Non-Exam Assessment

Computing Practical Project

AQA Advanced Subsidiary GCE in Computer Science

Dating Solution | Xcode, Swift, MySQL, Python

Contents

Analysis

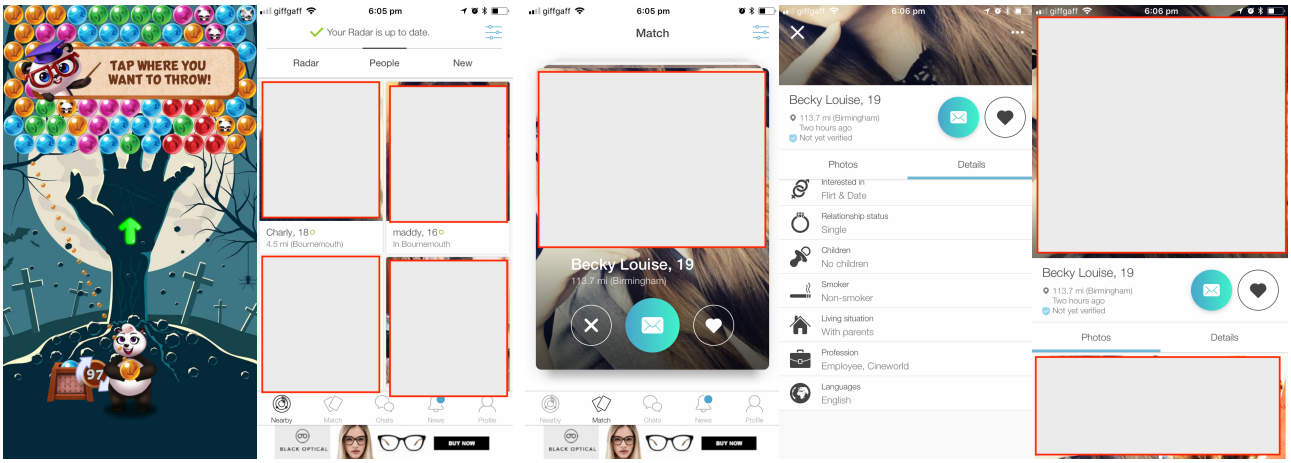
Background and Introduction

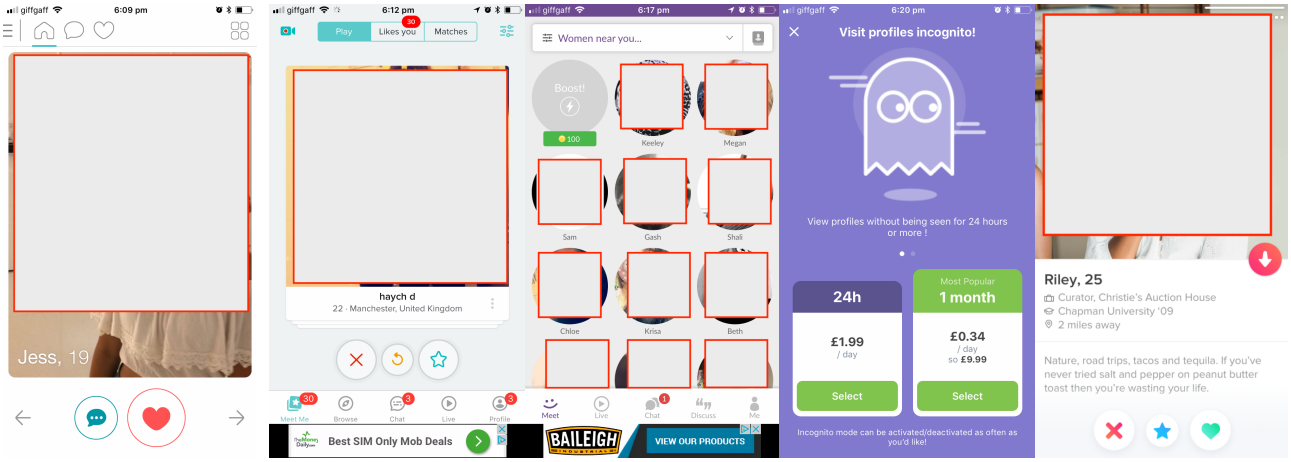
One of the founding natures of human behaviour is the desire to form a meaningful relationship with another person. Under this simple premise several companies have brought new innovative solutions to make the process of finding a partner more straightforward and accessible. In doing so those who have managed to provide a successful service have capitalised heavily on subscription based memberships, advertising and in app upgrades and perks. There are now many companies positioning services that meet different niche audiences who's needs aren't otherwise catered for.

