanese: ドラゴンボールZ

Information
Type: TV
Episodes: 291
Status: Finished Airing
Aired: Apr 26, 1989 to Jan 31, 1996
Premiered: Spring 1989
Broadcast: Wednesdays at 19:00 (JST)
Producers: Fuji TV
Licensors: Funimation
Studios: Toei Animation
Source: Manga
Genres: Action, Adventure, Comedy, Fantasy, Martial Arts, Shounen, Super Power
Duration: 24 min. per ep.
Rating: PG-13 - Teens 13 or older

Statistics
Score: 8.31[1] (scored by 361,221 users)
Ranked: #228[2]
Popularity: #70
Members: 524,682
Favorites: 21,013

Alternative Titles
English: My Hero Academia
Japanese: 僕のヒーローアカデミア

Information
Type: TV
Episodes: 13
Status: Finished Airing
Aired: Apr 3, 2016 to Jun 26, 2016
Premiered: Spring 2016
Broadcast: Sundays at 17:00 (JST)
Producers: Dentsu, Mainichi Broadcasting System, Movic, TOHO animation, Shueisha
Licensors: Funimation
Studios: Bones
Source: Manga
Genres: Action, Comedy, School, Shounen, Super Power
Duration: 24 min. per ep.
Rating: PG-13 - Teens 13 or older

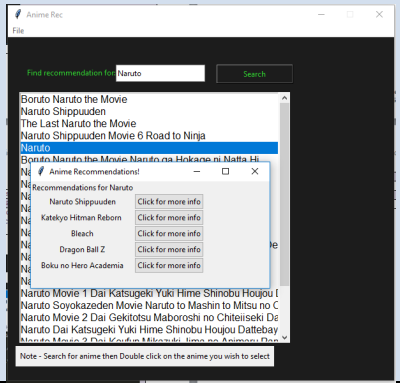
Statistics
Score: 8.43[1] (scored by 471,898 users)
Ranked: #146[2]
Popularity: #30
Members: 730,516
Favorites: 18,299

As you can see all the generated recommendations have very similar features to Naruto.

They all have very similar genres, episodes, rating descriptions, types and popularity.

05/04/2018 – Get Rec Screen prototype interview with client, Josh Santillan

Comment [MR110]: Client and success criteria review for the recommender system and get rec screen

Success criteria / requirement	Achieved / Not Achieved (with evidence)
Functional recommender system	Achieved 
Efficient recommender system	Not Achieved - Could be more efficient as there is definitely ways to reduce the code size
Response time	Achieved Executes in ;less than 40 seconds (20 seconds)

FINAL REGISTER SCREEN PROTOTYPE

Modifications from first prototype:

- More efficient (Requirement- Efficient recommender system now Achieved)
- Reduced response time to less than 3 seconds

Evidence:

Comment [MR111]: White box testing

BEFORE

```
from sklearn.preprocessing import MaxAbsScaler # Imports Scikit-learn's MaxAbsScaler module which scales each feature by its maximum absolute value
from sklearn.neighbors import NearestNeighbors # Imports sklearn's NearestNeighbors module which provides functionality for unsupervised and supervised neighbors-based learning methods
import numpy as np # Imports the Numpy library which adds support for large, multi-dimensional arrays and matrices
import pandas as pd # Imports the pandas library which is used for data manipulation and analysis
import re # Imports the re library which is used to read/write and match regular expressions


## STEP 1 - Formatting the csv file which contains all the anime data in a format that the K-Nearest-Neighbour (KNN) algorithm can understand ##

# Creating the anime.csv file and storing it in a pandas dataframe
anime = pd.read_csv("anime.csv")

# Returning the first row of the anime dataframe, i.e the headers
anime.head()

# Manually editing the number of episodes for certain animes which do not have a value for number of episodes
anime.loc[(anime["genre"]=="Hentai") & (anime["episodes"]=="Unknown"),"episodes"] = "1"
anime.loc[(anime["type"]=="OVA") & (anime["episodes"]=="Unknown"),"episodes"] = "1"
anime.loc[(anime["type"] == "Movie") & (anime["episodes"] == "Unknown")] = "1"
known_animes = {"Naruto Shippuuden":500, "One Piece":784,"Detective Conan":854, "Dragon Ball Super":86,
                "Crayon Shin-chan":942, "To Gi Oh Arc V":148,"Shirayuki no Hyoujo Season 2":25,
                "Boku no Hero Academia 2nd Season":25,"Little Witch Academia TV":25}
for k,v in known_animes.items():
    anime.loc[anime["name"]==k,"episodes"] = v

# Replacing the remaining animes which do not have a value for number of episodes with the median number of episodes of the entire dataset
anime["episodes"] = anime["episodes"].map(lambda x:np.nan if x=="Unknown" else x)
anime["episodes"].fillna(anime["episodes"].median(),inplace = True)

# Converting all the rating values to floats and replacing all animes which do not have a rating with the median rating of the entire data set
anime["rating"] = anime["rating"].astype(float)
anime["rating"].fillna(anime["rating"].median(),inplace = True)

# Converting categorical variable of type to an indicator variable
pd.get_dummies(anime[["type"]]).head()

# Converting all the values of members to the float data type
anime["members"] = anime["members"].astype(float)

# Creating a dataframe with all the anime features as the headings
anime_features = pd.concat([anime["genre"].str.get_dummies(sep=","),pd.get_dummies(anime[["type"]]),anime[["rating"]],anime[["members"]],anime["episodes"]],axis=1)
anime["name"] = anime["name"].map(lambda name:re.sub('[^A-Za-z0-9]+', " ", name))
anime_features.head()
anime_features.columns

## STEP 2 - Executing the KNN algorithm on the anime features dataframe

# Instantiating sklearn's MaxAbsScaler which scales each feature by its maximum absolute value
max_abs_scaler = MaxAbsScaler()

# Fit the data and then transform it
anime_feature = max_abs_scaler.fit_transform(anime_features)

# Runs SKNN's ball tree algorithm on the anime features dataframe and returns the the 6 most similar animes to every single anime in the data frame
nbrs = NearestNeighbors(n_neighbors=6, algorithm='ball_tree').fit(anime_features)
distances, indices = nbrs.kneighbors(anime_features)


# Function which returns the index of an anime
def get_index_from_name(name):
    return anime[anime["name"]==name].index.tolist()[0]

all_anime_names = list(anime.name.values)

# Function which gets all the anime names from a partial input
def get_id_from_partial_name(partial):
    for name in all_anime_names:
        if partial in name:
            print(name,all_anime_names.index(name))

# Function which returns the 5 most similar animes to a given anime using the results from the KNN algorithm
def get_rec(query=None,id=None):
    if id:
        for id in indices[id][1:]:
            print(anime.ix[id]["name"])
    if query:
        found_id = get_index_from_name(query)
        for id in indices[found_id][1:]:
            print(anime.ix[id]["name"])
```

AFTER

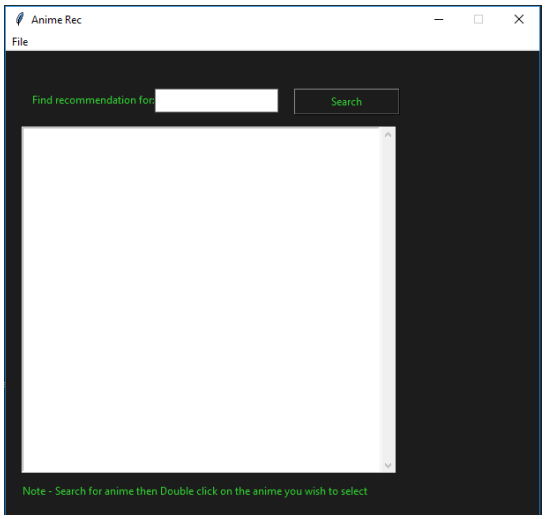
```
import pandas as pd
import pickle
import sqlite3
import webbrowser


anime = pd.read_pickle('anime')
anime_features = pd.read_pickle('anime_features')
filename = 'finalized_model.sav'
loaded_model = pickle.load(open(filename, 'rb'))
distances, indices = loaded_model



def get_ID(name):
    with sqlite3.connect("animeData.db") as  db: #anime database
            c = db.cursor()
            findID=("SELECT anime_id from t WHERE name=?")
            c.execute(findID,[name])

            results = c.fetchall()
            db.commit()
    return results


all_anime_names = list(anime.name.values)

idSearch = []

def get_possible_searches(partial):
    possibleSearch=[]
    for name in all_anime_names:
        if partial in name:
            possibleSearch.append(name)
    return possibleSearch

def get_index_from_name(name):
    return anime[anime["name"]==name].index.tolist()[0]

def get_similar_animes(query=None,id=None):
    rec=[]
    if id:
        for id in indices[id][1:]:
            print(anime.ix[id]["name"])
    if query:
        found_id = get_index_from_name(query)
        for id in indices[found_id][1:]:
            rec.append((anime.ix[id]["name"]))
    return rec
```

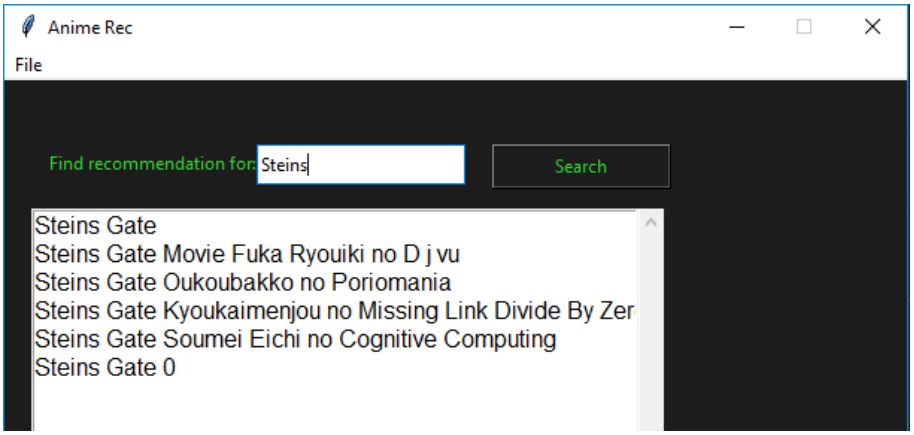
I also stored the finalised model (results of the KNN algorithm) in a sav file, which again can just be loaded by reading the file instead of running the KNN algorithm every time we launch the app

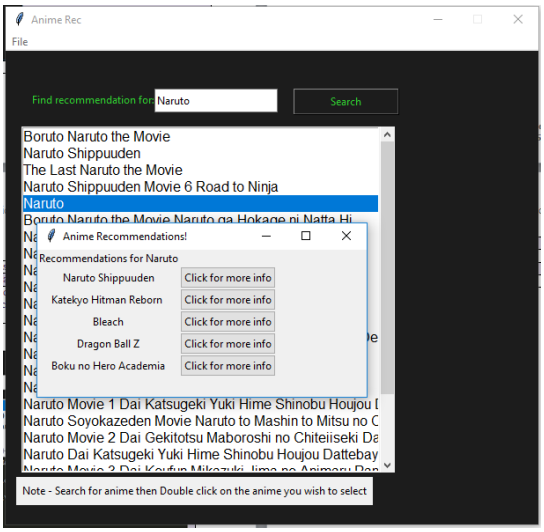
I stored the formatted anime and anime features data frames in a pickle file which can just be loaded again by reading the file. This Reduces the response time and space complexity of the algorithm as we don't have to format the data frame every time we run the app as it has already been done.

```
import pandas as pd
import pickle
import sqlite3
import webbrowser


anime = pd.read_pickle('anime')
anime_features = pd.read_pickle('anime_features')
filename = 'finalized_model.sav'
loaded_model = pickle.load(open(filename, 'rb'))
distances, indices = loaded_model



def get_ID(name):
    with sqlite3.connect("animeData.db") as  db: #anime database
            c = db.cursor()
            findID=("SELECT anime_id from t WHERE name=?")
            c.execute(findID,[name])

            results = c.fetchall()
            db.commit()
    return results


all_anime_names = list(anime.name.values)

idSearch = []

def get_possible_searches(partial):
    possibleSearch=[]
    for name in all_anime_names:
        if partial in name:
            possibleSearch.append(name)
    return possibleSearch

def get_index_from_name(name):
    return anime[anime["name"]==name].index.tolist()[0]

def get_similar_animes(query=None,id=None):
    rec=[]
    if id:
        for id in indices[id][1:]:
            print(anime.ix[id]["name"])
    if query:
        found_id = get_index_from_name(query)
        for id in indices[found_id][1:]:
            rec.append((anime.ix[id]["name"]))
    return rec
```

Both pieces of code above do the same thing but as you can see the second piece of code is much more efficient as we do not need to format the data frame or run the KNN algorithm every time we launch the app and instead we can run them once and store the results in a permanent file which can then be loaded into memory whenever we need it.







Josh Santillan, Lead client

06/04/2018 – Creating Browse screen

The browse screen is a simple feature of the app which will allow the user to browse for animes by name, genre or type.

Creating the front end of the browse screen

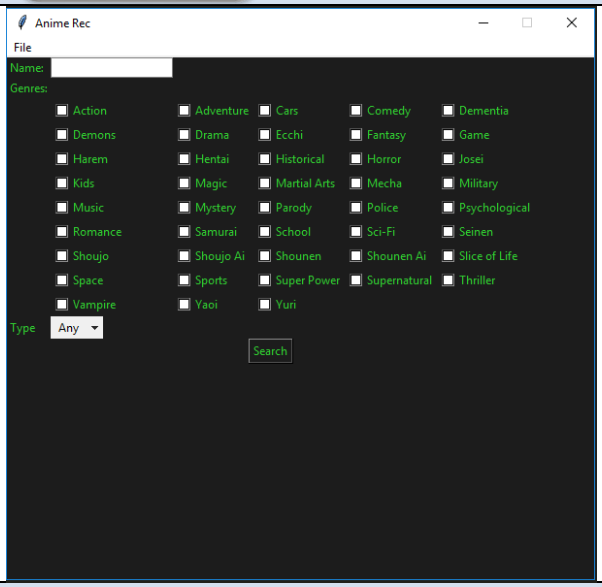
```
# Declaring the browse class which inherits from Tkinter's Frame class (Creating the browse screen)
class browse(tk.Frame):
    # Declaring browse class' constructor method
    def __init__(self, parent, controller):

        # Initialising Tkinter's Frame class
        tk.Frame.__init__(self, parent)

        # Creating and displaying the entry widget for the user to search by name along with a label for the entry widget
        NameLabel = ttk.Label(self, text="Name:")
        NameLabel.grid(row=0, sticky ='w')
        self.NameEntry = ttk.Entry(self)
        self.NameEntry.grid(row=0, column=1)

        # Creating and displaying a genres label for the checkbuttons
        GenreLabel = ttk.Label(self, text="Genres:")
        GenreLabel.grid(row=1, sticky='w')

        # List of all the genres
        self.GENRES = ["Action","Adventure","Cars","Comedy","Dementia",
                       "Demons","Drama","Ecchi","Fantasy","Game","Harem",
                      "Hentai","Historical","Horror","Josei","Kids",
                      "Magic","Martial Arts","Mecha","Military","Music",
                      "Mystery","Parody","Police","Psychological",
                      "Romance","Samurai","School","Sci-Fi","Seinen",
                      "Shoujo","Shoujo Ai","Shounen","Shounen Ai",
                      "Slice of Life","Space","Sports","Super Power",
                      "Supernatural","Thriller","Vampire","Yaoi","Yuri"]

        # Creating and displaying each genre ans a checkbutton, most efficient way in Tkinter
        self.vars = []
        self.chk = []
        for i in range(0,5):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var))
            self.chk[-1].grid(row=2, column=(i+1),sticky="w")
            self.vars.append(self.var)
        for i in range(5,10):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var))
            self.chk[-1].grid(row=(3),column=((i+1)-5),sticky="w")
            self.vars.append(self.var)
        for i in range(10,15):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var))
            self.chk[-1].grid(row=(4),column=((i+1)-10),sticky="w")
            self.vars.append(self.var)
        for i in range(15,20):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var))
            self.chk[-1].grid(row=(5),column=((i+1)-15),sticky="w")
            self.vars.append(self.var)
        for i in range(20,25):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var))
            self.chk[-1].grid(row=(6),column=((i+1)-20),sticky="w")
            self.vars.append(self.var)
        for i in range(25,30):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var))
            self.chk[-1].grid(row=(7),column=((i+1)-25),sticky="w")
            self.vars.append(self.var)
        for i in range(30,35):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var))
            self.chk[-1].grid(row=(8),column=((i+1)-30),sticky="w")
            self.vars.append(self.var)
        for i in range(35,40):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var))
            self.chk[-1].grid(row=(9),column=((i+1)-35),sticky="w")
            self.vars.append(self.var)
        for i in range(40,43):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var))
            self.chk[-1].grid(row=(10),column=((i+1)-40),sticky="w")
            self.vars.append(self.var)
```

Usually you would need these 3 lines for each check button so that's 3x43, which is 129 lines of code for 43 check buttons.

However through the use of multiple for loops and simple maths to configure the rows and columns to display the check buttons correctly I was able to reduce the lines of code to just 45 lines

```
# Creating and displaying the type label and combobox to allow the user to search by type
TypeLabel = ttk.Label(self, text="Type")
TypeLabel.grid(row=11,sticky="w")
self.type = tk.StringVar(self)
self.type.set("Any")
t = ttk.OptionMenu(self, self.type,"Any","TV", "Movie", "OVA", "Special")
t.grid(row=11, column=1,sticky="w")

# Creating and displaying the Search button which takes all the inputs from the user and querys the database with the given criteria
Search= ttk.Button(self, text="Search", command =self.query)
Search.grid(columnspan=10)
```

Browse screen front end testing

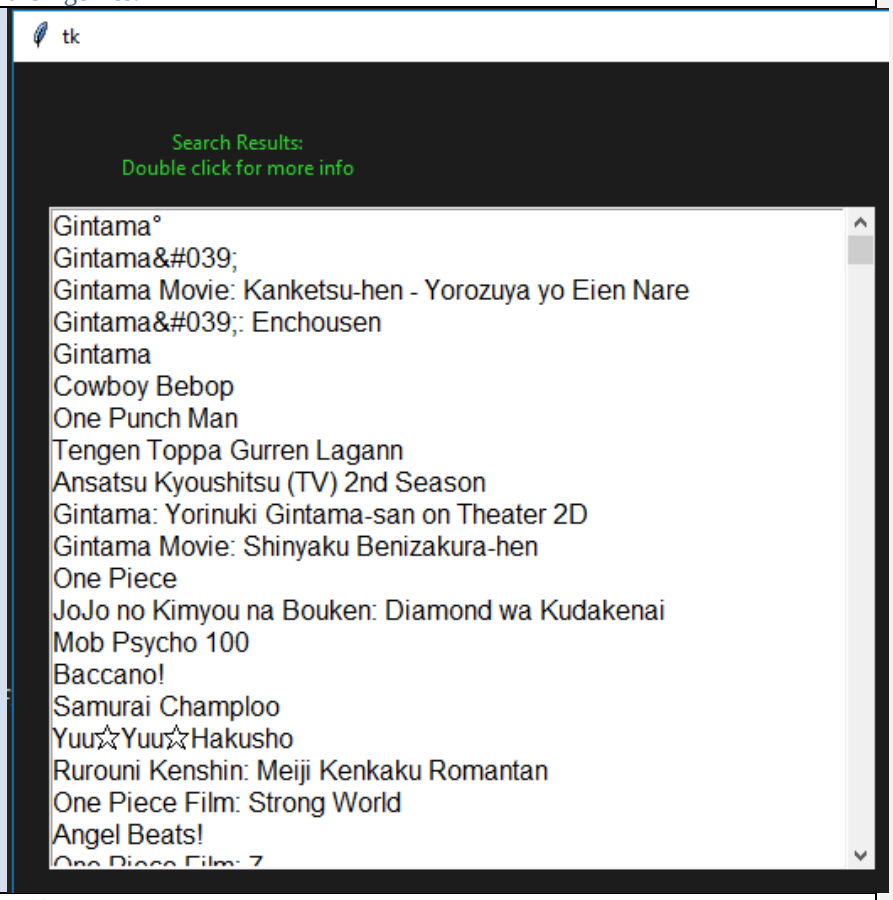
Test case	Check whether the Browse screen is displayed correctly
Expected result	
Actual result	
Modification	N/A

Adding functionality to the search button

Comment [MR112]: •Pop up window
•List box
•Scroll bar
•Check buttons
•SQL database query

```
# Function for the Search buttons, which querys the database with the given criteria inputted by the user
def query(self):

    # Creates a pop up window
    self.popup = tk.Tk()

    # Creates and display a title label for the pop up window
    self.title = ttk.Label(self.popup, text=("Search Results:\nDouble click for more info"), width=210, relief='flat')
    self.title.place(relx=0.04, rely=0.07, height=35, width=210)

    # Creates and displays a frame used to hold the listbox widget and scroll bar widget
    frm = tk.Frame(self.popup)
    frm.place(relx=0.03, rely=0.16, relheight=0.73, relwidth=0.7)

    # Creates and adds a scroll bar to the list box to allow the user to scroll up and down the search results
    scrollbar2 = tk.Scrollbar(frm, orient="vertical", highlightcolor="#d9d9d9", highlightbackground="#d9d9d9")
    scrollbar2.pack(side='right', fill='y')

    # Creating and displaying the Listbox widget
    self.listNodes2 = tk.Listbox(frm, width=100, yscrollcommand=scrollbar2.set, font=("Helvetica", 12))
    self.listNodes2.bind("<Double-Button-1>", self.OnDouble2)
    self.listNodes2.pack(expand=True, fill='y')
    scrollbar2.config(command=self.listNodes2.yview)

    # Button which returns the user to the browse screen and closes the search results pop up window
    button1 = tk.Button(self.popup, text="Return", command = self.return1)
    button1.pack(side='bottom')

    # Stores the whatever values are checked from the check buttons into an array
    states=[]
    for x in self.vars:
        states.append(x.get())
    genreIndex = [i for i, s in enumerate(states) if s == 1]
    self.genreResults=[]
    for x in genreIndex:
        self.genreResults.append(self.GENRES[x])

    # Gets the  value of the type from the type combobox and the value of the name from the name entry widget
    self.typeResult = self.type.get()
    self.animeName = self.NameEntry.get()

    # Connects to the anime database  and runs an sql query with the given criteria
    with sqlite3.connect("animeData.db") as  db: #anime database
        c = db.cursor()
        #self.genreResults is the list containing the genres i want to query
        user_input = self.genreResults  # for example: ['Action','Drama']
        convert_to_like_conditions = ' and '.join(list(map(lambda item : "genre like '%{0}%'".format(item), user_input)))
        c.execute("select * from t where %s" % convert_to_like_conditions )
        self.results = c.fetchall()
        print(self.results)
        db.commit()


    # Displays all the search results from the query in the listbox
    for i in range(0,len(self.results)):
        self.listNodes2.insert(i,self.results[i][1])

    # Pop up window dimensions configuration
    width_of_window = 700
    height_of_window = 538
    screen_width = self.popup.winfo_screenwidth()
    screen_height = self.popup.winfo_screenheight()
    x_coordinate = (screen_width//2) - (width_of_window//2)
    y_coordinate = (screen_height//2) - (height_of_window//2)
    self.popup.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window, x_coordinate, y_coordinate))
    self.popup.mainloop()

# Function which is executed when a search result in the listbox is double clicked
def OnDouble2(self, event):
    # Gets the value of the anime of the listbox node that was double clicked
    widget = event.widget
    selection=widget.curselection()
    value = widget.get(selection)
    i = index = [idx for idx, s in enumerate(self.results) if value in s][0]

    # Creates a myanimelist.net website for that anime and opens it up in the user's browser
    url = 'https://myanimelist.net/anime/'+self.results[i][0]+self.results[i][1]
    webbrowser.open(url)

# Function which is executed when the return button is clicked
def return1(self):
    self.popup.destroy()
```