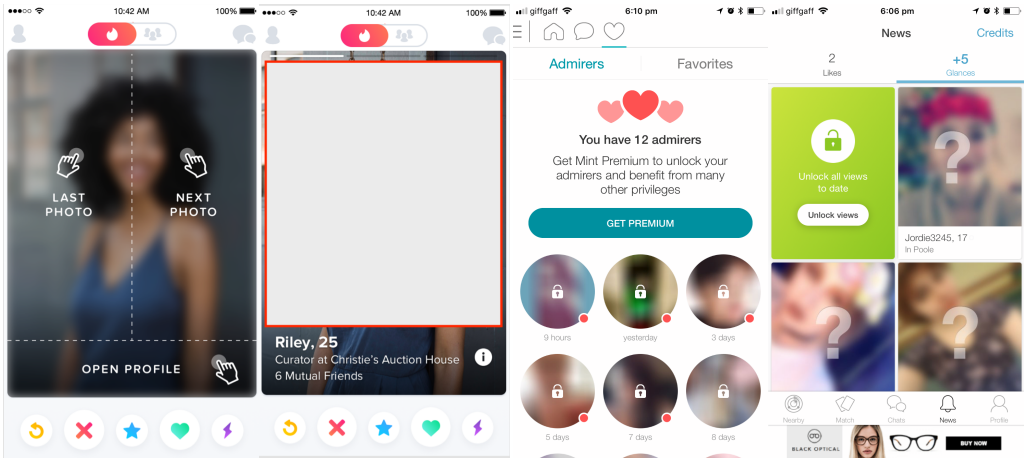
Current System

As dating is supposed to be such a simple process that anyone can access there has become a preference for an app only ecosystem over a cross-device one used by relatively view sites such as Match and OKCupid which may be intended for people of the older generation to use without a smart phone or those looking for something that is relatively 'safe for work'. While the solutions database could be implemented to power a website, most developers who are meeting only demands of the average millennial will focus on making a really good app that runs on iOS and Android. For this reason I didn't do any testing of services available through a web browser.

To make a full and proper investigation into the current solutions on the market I downloaded and used a selection of what are considered to be the more successful apps available which included: Tinder, Badoo, Bumble, eHarmony, Match, Zoosk, Happn and Huggle.







There were many common examples of how all the apps have evolved in tandem and just copied off each other with only a few posing unique selling points. This was of course in exception to apps designed for more complex alternative target markets. It is statistically proven that for this market segmentation the most popular revenue stream and pricing model is to make the app free to download but offer in app purchases or adds. The use of adverts displayed within the app enables all users to have an equal playing field without preference given to those who have bought extra credits and search ranking options. All had an extremely quick initial signup page that didn't bore the user while later prompting them to enter all their personal details to improve their experience. To make best use of space there was normally only a small menu while the rest of the screen was dedicated to presenting the user with a picture of their potential match. Some apps tried to stand out with unique features such as the use of location data to magically have potential matches meet on their daily commute. Many were clones of Tinder who's success was due to their early venture into the market and pure simplicity and design taste in the user experience which still feels very fun and fresh. Some were designed for people who were more serious and looking for people to meet exact specifications while others just gave a picture and description and were looking for friends.

Market & User Identification

The market for very 'normal', often relatively young, user looking to date is already saturated with enough dating solutions to please but for those in more niche markets there is still a demand for innovation. The data structure and interface of the my dating solution should not be based on the simple premise of male/female matches. Instead the solution will treat sex/gender as any other variable such as country of residence or ethnicity. This is very simple but makes the solution unique and very powerful in the different user bases it could attract. The target market would be users on a mobile device with a persistent internet.

To gather some information I made a user focus group chat with two participants of my target market. However the information gained was not relevant and had little effect on my eventual decisions. Instead I would focus on achieving a project with significant technical techniques and logic to perform error capturing etc as well as overcoming the challenge of a non binary match system.

Objectives

Ref	Objective
A	Signup - Allow the user to very quickly create an account with a secure password of their choice and a username that is not already taken
B	Login - Secure quick and easy

Ref	Objective
C	Logout and Quit - Allow user to easily navigate to logout and close the program so that they can save their data, close the program safety and not have their private details on screen should someone else share of have view of their screen
D	Profile Edit - Present user with nicely structured that instructs them to fill in personal details so that they can be better filtered by others
E	The following files should be editable by the user: Username, Password, Gender, Country, Ethnicity, Language, First Name, Last Name, Age, Picture, Bio
F	The following should allow the user to selected a value from a list of predefined values within a related table: Gender, Country, Ethnicity, Language
G	Data such as country and language should be sourced so that it suits all people, for example language should include different but distinct dialects and even small countries should be included
H	The program should be designed to allow for infinite definitions of gender instead of a more boolean approach ie either 1 or 0/male or female
I	Permissions should be in place to prevent users from editing certain data such as the listed values of predefined data like countries
J	Data validation to make sure further logic won't crash when for example users entered letters in the age field
K	Their should be a Match function that allows the user to see a picture of their potential match
L	The match function should allow the user to navigate to the filter
M	The Filter function should present the user with a similar interface to what they used to edit their profile with to enter options to filter details to select only the people with particular attributes for potential matching
N	The filter function should not update live but should perform the query once a button has been selected to commit the action
O	The Match should allow the user to either send a request to the currently shown user or to continue without any instead view the next person
P	There should be a view matches that allows the user to view a list of users who they have a mutual relationship with
R	A mutual relationship should only exist of be true if both users have requested each other. This means it is false if just one person like the other
S	The user should be able to select a user from the 'view matches' table which the takes them to a message so they are able to send and receive plain text messages
T	The 'Chat\Messages' should not allow for more then the two people in the relationship to communicate
V	The software should have an initial download cost and thus not have any adverts or paid membership subscription options
W	The database should be setup professionally with constraints and integrity assurance to prevent to from breaking

Proposed Solution Details

Pre Adaption
It would only make sense for the solution to be in the form of a website of mobile app. As I have a Mac and am more familiar with using Apple software I choose to use Apple's Xcode development environment and their Swift programming language to create the dating App for iOS devices. Building it within this system would then directly allow for the solution to be published to the App store world-wide for different iPhones and later adapted to work on other apple devices as well as there being possibility for porting the solution to android. As I only new Python I would have to learn both Xcode, Swift and any other technologies to complete the project in this form. Before undertaking the project I looked at the different ways I could learn and bought access to an online video course which would give be an in-depth and comprehensive education and teach me the skills needed to complete the project.

I also did research into how to publish Apps to the App store including securing a Developer ID and the associated fees. The course I partially finished was hosted on the Udemy platform.
Udemy Course

Post Adaption
It became quite clear late into the project that I wasn't capable of completing this project in the way I had intended. This was because I wasn't clever enough to learn the new technologies within the timeframe for the project as well as writing the final solution and documentation. This meant that I had to try and complete the project in Python as I had already learnt that through the previous year using Code Academy. I would also need to learn SQL for creating the data structure and TKinter for making GUIs.

Modelling of the problem

Acceptable limitations

Pre Adaption
There shouldn't have been any limitations using this method other then the trial tier used on the chosen cloud solution with certain scalability restraints as well as the cost of publishing to the App Store.

Post Adaption
Considering the change in plan I would have to consider several large limitations, mainly the platform and graphical user interface. The application would only be able to run on a desktop and their would be limitation to its accessibility as it would be executed within a development environment with all dependencies preinstalled on the system running it instead of the entire solution being complied for use on other variations of system setup. For example the solution could not be run on a users computer unless they had a certain version of Python installed and the PIL library. Using Python and SQL there would also be some limitation to how images could be stored. I would not be able to allow users to upload their picture very easily.

Data Volumes and Scalability

Pre Adaption
So that people would be able to access live up-to-date data I knew the App would have to be able to communicate with a server. For this to function the code would have to be written to interact with an interface that hooked into a free membership to a Public Cloud platform. There are different services which I tried including AWS Amazon Web Services and the Google Cloud Platform. After following the Udmey course I would have to use Heroku with the addition of Parse to achieve a Client - Server model. The would be an industry recognised setup and would have the potential to see huge scalability while also being robust and secure.

Post Adaption
There are different types of SQL and while it was easier for me to write SQLite from what I knew I tried to learn MySQL so that the program would theoretically be able to connect to a server hosted online. I choose to use MySQL locally vs implementing a Postgres database hosted on Heroku as there was complications with making a connection over the school network as it would need to go through the strict filter/firewall SmmothWall. The use of MySQL would allow me full scalability if the project was to be hosted within a public cloud where I could pay for more resources based on user demand.

Origin or data
There is two different types of data set for this project; user data and options for the lookup tables. Options for the lookup table can only be edited within development and would rarely be changed. User information would be partly user defined such as 'bio' or select from a lookup table. Before marketing the solution a pool or starting users would have to be used within on geo region for testing and to create a snowball effect of more people joining. While there are only four lookup tables this could be increased and already offers a large number of permutations.

Log/Diary of Research Collection
Appendix(Udemy Course)
Appendix(Bibliography)

Documented design

Algorithms

The program will be split up into several functions which are called either from the 'mainMenu' function or another such as 'message'. This would mean that a complicated procedure such as messaging a user where the users input was required each time to see whether thy wanted to proceed could be made quite simple and not require code to be copy and pasted. Globals should be used so that they can be accessed throughout the program instead of having to pass between functions and continuously set flags.

Data structures

MySQL is a fundamental of many services provided by some of the worlds largest technology companies such as: YouTube, PayPal, Google, Facebook, Twitter, eBay, Cisco, LinkedIn, Uber, Netflix, Walmart, WeChat, Tencent…. and many others. MySQL is developed by Oracle which is a highly profitable and recognised corporation.

Other Data Types

Apart from a database running locally, there are will be other data types used throughout the main program. Each time the predefined 'fetchall' function is executed using the cursor the result will be assigned to a value using python in the form or a tuple. Then certain values can be accessed from within the tuple by using an index for example "results[0]". However when storing multiple records in the form of a matrix or table the tuple must become multi dimensional and so there are two methods for accessing each row. The first is to use a 'for' loop where each loop assigns a record to the value 'row' for use within the loop. For example "for row in results:", and then "row[0]". However when accessing a two dimensional tuple the use of to values for an index can be made; for example "user[3][9]".