Search button testing

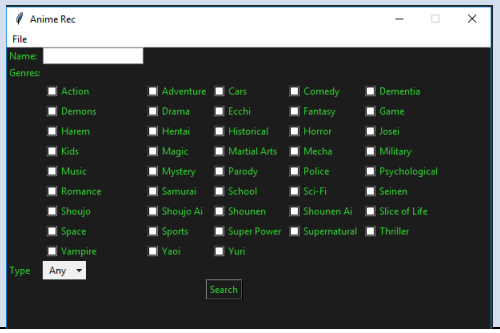
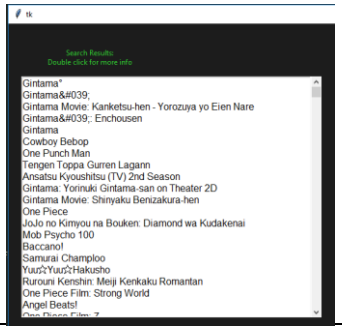
Test Case	**Check that browsing works**
Test steps	1. Select Action and Comedy. 2.Click Search
Expected Result	A new window with a list of animes that have both Action and Comedy as their genres.
Actual Result	
Modification	N/A

Comment [MR113]: Black box and usability testing

Test Case	**Check that double clicking on a result open its myanimelist.net website in the browser.**
Test steps	Double click on a result
Expected Result	That results myanimelist.net website should be displayed
Actual Result	
Modification	N/A

07/04/2018 – Browse screen prototype interview with client, Josh Santillan

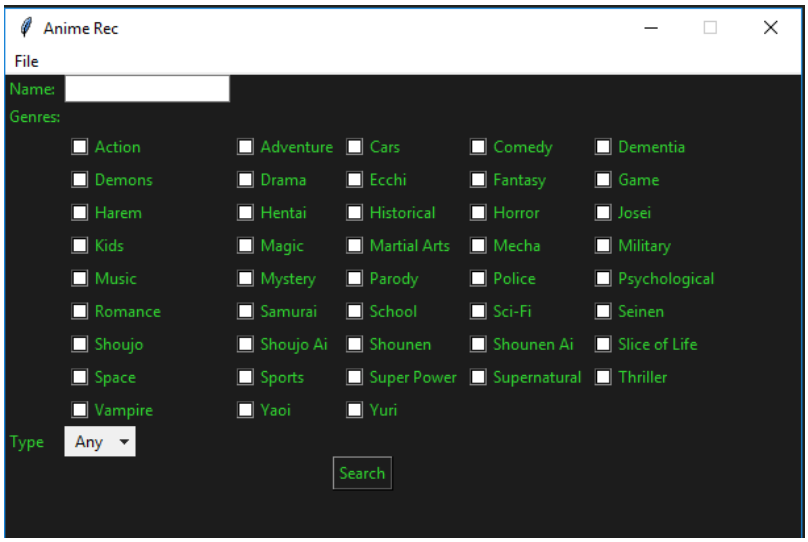
Comment [MR114]: Client and success criteria review

Success criteria / requirement	**Achieved / Not Achieved (with evidence)**
Colour scheme set 1 	Achieved 
Browsing tool – A tool that allows the user to search for animes by genre rather than just name	Achieved 

FINAL BROWSE SCREEN PROTOTYPE

Modifications from first prototype:

N/A



Josh Santillan, Lead client

08/04/2018 – Creating Pictionary screen

CLIENT ANALYSIS AND DESIGN UPDATE

Comment [MR115]: New requirements

After some further discussion with my clients we have revised the design of the Pictionary screen and have come up with some clear requirements.

The Pictionary screen is a mini game whereby the user has to guess the name of the anime whose picture is shown on the screen. The user will be given 4 choices along with the image to guess.

If the user guesses wrong then it will be game over and the user's highest score will be stored in the database. There is a maximum of 60 questions if the user gets all the questions right then they win the game.

Breakdown of the Pictionary screen

I have broken the Pictionary screen down into 3 main sections:

- Create questions (60 images, each with 4 choices, 1 being the correct answer)
- Display the question (Display the image along with 4 answers)
- Check the answer (Take user input and compare with correct answer)
- Update question or End game

Creating the questions

The questions and choices need to be read by Python in order to display them and so to do this I decided to create the questions in a tab delimited text file. A tab delimited text file is where values are separated by tab indentations, this way we can easily distinguish between different values.

First 10 out of 60 questions (questions.txt)

```
image   answer  choices
0       Dragon Ball     "Dragon Ball,One Piece,Naruto,Hunter X Hunter"
1       One Piece       "Dragon Ball,One Piece,Naruto,Hunter X Hunter"
2       Naruto  "Dragon Ball,One Piece,Naruto,Hunter X Hunter"
3       Puella Magi Madoka Magica       "Puella Magi Madoka Magica,Rurouni Kenshin,Samurai Champloo,Ninja Scroll"
4       Rurouni Kenshin "Puella Magi Madoka Magica,Rurouni Kenshin,Samurai Champloo,Ninja Scroll"
5       Samurai Champloo        "Puella Magi Madoka Magica,Rurouni Kenshin,Samurai Champloo,Ninja Scroll"
6       Hunter X Hunter "Dragon Ball,One Piece,Naruto,Hunter X Hunter"
7       Fullmetal Alchemist     "Fullmetal Alchemist,D. Gray Man,Dragon Ball,One Piece"
8       Ninja Scroll    "Puella Magi Madoka Magica,Rurouni Kenshin,Samurai Champloo,Ninja Scroll"
9       Death Note      "Death Note,D. Gray Man,Dragon Ball,One Piece"
10      D. Gray Man     "Fullmetal Alchemist,D. Gray Man,Dragon Ball,One Piece"
```

Displaying questions

The code below reads the data from the questions.txt file and stores it into memory to be used to display the questions.

```
## READING THE QUESTIONS.TXT FILE AND STORING THE VALUES IN A NUMPY ##

# Opens file in read mode
f = open('questions.txt', 'r')

# Temporary Python array
data = []

# For each line in the text file it splits the line based on tabs
for row_num, line in enumerate(f):
    values = line.strip().split('\t')
    # first line is the header (image, answer)
    if row_num == 0:
         # Stores the haeader in an array
         header = values
    else:
        data.append([v for v in values])

# Converts Python array to a numpy array
question_data = np.array(data)


# Close the file
f.close()
## TESTING PURPOSES ##
    #print(question_data)
    #print(header)

## RANDOMIZE THE QUESTIONS BEFORE EACH RUN ##
np.random.shuffle(question_data)
    #print(question_data)# TESTING
```

Now that we have the data for each question in memory we have to display it onto the window. The code below does this:

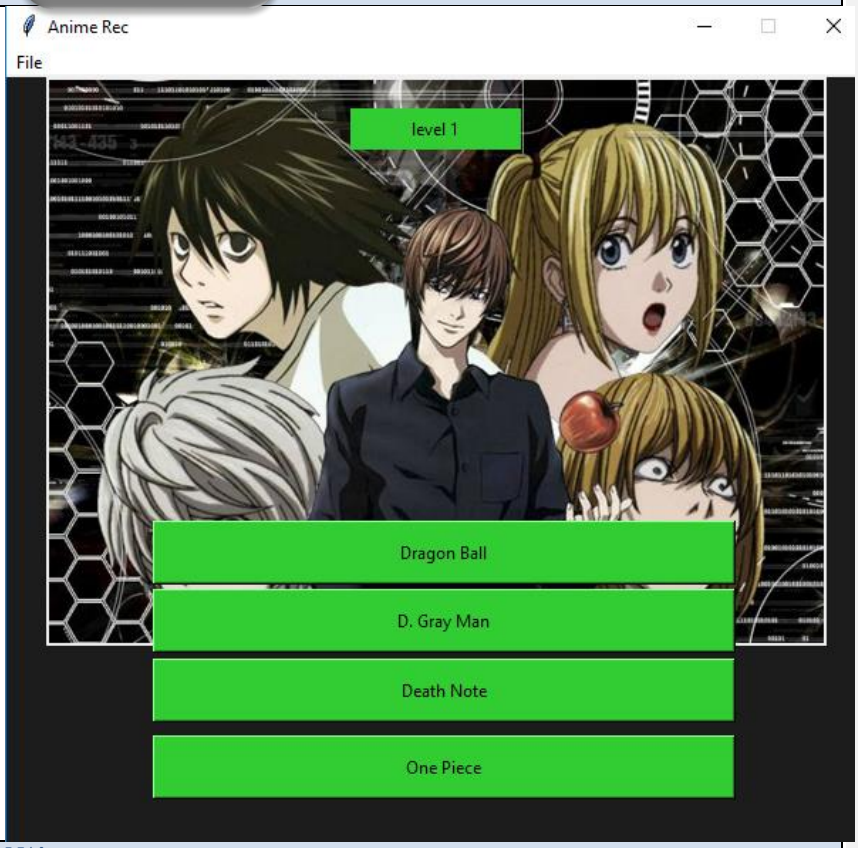
```
# Declares the Pictionary class which inherits from Tkinters frame class (Creates the Pictionary screen)
class pictionary(tk.Frame):
    # Constructor method for the Pictionary class
    def __init__(self, parent, controller):

        # Initalises Tkinter's Frame class
        tk.Frame.__init__(self, parent)

        # Stores controller as a variable of the pictionary class
        self.controller =controller

        # Setting count to 0 at the start (indicating we are at the first question)
        self.count =0

        # Number of questions set to 10 for testing purposes
        self.max_questions = 10

        # Executes the display_question function (declared below) with the question image, answer and choices for the first question from the text file
        self.display_question(question_data[self.count][0],question_data[self.count][1],question_data[self.count][2])


    # Function which display the question onto the window. Takes a questions image, answer and choices as parameters.
    def display_question(self, image, answer, choices):

        # Loads the image in a way that Python can read it
        img = Image.open(image+'.jpg')
        # Stores the loaded image into the variable, photo
        photo = ImageTk.PhotoImage(img)

        # print('answer:',answer) ## Testing purposes to check the correct answer

        # Seperates the choices for the question into single elements so they can be displayed on the window
        choices = choices.replace('"','')
        choices = [x.strip() for x in choices.split(',')]
        # Randomizes the choices
        random.shuffle(choices)

        # Creating and displaying the label that contains the image that is to be guessed
        self.anime_image = tk.Label(self, image=photo,relief='flat')
        self.anime_image.image = photo
        self.anime_image.pack()


        # Creating and displaying the label which displays the current level to the user i.e their score
        level = 'level',self.count
        self.level = tk.Label(self, text=level,borderwidth=0, relief='flat')
        self.level.place(relx=0.4, rely=0.04, height=29, width=120)

        # Creating and displaying the Buttons which represent the 4 available choices to the user
        # Note the command for each button passes the value of the choice to the check answer function so that the users choice can be compared with the-
        # correct answer
        # Check Function yet to be created
        self.answer_a = tk.Button(self,text=choices[0],command = lambda:self.check(choices[0],answer),width=410)
        self.answer_a.place(relx=0.17, rely=0.58, height=45, width=410)

        self.answer_c = tk.Button(self,text=choices[1],command = lambda:self.check(choices[1],answer),width=410)
        self.answer_c.place(relx=0.17, rely=0.76, height=45, width=410)

        self.answer_b = tk.Button(self,text=choices[2],command = lambda:self.check(choices[2],answer),width=410)
        self.answer_b.place(relx=0.17, rely=0.67, height=45, width=410)

        self.answer_d = tk.Button(self,text=choices[3],command = lambda:self.check(choices[3],answer),width=410)
        self.answer_d.place(relx=0.17, rely=0.86, height=45, width=410)
```

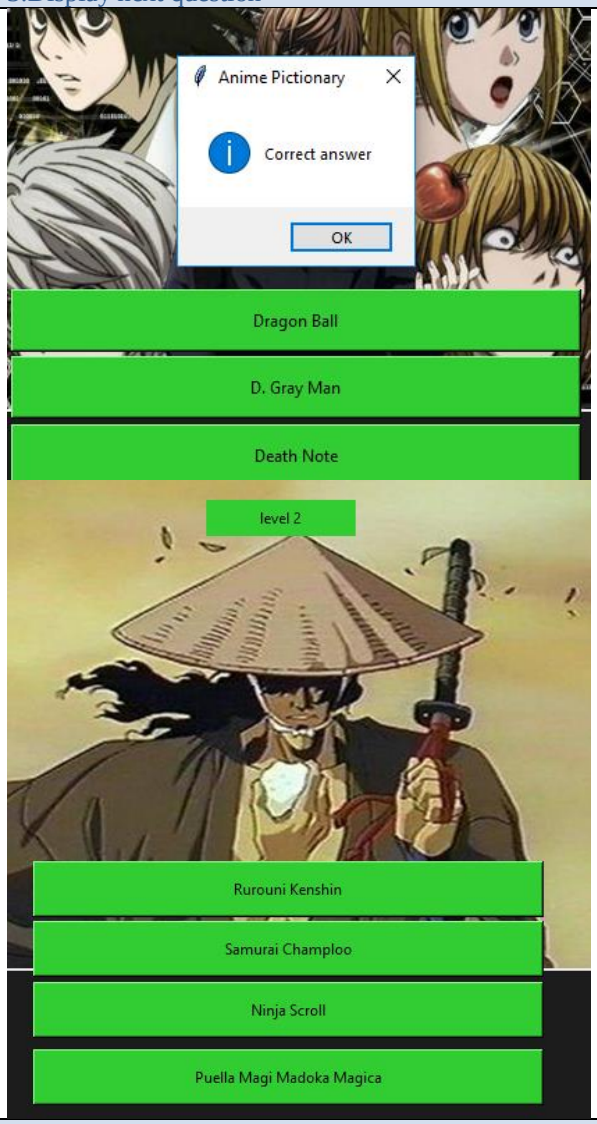
Checking the answer and updating the question

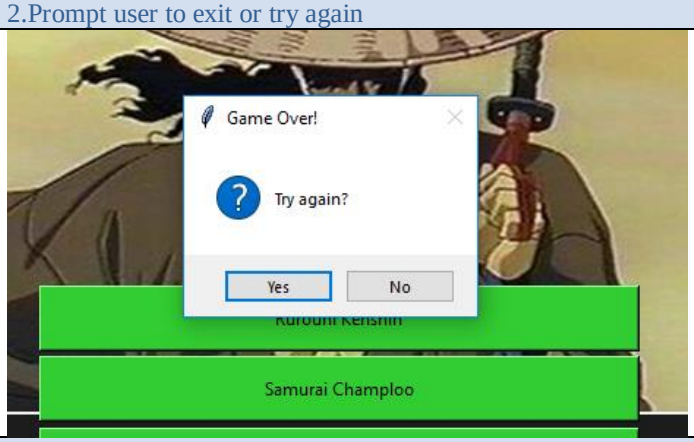
Now we need to create a function that checks the user answer with the correct answer. If the user gets the question right then we display the next question or if the user gets it wrong then its game over.

```
# Function that checks the user's answer. The function takes the user's answer and the correct answer as parameters
def check(self,user_answer,correct_answer):

    # Checks if this is the last question, if it is then the user has won the game
    if self.count == self.max_questions:
        # Instantiating Tkinter's messagebox wwhich asks the use if they want to play again
        user_answer=tkinter.messagebox.askquestion("You won", "Play again?")
        # if the user wants to play again then we reset the game
        if user_answer == 'yes':
            np.random.shuffle(question_data)
            self.anime_image.forget()
            self.answer_a.forget()
            self.answer_b.forget()
            self.answer_c.forget()
            self.answer_d.forget()
            self.count =0
            self.display_question(question_data[self.count][0],question_data[self.count][1],question_data[self.count][2])
        # If they dont then we reset the game and return to the menu screen
        else:
            self.forget()
            self.controller.show_frame(menu
        return

    # If the user's answer is correct then we display a correct prompt and display the next question
    if user_answer == correct_answer:
        # Instantiating Tkinter's messagebox with the given  window title and message respectively
        tkinter.messagebox.showinfo("Anime Pictionary","Correct answer")

        # Increment count, i.e. move onto next question
        self.count += 1

        # Removing the current image and choices on screen. I.E. Removing the current question
        self.anime_image.forget()
        self.answer_a.forget()
        self.answer_b.forget()
        self.answer_c.forget()
        self.answer_d.forget()

        # Ececutes the display_question function with the image,correct answer and choices for the next question
        self.display_question(question_data[self.count][0],question_data[self.count][1],question_data[self.count][2])

    # Else if the user gets the questions wrong then its Game over!
    else:
        # Instantiating Tkinter's messagebox asking the user if they want to try agian
        user_answer=tkinter.messagebox.askquestion("Game Over!", "Try again?")

        # If they want to try  again then we reset the game
        if user_answer == 'yes':
            np.random.shuffle(question_data)
            self.anime_image.forget()
            self.answer_a.forget()
            self.answer_b.forget()
            self.answer_c.forget()
            self.answer_d.forget()
            self.count =0
            self.display_question(question_data[self.count][0],question_data[self.count][1],question_data[self.count][2])
        # If they dont then we reset the game and return to the menu screen
        else:
            self.forget()
            self.controller.show_frame(menu)
```

Pictionary screen testing

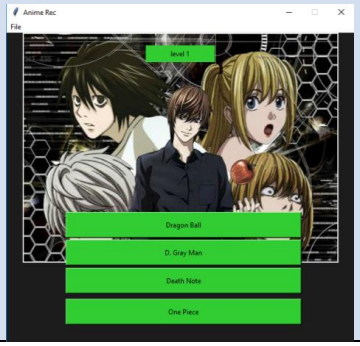
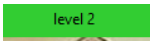
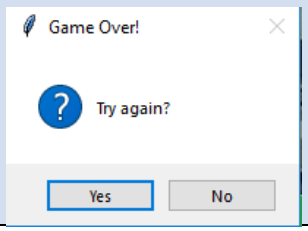
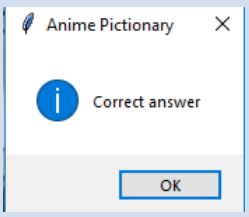
Test Case	Check whether the Pictionary screen is displayed correctly
Expected result	
Actual result	
Modification	N/A

Test Case	Check result in the case of correct answer
Expected result	1.Display correct prompt 2.Increment score 3.Display next question
Actual result	
Modification	N/A

Test Case	Check result in the case of incorrect answer
Expected result	1.Display game over screen 2.Prompt user to exit or try again
Actual result	
Modification	N/A

09/04/2018 – Get Rec Screen prototype interview with client, Josh Santillan

Comment [MR116]: Client and success criteria review for the Pictionary screen

Success criteria / requirement	Achieved / Not Achieved (with evidence)
Consistent anime theme	Achieved 
Score	Achieved 
Validation for questions	Achieved  

Josh Santillan, Lead client

09/04/2018 – Creating Profile screen

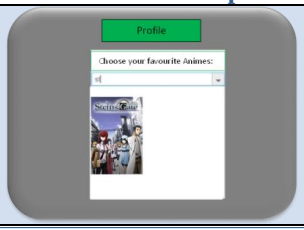
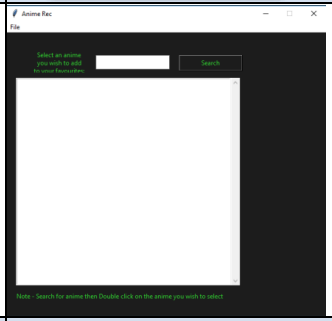
The profile screen will allow the user to add their favourite animes to a list which is permanently stored in a data storage file.

In order to do this I decided to use Tkinter's Listbox widget along with an entry widget which will allow the user to select an anime which will be stored in the user's favourites list.