Object orientation

There are different methods for implementing the actual running of TKinter as is works in operation with the opening system to display aa window. The first method it to base the entire program around the use of TKinter and for the main program call to be 'mainloop()' on the initial TKinter solution. This will not be suitable for this project as the GUI is to only be used for the matching function and thus is not needed through the rest of the program. I also need to considered the timeframe of the project and that may main skills are in Python. So for this reason despite that using this method would allow me to demonstrated object oriented programming, I have planned to implement TKiner either nested within a 'while' loop or defined and called within a separate 'displayPicture' function. Apart from this Apart from this and the use of tuples there has been no use of objects as such within the solution.

Data Dictionary

Table	Field	Example	Type	MySQL	Default	Key
User	UserID	1	Integer	int(11)	AUTO_INCREMENT	PK
	Username	James13	String	varchar(40)	UNIQUE	
	Password	Password1!	String	varchar(50)		
	FirstName	Bob	String	varchar(80)		
	LastName	Smith	String	varchar(80)		
	GenderID	1	Integer	int(11)	1	FK
	CountryID	2	Integer	int(11)	1	FK
	EthID	30	Integer	int(11)	1	FK

Table	Field	Example	Type	MySQL	Default	Key
	LanID	2	Integer	int(11)	1	FK
	Age	18	Integer	int(11)		
	Bio	I like Cake!	String	varchar(500)		
Gender	GenderID	1	Integer	int(11)	AUTO_INCREMENT	PK
	Gender	Male	String	varchar(80)	"None"	
Country	CountryID	1	Integer	int(11)	AUTO_INCREMENT	PK
	Country	France	String	varchar(80)	"None"	
Eth	EthID	1	Integer	int(11)	AUTO_INCREMENT	PK
	Eth	Welsh	String	varchar(80)	"None"	
Lan	LanID	1	Integer	int(11)	AUTO_INCREMENT	PK
	Lan	English	String	varchar(80)	"None"	
Match	MatchID	1	Integer	int(11)	AUTO_INCREMENT	PK
	FromID	10	Integer	int(11)		
	ToID	4	Integer	int(11)		
	State	0	Bool	int(11)		
Message	MessageID	5	Integer	int(11)	AUTO_INCREMENT	PK
	FromID	4	Integer	int(11)		
	ToID	2	Integer	int(11)		
	Message	Hi There :)	String	char(200)		

MySQL Database Server

As earlier mentioned it wasn't possible within the school network to setup a connection to an external database server so the MySQL server would have to be hosted locally. To achieve this I tried running an SQL sever on my Macintosh but eventually settled on running Xampp on windows which functioned very well.

To setup the connection to the database running locally using Xampp for use by the Python code I would install SQLyog. This would allow me to execute SQL scripts as well as make sure the tables had been setup correctly as intended in the code.

Interfacing & Creating the Database

To create the database I could have chosen used a GUI within an ORDMS object oriented relational database management system such as Microsoft Access but for the purpose of this project I would utilise a database viewer application only for data input and testing purposes. To create and setup the database I would wrote a MySQL script using Visual Studio Code and an SQL plugin that appropriately highlighted different code elements and errors. To execute the script I would use SQLyog on an existing database connection referenced within the SQL script.

Database Design

Relationships

To meet the criteria of having the program allow for more matches to exist then just binary male to female ones I could't implement two user tables; User_Male, User_Female. Instead I would have to have one User table and then have a field in a related table for the Gender. The same would be for: Language, Ethnicity, Age, Country. To achieve this there would be relationship that allowed one user to have one related record from each lookup table.

There would also need to be a match table. However there was much confusion to how this would be setup. This is because Users form the same 'User' table would have to be related to each other. The relationships would also need to be mutual where a record should not be created for a relationship unless both users wanted to be in it. One solution was to create a second alias 'User' table which would be related to the original User table with a composite key. However I couldn't understand how this would be implemented to allow the relationship record to only be created when both users made an interaction.

To overcome this problem I decided to have an unrelated relationship/match table that created a record as soon as User 1 requested User 2. Then that record would only be made 'True' if User 2 tried to make a request with User 1. This would result in a table with a match record including the 'UserID' for both users and that their match was True. This table could then be used to aid the messaging functionality. All of this would have to be coded in the main program and prevent and integrity issues. The problem with a composite table is that it relied on one user creating the match which was not suitable as it was of vital importance that both users would have to agree for a match to exist. Below the code has been simplified to show the logic of how this would be implemented with the 'row' indexes corresponding to the fetched results from a previous query made on the user table.

```
name = results[0][0] + " " + results[0][1]
#If Match record exists then continue
if results2[0][3] == "1":
    #If state is True then tell user they can't match again with same user
    print("\nYou have already matched with User “,name)
elif results2[0][3] == "0":
    #If state is false then update status to true andprint that the user has made a new match!
    print("\nCongratulations,   You   have    Matched    with",name)
          sql = "UPDATE 'Match' SET State='1' WHERE MatchID= results2[0][0]"
    c.execute(sql) db.commit()
else:
    #Else if match does not exist then insert new match record with state as false
    sql = "INSERT INTO 'Match' (FromID, ToID, State) VALUES ('UserID, results[0][9], 0)"
    c.execute(sql) db.commit()
```

To allow the user to message their match there would be a separate message table holding the messages of all users form all matches. This would need to hold data on which user if was from and who it was too.

Normalisation

First

Instead of printing all the Genders, Countries, Languages and Ethnicities from a single cell with comma delimitated values each option was given a row and unique ID within a respective table as can be seen in 'FINAL SOLUTION' diagram. The 'User' table then one holds one foreign key for each respective lookup table.

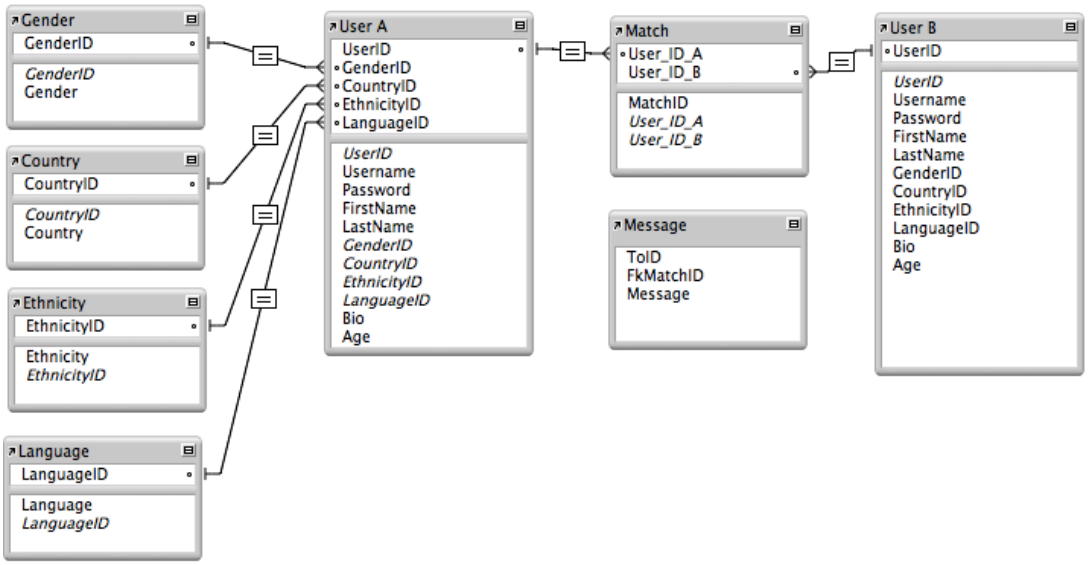
Second

This may have been of concern when using the original database relationship model as can be seen in the 'COMPOSITE KEY METHOD' diagram. However using the final solution each table had its own primary key on which the records were dependent including the lookup tables.

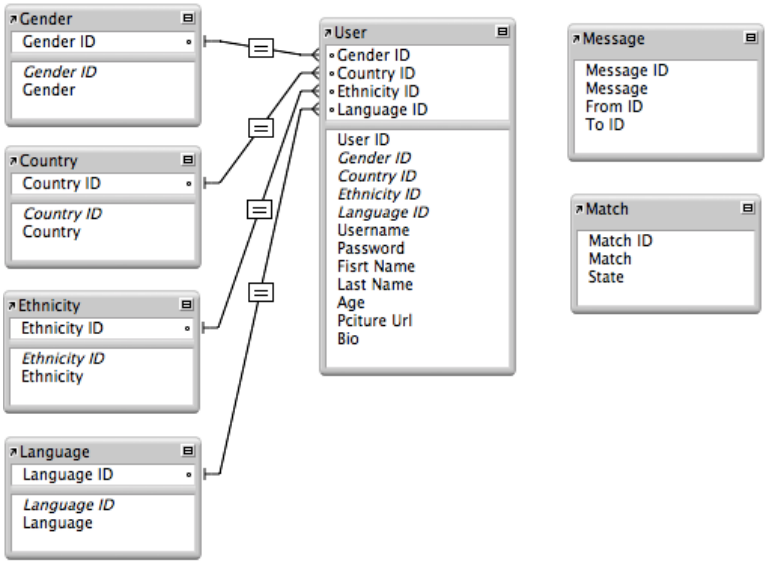
Third

Originally is was decided that a field named 'PictureURL' would exist. However once the selection of hardware and platform had been decided it became clear that all pictures would be stored within one folder in the root directory of the source code being run. For this to function the only information needed would be a unique identifier for the photo while the full path and picture format suffix could be set within the MySQl query using string concatenation with Python. This meant that the replacement field 'PictureID' would have the same default value and auto increment function as the 'UserID' field. However this is in violation of SQL database integrity rules so this would not be possible. In accordance with the third rule of normalisation it was also not possible to make the 'PictureID' field dependant on the 'UserID' field. For this reason the 'UserID' was fetched within the main program and used for the unique identifier in the 'displayPicture' function.