Creating the front-end of the profile screen

```
# Declaring the class for the Profile screen, which inherits from Tkinter's Frame class
class profile(tk.Frame):

    # Declaring the constructor method for the profile class, which runs when the class is instantiated
    def __init__(self, parent, controller):
        # Initialising Tkinter's Frame class
        tk.Frame.__init__(self, parent, bg='grey11')

        # Storing controller as a variable of the profile class
        self.controller = controller

        # Creating and displaying the label for the entry widget which will be used so that the user can enter which anime they wish to add to their favourites
        self.getRec_label = tk.Label(self, text='Select an anime\nyou wish to add\nto your favourites:', width=210, relief='flat', bg = 'grey11', fg='lime green', borderwidth=2,highlightbackground="lime green" )
        self.getRec_label.place(relx=0.00, rely=0.07, height=38, width=200)

        # Creating and displaying the Entry widget which will allow the user to input the name of an anime
        self.search_entry = ttk.Entry(self)
        self.search_entry.place(relx=0.28, rely=0.08, relheight=0.05, relwidth=0.23)

        # Creating and displaying a button, which when clicked will output the search results of the user's input
        self.search_butt = tk.Button(self, text='Search', command= self.populate, bg = 'grey11', fg='lime green', borderwidth=2,highlightbackground="lime green" )
        self.search_butt.place(relx=0.54, rely=0.08, height=30, width=120)



        # Creating and displaying a frame to contain the list box and scrollbar widget
        frm = tk.Frame(self)
        frm.place(relx=0.03, rely=0.16, relheight=0.73, relwidth=0.7)

        # Creating and displaying a scroll bar on the right of the frame which is used to scroll up and down the list box
        scrollbar = tk.Scrollbar(frm, orient="vertical", highlightcolor="#d9d9d9", highlightbackground="#d9d9d9")
        scrollbar.pack(side='right', fill='y')

        # Creating and displaying a Listbox widget within the same frame as the scrollbar
        self.listNodes = tk.Listbox(frm, width=100, yscrollcommand=scrollbar.set, font=("Helvetica", 12))
        self.listNodes.bind("<Double-Button-1>", self.OnDouble)
        self.listNodes.pack(expand=True, fill='y')
        scrollbar.config(command=self.listNodes.yview)

        # Creating and displaying a label to give the user some instructions as to how to use the list box.
        self.getRec_note = tk.Label(self, text='Note - Search for anime then Double click on the anime you wish to select', bg = 'grey11', fg='lime green')
        self.getRec_note.place(relx=0.02, rely=0.9, relheight=0.06, relwidth=0.67)
```

Testing the front end of the profile screen

Test Case	Check whether the profile screen is displayed correctly
Expected result	
Actual result	
Modification	N/A

Adding functionality to the search button

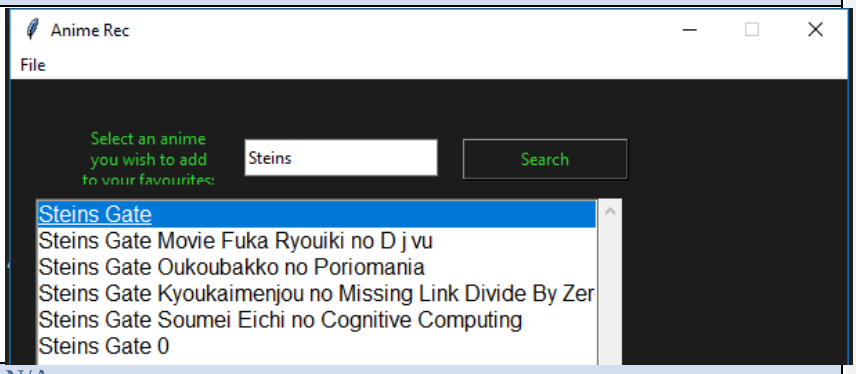
When the search button is clicked the aim is to populate the list box with the search results of all the animes that contain the user's input in their name. The code below achieves this.

Note – the function below is executed when the search button is clicked

```
# Function which populates the List box with the search results from the user's input.
def populate(self):
    self.listNodes.delete(0, 'end')
    partial=string.capwords(self.search_entry.get())
    possibleSearch= get_possible_searches(partial)
    for i in range(0, len(possibleSearch)):
        self.listNodes.insert(i, possibleSearch[i])
```

The function gets the entry from the entry widget and then runs the get_possible_searches function with the user's entry which returns all the anime that contain the user's entry in their name. It then loops through each anime in the results and adds it to the list box.

Search button testing

Test Case	**Check functionality of selecting a favourite anime**
Test steps	1.Enter anime name 2.Click the find Button
Test data	"Steins"
Expected result	Display all the anime which contain the characters 'Steins' in their name
Actual result	
Modification	N/A

Creating the function that will add the selected anime to the user's favourite list

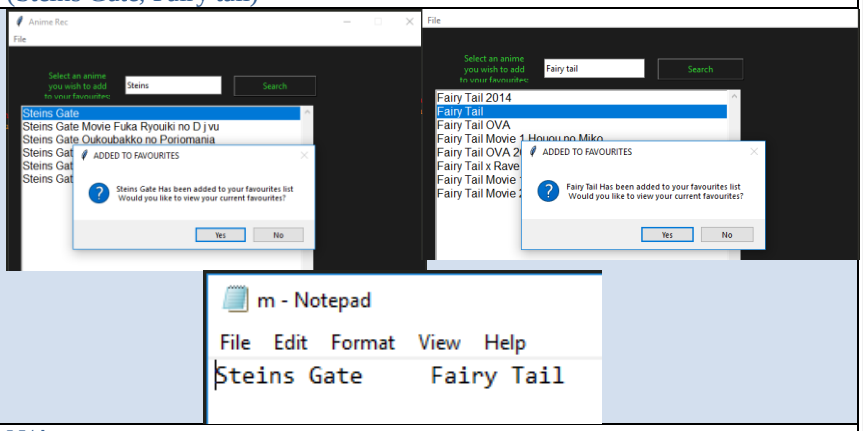
Firstly we need to get the value of the anime that was selected by the user from the list box:

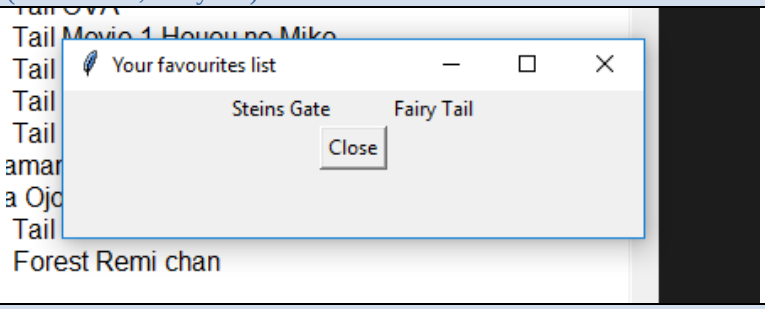
```
    # Function which adds the selected anime to the user favourites
    def OnDouble(self, event):
        # Gets the value of the node that was selected from the list box and stores it in the varibale, value
        widget = event.widget
        selection=widget.curselection()
        value = widget.get(selection)
```

We then have to store this value in permanent storage file which can be loaded when the user wants to view their favourites.

```
# Function which adds the selected anime to the user favourites
def OnDouble(self, event):
    # Gets the value of the node that was selected from the list box and stores it in the varibale, value
    widget = event.widget
    selection=widget.curselection()
    value = widget.get(selection)

    # Gets the value from does file exist function
    # if th efile exists then we open the file in append mode and append the selected anime to the users favourite list
    if self.does_file_exist(usr_name+'.txt'):
        with open(usr_name+'.txt', "a") as f:
            f.write(value+'\t')
    # Else if the file does not exist then we create it using the write mode and write the selected value to the file
    else:
        with open(usr_name+'.txt', "w") as f:
            f.write(value+'\t')


    # Once the selected anime has been added to the users favourites list we give them the option to view their cirrent favourites list
    user_choice = tk.messagebox.askquestion("ADDED TO FAVOURITES",value+" Has been added to your favourites list\n Would you like to view your current favourites?")

    # If they select yes  then we execyte the view_favourites function which displays the current favourites
    if user_choice == 'yes':
        self.view_favourites()
    # If they select no then we just close the message box and return to the profile screen
    else:
        pass

# Function that displays the users curerent favourite animes on a pop up window
def view_favourites(self):
    # Creates the pop up window
    pop_up = tk.Toplevel()
    pop_up.title('Your favourites list')

    # retrieves the users current favourite list by reading it from the text file
    with open(usr_name+'.txt', "r") as f:
        for lines in f:
            if usr_name in lines:
                favs = lines
    # Creating and displaying the label which will display the favourites list on screen
    favs_label = tk.Label(pop_up, text = favs)
    favs_label.pack()

    # Creating and displaying a button which will close the screen and return to the profile screen when the user is done
    close = tk.Button(pop_up, text= "Close", command = self.forget())
    close.pack()

    # Pop up windows mainloop
    pop_up.mainloop()


# Function to check whether a file already exist using the path.exists function from the os library
def does_file_exist(self,file_name):
    return os.path.exists(file_name)
```

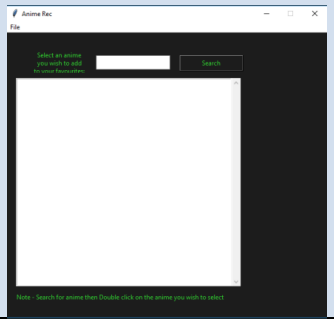
Profile back end testing

Test Case	**Check that the selected anime is add to the users favourite list which is stored in a text file**
Test data	(select these animes) Steins Gate, Fairy tail
Expected result	The names of the selected animes should be in the text file (Steins Gate, Fairy tail)
Actual result	
Modification	N/A

Test Case	**Check that the users favourite anime list is correctly displayed on the screen**
Expected result	The names of the users current favourite animes should be displayed on screen (Steins Gate, Fairy tail)
Actual result	
Modification	N/A

09/04/2018 – Profile prototype interview with client, Josh Santillan

Comment [MR117]: Client and success criteria review for the profile screen

Success criteria / requirement	Achieved / Not Achieved (with evidence)
Consistent anime theme	Achieved 
Feature which allows the user to store their favourite animes	Achieved 
Feature which adds the user's favourite list from this app to their myanimelist.net account	Not Achieved Reason – myanimlist.net API documentation does not have any details on how to link and modify a myanimelist.net's user's account.

Josh Santillan, Lead client

Final Anime Recommender app alpha testing

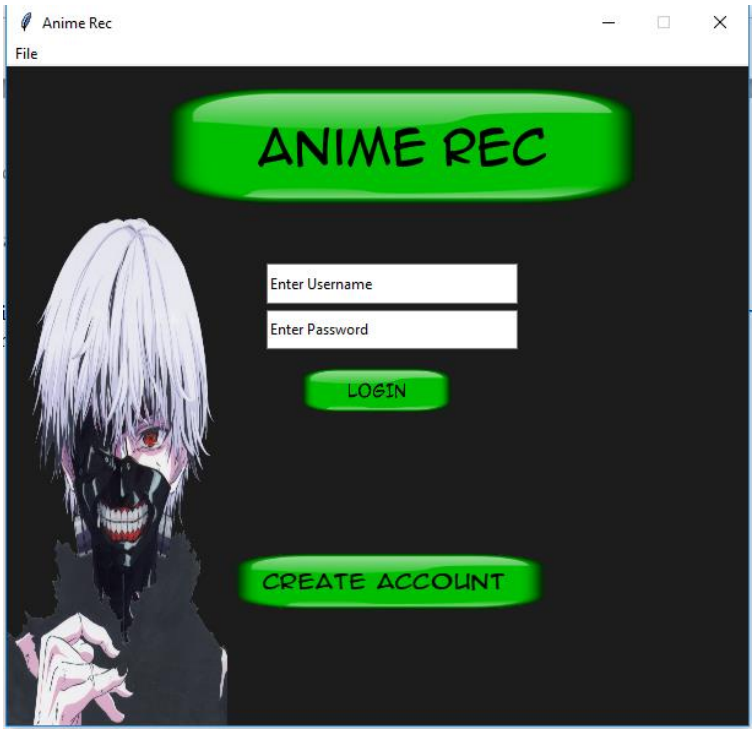
Comment [MR118]: Alpha testing for the final prototype of the entire system

During this phase of testing on the nearly completed system I will be using the app like a user in order to find any bugs in an attempt to find them.

Launching the app

Expected result: Loads the home screen

Actual result:



The app launched as expected with no hiccups or errors and the home screen is displayed correctly.

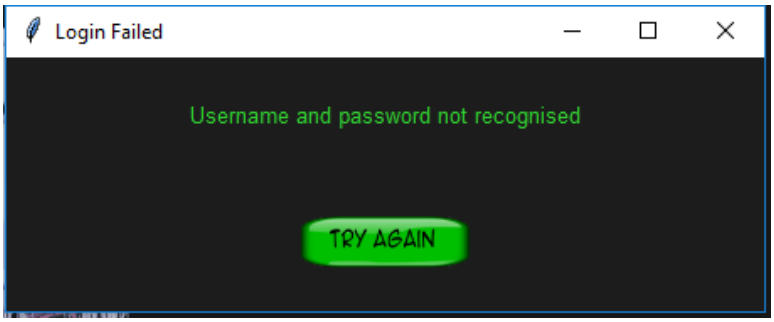
STATUS: PASSED TESTING

Comment [MR119]: Passing off each section once I am happy that it fulfils the test criteria and that there are no bugs.

Attempting to log in with an account that doesn't exist

Expected result: Display login fail prompt

Actual result:

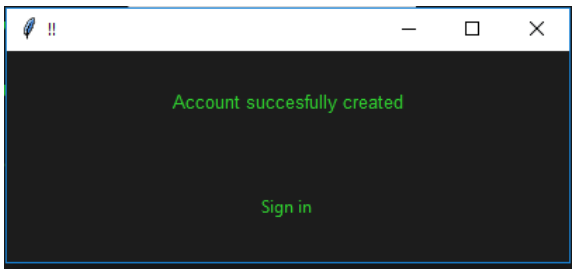


Comment: Works as intended, displaying a prompt saying the login details are not recognised and prompting the user to try again.

STATUS: PASSED TESTING

Attempting to create an account

Actual result:



Comment: Works as intended, creating an account and storing it in the accounts database
However my clients and I have spotted that the sign in button does not match the other buttons in our app so I have decided to create an image for the sign in button.
Modification:

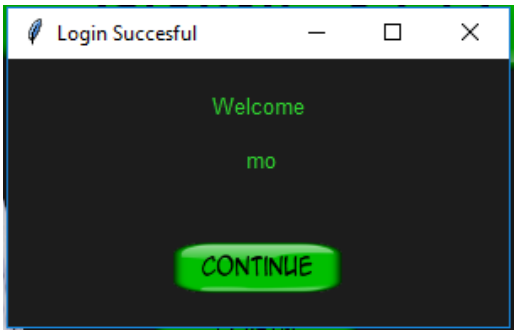
Comment [MR120]: Wherever I have found a bug or something I'm not happy with for any section below I have stated it like I have done here

STATUS: PASSED TESTING

Attempting to log in with the created account

Expected result: Logins successfully

Actual result:



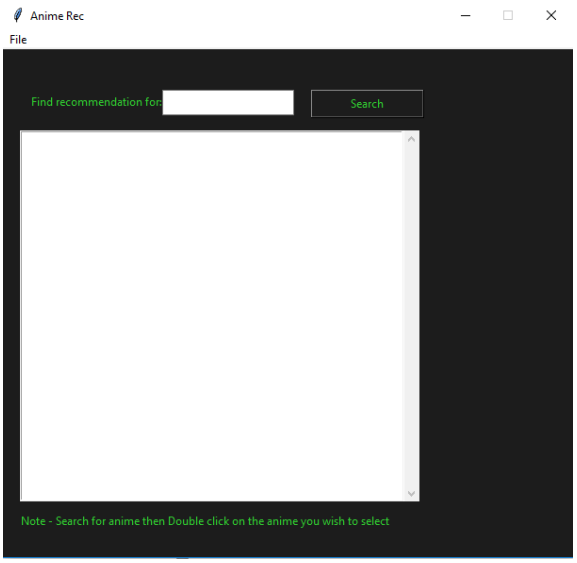
Comment: Works as intended, logging into the created account and displaying a prompt welcoming the user with their username

STATUS: PASSED TESTING

Attempting to go to the Get rec screen from the menu

Expected result: Displays the Get Rec screen

Actual result:



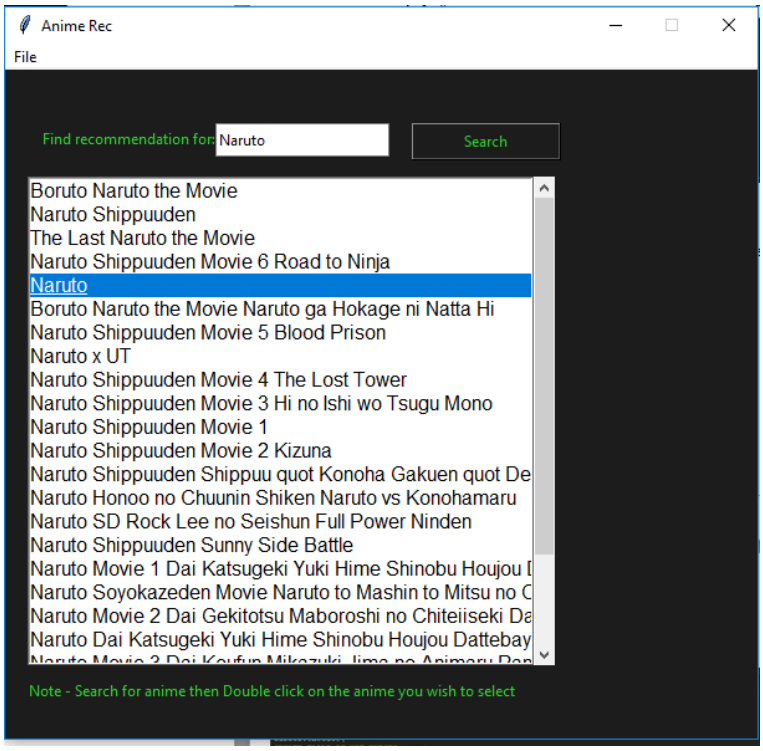
Comment: Works as intended, displaying the Get Rec screen as designed.

STATUS: PASSED TESTING

Searching for an anime to get a recommendation for

Expected result: Populates the list box with the search results

Actual result:



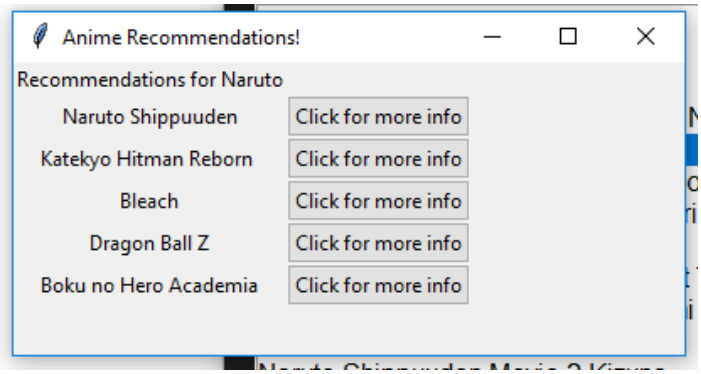
Comment: Works as intended, populating the list box with all animes that contain Naruto in their name allowing the user to select the anime they intended to search for.

STATUS: PASSED TESTING

Getting recommendations for a selected anime

Expected result: Generates 5 similar animes for the anime Naruto

Actual result:

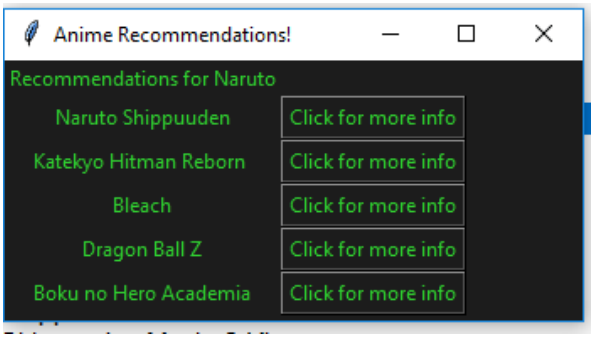


Comment: Works as intended (Generated animes are similar to Naruto) but the aesthetics don't match the theme of the app.

MODIFICATIONS:

Adding code to the configurations of the widgets

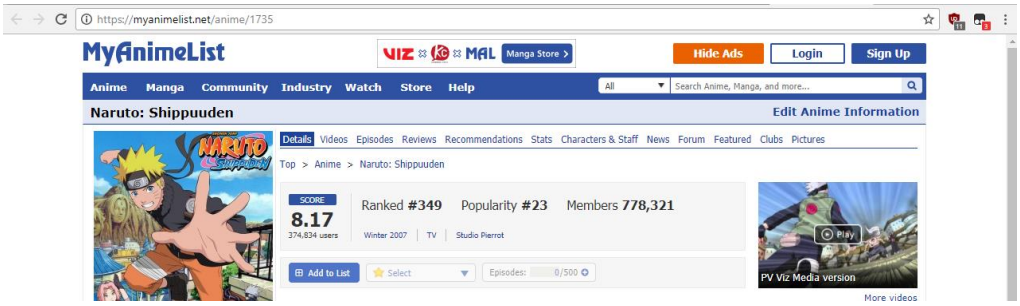
```
bg='greyll',fg='limegreen'
```



STATUS: PASSED TESTING

Checking if the Click for more info button for the recommendations work

Clicking on the 'CLICK FOR MORE INFO' button for the anime Naruto Shippuuden (the first one)



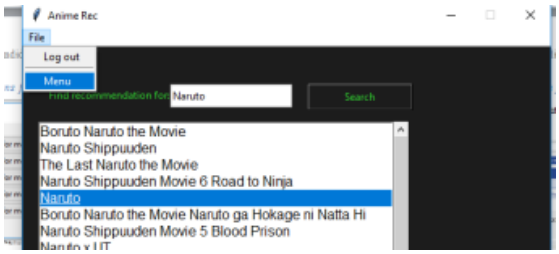
Comment: Works as intended, click on the button opens up the myanimlist.net website for 'Naruto Shippuden'

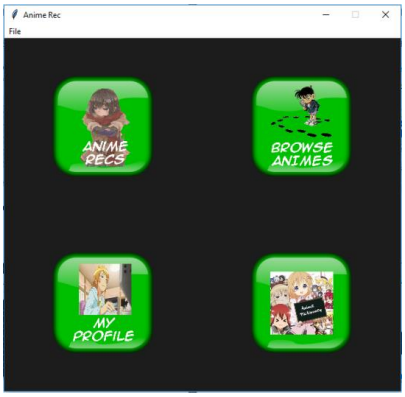
STATUS: PASSED TESTING

Checking if the Menu button works in the Menu bar

Expected result: Switch from the Get Rec screen to the Menu Screen when the Menu button is clicked.

Actual result:





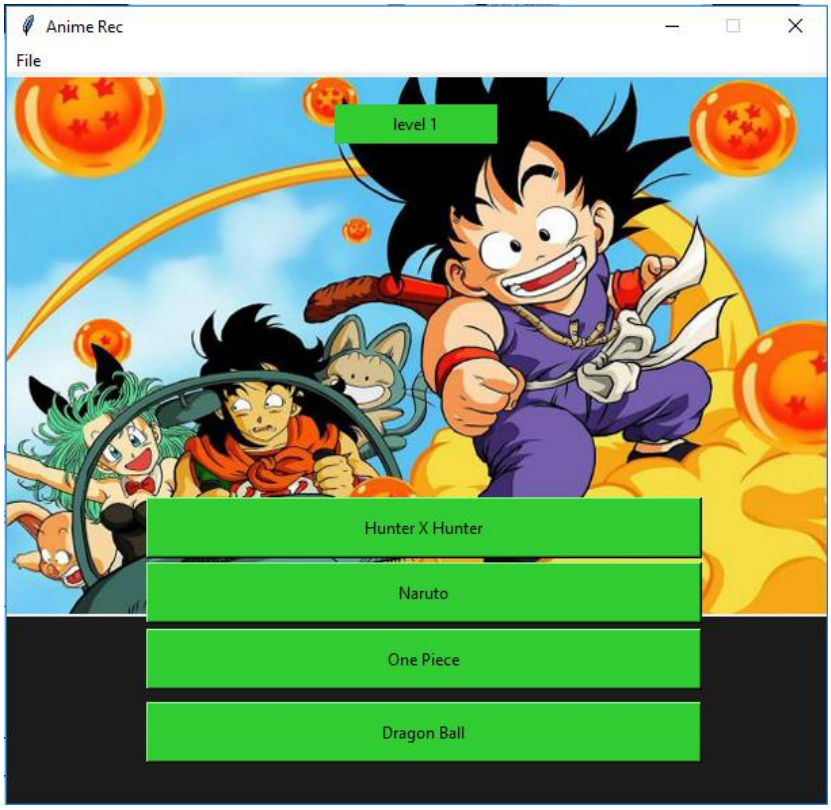
Comment: Works as intended, switches from the Get rec screen to the Menu screen

STATUS: PASSED TESTING

Attempting to go to the Pictionary screen from the Menu

Expected result: Switch from the Menu screen to the Pictionary screen

Actual results:



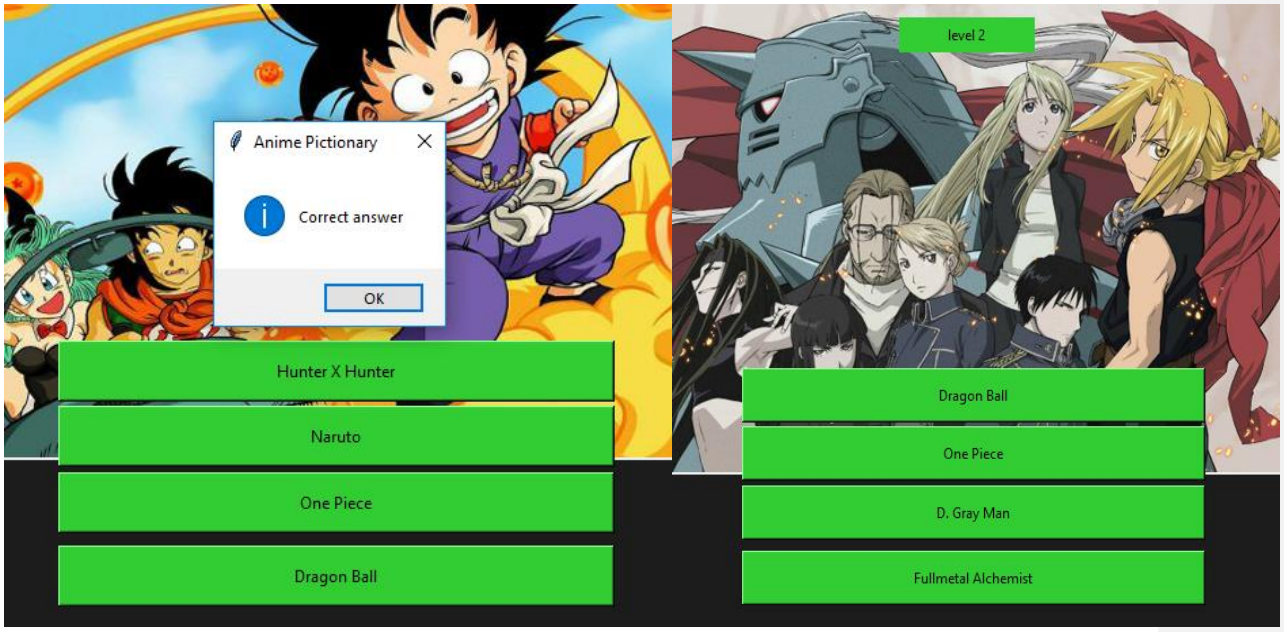
Comment: Works as intended, switches from the Menu screen to the Pictionary screen

STATUS: PASSED TESTING

Selecting the correct answer in the Pictionary quiz

Expected Result: Displays a prompt saying that we selected the correct answer and then displays the next question (correct answer = Dragon Ball)

Actual result:



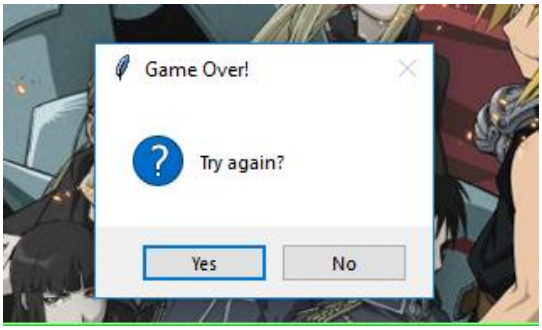
Comment: Works as intended displays a prompt saying that we got the correct answer and switches to the next question when we click ok.

STATUS: PASSED TESTING

Selecting the wrong answer in the Pictionary quiz

Expected Result: Display a prompt saying that the game is over and ask the user if they want to try again or return to the menu.

Actual result:



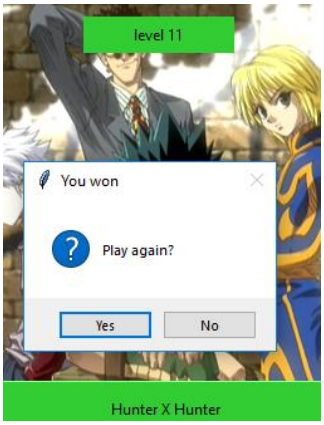
Comment: Works as intended, display Game Over prompt and askes the user if they wish to try again.

STATUS: PASSED TESTING

Reaching the end of the quiz

Expected result: Display a prompt saying that the user has won the game.

Actual result:



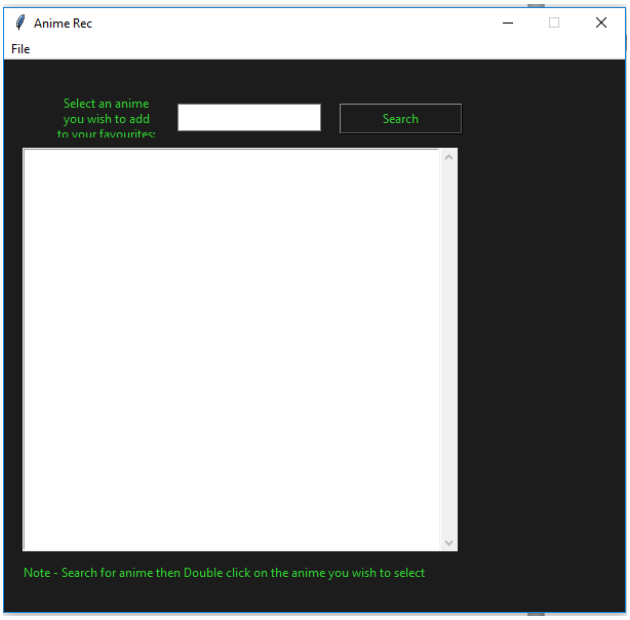
Comment: Works as intended, displays you won prompt

STATUS: PASSED TESTING

Attempting to go to the Profile screen from the Menu screen

Expected result: Switch from the Menu screen to the Pictionary screen

Actual results:



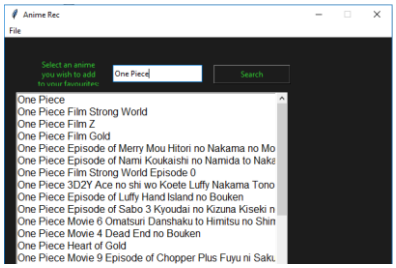
Comment: Works as intended, switches from the menu screen to the Profile screen

STATUS: PASSED TESTING

Searching for an anime to add to favourites

Expected result: (test data: 'One Piece') Populates the list box with the search results for the anime 'One Piece'

Actual result:



Comment: Works as intended, populating the list box with all animes that contain 'One Piece' in their name allowing the user to select the anime they intended to search for.

STATUS: PASSED TESTING

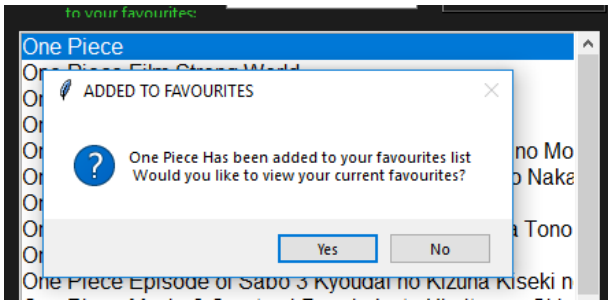
Adding the selected anime to user's favourites list

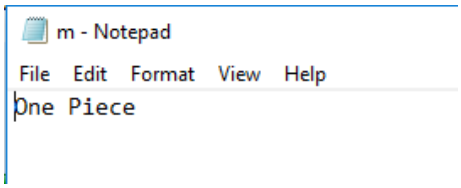
Expected result: Add the selected anime to the favourites list which is stored a text file and display a prompt saying that the anime was added to the favourites list and give the user an option to view their current favourites list.

Test Data: 'One Piece'

Therefore we expect the anime 'One Piece' to be added to the favourites list

Actual Result:





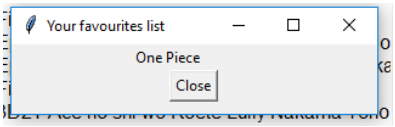
Comment: Works as intended, displaying the prompt and successfully adding the selected anime to the favourites list in the text file

STATUS: PASSED TESTING

Viewing the current anime favourites list

Expected result: Display the current favourites list which only contains 'One Piece' at the moment.

Actual Result:



Comment: Works as intended, displaying the current anime favourites list which currently only contains One Piece

STATUS: PASSED TESTING

Profile screen modification

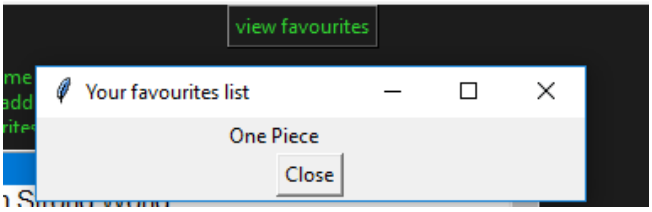
During the testing of the profile screen I realised the only way for a user to check their current favourites is to add an anime to the list first which is inconvenient.

To solve this I have decided to add a button on the main profile screen which will allow the user to view their current favour rites without adding an animes to the list to do so.

Code modification:

Creating a button which will call the function which will display the user's current favourite animes on the screen.

```
# Creating and displaying a button, which when clicked displays the user's current favourite anime list
self.view_butt = tk.Button(self, text='view favourites', command = self.view_favourites, bg = 'grey11', fg='lime green', borderwidth=2,highlightbackground="lime green")
self.view_butt.pack(side='top')
```



So now we have successfully added a button to the main profile screen which will display the user's favourite anime list.

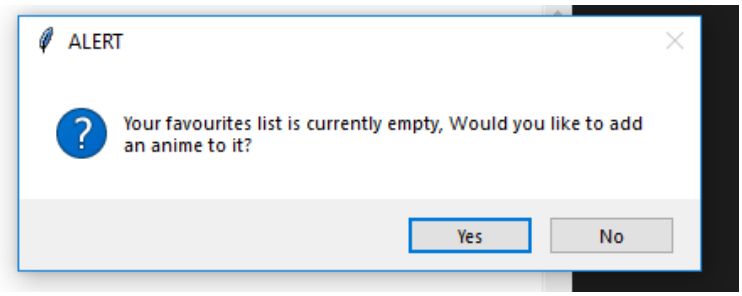
However, everything worked fine in this initial test run as we had already created a favourites list when we added our first anime to it however I re ran the problem with an account that hadn't created a favourites list and this error popped up:

```
w_favourites
    with open(usr_name+'.txt', "r") as f:
FileNotFoundError: [Errno 2] No such file or directory: 'm.txt'
```

This error pop up because the view favourites function is called and is trying to open the user's favourite list text file although it doesn't exist yet so the solution to this problem is by creating a prompt that says to the user that they have nothing in their favourites list at the moment.

Fix:

```
# Function that displays the users curerent favourite animes on a pop up window
def view_favourites(self):

    # UPDATE - added steps that will check if the favourites list text file exists
    # If it doesnt then we prompt the user that their list is empty
    if self.does_file_exist(usr_name+'.txt') == False:
        # Giving them an option to add an anime to their favourites list  or not
        user_choice = tk.messagebox.askquestion("ALERT","Your favourites list is currently empty, Would you like to add an anime to it?")

        # terminates the function regardless of the user choice thus preventing the error yet still keeping the program functional
        if user_choice == 'yes':
            return
        else:
            return
```



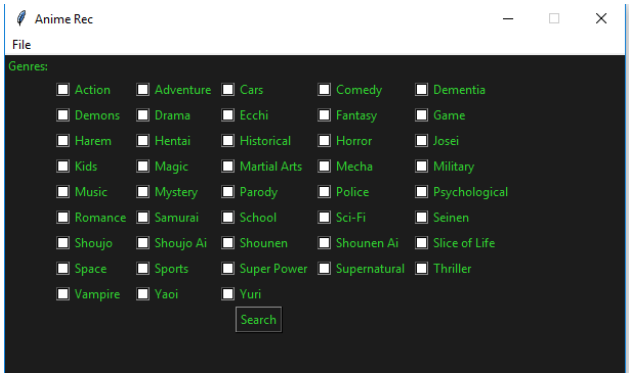
Comment: Now the Profile screen is fully functional and has improved usability functions.

STATUS: PASSED TESTING

Attempting to go to the Browse screen from the Menu screen

Expected result: Switch from the Menu screen to the Pictionary screen

Actual results:



Comment: Works as intended, switches from the menu screen to the browse screen

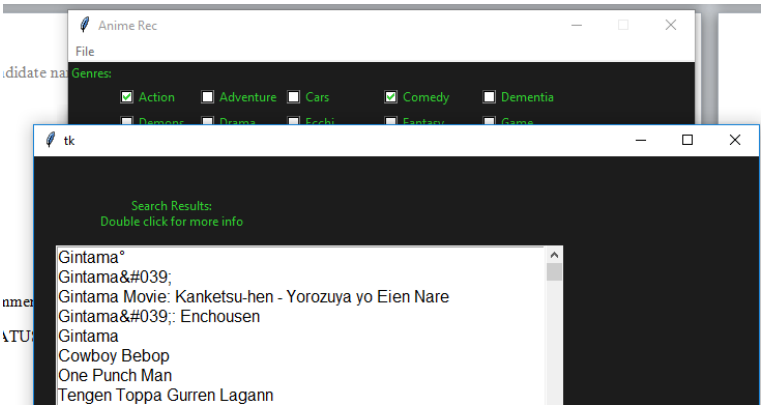
STATUS: PASSED TESTING

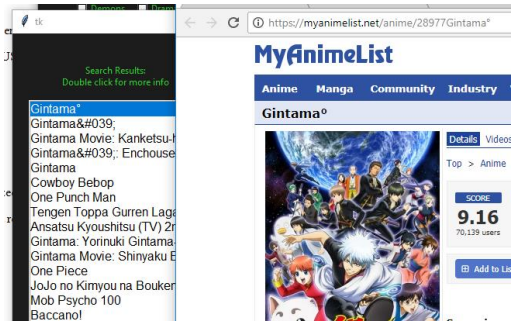
Searching for anime in the browse screen and web link

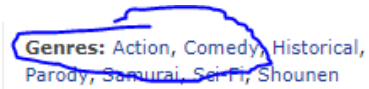
Test Data: Action and Comedy

Expected result: Display all anime have both Action and Comedy as a genre and double clicking on a search result should open the myanimelist.net web page for that anime

Actual result:







Comment: Works as intended, Display all anime with Action and Comedy as their genres and opens the right web link for it's respective anime.

STATUS: PASSED TESTING

Clients review on final prototype complete alpha testing

After my clients had reviewed the results of my alpha testing on the final prototype they are satisfied that there are no more bugs within the program and that the program has validation wherever it is needed which means that it will not break down In any circumstances where the user inputs the invalid values or even if they input valid values.

Building the application into an executable and building an installer for the executable

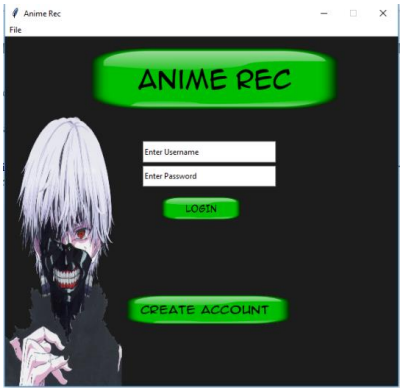
In order to compile the application (Python scripts, images, files …) into an executable file which will work on any 64 but version of windows I will be using a Python library called **cx_Freeze**. **cx_Freeze** is a set of scripts and modules for freezing Python scripts into executables, in much the same way that py2exe and py2app do. Unlike these two tools, **cx_Freeze** is cross platform and should work on any platform that Python itself works on.

The script that freezes my application into an executable:

```
import cx_Freeze # Importing the cx_freeze module
from cx_Freeze import * # Ensuring everything is imported from cx_freeze as the chance for errors during the build is high
import sys # Importing the system library which aids in building the executable
import os # Importing the os library which aids in building the executable
import matplotlib # Importing the matplotlib library which aids in building the executable


# Manually declaring the path for tcl which will help build our executable as we will need Tkinter to run our app
os.environ['TCL_LIBRARY'] = r'C:\Users\mar71\AppData\Local\Programs\Python\Python36\tcl\tcl8.6'
os.environ['TK_LIBRARY'] = r'C:\Users\mar71\AppData\Local\Programs\Python\Python36\tcl\tk8.6'

base = None

if sys.platform =='win32':
    base = "Win32GUI"

executables = [cx_Freeze.Executable("main.py", base=base)]

# Stating what modules and files our app needs so they can be included in the build
cx_Freeze.setup(
    name = "AnimeRec",
    options = {"build_exe":{"packages":["tkinter.ttk","tkinter","sqlite3","numpy","pandas","string","webbrowser","pickle","PIL","PIL.ImageTk","PIL.Image","random","glob","os","sys","tkinter.messagebox"],
                            "include_files":["content_based_system.py","AccountDB.py",
                                             "AnimeRecs.png","browse_button.png","BrowseAnimes.png","con_button.png","finalized_model.sav","anime","get_rec_button.png",
                                             "questions.txt","Help.png","Home_title.png","Ken_Masked.png","login.png","Login_button.png","MyProfile.png","pictionary_button.png",
                                             "profile_button.png","Reg_Button.png","register_button.png","sign_in_button.png","title.png","Try_Again_Button.png",
                                             "tk86t.dll","tcl86t.dll","animeData.db","anime_features","0.jpg","1.jpg","2.jpg","3.jpg","4.jpg","5.jpg",
                                             "6.jpg","7.jpg","8.jpg","9.jpg","10.jpg"
                                             ]}},
    version = "0.01",
    description="AnimeRec",
    executables = executables
    )
```

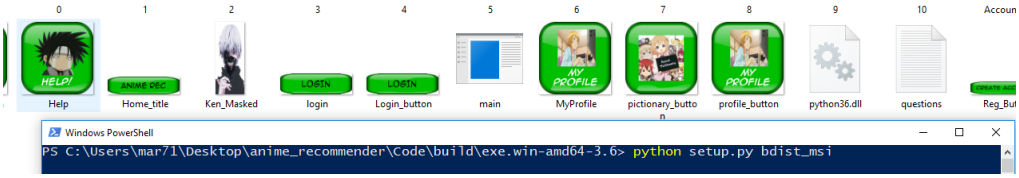
Running this script creates this folder containing a functional executable:

Proof that the executable works on all 64 bit versions of windows:



However the build folder with the executable and its files is a bit messy so let's clean it up by making a single installer file which will install our app and its required file in a folder of the user's choice just like installing any other program on windows.

Running this command in the command line in the directory of the setup.py file and the executable:



Creates an installer for my app:

*Note – mediafire download link for the installer:
http://www.mediafire.com/file/r62epfgjdbmvexp/MollerAnimeRec-0.01-amd64.msi

Client review of alpha testing

Comment [MR121]: Client sign off and feedback of development

Josh Santillan: I have reviewed the results of the alpha testing you have conducted and after the improvements and further modifications you have made to root out bugs and flawas of the system. I am happy to sign off and mark the system development of the Anime recommender app as complete.