COMPOSITE KEY METHOD



FINAL SOLUTION



User Interface Design

Pre Adaption
This Udemy course also included lots of assets which I would be able to use to further enhance the user experience of the solution. Using Xcode's interface I would be able to use a design environment to drag and drop in user interface elements which could then be linked to code within the editor. Appendix(Udemy Course)

Post Adaption
The user interface would be restricted by my ability to learn TKinter. All the buttons and user input files would be uniform and conform to one style. I would either create a set of windows using it or just one textual user interface through a command line prompt with the exception of using TKinter to display profile images. There are easily understandable prompts for the user and the program performs actions very quickly.

LOGIN

Username username1
Password password1

FILTER

Gender ID 1
Country ID 1
Ethnicity ID 1
Language ID 1
Age 18 18

PROFILE

Username username1
Password password1
Fisrt Name firstname1
Last Name lastname1
Age 18
Pciture Url https://goo.gl/2UMSmd
Bio Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also

Gender ID 1
Country ID 1
Ethnicity ID 1
Language ID 1

Commit Changes

MATCH

Edit Profile | Filter | View Matches | Logout

Request | Next

SELECT MATCH


Fisrt Name ::Fisrt Name Last Name ::Last Name


Fisrt Name ::Fisrt Name Last Name ::Last Name

MESSAGE

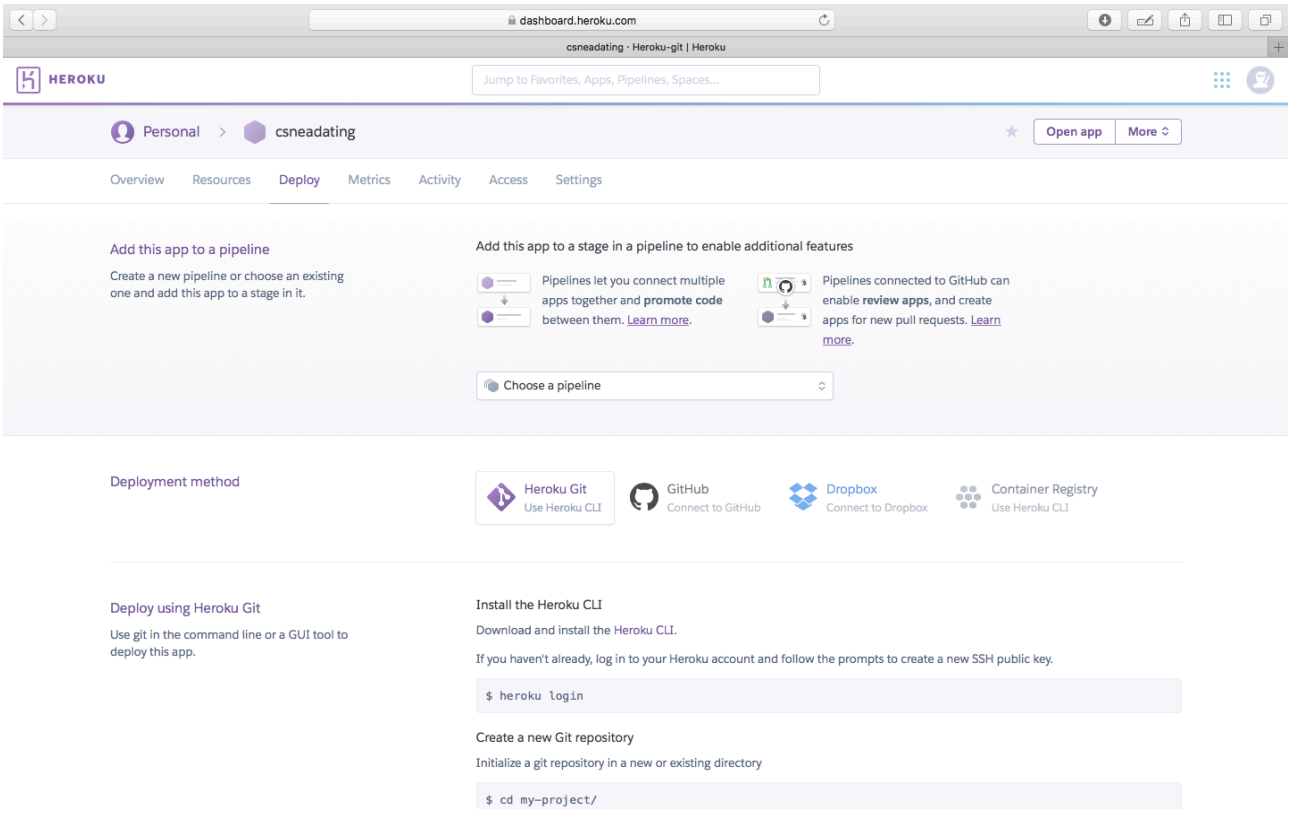
From ID | To ID | Message
From ID | To ID | Message

From ID | To ID | Message
From ID | To ID | Message

Hardware Design and Selection

I originally choose to use a free tier of Heroku as this would mean I wouldn't have to worry about additional setup complications. However it wasn't possible to make the necessary connection through the school's network so MySQL was chosen which could in theory be run using Heroku, AWS, Google Cloud Platform or a one of many other public cloud providers. While I use a Macintosh as my daily driver I utilised Parallels so that I was able to run certain software only available for Windows.