Josh Santillan, Lead client

EVALUATION

Testing to inform evaluation (done by myself alongside Hannah Arifi, Client)

Comment [MR122]: Testing for evaluation

App installation testing

*Note – mediafire download link for the installer:
http://www.mediafire.com/file/r62epfgjdbmvexp/MollerAnimeRec-0.01-amd64.msi

Test Case	Test steps/Input	Expected results	Actual result	Requirement met?	Screen shots no.
1.Check if the installer correctly install the anime recommender app	-Launch the installer named (MollerAnimeRec-0.01-amd64) -Select a path to install the app and click next(allow if UAC pops up) -Click finish -Go to installation path and launch main.exe	Launch app successfully (Display Home screen)	As expected launches without any error and displays the Home screen	YES	1

Home screen testing

Test Case	Test steps/Input	Expected results	Actual result	Requirement met?	Screen shots
2. Login functionality	**VALID**: Correct login details are entered	Logins successfully and displays menu screen	Login successfully and Menu screen is displayed	YES	2
	INVALID: Incorrect login details are entered	Login failed pop up is displayed and the user is prompted to try again	Login fail pop up screen with button for user to try again	YES	2.1

	INVALID: nothing is entered in the username and password entry fields	Login failed pop up is displayed and the user is prompted to try again	Login fail pop up screen with button for user to try again	YES	2.2
3. Create account button functionality	**VALID : Create account** button is clicked	Switch to Register screen	Register screen is displayed	YES	3
	INVALID: Enter button is clicked	Nothing should happen	Nothing happens	YES	N/A

Register screen testing

Test Case	**Test steps/Input**	**Expected results**	**Actual result**	**Requirement met?**	**Screen shots**
4. Register button functionality	**VALID**: Fill in account registration details correctly leaving no entry fields empty	A pop up saying that the account is successfully created and give the option for the user to sign in	A pop up window with the expected message and options is displayed	YES	4
	INVALID: Leave one or more entry fields empty	a pop up saying 'Please fill all the fields' should be displayed with the option to try again	A pop up window with the expected message and options is displayed	YES	4.1
	INVALID: Enter two different passwords in the password and re-enter password fields	Password mismatch pop up should be displayed with the option to try again	Password mismatch pop up is displayed with the expected message and option to try again	YES	4.2

Menu screen testing

Test Case	**Test steps/Input**	**Expected results**	**Actual result**	**Requirement met?**	**Screen shots**
5. Anime Rec button functionality testing	**VALID:** Click on the anime rec button	Switch to the Get Recommendation screen	Displays the Get Rec screen	YES	5

	INVALID: Enter button is clicked	Nothing should happen	Nothing happens	YES	N/A
6. Browse animes button functionality testing	**VALID:** Click on the anime rec button	Switch to the browse screen	Displays the browse screen	YES	6
	INVALID: Enter button is clicked	Nothing should happen	Nothing happens	YES	N/A
7. My profile button functionality testing	**VALID:** Click on the My profile **button**	Switch to the profile screen	Displays the profile screen	YES	7
	INVALID: Enter button is clicked	Nothing should happen	Nothing happens	YES	N/A
8. Anime Pictionary button functionality testing	**VALID:** Click on the a **Anime** Pictionary **button**	Switch to the Pictionary screen	Displays the Pictionary screen	YES	8
	INVALID: Enter button is clicked	Nothing should happen	Nothing happens	YES	N/A
9. Check if the logout option from the File bar works	**VALID: Click on the file option from the menu bar and then click logout**	The user should be logged out and the app should reset returning to the home screen	Logged out successfully and returns to the home screen	YES	9 and 1

Get Recommendation screen testing

Test Case	Test steps/Input	Expected results	Actual result	Requirement met?	Screen shots
10. Anime search functionality	**VALID:** Type an anime name ('Naruto') and click on the search button	The Listbox should be populated with animes that contain 'Naruto' in their name.	The listbox is populated with animes that have similar names to 'Naruto'	YES	10
11. Check whether recommendation is generated when an anime is selected.	**VALID:** *(List box is currently populated with search results of 'Naruto')* Double click on the anime called 'Naruto'	A new pop up window shgould be displayed with 5 animes that are similar to 'Naruto. Also at the side of each recommendation their should be a	Pop up window is displayed with 5 animes that are similar to 'Naruto'	YES	11

		button that says click for more info.			
12. Check whether the click for more info button is functional	**VALID:***(Currently on the recommendation pop up window)* Click on the more info button for 'Bleach	The myanimelist.net website for bleach should open in the users browser.	https://myanimelist.net/anime/269 is displayed in browser	YES	12
13. Evaluate the recommender systems accuracy	Check each feature of the generated recommendations and compare them to the anime that the user entered to find recommendations for.	The generated recommendations should have a lot of the same features 'Naruto'	The generated animes were indeed very similar to 'Naruto'	YES	13

Profile screen testing

Test Case	Test steps/Input	Expected results	Actual result	Requirement met?	Screen shots
14. Check functionality of selecting a favourite anime	VALID: (currently on profile screen) 1.Enter anime name ('Naruto') 2.Click the search Button	display all animes containing 'Naruto' in their name in the scrollbar widget	The listbox is populated with animes that have similar names to 'Naruto'	YES	14
15. Adding the selected anime to user's favourites list	VALID: Double click on the anime called 'Naruto'	Display prompt saying that 'Naruto' has been added to your favourites and 'Naruto' should be added to the favourites text file	Expected prompt is shown and 'Naruto' is in the favourites text file	YES	15
16. Check whether the users current favourites list is correctly displayed	VALID; Click on yes on the prompt when you add an anime to your favourites or click on the view favourites button on the main profile screen	Display the user's current favourites list which only has 'Naruto' in it at the moment	Favourites list is displayed in pop up with only 'Naruto' in it	YES	16

Browse screen testing

Test Case	Test steps/Input	Expected results	Actual result	Requirement met?	Screen shots
17. Check that browsing for an anime works	1. Select Action and Comedy. 2.Click Search	A new window with a list of animes that have both Action and Comedy as their genres should be displayed	List box populated with search results for action and comedy animes.	YES	17
18. Checking whether double clicking on a search result works	**VALID:** double click on the 'Gintama' search result.	The myanimelist.net website for 'Gintama' should open in the users browser.	https://myanimelist.net/anime/269 is displayed in browser	YES	18
19. Check if the Menu option from the File bar works	**VALID:** Click on the file option from the menu bar and then click logout	Swith to the Menu screen	Returned to the menu screen	YES	19

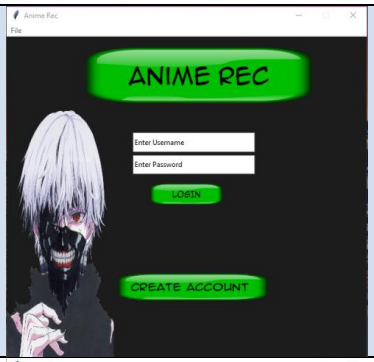
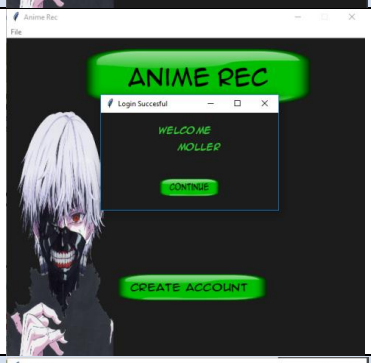
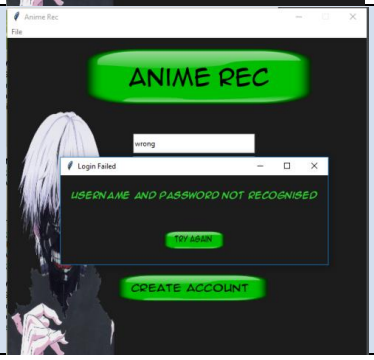
Pictionary screen testing

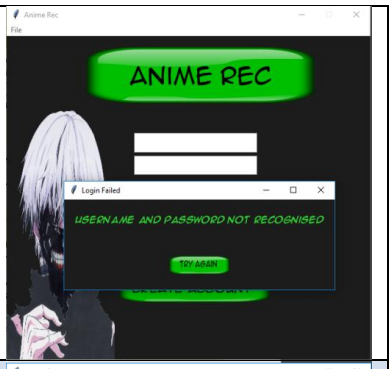
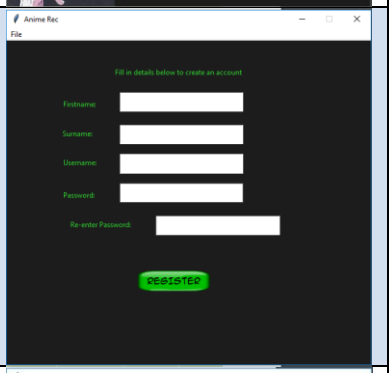
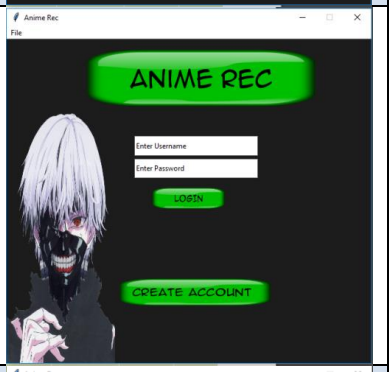
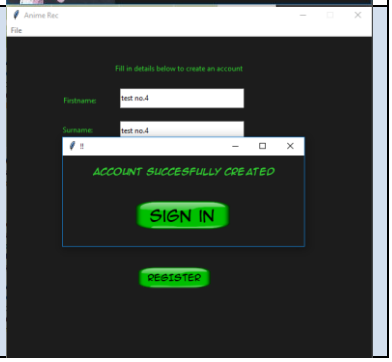
Test Case	Test steps/Input	Expected results	Actual result	Requirement met?	Screen shots
20. Check result in the case of selecting the correct answer	(Depends on the current image) While I was doing this test the correct answer was 'Hunter X Hunter' So I selected it.	1.Display correct prompt 2.Increment score 3.Display next question	All 3 expected points were met.	YES	20
21. Check result in the case of selecting the WRONG answer	(Depends on the current image) While I was doing an incorrect answer was: 'Ninja scroll'	1.Display game over screen 2.Prompt user to exit or try again	Both expected points were met	YES	21
22. Finishing the quiz	Answer all the questions correctly	1.Display you won screen 2.Prompt user to play again or try again	Both expected points were met	YES	22
23. Check if the Menu option from the File bar works	**VALID:** Click on the file option from the menu bar and then	Swith to the Menu screen	Returned to the menu screen	YES	23

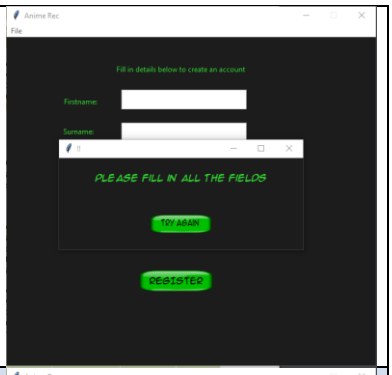
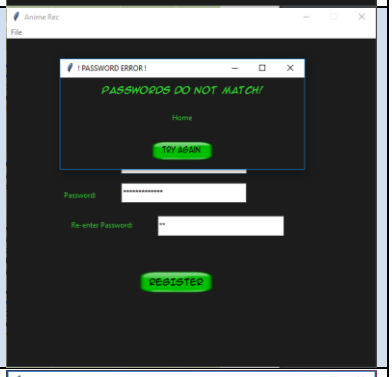
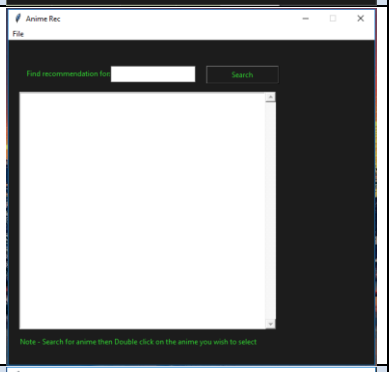
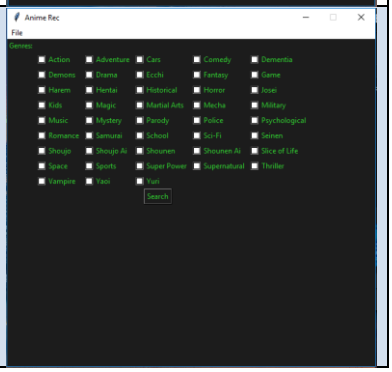
	click logout				

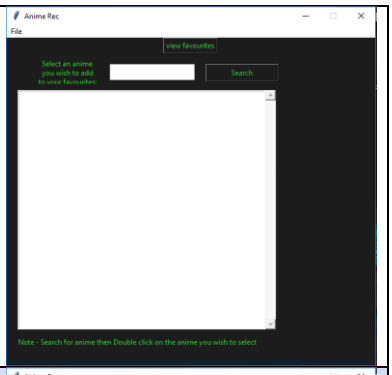
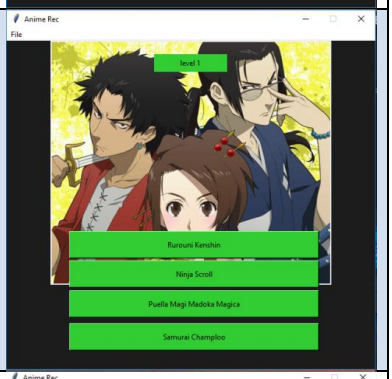
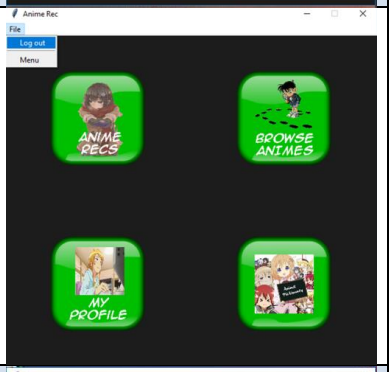
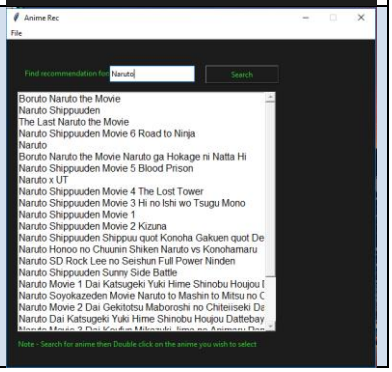
Josh Santillan, Lead client

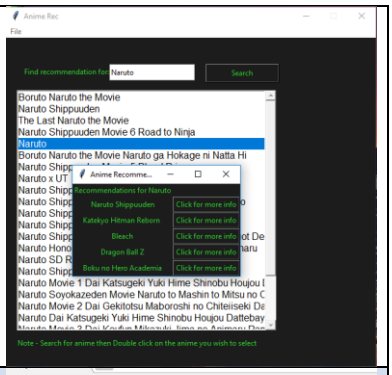
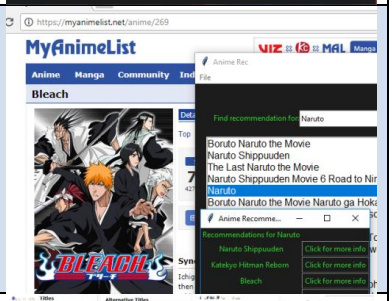
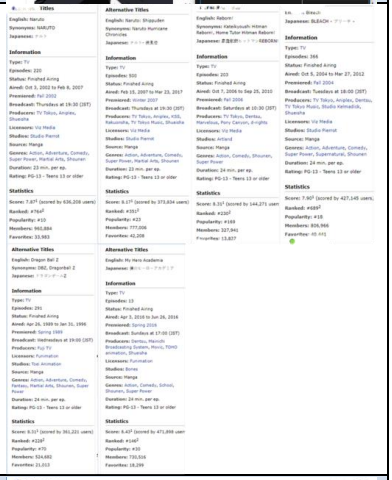
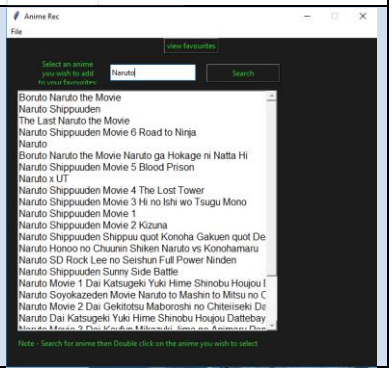
Screenshot for the testing above

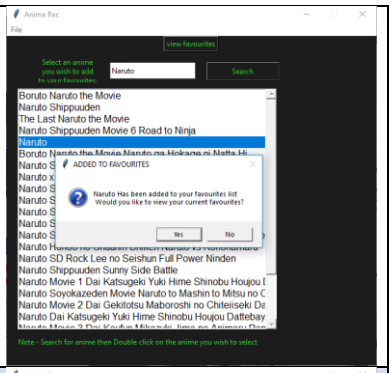
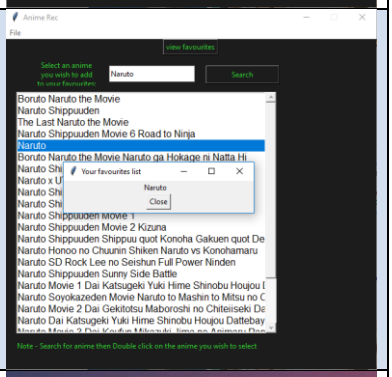
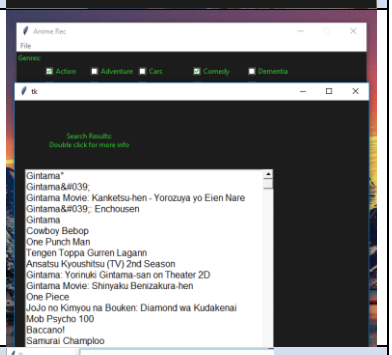
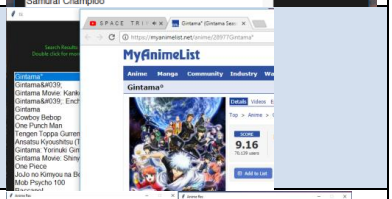
Screen shot number	Screenshot
1	
2	
2.1	

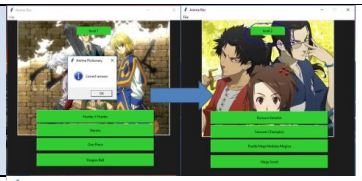
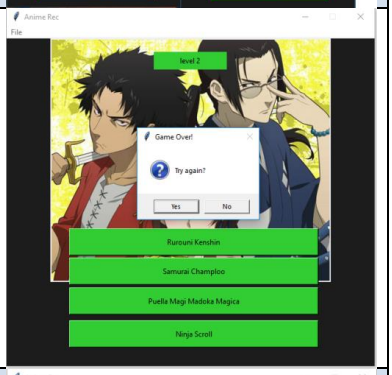
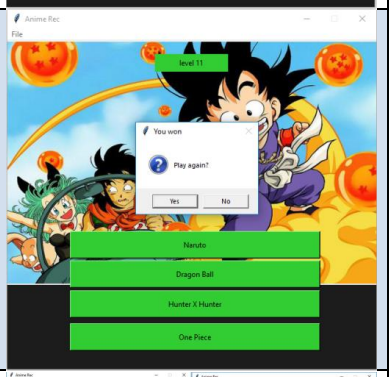
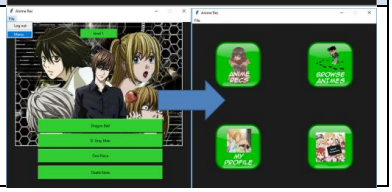
2.2	 Anime Rec File ANIME REC Login Failed USERNAME AND PASSWORD NOT RECOGNISED TRY AGAIN
3	 Anime Rec File Fill in details below to create an account Firstname: Surname: Username: Password: Re-enter Password: REGISTER
3.1	 Anime Rec File ANIME REC Enter Username Enter Password LOGIN CREATE ACCOUNT
4	 Anime Rec File Fill in details below to create an account Firstname: test no.4 Surname: test no.4 ACCOUNT SUCCESFULLY CREATED SIGN IN REGISTER

4.1	 Anime Rec File Fill in details below to create an account Firstname: Surname: PLEASE FILL IN ALL THE FIELDS TRY AGAIN REGISTER
4.2	 Anime Rec File ! PASSWORD ERROR ! PASSWORDS DO NOT MATCH! Home TRY AGAIN Password: Re-enter Password: REGISTER
5	 Anime Rec File Find recommendation for Search Note - Search for anime then Double click on the anime you wish to select
6	 Anime Rec File Genres Action Adventure Cars Comedy Dementia Demons Drama Ecchi Fantasy Game Harem Hentai Historical Horror Josei Kids Magic Martial Arts Mecha Military Music Mystery Parody Police Psychological Romance Samurai School Sci-Fi Seinen Shoujo Shoujo Ai Shounen Shounen Ai Slice of Life Space Sports Super Power Supernatural Thriller Vampire Yaoi Yuri Search

7	 Anime Rec File view favourites Select an anime you wish to add to your favourites Search Note - Search for anime then Double click on the anime you wish to select
8	 Anime Rec File level 1 Rurouni Kenshin Ninja Scroll Puella Magi Madoka Magica Samurai Champloo
9	 Anime Rec File Log out Menu ANIME RECS BROWSE ANIMES MY PROFILE
10	 Anime Rec File Find recommendation for Naruto Search Boruto Naruto the Movie Naruto Shippuuden The Last Naruto the Movie Naruto Shippuuden Movie 6 Road to Ninja Naruto Boruto Naruto the Movie Naruto ga Hokage ni Natta Hi Naruto Shippuuden Movie 5 Blood Prison Naruto x UT Naruto Shippuuden Movie 4 The Lost Tower Naruto Shippuuden Movie 3 Hi no Ishi wo Tsugu Mono Naruto Shippuuden Movie 1 Naruto Shippuuden Movie 2 Kizuna Naruto Shippuuden Shippuu quot Konoha Gakuen quot De Naruto Honoo no Chuunin Shiken Naruto vs Konohamaru Naruto SD Rock Lee no Seishun Full Power Ninden Naruto Shippuuden Sunny Side Battle Naruto Movie 1 Dai Katsugeki Yuki Hime Shinobu Houjou Naruto Soyokazeden Movie Naruto to Mashin to Mitsu no Naruto Movie 2 Dai Gekitotsu Maboroshi no Chiteiseki Naruto Dai Katsugeki Yuki Hime Shinobu Houjou Dattebay Note - Search for anime then Double click on the anime you wish to select

11	
12	
13	
14	

15	
16	
17	
18	
19	

20	
21	
22	
23	

Josh Santillan, Lead client

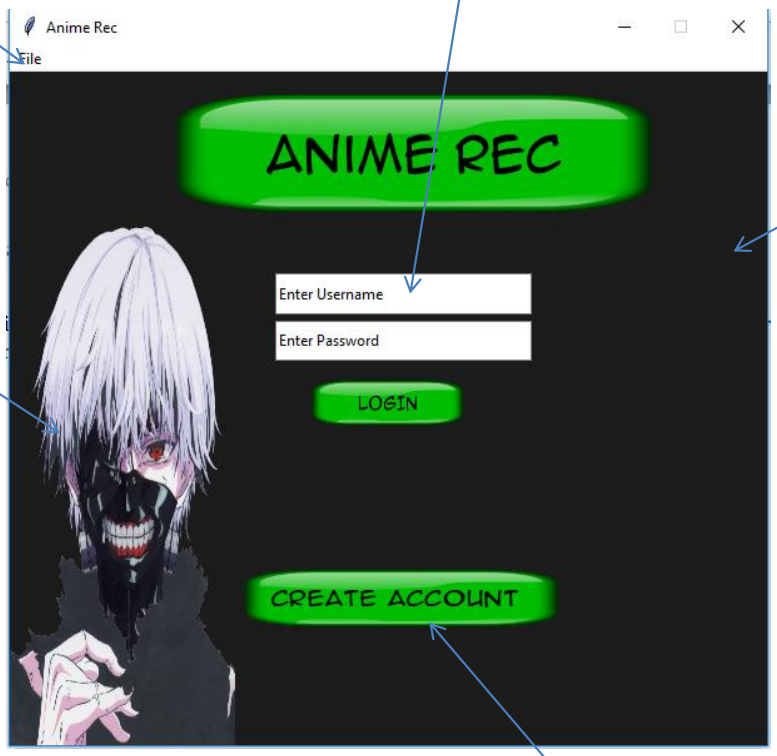
Comment [MR123]: Evaluation of usability features

Usability features

GUI

Place holder text indicating what to enter in the fields rather than labels to keep it minimalistic and clear and to aid the user

App and navigation options allowing the user to logout, exit or navigate through screens



Grey11 background colour to match colour scheme

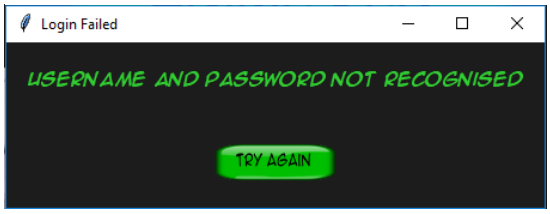
Anime related image to create a sense of familiarity with the users and to make the app more appealing

Clear custom buttons that match the anime theme with Anime Ace font and colour scheme with its green and black colours

Validation

Comment [MR124]: Evaluation of validation/robustness

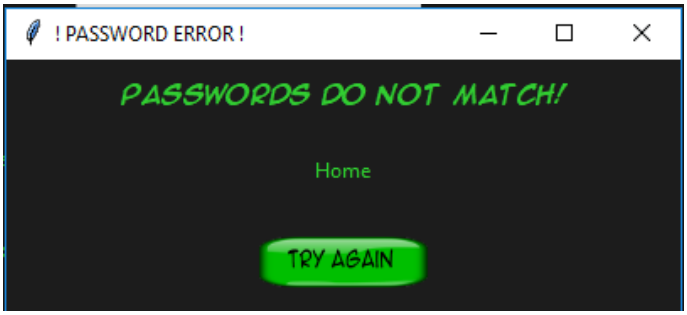
Validation if the user enters the wrong login details allowing them to try again



Validation if the user enters the correct login details allowing them to continue

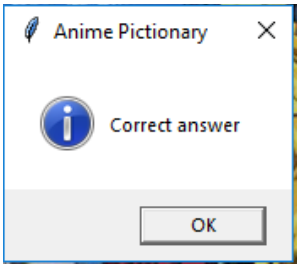
Validation if the user leaves an entry field blank

Validation if the user enter mismatching password

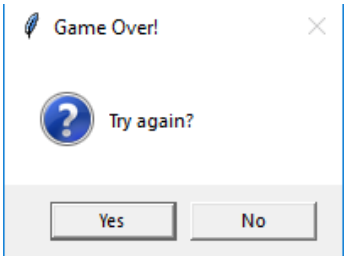


Validation if the user successfully creates an account

Validation if the user selects a correct answer



Validation if the user selects an incorrect answer



Comment [MR125]: Stakeholders feedback on GUI and features

GUI and features interview with Hannah Arifi (Client)

After Hannah had finished testing the entire system I decided to conduct a brief interview in order to get her feedback on each section of our app.

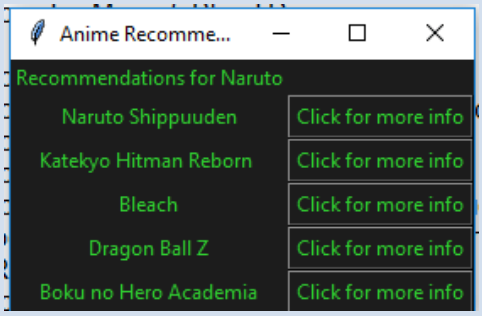
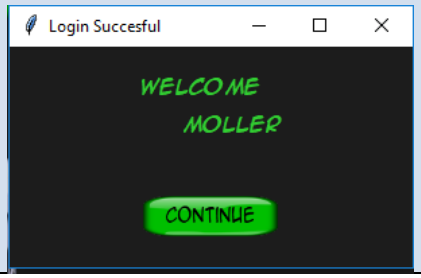
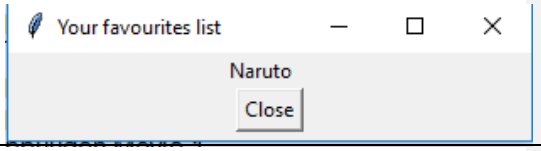
Feature	Hannah's feedback
GUI's colour scheme and anime theme	'I was very pleased with the aesthetics of the GUI. As not only was the entire app consistent with our chosen colour scheme but it also had things that were associated with anime that were integrated into the GUI in the right way.'
Home screen	'After testing the Home screen myself I concur that it has fulfilled all of its intended purposes such as having a fully functional login function which has validation and is robust so that even in the case of invalid data the program will not break. Furthermore, the home screen has a very nice and minimalistic design which is consistent with our theme and these factors aid in making it easy to navigate through for a user.'
Register screen	'The register screen is consistent with our theme and successfully fulfils its intended purpose of allowing a user to create an account. Furthermore the register function is very robust and includes validation as even in the case of invalid data the program will not break.'
Menu Screen	'The Menu screen is very pleasing to the eye with the use of the custom made anime themed buttons, which also make it very clear to the user what options are available for them to select from.'
Get Recommendation screen	'Going beyond the scope of the tests for the recommendations system I tried it myself with a few other animes that I have already seen and I've got to say that it is pretty accurate. For example the recommendations I received for the anime Naruto were all of similar genre's, plots, rating and I had already seen some of the recommended animes which means I can back up that they are in fact very similar to Naruto. Apart from the recommender system itself which is also very fast in its response time, it is also extremely easy to find a recommendation with the few steps of searching for anime, selecting it and getting the recommendations displayed in a pop up window. Very Nice!'

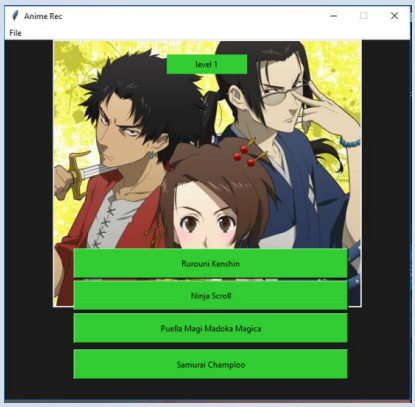
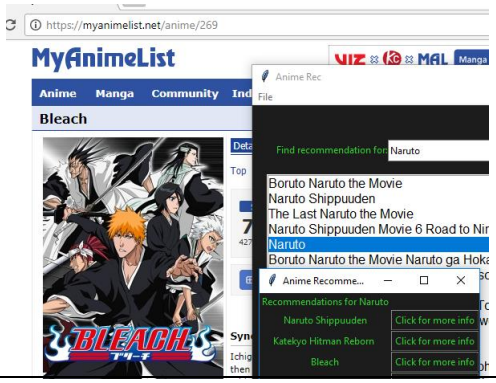
Profile Screen	'The profile screen is consistent with our theme and also fulfils the requirement of allowing the user to add animes to a list which will act as the user's favourites. Furthermore, it is very easy to add an anime to your favourites by simply searching for it using a list box and an entry widget and then adding to your favourites by clicking on the anime.'
Pictionary Screen	'I was very happy with the anime Pictionary screen as it worked really well and is a fun feature for our app which will be beneficial in attracting users. The quiz also has a lot of validation making it robust as even in the case of the user entering invalid data the program will not break. '
Browse Screen	'The browse screen is also consistent with our chosen theme and fulfils its very limited and specific purpose of allowing a user to search for animes based on genres. Furthermore, the browse screen is very robust as even when you search for another anime the results from the previous search are cleared from the list box(widget used to display the search results)'

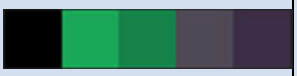
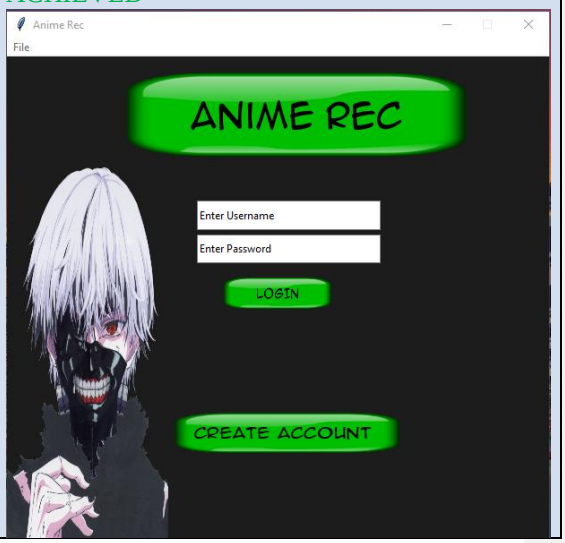
Josh Santillan, Lead client

Have the Requirements been met?

Comment [MR126]: Reviewing and justifying all the success criteria's and whether they have been met or not

REQUIREMENT	JUSTIFICATION	HAS IT BEEN ACHIEVED?
Content based Recommender system - This is the algorithm that will be used to generate recommendations. It works by calculating the similarities between animes using observations and features.	This feature is compulsory at it is the main solution to my problem which is how to find recommendations for animes. Using a content based recommender system will solve this problem as it will calculate similarities between animes and using these similarities we can recommend animes which are the most similar compared to each other.	ACHIEVED 
Response time - The time taken for the recommender system to compute through the anime dataset and yield recommendations.	From my research I have found that finding recommendations is possible to do manually, however this is time consuming. Therefore, for my app to be effective it must yield a faster response time than the current methods. My aim for the response time is in the worst case 40 seconds, as we computing with thousands of animes.	ACHIEVED The app launches in less than 2 seconds
User accounts - Each user should have their own account which will be used to store their preferences to a database.	Having user accounts will save users from having to re-enter their preferences every time they use our app. (as suggested by Thomas Wojenski)	ACHIEVED 
Profile - Feature that allows users to add their favourite animes to a list	This will aid in providing recommendations automatically to users rather than having them manually input animes for which they wish to	ACHIEVED 

	find similar animes to. In addition, they can share their preferences with other users so as to help each other to find animes to watch.	
Browsing tool - Tool users can use to find animes from the database by genre	This will allow for a broader search, enabling users who wish to find animes that meet certain genre criteria.	ACHIEVED 
Instructions- Notes for users who are encountering difficulties/problems	This feature will make our app more user friendly and can help prevent users having misunderstanding as to how to use the app.	NOT ACHIEVED – Solve by including a page with instructions, although the labels and buttons make the app fairly easy to use.
Anime Pictionary- Mini game where users have to guess the anime name based of an image of said anime.	Including an extra feature such as a game like this greatly improves the app's appeal to users and can help build a friendlier user environment through the sense of familiarity by implementing a game related to anime. Furthermore, from my research it has been concluded that such a feature is would be highly demanded and appreciated.	ACHIEVED 
Anime discussion sites- Embedding links to anime discussion sites within the app.	The anime community like to converse between each other sharing their opinions and giving advice on recommendations. Therefore, instead of implementing our own discussion feature, it was suggested to me to link more popular anime discussion sites such as MyAnimeList. This way, users can experience a	ACHIEVED 

	bigger and more active community which will enable them to discuss the recommendations they received from my app and evaluate or perhaps even recommend them to other users.	
Colour scheme 	Colour scheme that should remain consistent throughout the entire app, as chosen by popular opinion from questionnaire.	ACHIEVED 
Anime theme - Anime related fonts, images and quotes embeded genorously throughout the app.	Having an anime related theme to the app, will create an emotional connection to users, creating a more familiar and friendly environment. Furthermore, it will bring vibrancy and uniqueness to my app.	

Josh Santillan, Lead client

Comment [MR127]: Limitations

Limitations

Limitations	Justification
Collaborative filtering recommender system - Collaborative filtering analyses similar users and what Animes they prefer. For example, if the user looking for a recommendation had previously enjoyed watching Anime 'x', and another user who also enjoyed watching anime 'x' liked anime 'y' then anime 'y' may be a suitable recommendation for the user.	Unfortunately due to time and resource constraints I was unable to implement a hybrid recommender system whereby I use content based and collaborative filtering to generate recommendations. The downside to not being able to implement a hybrid recommender system is that the generated recommendations may not be as accurate as they could potentially be by using a hybrid recommender system. However, if later on we decided to implement a collaborative filtering system in our app it will be really easy to do so as I have coded the current content based recommender system min a separate Python script and have also saved the formatted anime data in a permanent storage file which means that integrating another recommender system in our app will be as easy as attaching blocks of Lego to each other.
Descriptions for each anime	I also did not have the time to web scrape descriptions for each anime however I did manage to add web links to pages that do have descriptions about each anime, the only downside to this is that the user would require an internet connection.
MyAnimeList account linking	In addition, I was unable to create a function which would add the anime from a user's favourites list from this app to their myanimelist.net account due to a lack of knowledge in regards to the API provided by myanimelist.net. This feature could have been solved if I had gotten in touch with a developer from the myanimelist.net website, however I did not choose to do this as it is not essential to the solution of my problem and I did not want to impose on developers who are already busy maintain their own system.

Comment [MR128]: Features that could be improved

Features that could have been improved

Feature	Justification
Home screen anime character image	The current app only has one image of the character Ken from the anime Tokyo Ghoul on the front of the home screen; although having the same home page will bring familiarity to our app there could possibly be future development to add more characters so every time the user launches the app the character on the home screen changes. Even though this is a minor improvement it does help in mixing things up and keeping our app fresh.
Recommender system	The current recommender system could be improved by combining it with a collaborative filtering recommender system. The addition of a collaborative filtering system will make the generated recommendations more accurate and précises as they will include the opinions of other users who have already seen these animes that we are recommending.
Anime Pictionary	The current Pictionary quiz has 60 images (questions) however after a while users may already have answered all of them or the current questions may be out of date and so if we could implement a feature whereby the app could be updated to include more recent animes in the Pictionary quiz. This will make users use our app for a longer amount of time rather than a one off use.
Anime dataset	The current dataset used by the recommender system was populated by web scraping however the packaged app will have that fixed dataset, which means it won't include new anime releases that come out after I packaged the app. The can be solved by implementing an update whereby a new dataset is populated using the same web scraping script I used to populate the first one but this time it will be an up to date data set. Having an up to date anime data set will improve the accuracy of our recommender system and also we will be in touch with new anime fans.

Comment [MR129]: System maintenance

MAINTENANCE

Feature	Justification
Adding/Changing images	If in future development any images need to be changed or added this can be done fairly easily. For instance, in my code I have annotated where I have declared the image for each widget with appropriate names such as 'register_button_img'. This means that all a developer has to do to change the image for the register button is locate the variable 'register_button_img' and change the path of that variable to the path of the new image file.
Updating the anime data set	To update the anime data set a developer would have to re-run the web scraper script which I used to create the initial dataset and then replace the generated dataset with the current one in the programs directory. This is a bit more confusing process than changing images so further documentation may be needed.
Changing the colour scheme/font	Changing the colour scheme or font of the app is really easy as in the code I have annotated where I declare these settings for each screen. So In the case of changing the background colour of the home screen, a developer would have to locate the Home class and in the parameters of the Frame constructor within the class change bg = '' to whatever colour they wish.
Recommender system	Integrating a collaborative system is simple as I have separated the recommender system and the data set into a different script to the main script. However, the problem arises of actually crating the collaborative filtering system as it would require knowledge of machine learning algorithms.

Josh Santillan, Lead client

CODE LISTINGS

Comment [MR130]: complete annotated code listings

Accounts.py

```
import sqlite3 # Imports the sqlite3 library which is an embedded relational database management system

with sqlite3.connect("Accounts.db") as db: # Connects to the database, 'Accounts.db', if it doesn't exist then it is created
    cursor = db.cursor() # Creates a cursor object which is used to traverse the database

## Executes a sql query on the database which creates a table called 'user' with the fields: 'userID','username','firstname','surname' and 'password',
## if they do not already exist.
cursor.execute( '''
CREATE TABLE IF NOT EXISTS user(
userID INTEGER PRIMARY KEY,
username VARCHAR(20) NOT NULL,
firstname VARCHAR(20) NOT NULL,
surname VARCHAR(20) NOT NULL,
password VARCHAR(20) NOT NULL,
highscore INTEGER,
favourites TEXT);
''')

db.commit() # Makes changes made to the database permanent

cursor.execute("SELECT * FROM user") # Selects all the data from the table 'user'

db.close() # Closes the connection to 'Accounts.db' so that it can be accesed by other processes; avoiding locking of the database.
```

Content_based_sytem.py

```
from sklearn.preprocessing import MaxAbsScaler # Imports sklearn's MaxAbsScaler module which Scales each feature by its maximum absolute value.
from sklearn.neighbors import NearestNeighbors # Imports sklearn's NearestNeighbors module which provides functionality for unsupervised and supervised neighbors-based learning methods
import numpy as np # Imports the Numpy library which adds support for large, multi-dimensional arrays and matrices
import pandas as pd # Imports the pandas library which is used for data manipulation and analysis
import re #  Imports the re library which is used to read/write and match regular expressions



## STEP 1 - Formatting the csv file which contains all the anime data  in a formaat that the K-Nearest-Neighbour (KNN) algorithm can understand ##

# Opening the anime.csv file and storing it in a pandas dataframe
anime = pd.read_csv("anime.csv")

# Returns the first row  of the anime dataframe, i.e the headers
anime.head()

# Manually setting the number of episodes for certain animes which do not have a value for number of episodes
anime.loc[(anime["genre"]=="Hentai") & (anime["episodes"]=="Unknown"),"episodes"] = "1"
anime.loc[(anime["type"]=="OVA") & (anime["episodes"]=="Unknown"),"episodes"] = "1"
anime.loc[(anime["type"] == "Movie") & (anime["episodes"] == "Unknown")] = "1"
known_animes = {"Naruto Shippuuden":500, "One Piece":784,"Detective Conan":854, "Dragon Ball Super":86,
                "Crayon Shin chan":942, "Yu Gi Oh Arc V":148,"Shingeki no Kyojin Season 2":25,
                "Boku no Hero Academia 2nd Season":25,"Little Witch Academia TV":25}
for k,v in known_animes.items():
    anime.loc[anime["name"]==k,"episodes"] = v

# Replacing the remaining animes which do not have a value for number of episodes with the median number of episodes of the entire dataset
anime["episodes"] = anime["episodes"].map(lambda x:np.nan if x=="Unknown" else x)
anime["episodes"].fillna(anime["episodes"].median(),inplace = True)

# Converting all the rating values to floats and replacing all animes which do not have a rating with the median rating of the enitre data set
anime["rating"] = anime["rating"].astype(float)
anime["rating"].fillna(anime["rating"].median(),inplace = True)

# Converting categorical variable of type to an indicator variable
pd.get_dummies(anime[["type"]]).head()

# Converting all the values of members to the float data type
anime["members"] = anime["members"].astype(float)

# Creating a dataframe with all the anime features as the headings
anime_features = pd.concat([anime["genre"].str.get_dummies(sep=","),pd.get_dummies(anime[["type"]]),anime[["rating"]],anime[["members"]],anime["episodes"]],axis=1)
anime["name"] = anime["name"].map(lambda name:re.sub('[^A-Za-z0-9]+', " ", name))
anime_features.head()
anime_features.columns

## STEP 2 - Executing the KNN algorithm on the anime features dataframe

# Instantiating sklearn's MaxAbsScaler which scales each feature by its maximum absolute value
max_abs_scaler = MaxAbsScaler()

# Fit to data and then transforms it
anime_features = max_abs_scaler.fit_transform(anime_features)

# Runs KNN's ball tree algorithm on the anime features dataframe and returns the the 6 most similar animes to every single anime in the data frame
nbrs = NearestNeighbors(n_neighbors=6, algorithm='ball_tree').fit(anime_features)
distances, indices = nbrs.kneighbors(anime_features)



# Function which returns the index of an anime
def get_index_from_name(name):
    return anime[anime["name"]==name].index.tolist()[0]

all_anime_names = list(anime.name.values)

# Function which gets the full anime name from a partial input
def get_id_from_partial_name(partial):
    for name in all_anime_names:
        if partial in name:
            print(name,all_anime_names.index(name))

# Function which return the 5 most  similar animes to a given anime using the results from the KNN algorithm
def get_rec(query=None,id=None):
    if id:
        for id in indices[id][1:]:
            print(anime.ix[id]["name"])
    if query:
        found_id = get_index_from_name(query)
        for id in indices[found_id][1:]:
            print(anime.ix[id]["name"])
```