I already had Python 2 installed on both my Macintosh and virtual Windows instance so to be able to run Python 3 without uninstalling Python 2 I installed Anaconda on Windows which enabled me to run initiate a Python 3 shell from within the command prompt using "activate py36'.



System security Data Integrity

While MySQL is very secure the overall solution would have vulnerabilities. However this was not covered by the scope of the project specification.

Development Log

I switched from using Xcode and Swift to using Python to achieve this project due to my own ability and the time constraints. I switched from using SQLite to MySQL so that I would be able to achieve a client server model. Initial development was made using Python in IDLE and Jet Brain's IDEA PyCharm before migrating to Visual Studio Code. From here code was run and tested using the windows command prompt. SQLyog was also used for executing SQL scripts, creating the data and importing data.

Technical solution

Source Code

The original code was written using Visual Studio Code but has been copied with formatting into two separate documents. In some cases the MySQl queries implement within the Python file have been formatted so they are more able to read. The lighter green is used to highlight certain comments as titles for main purposes of code blocks. Comments are used throughout to illustrate different techniques used and to example the function of the code.

Appendix(MySQL)
Appendix(Python)

Screencast

https://youtu.be/kY1O3cNeuFs

The following in demonstrated:

- Realtime running of a MySQL server locally using Xampp
- Creation of a database with a connection to the server at root using SQLyog
- SQL script being edited to function with the database
- SQL Script being executed on the server and successfully altering the target database
- Identifying the SQL specially responsible for creating a particular attribute of the database an an example
- Running both Python version 2 and 3 on the same system using Anaconda as shown using the command line
- Setting the root directory of the command prompt session to that of the directory holding the source code
- Executing the Python file and identifying the code responsible for printing the message shown in the command prompt
- Importing data into the database using SQLyog to import pre made CSV files for each table. The lookup tables must be created first before the 'User' table. The errors shown are because the first lien fo each CSV file golds the fields names
- Main Python file is executed and live functionality and its use of the database is demonstrated by logging in to 'U1' and then deleting it form the command prompt before confirming this change by refreshing the table view of 'User' within SQLyog

Following actions demonstrated can be seen referenced in the table of testing which identifies each action and it's resulting outcome.

Testing

Invalid Data
- Input of letter or special character in 'int' field
- Too many characters entered for field memory allocation
- Option does not exist, for example option must be in [1,4,5,6,8]
- selection out of range, for example only 10 gender options and option 12 entered
- No data entered, can be valid for varchar() but may require dealt value
- keyboard interrupt such as "Ctrl + " causes main loop to break and program exit

Scope	Action	Outcome
Main Program	Keyboard Interrupt e.g Ctrl + C	break from program
	closed TKinter window	some stalling of GUI
	Didn't close program and used other at same time	experienced some lag and TKinter not responding
Main Menu	out of range	retuned to 'mainMenu' with error message
	wrong data type	retuned to 'mainMenu' with error message
	valid option	execute correct function for selection
	option in selection that requires logging in	retuned to 'mainMenu' with error message
Signup	using existing details	retuned to 'mainMenu' with error message
	password strength invalid	retuned to 'signup' with error message
	Username and Password valid	continued to 'login'
Login	on existent details	retuned to 'mainMenu' with error message
	no details	retuned to 'mainMenu' with error message
	mismatching details	retuned to 'mainMenu' with error message
	Username and Password valid	retuned to 'mainMenu'
Password Strength	no capital letter	retuned to 'passwordStrength' with error message
	no letters	retuned to 'passwordStrength' with error message
	no numbers	retuned to 'passwordStrength' with error message
	not enough characters	retuned to 'passwordStrength' with error message
	to many characters	retuned to 'passwordStrength' with error message
	Password valid	continued to finish 'signup' with success message
	Password already exists and is valid	continued to finish 'signup' with success message
Delete User	continue yes	goodbye message printed and account removed

Scope	Action	Outcome
	continue no	retuned to 'mainMenu'
	invalid option	retuned to 'mainMenu'
Logout	no input required	functioned correctly
Profile	index out or range	retuned to 'mainMenu' with error message
	invalid data type	retuned to 'mainMenu' with error message
	Exiting from profile function	retuned to 'mainMenu'
	valid option	continued to 'profile'
	editing same information again	continued to 'profile'
	Password validation for changing password	looped through 'passwordStrength' until valid
Match	'filterUsers' returns false	retuned to 'mainMenu' with error message
	exit selected	breaks from 'match' and returns to 'mainMenu'
	invalid selection	breaks from 'match' and returns to 'mainMenu'
	request made to user with existing match	continues to next user in 'filterResults'
	request made to user with existing rerun request	continues to next user in 'filterResults' and print message
	request made to same user	continues to next user in 'filterResults'
Filter Users	index out or range	returns to 'match' with True
	invalid data type	returns to 'match' with True
	valid options	returns to 'match' with True
	valid option	continues
Message		
	no results from 'viewMatches'	retuned to 'mainMenu'
	invalid option for "continue"	retuned to 'mainMenu'
	'yes' option selected	continues
	invalid option for "send message"	retuned to 'mainMenu'
	yes' option selected	continues to 'showMessages', 'sendMessage', 'showMessages'
View Matches	no matches	nothing returned, this is issue as should return false and break from message
	matches found	matches printed