main.py

```
import tkinter as tk # Imports the Tkinter library which is  used to create the GUI
from tkinter import ttk # Imports the ttk module from Tkinter which is used for Widget styling
import sqlite3 # Imports the sqlite3 library which is an embedded relational database management system
from content_based_system import get_ID, get_possible_searches,get_similar_animes,get_index_from_name # Imports functions from the local external Python script, content_based_system.py
import AccountDB # Imports the local external script, AccountDB.py
import numpy as np # Imports the Numpy library which provides a variety of mathmatical functions and functions for manipulating large Data structures
import string # Imports the String Library which provides many common string operations
import webbrowser  # Imports the Webbrowser Library which provides a high-level interface to allow displaying Web-based documents to users.
from PIL import ImageTk, Image # Imports PIL library used for loading images
import random # Imports Pythons random library used to randomize the questions for the pictioanry screen
import tkinter.messagebox # Imports Tkinter's message box widget
import csv # Imports csv module used to read/write/modify csv files
# The modules below are used to restart the app, logout and also search for files in the programs directory.
import glob
import os
import sys

# Font settings for the GUI
LARGE_FONT= ("Anime Ace Bold", 12)
NORM_FONT =("Anime Ace Bold", 10)
SMALL_FONT =("Anime Ace Bold", 8)



##
''' (STEP 1) CREATING THE INTERFACE CLASS - Instead of making a window for each of our screens, i decided to create just one window
and instead make multiple Tkinter Frame widgets which will represent the screens. I then plan to create an invisible Frame which will
act as a container to hold all the screens. So then if we want to change to a different screen we can do so by raising the screen
that we want to see to the top of the container.

I chose to do this in this way as creating multiple windows can be very inefficient on memory and cause cause delays and freezes in
runtime of the application.Whereas, the use of frames is much more generous on memory consumption.
'''
##
# Declaring the Interface class which inherits from Tkinter's Tk class
class interface(tk.Tk):

    # Declaring interface's constructor method and following convention by stating the default parameters
    def __init__(self, *args, **kwargs):
        # Initialising Tkinter (Creates the window)
        tk.Tk.__init__(self, *args, **kwargs)

        # Setting the title of the window
        tk.Tk.wm_title(self, "Anime Rec")

        # Creating a Tkinter Frame widget which will act as a container
        container = tk.Frame(self)

        # Displaying/putting the container onto the window and making it fill the entire window
        container.pack(side="top", fill="both", expand=True)

        # Giving the container priority over all other widgets
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        # Declaring a dictionary which will contain all the frames that we will make
        self.frames = {}

        # For loop which puts each frame in to the container and appends each container item to the self.frames dictionary
        for F in {home, register, menu, profile, get_rec, browse, pictionary}: # These are classes that are yet to be created
            frame = F(container, self)
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky="nsew")

        # Sets the Home screen to the top of the container by defualt (displays the home screen by default)
        self.show_frame(home)

        # Creating and displaying a navigation bar which will allow the user to logout and exit the app
        menubar = tk.Menu(container)
        filemenu = tk.Menu(menubar, tearoff = 0)
        filemenu.add_command(label="Log out",command=self.logout )
        filemenu.add_separator()
        filemenu.add_command(label="Menu",command=lambda: self.show_frame(menu))
        menubar.add_cascade(label="File", menu=filemenu)
        tk.Tk.config(self, menu=menubar)

        # This function will be called whenever we want to change screens, it does this by taking the screen we want to change to
        # as a parameter and using Tkinters tkraise() we can raise that screen to the top of the container

    def show_frame(self, cont):
        frame = self.frames[cont]
        frame.tkraise()

    def logout(self):
        os.execl(sys.executable, sys.executable, *sys.argv)
```

```
class home(tk.Frame): # Class Home inherits from Tkinter's Frame class (Creating a frame for the Home screen)
    def __init__(self, parent, controller): # Constructor method for home class
        tk.Frame.__init__(self, parent, bg = 'grey11') # Initialises the Frame class nad sets background colour to follow our colour scheme

        self.controller = controller # Assigns the controller variable to a variable of the class home

        ## CREATING AND DISPLAYING THE HOME TITLE ##
        self.title_img = tk.PhotoImage(file="Home_Title.png") # Loads image
            # Creates Label widget containing the image
        self.title_label = tk.Label(self, width=392, image=self.title_img, relief='flat', background =  'grey11')
        self.title_label.place(relx=0.21, rely=0.02, height=108, width=392) # Places widget onto window


        self.ken = tk.PhotoImage(file="Ken_Masked.png")
        self.ken_label = tk.Label(self, width=392, image=self.ken, relief='flat', background =  'grey11')
        self.ken_label.place(relx=0, rely=0.2, height=450, width=180)

        ## CREATING AND DISPLAYING THE LOGIN BUTTON ##
        login_img = tk.PhotoImage(file = "Login_button.png") # Loads image
            # Creates Button widget containing the image
        self.login_butt = tk.Button(self,image=login_img,command = self.login, borderwidth=0,background ='grey11', activebackground='grey11')
        self.login_butt.image=login_img # keeps a reference of the widgets image
        self.login_butt.place(relx=0.4, rely=0.46, height=35, width=120) # Places widget onto window

        ## CREATING AND DISPLAYING THE REGISTER BUTTON ##
        reg_img = tk.PhotoImage(file = "Reg_Button.png")
        self.register_butt = tk.Button(self, image=reg_img,command = lambda:controller.show_frame(register), borderwidth=0,background ='grey11', activebackground='grey11')
        self.register_butt.image = reg_img
        self.register_butt.place(relx=0.31, rely=0.74, height=45, width=250)

        ## CREATING AND DISPLAYING USERNAME ENTRY WIDGET ##
        self.username_entry = ttk.Entry(self) # Creates usenrmae entry widget
        self.username_entry.insert(0, 'Enter Username') # Sets placeholder text
        # Clears placeholder text when you click in the entry field
        self.username_entry.bind("<FocusIn>", lambda args: self.username_entry.delete('0', 'end'))
        self.username_entry.place(relx=0.35, rely=0.3, relheight=0.06, relwidth=0.34) # Places widget onto window

        ## CREATING AND DISPLAYING PASSWORD ENTRY WIDGET ##
        self.password_entry = ttk.Entry(self)

        self.password_entry.insert(0, 'Enter Password')
        self.password_entry.bind("<FocusIn>", lambda args: self.password_entry.delete('0', 'end'))
        self.password_entry.place(relx=0.35, rely=0.37, relheight=0.06, relwidth=0.34)



    def login(self): # Login function which is executed when the login button is clicked
        username = self.username_entry.get() # Gets the value from the username entry widget
        password = self.password_entry.get() # Gets the value from the password entry widget

        with sqlite3.connect("Accounts.db") as  db:# Connects to the Accounts database and checks if the login credentials are valid
            cursor = db.cursor() # Creates a cursor object used to traverse the database
        find_user = ("SELECT * FROM user WHERE username = ? AND password =?") # runs a sql query on the database to check login validity
        cursor.execute(find_user,[(username),(password)])
        results = cursor.fetchall() # Stores the query results in variable called 'results'
        if results:
            for i in results:
                self.name=i[2]
            self.login_success(username) # if credentials are valid then the LoginSuccess() is executed
        else:
            self.login_fail() # if credentials are invalid then the LoginFail() is executed
        db.commit() # Makes changes made to the database permanent
```

```
def login_success(self,username):# Executed if Login is succesful
    global usr_name
    usr_name = username
    self.popup = tk.Toplevel() # Creates a pop-up window
    self.popup.wm_title("Login Succesful")

    # Creates and displays a welcome message on the pop up window
    label = ttk.Label(self.popup,text=("Welcome\n\n     "+self.name), font =NORM_FONT, background='grey11', foreground='limegreen')
    label.pack(side='top', expand=True)

    # Creates and displays a continue button, which is binded to the LSMenu function
    con_img = tk.PhotoImage(file="con_button.png")
    c = tk.Button(self.popup, image= con_img, command=self.ls_menu, borderwidth=0,background ='grey11', activebackground='grey11')
    c.image = con_img
    c.pack(side='bottom', expand=True)

    # Sets the dimensions of the pop-up window
    width_of_window = 200
    height_of_window = 150
    screen_width = self.popup.winfo_screenwidth()
    screen_height = self.popup.winfo_screenheight()
    x_coordinate = (screen_width//2) - (width_of_window//2)
    y_coordinate = (screen_height//2) - (height_of_window//2)
    self.popup.geometry("%dx%d" % (width_of_window, height_of_window))
    self.popup.configure(bg='grey11')
    self.popup.mainloop() # ends the loop for the pop up window

def ls_menu(self):
    self.popup.destroy() # Closes the login sucessful pop up window
    self.username_entry.delete(0,'end') # Clears the username and password entry fields
    self.password_entry.delete(0,'end')
    self.controller.show_frame(menu) # Displays the Menu screen

def login_fail(self): #  Executed if user login fails
    self.popup = tk.Toplevel() # Creates a pop-up window
    self.popup.wm_title("Login Failed") # Sets windows title

    label = ttk.Label(self.popup,text=("Username and password not recognised"), font =NORM_FONT, background='grey11', foreground='limegreen')# Error message (Label widget)
    label.pack(side='top', expand=True)

    try_img = tk.PhotoImage(file="Try_Again_Button.png")
    try_again = tk.Button(self.popup, image= try_img, command=self.lf_home, borderwidth=0,background ='grey11', activebackground='grey11')
    try_again.image = try_img
    try_again.pack(side='bottom', expand=True)

    # Sets the dimensions of the pop-up window
    width_of_window = 450
    height_of_window = 150
    screen_width = self.popup.winfo_screenwidth()
    screen_height = self.popup.winfo_screenheight()
    x_coordinate = (screen_width//2) - (width_of_window//2)
    y_coordinate = (screen_height//2) - (height_of_window//2)
    self.popup.geometry("%dx%d" % (width_of_window, height_of_window))
    self.popup.configure(bg='grey11')
    self.popup.mainloop()# ends the loop for the pop up window

def lf_home(self):
    self.popup.destroy() # Closes the login failed pop up window
    self.controller.show_frame(home) # Displays the Menu screen
```

```
class register(tk.Frame): # Declaring register class which inherits from Tkinters Frame class (Creating Register screen)
    def __init__(self, parent, controller):# Constructor method for register class
        tk.Frame.__init__(self, parent, bg='grey11') # Initialising Tkinters Frame class
        self.controller = controller # Assigning controller as a variable of the register class
        ## CREATING AND DISPLAYING REGISTER TITLE LABEL WIDET ##
        reg_title = tk.Label(self,text='Fill in details below to create an account', background='grey11', foreground='limegreen')
        reg_title.place(relx=0.21, rely=0.07, height=29, width=318)
        ## CREATING AND DISPLAYING THE USERNAME LABEL WIDGET ##
        reg_username = tk.Label(self, text='Username:' , background='grey11', foreground='limegreen')
        reg_username.place(relx=0.13, rely=0.35, height=29, width=87)
        ## CREATING AND DISPLAYING THE FIRSTNAME LABEL WIDGET ##
        reg_firstname = tk.Label(self, text='Firstname:', background='grey11', foreground='limegreen')
        reg_firstname.place(relx=0.13, rely=0.17, height=29, width=85)
        ## CREATING AND DISPLAYING THE SURNAME LABEL WIDGET ##
        reg_surname = tk.Label(self, text='Surname:' , background='grey11', foreground='limegreen')
        reg_surname.place(relx=0.13, rely=0.26, height=29, width=78)
        ## CREATING AND DISPLAYING THE PASSWORD LABEL WIDGET ##
        reg_pass = tk.Label(self,text='Password:' , background='grey11', foreground='limegreen')
        reg_pass.place(relx=0.13, rely=0.45, height=29, width=84)
        ## CREATING AND DISPLAYING THE RE-ENTER PASSWORD LABEL WIDGET ##
        reg_pass2 = tk.Label(self, text='Re-enter Password:' , background='grey11', foreground='limegreen')
        reg_pass2.place(relx=0.13, rely=0.54, height=29, width=156)
        ## CREATING AND DISPLAYING THE FIRSTNAME ENTRY WIDGET ##
        self.reg_firstname_entry = tk.Entry(self,cursor="ibeam")
        self.reg_firstname_entry.place(relx=0.31, rely=0.16, relheight=0.06, relwidth=0.34)
        ## CREATING AND DISPLAYING THE SURNAME ENTRY WIDGET ##
        self.reg_surname_entry = tk.Entry(self,cursor="ibeam")
        self.reg_surname_entry.place(relx=0.31, rely=0.26, relheight=0.06, relwidth=0.34)
        ## CREATING AND DISPLAYING THE USERNAME ENTRY WIDGET ##
        self.reg_username_entry = tk.Entry(self,cursor="ibeam")
        self.reg_username_entry.place(relx=0.31, rely=0.35, relheight=0.06, relwidth=0.34)
        ## CREATING AND DISPLAYING THE PASSWORD ENTRY WIDGET ##
        self.reg_pass_entry = ttk.Entry(self,show="*" ,cursor="ibeam")
        self.reg_pass_entry.place(relx=0.31, rely=0.44, relheight=0.06, relwidth=0.34)
        ## CREATING AND DISPLAYING THE RE-ENTER PASSWORD ENTRY WIDGET ##
        self.reg_pass2_entry = tk.Entry(self,show="*" ,cursor="ibeam")
        self.reg_pass2_entry.place(relx=0.41, rely=0.54, relheight=0.06, relwidth=0.34)
        ## CREATING AND DISPLAYING THE REGISTER BUTTON ##
        reg_img = tk.PhotoImage(file='register_button.png') # Adding image to register button
        self.reg_butt = tk.Button(self, image=reg_img, command = self.create, borderwidth=0,background ='grey11', activebackground='grey11', foreground='limegreen')
        self.reg_butt.image=reg_img # Keeping a reference of the image
        self.reg_butt.place(relx=0.36, rely=0.71, height=35, width=120)
```

```
def create(self):
    # Gets the entered values for all entery widgets
    username=self.reg_username_entry.get()
    firstname=self.reg_firstname_entry.get()
    surname=self.reg_surname_entry.get()
    password=self.reg_pass_entry.get()
    password1=self.reg_pass2_entry.get()
    values = [username, firstname, surname, password, password1]

    # Check to see if a entry widget is empty
    empty = True
    count = 0

    while empty == True and count != 5: # While loop that sets empty to True if a value is blank for an entry widget
        if len(values[count]) > 0:
            empty = False
        else:
            empty = True
            break
        count+=1

    if empty == True: # If one or more entry widgets are blank the appropriate prompt message is displayed
        self.fill_in_all_details()
    else: # Else if no entry widget is blank continue
        with sqlite3.connect("Accounts.db") as  db: # Connects to the Accounts database
            cursor = db.cursor() #  Creates cursor object to traverse through the database
            findUser = ("SELECT * FROM user WHERE username = ?") # SQL query to check if that username already exists
            cursor.execute(findUser,[(username)])
        # If username exists execute userTaken() (pop-up window that prompts that the username is taken/ already exists)
        if cursor.fetchall():
            self.user_taken()
        # Else if username is not taken then gets all the values from the other entry widgets
        else:
            if password != password1: # VALIDATION - checks to see whether the initial password matches the re-entered password
                self.pass_miss_match() # If they dont match then it executes passMissMatch function
                self.controller.show_frame(register)
            ## SQL query to Write users account details to the Accounts database
            insertData = '''INSERT INTO user (username,firstname,surname,password)
            VALUES(?,?,?,?)'''
            cursor.execute(insertData,[(username),(firstname), (surname), (password)])
            db.commit() # Makes changes made to the database permanent
            self.acc_created() # Executes the accCreated() function

def r2_register(self): # Executed if the user wants to try again in creating an account
    self.popup.destroy()
    self.controller.show_frame(register)
def r2_home(self): # Executed if the user wants to return to the home screen
    self.popup.destroy()
    self.controller.show_frame(home)

def user_taken(self): # userTaken function, executed if the username already exists in the database
        self.popup = tk.Toplevel() #Creates user taken pop-up window
        self.popup.wm_title("!USER TAKEN!") # Window title
        ## CREATING AND DISPLAYING THE USER TAKEN MESSAGE LABEL WIDGET ##
        label = tk.Label(self.popup,text="User already exists!", font =NORM_FONT, borderwidth=0,background ='grey11',foreground='limegreen')
        label.pack(side='top', expand=True)
        ## CREATING AND DISPLAYING THE TRY AGAIN BUTTON WIDGET##
        try_img = tk.PhotoImage(file="Try_Again_Button.png")
        try_again = tk.Button(self.popup, image= try_img, command=self.r2_register, borderwidth=0,background ='grey11', activebackground='grey11')
        try_again.image = try_img
        try_again.pack(side='bottom', expand=True)
        ## CREATING AND DISPLAYING THE HOME BUTTON WIDGTET ##              #  Configure widgets to match colour scheme
        button1 = tk.Button(self.popup, text="Home", command=self.r2_home, borderwidth=0,background ='grey11', activebackground='grey11', foreground='limegreen')
        button1.pack(side='bottom', expand=True)

        ## CENTERING THE POP-UP WINDOW ##
        width_of_window = 250
        height_of_window = 150
        screen_width = self.popup.winfo_screenwidth()
        screen_height = self.popup.winfo_screenheight()
        x_coordinate = (screen_width//2) - (width_of_window//2)
        y_coordinate = (screen_height//2) - (height_of_window//2)
        self.popup.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window, x_coordinate, y_coordinate))
        self.popup.configure(bg='grey11') # Set windows background colour to match client requested colour scheme
        self.popup.mainloop() ## User taken pop-up window loop
```

```
def pass_miss_match(self): # passMissMatch function, executed if the two entered passwords by the the user do not match
        self.popup = tk.Toplevel() #  Creates password mismatch pop-up window
        self.popup.wm_title("! PASSWORD ERROR !") # Window title

        ## CREATING AND DISPLAYING THE ERROR MESSAGE LABEL WIDGET ##
        label = tk.Label(self.popup,text="Passwords do not match!", font =NORM_FONT, borderwidth=0,background ='grey11',foreground='limegreen')
        label.pack(side='top', expand=True)
        ## CREATING AND DISPLAYING THE TRY AGIAN BUTTON WIDGET ##
        try_img = tk.PhotoImage(file="Try_Again_Button.png")
        try_again = tk.Button(self.popup, image= try_img, command=self.r2_register, borderwidth=0,background ='grey11', activebackground='grey11')
        try_again.image = try_img
        try_again.pack(side='bottom', expand=True)
        ## CREATING AND DISPLAYING THE HOME BUTTON WIDGET ##
        button1 = tk.Button(self.popup, text="Home", command=self.r2_home, borderwidth=0,background ='grey11', activebackground='grey11', foreground='limegreen')
        button1.pack(side='bottom', expand=True)
        ##  CENTERING THE POP-UP WINDOW ##
        width_of_window = 400
        height_of_window = 150
        screen_width = self.popup.winfo_screenwidth()
        screen_height = self.popup.winfo_screenheight()
        x_coordinate = (screen_width//2) - (width_of_window//2)
        y_coordinate = (screen_height//2) - (height_of_window//2)
        self.popup.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window, x_coordinate, y_coordinate))
        self.popup.configure(bg='grey11')
        self.popup.mainloop() # Password mismatch pop-up window main loop

def acc_created(self): # accCreated function, executed in the case when the account is succesfully created
        self.popup = tk.Toplevel() # Creates the account created pop up window
        self.popup.wm_title("!!") # Setting the windows title

        ## CREATING AND DISPLAYING THE SUCCESS MESSAGE, LABEL WIDGET ##
        label = tk.Label(self.popup,text="Account succesfully created", font =NORM_FONT, borderwidth=0,background ='grey11',foreground='limegreen')
        label.pack(side='top', expand=True)
        ## CREATING AND DISPLAYING THE SIGN IN BUTTON WIDGET ##


        # Creates and displays a continue button, which is binded to the LSMenu function
        sign_img = tk.PhotoImage(file="sign_in_button.png")
        b = tk.Button(self.popup, image= sign_img, command=self.r2_home, borderwidth=0,background ='grey11', activebackground='grey11')
        b.image = sign_img
        b.pack(side='bottom', expand=True)

        ## CENTERING THE POP-UP WINDOW ##
        width_of_window = 400
        height_of_window = 150
        screen_width = self.popup.winfo_screenwidth()
        screen_height = self.popup.winfo_screenheight()
        x_coordinate = (screen_width//2) - (width_of_window//2)
        y_coordinate = (screen_height//2) - (height_of_window//2)
        self.popup.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window, x_coordinate, y_coordinate))
        self.popup.configure(bg='grey11')
        self.popup.mainloop() # Account created pop up window main loop

def fill_in_all_details(self): # function that is executed in the case of one or more mepty entry fields
        self.popup = tk.Toplevel() # Creates the empty entry  pop up window
        self.popup.wm_title("!!") # Setting the windows title

        ## CREATING AND DISPLAYING THE ERROR MESSAGE, LABEL WIDGET ##
        label = tk.Label(self.popup,text="Please fill in all the fields", font =NORM_FONT, borderwidth=0,background ='grey11',foreground='limegreen')
        label.pack(side='top', expand=True)
        ## CREATING AND DISPLAYING THE SIGN IN BUTTON WIDGET ##
        try_img = tk.PhotoImage(file="Try_Again_Button.png")
        try_again = tk.Button(self.popup, image= try_img, command=self.r2_register, borderwidth=0,background ='grey11', activebackground='grey11', foreground='limegreen')
        try_again.image = try_img
        try_again.pack(side='bottom', expand=True)
        ## CENTERING THE POP-UP WINDOW ##
        width_of_window = 400
        height_of_window = 150
        screen_width = self.popup.winfo_screenwidth()
        screen_height = self.popup.winfo_screenheight()
        x_coordinate = (screen_width//2) - (width_of_window//2)
        y_coordinate = (screen_height//2) - (height_of_window//2)
        self.popup.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window, x_coordinate, y_coordinate))
        self.popup.configure(bg='grey11')
        self.popup.mainloop() # fill_in_all_deatils pop up window main loop
```

```
# Declaring the menu class whioch inherits from Tkinter's Frame class
class menu(tk.Frame):

    # Declaring menu's constructor method which is called when the class in instantiated
    def __init__(self, parent, controller):

        # Initialises Tkinter's Frame class
        tk.Frame.__init__(self, parent, bg='grey11') # Setting the background of the frame to 'grey11'

        # Storking controller as a varioable of the menu class
        self.controller = controller

        # Loading the image for the Get rec button
        get_rec_img = tk.PhotoImage(file="get_rec_button.png")
        # Shrinks the image
        get_rec_img = get_rec_img.subsample(3)
        # Crerating the get_rec_button with get_rec_img as its image and binding it with the function to display the get rec screen
        get_rec_button = tk.Button(self,image=get_rec_img, borderwidth = 0,command=lambda: controller.show_frame(get_rec), background = "grey11")
        # Keeping a reference of get_rec_button's image
        get_rec_button.image=get_rec_img
        # Displaying the Button widget on the window
        get_rec_button.grid(row=0,sticky="nsew")


        # Loading the image for the browse button
        browse_button_img = tk.PhotoImage(file="browse_button.png")
        # Shrinks the image
        browse_button_img = browse_button_img.subsample(3)
        # Crerating the browse_button with browse_button_img as its image and binding it with the function to display the browse screen
        browse_button = tk.Button(self, image= browse_button_img, borderwidth = 0,command=lambda: controller.show_frame(browse), background = "grey11")
        # Keeping a reference of browse_button's image
        browse_button.image=browse_button_img
        # Displaying the Button widget on the window
        browse_button.grid(row=0,column=1, sticky="nsew")

        # Loading the image for the profile button
        profile_button_img = tk.PhotoImage(file="profile_button.png")
        # Shrinks the image
        profile_button_img = profile_button_img.subsample(3)
        # Crerating the profile_button with profile_button_img as its image and binding it with the function to display the profile screen
        profile_button = tk.Button(self,image=profile_button_img, borderwidth = 0,command=lambda: controller.show_frame(profile), background = "grey11")
        # Keeping a reference of profile_button's image
        profile_button.image=profile_button_img
        # Displaying the Button widget on the window
        profile_button.grid(row=1,sticky="nsew")

        # Loading the image for the pictionary button
        pictionary_button_img = tk.PhotoImage(file="pictionary_button.png")
        # Shrinks the image
        pictionary_button_img = pictionary_button_img.subsample(3)
        # Crerating the pictionary_button with pictionary_button_img as its image and binding it with the function to display the pictionary screen
        pictionary_button = tk.Button(self,image=pictionary_button_img, borderwidth = 0,command=lambda: controller.show_frame(pictionary), background = "grey11")
        # Keeping a reference of pictionary_button's image
        pictionary_button.image=pictionary_button_img
        # Displaying the Button widget on the window
        pictionary_button.grid(row=1,column=1, sticky="nsew")

        # Configuring the priority of the widgets
        self.columnconfigure(0, weight=1)
        self.columnconfigure(1, weight=1)
        self.rowconfigure(0, weight=1)
        self.rowconfigure(1, weight=1)
```

```
# Declaring the class for the Get Recommendation screen as get_rec, which inherits from Tkinter's Frame class.
class get_rec(tk.Frame):

    # Declaring the constructor method for the get_rec class, which runs w henb the class is instantiated
    def __init__(self, parent, controller):
        # Intialising Tkinter's Frame  class
        tk.Frame.__init__(self, parent, bg='grey11') #  Setyting backgroung to 'grey11'
        # Storing controller as a variable of the get_rec class
        self.controller = controller

        # Creating and displaying the label for the entry widget  which will be used so that the user can enter which anime they wish to find recommendations for
        self.getRec_label = tk.Label(self, text='Find recommendation for:', width=210, relief='flat', bg = 'grey11', fg='lime green', borderwidth=2,highlightbackground="lime green" )
        self.getRec_label.place(relx=0.00, rely=0.07, height=35, width=200)

        # Creating and displaying the Entry widget which will allow the user to input an anime
        self.search_entry = ttk.Entry(self)
        self.search_entry.place(relx=0.28, rely=0.08, relheight=0.05, relwidth=0.23)

        # Creating and displaying a button, which when clicked will output the search reesults of the user's input
        self.search_butt = tk.Button(self, text='Search', command= self.populate, bg = 'grey11', fg='lime green', borderwidth=2,highlightbackground="lime green" )
        self.search_butt.place(relx=0.54, rely=0.08, height=30, width=120)



        # Creating and  displaying a frame to contain the list box
        frm = tk.Frame(self)
        frm.place(relx=0.03, rely=0.16, relheight=0.73, relwidth=0.7)

        # Creating and displaying a scroll bar  used to scroll up and down  the list box
        scrollbar = tk.Scrollbar(frm, orient="vertical", highlightcolor="#d9d9d9", highlightbackground="#d9d9d9")
        scrollbar.pack(side='right', fill='y')


        # Creating a displaying a List box widget within the same frame as the scroll bar
        self.listNodes = tk.Listbox(frm, width=100, yscrollcommand=scrollbar.set, font=("Helvetica", 12))
        # When a node from the List box is double clicked then the OnDouble function is executed
        self.listNodes.bind("<Double-Button>" , self.OnDouble)
        self.listNodes.pack(expand=True, fill='y')
        scrollbar.config(command=self.listNodes.yview)

        # Creating and displaying a label to give the user some instructions as to how to use this feature.
        self.getRec_note = tk.Label(self, text='Note - Search for anime then Double click on the anime you wish to select')
        self.getRec_note.place(relx=0.02, rely=0.9, relheight=0.06, relwidth=0.67)

    # Function which displays pop-up window which displays the generated recommendations to the user
    def OnDouble(self, event):
        # Creating a pop up window
        self.popup = tk.Tk()
        self.popup.wm_title("Anime Recommendations!")

        # Get the anime that was selected from the List Box widget
        widget = event.widget
        selection=widget.curselection()
        value = widget.get(selection[0])

        # Storing the results of the recommender system in the list anime
        anime = get_rec(value)
        t_text = "Recommendations for "+value

        # Creating and displaying a label widget which tells the user th anime for which the recommendations are for
        title = ttk.Label(self.popup, text = t_text )
        title.grid(row=0, sticky = "ne")


        # Displays each recommendation with their respective link for more information
        a = ttk.Label(self.popup,text=anime[0])
        a.grid(row=1, column=0)
        b = ttk.Button(self.popup,text='Click for more info', command=lambda:self.openLink(anime[0]))
        b.grid(row=1, column=1)

        a1 = ttk.Label(self.popup,text=anime[1])
        a1.grid(row=2, column=0)
        b = ttk.Button(self.popup,text='Click for more info',command=lambda:self.openLink(anime[1]))
        b.grid(row=2, column=1)

        a2 = ttk.Label(self.popup,text=anime[2])
        a2.grid(row=3, column=0)
        b = ttk.Button(self.popup,text='Click for more info',command=lambda:self.openLink(anime[2]))
        b.grid(row=3, column=1)

        a3 = ttk.Label(self.popup,text=anime[3])
        a3.grid(row=4, column=0)
        b = ttk.Button(self.popup,text='Click for more info',command=lambda:self.openLink(anime[3]))
        b.grid(row=4, column=1)

        a4 = ttk.Label(self.popup,text=anime[4])
        a4.grid(row=5, column=0)
        b = ttk.Button(self.popup,text='Click for more info',command=lambda:self.openLink(anime[4]))
        b.grid(row=5, column=1)

        # Pop up window main loop
        self.popup.mainloop()

    # function that is executed when the search button is clicked; which populates the list box with the search results
    def populate(self):
        self.listNodes.delete(0, 'end')
        partial=string.capwords(self.search_entry.get())
        possibleSearch= get_possible_searches(partial)
        for i in range(0, len(possibleSearch)):
            self.listNodes.insert(i, possibleSearch[i])

    # function that open the myanimelist.net website for the given anime
    def openLink(self, anime):
        try:
            a_id=get_ID(anime)
            url = "https://myanimelist.net/anime/"+a_id[0][0]
            webbrowser.open(url)
        except IndexError:
            print("Link broken")
```

```
# Declaring the browse class which inherits from Tkinter's Frame class (Creating the browse screen)
class browse(tk.Frame):
    # Declaring browse class' constructor method
    def __init__(self, parent, controller):

        # Initialising Tkinter's Frame class
        tk.Frame.__init__(self, parent, bg='greyll')



        # Creating and displaying a genres label for the checkbuttons
        GenreLabel = tk.Label(self, text="Genres:", fg='limegreen', bg='greyll')
        GenreLabel.grid(row=1, sticky='w')

        # List of all the genres
        self.GENRES = ["Action","Adventure","Cars","Comedy","Dementia",
                  "Demons","Drama","Ecchi","Fantasy","Game","Harem",
                 "Hentai","Historical","Horror","Josei","Kids",
                 "Magic","Martial Arts","Mecha","Military","Music",
                 "Mystery","Parody","Police","Psychological",
                 "Romance","Samurai","School","Sci-Fi","Seinen",
                 "Shoujo","Shoujo Ai","Shounen","Shounen Ai",
                 "Slice of Life","Space","Sports","Super Power",
                 "Supernatural","Thriller","Vampire","Yaoi","Yuri"]

        # Creating and displaying each genre ans a checkbutton, most efficient way in Tkinter
        self.vars = []
        self.chk = []
        for i in range(0,5):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var, fg='limegreen', bg='greyll'))
            self.chk[-1].grid(row=2, column=(i+1),sticky="w")
            self.vars.append(self.var)
        for i in range(5,10):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var, fg='limegreen', bg='greyll'))
            self.chk[-1].grid(row=(3),column=((i+1)-5),sticky="w")
            self.vars.append(self.var)
        for i in range(10,15):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var, fg='limegreen', bg='greyll'))
            self.chk[-1].grid(row=(4),column=((i+1)-10),sticky="w")
            self.vars.append(self.var)
        for i in range(15,20):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var, fg='limegreen', bg='greyll'))
            self.chk[-1].grid(row=(5),column=((i+1)-15),sticky="w")
            self.vars.append(self.var)
        for i in range(20,25):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var, fg='limegreen', bg='greyll'))
            self.chk[-1].grid(row=(6),column=((i+1)-20),sticky="w")
            self.vars.append(self.var)
        for i in range(25,30):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var, fg='limegreen', bg='greyll'))
            self.chk[-1].grid(row=(7),column=((i+1)-25),sticky="w")
            self.vars.append(self.var)
        for i in range(30,35):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var, fg='limegreen', bg='greyll'))
            self.chk[-1].grid(row=(8),column=((i+1)-30),sticky="w")
            self.vars.append(self.var)
        for i in range(35,40):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var, fg='limegreen', bg='greyll'))
            self.chk[-1].grid(row=(9),column=((i+1)-35),sticky="w")
            self.vars.append(self.var)
        for i in range(40,43):
            self.var = tk.IntVar()
            self.chk.append(tk.Checkbutton(self, text=self.GENRES[i], variable=self.var, fg='limegreen', bg='greyll'))
            self.chk[-1].grid(row=(10),column=((i+1)-40),sticky="w")
            self.vars.append(self.var)


        # Creating and displaying the Search button which takes all the inputs from the user and querys the database with the given criteria
        Search= tk.Button(self, text="Search", command =self.query, fg='limegreen', bg='greyll')
        Search.grid(columnspan=10)
```

```
# Function for the Search buttons, which querys the database with the given criteria inputted by the user
def query(self):

    # Creates a pop up window
    self.popup = tk.Tk()
    self.popup.configure(bg='grey11')

    # Creates and display a title label for the pop up window
    self.title = tk.Label(self.popup, text=("Search Results:\nDouble click for more info"), width=210, relief='flat',bg='grey11',fg='limegreen')
    self.title.place(relx=0.04, rely=0.07, height=35, width=210)

    # Creates and displays a frame used to hold the listbox widget and scroll bar widget
    frm = tk.Frame(self.popup)
    frm.place(relx=0.03, rely=0.16, relheight=0.73, relwidth=0.7)

    # Creates and adds a scroll bar to the list box to allow the user to scroll up and down the search results
    scrollbar2 = tk.Scrollbar(frm, orient="vertical", highlightcolor="#d9d9d9", highlightbackground="#d9d9d9")
    scrollbar2.pack(side='right', fill='y')

    # Creating and displaying the Listbox widget
    self.listNodes2 = tk.Listbox(frm, width=100, yscrollcommand=scrollbar2.set, font=("Helvetica", 12))
    self.listNodes2.bind("<Double-Button-1>", self.OnDouble2)
    self.listNodes2.pack(expand=True, fill='y')
    scrollbar2.config(command=self.listNodes2.yview)

    # Button which returns the user to the browse screen and closes the search results pop up window
    button1 = tk.Button(self.popup, text="Return", command = self.return1,bg='grey11',fg='limegreen')
    button1.pack(side='bottom')

    # Stores the whatever values are checked from the check buttons into an array
    states=[]
    for x in self.vars:
        states.append(x.get())
    genreIndex = [i for i, s in enumerate(states) if s == 1]
    self.genreResults=[]
    for x in genreIndex:
        self.genreResults.append(self.GENRES[x])


    # Connects to the anime database  and runs an sql query with the given criteria
    with sqlite3.connect("animeData.db") as  db: #anime database
        c = db.cursor()
        #self.genreResults is the list containing the genres i want to query
        user_input = self.genreResults  # for example: ['Action','Drama']
        convert_to_like_conditions = ' and '.join(list(map(lambda item : "genre like '%{0}%'".format(item), user_input)))
        c.execute("select * from t where %s" % convert_to_like_conditions )
        self.results = c.fetchall()
        db.commit()


    # Displays all the search results from the query in the listbox
    for i in range(0,len(self.results)):
        self.listNodes2.insert(i,self.results[i][1])

    # Pop up window dimensions configuration
    width_of_window = 700
    height_of_window = 538
    screen_width = self.popup.winfo_screenwidth()
    screen_height = self.popup.winfo_screenheight()
    x_coordinate = (screen_width//2) - (width_of_window//2)
    y_coordinate = (screen_height//2) - (height_of_window//2)
    self.popup.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window, x_coordinate, y_coordinate))
    self.popup.mainloop()

# Function which is executed when a search result in the listbox is double clicked
def OnDouble2(self, event):
    # Gets the value of the anime of the listbox node that was double clicked
    widget = event.widget
    selection=widget.curselection()
    value = widget.get(selection)
    i = index = [idx for idx, s in enumerate(self.results) if value in s][0]

    # Creates a myanimelist.net website for that anime and opens it up in the user's browser
    url = 'https://myanimelist.net/anime/'+self.results[i][0]+self.results[i][1]
    webbrowser.open(url)

# Function which is executed when the return button is clicked
def return1(self):
    self.popup.destroy()
```

```
# Declaring the class for the Profile screen, which inherits from Tkinter's Frame class
class profile(tk.Frame):

    # Declaring the constructor method for the profile class, which runs when the class is instantiated
    def __init__(self, parent, controller):
        # Initialising Tkinter's Frame class
        tk.Frame.__init__(self, parent, bg='grey11')

        # Storing controller as a variable of the profile class
        self.controller = controller

        # Creating and displaying the label for the entry widget which will be used so that the user can enter which anime they wish to add to their favourites
        self.getRec_label = tk.Label(self, text='Select an anime\nyou wish to add\nto your favourites:', width=210, relief='flat', bg = 'grey11', fg='lime green', borderwidth=2,highlightbackground="lime green" )
        self.getRec_label.place(relx=0.00, rely=0.07, height=38, width=200)

        # Creating and displaying the Entry widget which will allow the user to input the name of an anime
        self.search_entry = ttk.Entry(self)
        self.search_entry.place(relx=0.28, rely=0.08, relheight=0.05, relwidth=0.23)

        # Creating and displaying a button, which when clicked will output the search results of the user's input
        self.search_butt = tk.Button(self, text='Search', command= self.populate, bg = 'grey11', fg='lime green', borderwidth=2,highlightbackground="lime green" )
        self.search_butt.place(relx=0.54, rely=0.08, height=30, width=120)

        # Creating and displaying a button, which when clicked displays the user's current favourite anime list
        self.view_butt = tk.Button(self, text='view favourites', command = self.view_favourites, bg = 'grey11', fg='lime green', borderwidth=2,highlightbackground="lime green")
        self.view_butt.pack(side='top')



        # Creating and displaying a frame to contain the list box and scrollbar widget
        frm = tk.Frame(self)
        frm.place(relx=0.03, rely=0.16, relheight=0.73, relwidth=0.7)

        # Creating and displaying a scroll bar on the right of the frame which is used to scroll up and down the list box
        scrollbar = tk.Scrollbar(frm, orient="vertical", highlightcolor="#d9d9d9", highlightbackground="#d9d9d9")
        scrollbar.pack(side='right', fill='y')

        # Creating and displaying a Listbox widget within the same frame as the scrollbar
        self.listNodes = tk.Listbox(frm, width=100, yscrollcommand=scrollbar.set, font=("Helvetica", 12))
        self.listNodes.bind("<Double-Button-1>", self.OnDouble)
        self.listNodes.pack(expand=True, fill='y')
        scrollbar.config(command=self.listNodes.yview)

        # Creating and displaying a label to give the user some instructions as to how to use the list box.
        self.getRec_note = tk.Label(self, text='Note - Search for anime then Double click on the anime you wish to select', bg = 'grey11', fg='lime green')
        self.getRec_note.place(relx=0.02, rely=0.9, relheight=0.06, relwidth=0.67)

    # Function which populates the List box with the search results from the user's input.
    def populate(self):
        self.listNodes.delete(0, 'end')
        partial=string.capwords(self.search_entry.get())
        possibleSearch= get_possible_searches(partial)
        for i in range(0, len(possibleSearch)):
            self.listNodes.insert(i, possibleSearch[i])
    # Function which adds the selected anime to the user favourites
    def OnDouble(self, event):
        # Gets the value of the node that was selected from the list box and stores it in the varibale, value
        widget = event.widget
        selection=widget.curselection()
        value = widget.get(selection)

        # Gets the value from does file exist function
        # if th efile exists then we open the file in append mode and append the selected anime to the users favourite list
        if self.does_file_exist(usr_name+'.txt'):
            with open(usr_name+'.txt', "a") as f:
                f.write(value+'\t')
        # Else if the file does not exist then we create it using the write mode and write the selected value to the file
        else:
            with open(usr_name+'.txt', "w") as f:
                f.write(value+'\t')


        # Once the selected anime has been added to the users favourites list we give them the option to view their cirrent favourites list
        user_choice = tk.messagebox.askquestion("ADDED TO FAVOURITES",value+" Has been added to your favourites list\n Would you like to view your current favourites?")

        # If they select yes  then we execyte the view_favourites function which displays the current favourites
        if user_choice == 'yes':
            self.view_favourites()
        # If they select no then we just close the message box and return to the profile screen
        else:
            pass



    # Function to check whether a file already exist using the path.exists function from the os library
    def does_file_exist(self,file_name):
        return os.path.exists(file_name)

    # Function that displays the users curerent favourite animes on a pop up window
    def view_favourites(self):

        # UPDATE - added steps that will check if the favourites list text file exists
        # If it doesnt then we prompt the user that their list is empty
        if self.does_file_exist(usr_name+'.txt') == False:
            # Giving them an option to add an anime to their favourites list  or not
            user_choice = tk.messagebox.askquestion("ALERT","Your favourites list is currently empty, Would you like to add an anime to it?")

            # terminates the function regardless of the user choice thus preventing the error yet still keeping the program functional
            if user_choice == 'yes':
                return
            else:
                return


        # retrieves the users current favourite list by reading it from the text file
        with open(usr_name+'.txt', "r") as f:
            for lines in f:
                    favourites_list = lines
        # Creates the pop up window
        pop_up = tk.Toplevel()
        pop_up.title('Your favourites list')



        favs = tk.StringVar()
        favs.set(favourites_list)
        # Creating and displaying the label which will display the favourites list on screen
        label = tk.Label(pop_up, text = favs.get())
        label.pack()

        # Creating and displaying a button which will close the screen and return to the profile screen when the user is done
        close = tk.Button(pop_up, text= "Close", command = pop_up.destroy)
        close.pack()

        # Pop up windows mainloop
        pop_up.mainloop()
```

```
# Opens file in read mode
f = open('questions.txt', 'r')

# Temporary Python array
data = []

# For each line in the text file it splits the line based on tabs
for row_num, line in enumerate(f):
    values = line.strip().split('\t')
    # first line is the header (image, answer)
    if row_num == 0:
         # Stores the haeader in an array
         header = values
    else:
        data.append([v for v in values])

# Converts Python array to a numpy array
question_data = np.array(data)


# Close the file
f.close()
## TESTING PURPOSES ##
    #print(question_data)
    #print(header)

## RANDOMIZE THE QUESTIONS BEFORE EACH RUN ##
np.random.shuffle(question_data)
    #print(question_data)# TESTING
```

```
# Declares the Pictionary class which inherits from Tkinters frame class (Creates the Pictionary screen)
class pictionary(tk.Frame):
    # Constructor method for the Pictionary class
    def __init__(self, parent, controller):

        # Initalises Tkinter's Frame class
        tk.Frame.__init__(self, parent, bg='grey11')

        # Stores controller as a variable of the pictionary class
        self.controller =controller
        self.parent = parent

        # Setting count to 0 at the start (indicating we are at the first question)
        self.count =0

        # Number of questions set to 10 for testing purposes
        self.max_questions = 10

        # Executes the display_question function (declared below) with the question image, answer and choices for the first question from the text file
        self.display_question(question_data[self.count][0],question_data[self.count][1],question_data[self.count][2])


    # Function which display the question onto the window. Takes a questions image, answer and choices as parameters.
    def display_question(self, image, answer, choices):

        # Loads the image in a way that Python can read it
        img = Image.open(image+'.jpg')
        # Stores the loaded image into the variable, photo
        photo = ImageTk.PhotoImage(img)

        # print('answer:',answer) ## Testing purposes to check the correct answer

        # Seperates the choices for the question into single elements so they can be displayed on the window
        choices = choices.replace('"','')
        choices = [x.strip() for x in choices.split(',')]
        # Randomizes the choices
        random.shuffle(choices)

        # Creating and displaying the label that contains the image that is to be guessed
        self.anime_image = tk.Label(self, image=photo,relief='flat')
        self.anime_image.image = photo
        self.anime_image.pack()


        # Creating and displaying the label which displays the current level to the user i.e their score
        level = 'level',self.count+1
        self.level = tk.Label(self, text=level,borderwidth=0, relief='flat', bg='limegreen', fg='black')
        self.level.place(relx=0.4, rely=0.04, height=29, width=120)

        # Creating and displaying the Buttons which represent the 4 available choices to the user
        # Note the command for each button passes the value of the choice to the check answer function so that the users choice can be compared with the-
        # correct answer
        self.answer_a = tk.Button(self,text=choices[0],command = lambda:self.check(choices[0],answer),width=410, bg='limegreen', fg='black')
        self.answer_a.place(relx=0.17, rely=0.58, height=45, width=410)

        self.answer_c = tk.Button(self,text=choices[1],command = lambda:self.check(choices[1],answer),width=410, bg='limegreen', fg='black')
        self.answer_c.place(relx=0.17, rely=0.76, height=45, width=410)

        self.answer_b = tk.Button(self,text=choices[2],command = lambda:self.check(choices[2],answer),width=410, bg='limegreen', fg='black')
        self.answer_b.place(relx=0.17, rely=0.67, height=45, width=410)

        self.answer_d = tk.Button(self,text=choices[3],command = lambda:self.check(choices[3],answer),width=410, bg='limegreen', fg='black')
        self.answer_d.place(relx=0.17, rely=0.86, height=45, width=410)
```

```
    # Function that checks the user's answer. The function takes the user's answer and the correct answer as parameters
    def check(self,user_answer,correct_answer):

        # Checks if this is the last question, if it is then the user has won the game
        if self.count == self.max_questions:
            # Instantiating Tkinter's messagebox wwhich asks the use if they want to play again
            user_answer=tkinter.messagebox.askquestion("You won", "Play again?")
            # if the user wants to play again then we reset the game
            if user_answer == 'yes':
                np.random.shuffle(question_data)
                self.anime_image.forget()
                self.answer_a.forget()
                self.answer_b.forget()
                self.answer_c.forget()
                self.answer_d.forget()
                self.count =0
                self.display_question(question_data[self.count][0],question_data[self.count][1],question_data[self.count][2])
            # If they dont then we reset the game and return to the menu screen
            else:
                self.forget()
                self.controller.show_frame(menu)
            return

        # If the user's answer is correct then we display a correct prompt and display the next question
        if user_answer == correct_answer:
            # Instantiating Tkinter's messagebox with the given  window title and message respectively
            tkinter.messagebox.showinfo("Anime Pictionary","Correct answer")

            # Increment count, i.e. move onto next question
            self.count += 1

            # Removing the current image and choices on screen. I.E. Removing the current question
            self.anime_image.forget()
            self.answer_a.forget()
            self.answer_b.forget()
            self.answer_c.forget()
            self.answer_d.forget()

            # Ececutes the display_question function with the image,correct answer and choices for the next question
            self.display_question(question_data[self.count][0],question_data[self.count][1],question_data[self.count][2])

        # Else if the user gets the questions wrong then its Game over!
        else:
            # Instantiating Tkinter's messagebox asking the user if they want to try agian
            user_answer=tkinter.messagebox.askquestion("Game Over!", "Try again?")

            # If they want to try  again then we reset the game
            if user_answer == 'yes':
                np.random.shuffle(question_data)
                self.anime_image.forget()
                self.answer_a.forget()
                self.answer_b.forget()
                self.answer_c.forget()
                self.answer_d.forget()
                self.count =0
                self.display_question(question_data[self.count][0],question_data[self.count][1],question_data[self.count][2])
            # If they dont then we reset the game and return to the menu screen
            else:
                self.forget()
                self.controller.show_frame(menu)


# Instantiates the Interface class as the object app
app = interface()
# Sets the geometry of the app and centers it in the center of the users monitor
width_of_window = 605
height_of_window = 538
screen_width = app.winfo_screenwidth()
screen_height = app.winfo_screenheight()
x_coordinate = (screen_width//2) - (width_of_window//2)
y_coordinate = (screen_height//2) - (height_of_window//2)
app.geometry("%dx%d+%d+%d" % (width_of_window, height_of_window, x_coordinate, y_coordinate))
app.resizable(False, False)
app.configure(background='grey11')
# Main loop of the entire system
app.mainloop()
```