Scope	Action	Outcome
Select Match	valid option	
	invalid option	print no matches found and continue with main 'message' function
Show Messages	no messages	none shown
	one or many messages found	all displayed correctly
Send Message	message valid	message sent and exit function
	Message empty	continue but print message

Evaluation

Objectives

Ref	Evaluation
A	Very quick signup process using command line with no lag. Username is forced to be unique using 'UNIQUE' constraint within SQL and check done within Python code
B	Login is done very quickly and easily using command line. When program is exited all user information of flushed form memory and globals are cleared for security
C	Program can be securely and safety quit and exited, This can also be done very quickly from 'mainMenu'
D	Profile' function utilises related lookup tables so that is possible to make a selection from predefined options. This means there is no confusion with spelling long values such as the name of a country. Min and Max age are taken form he user so they are able to select a age range.
E	All of the fields stated in the objectives were achieved: Username, Password, FirstName, LastName, Age Picture, Bio
F	All of the fields stated in the objectives for being stored within lookup tables were achieved: Gender, Country, Ethnicity, Language
G	A few different sources were accessed to download Excel files with lists of countries etc. These were then imported into the database using SQLyog
H	This was achieved by using Facebooks gender list and storing it in a lookup table
I	No SQL queries were made that would allow the user to edit other fields or alter tables that wen't intended for a common us case
J	Error capturing was implemented both for letters in the age field and at other times were the program relied on certain values being entered correctly
K	This was achieved by performing a query that returned all relevant matches to the current user where the 'State' was 1 or True
L	This was achieved and allows the user to save a the results of a filter query in a global which can be access for the purpose fo matching at any time during the login session
M	This was achieved, to start writing the 'filterUsers' function the code from the 'profile' function was copied and then altered
N	This was done so that the user could message a user and then return to matching while not having to execute the filter function form start again
O	While' and 'for' loops were used to allow the user to iterate over each user and make a decision to whether they made a request to them or past to the next profile
P	This was achieved by querying the 'Match' table for records relating to the current 'UserID' global where the state cell was set to "1"
R	A "1" was used to indicat the relationship in the match table was mutually held with another user and thus valid/True
S	This was complicated but achieved by nesting smaller functions within 'while' loops within the main 'Message' function.
T	This was achieved by only allowing a chat to be initiated on the basis of the validly of a record in the 'Match' table

Ref	Evaluation
V	This would be achieved later once this project was put into production but currently this was not achieved in the state of the project
W	This was achieved using the industry standard MySQl and constrains and keys. The SQL queries and Python was also written to validate data and accept key Errors and erroneous data

User Testing & Feedback
It was deemed inappropriate to reengage the focus group consulted through the analysis stage due to the huge change in project direction. Most of the feedback gained was irreverent and had little impact on decision made through the project development.

Some problems were identified through final testing:
- The 'message' function continues even if 'viewMatches' has returned no results
- An error message is not displayed and the 'message' function continues if invalid input is made within 'selectMatch'
- The statement showing the count of users found does not return the actual value of records found
- The TKinter window remains open when the 'match' function is no longer in use which causes problems further on into the program us
- The implementation of TKinter should be optimised for performance so that a loop is not always running to maintain it which uses system memory

Self Reflection and Review
The initial aspiration was not just to achieve a project for the purposes of the NEA specification but also to gain knowledge and real experience in using Xcode and Swift for the purposes of creating an App. In this respect I had set my aspirations of achievement higher then were reasonable for the timeframe. Because of this there is of course a certain amount of guilt in failure and disappointment. However once the project objectives and vision was revised and paired down with potential expansions such TKinter, MySQL the project became manageable and was eventually achieved. This outlook and final approach should be considered when making a final evaluation of the project as there are lots of obvious things that would need to have been done differently should this solution have seriously been intended for commercial use.

Further Development
There should be a facility to allow the user to upload pictures themselves in a secure fashion to a cloud service. There should be a user friendly GUI that is initiative and looks fun and fresh. The whole project should be compiled and packaged. The solution be easy to download and install using a utility and should be designed to prevent people for using it without a license code purchased. Security function should be implemented to prevent the user form performing SQL injection. The code should be written in such a way that it is very easy for further lookup tables to be added without lots of manual changes being required to be made in the source code. The solution should be able to operate cross platform on Mobile and desktop devices. There should be a strong branding and theme so that people feel hooked into the system and will tel other people about it. The system could be given a GUI using TKinter or left in its current state with a TUI textual user interface. The advantage of using a TUI is the fast loading speed and the ability to work with thousands of filter options and records.