FESTIVAL FINDER PROGRAM

OCR COMPUTING A-LEVEL COURSEWORK

George Wellington // Candidate Number 2478

Centre 52423

Contents

Investigation and Analysis

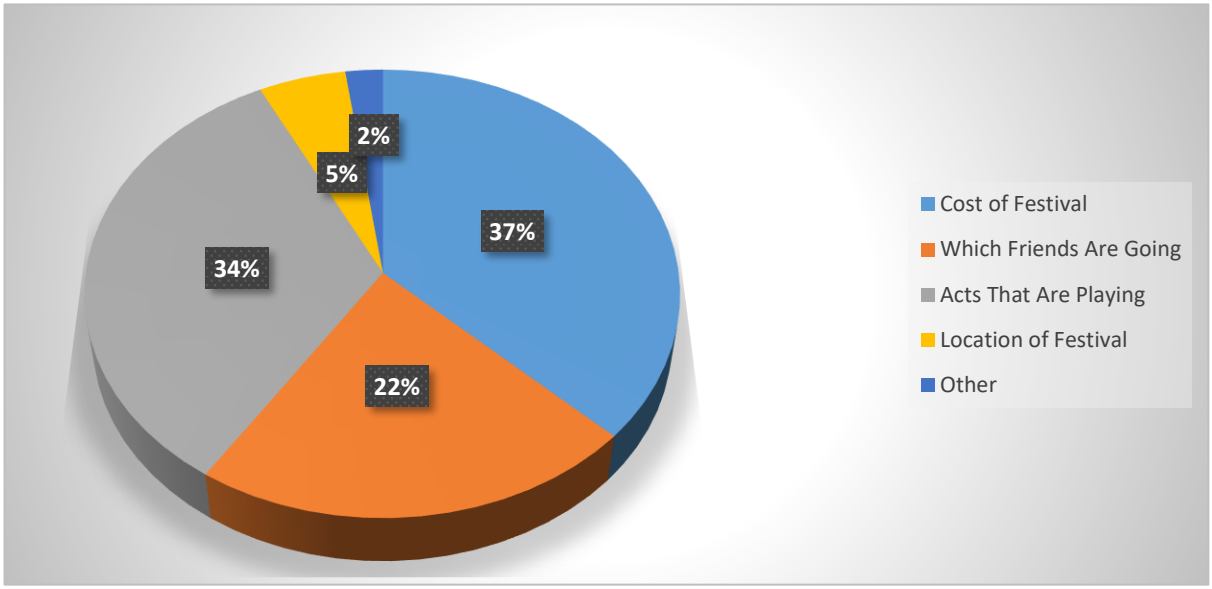
Background Information and The End-User

Music festivals, usually held annually and outside, are a massive industry. This project is going to revolve around a typical rock festival, the best examples being Glastonbury or Reading. The festivals have multiple stages and multiple acts and can attract a user base of several hundred thousand people.

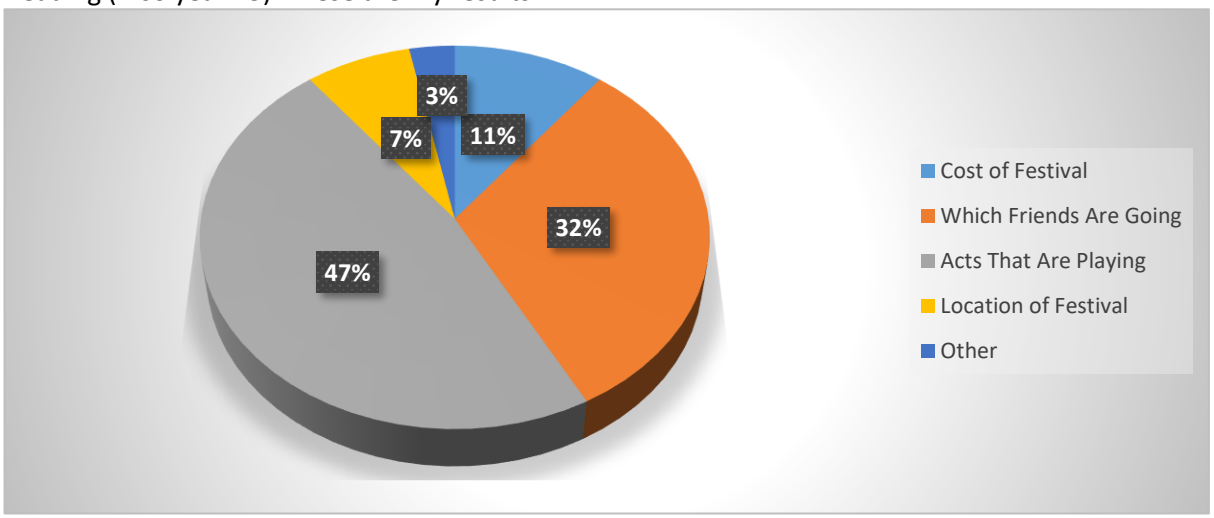
Festival Republic, my target market and focus group, attracts 500,000 customers/users per year. Thus, they need rigorous organisation and a database management; a system to buy and organise their ticket selling. Festival Republic is a music promoter based in the UK, found in 1982 by John Vincent Power as the Mean Fiddler Group. It started as a music promoting and venue-management business, but recognised the rising market and potential revenue of music festivals towards the end of the 1980s and took control of the Reading Festival brand. Following acquisition by Hamsard Ltd in 2005, the focus of the company became more on music festivals, most notable in the UK, with the old business, venues and non-festival related assets sold on in 2007 and the name was changed to Festival Republic. It's a private limited company, based at Regent Arcade House, Argyll Street, London, W1F 7TS. Festival Republic's flagship events are the Reading and Leeds Festivals. These are a pair of annual events which take place simultaneously on the Friday, Saturday, and Sunday of the August bank holiday weekend, sharing the same timeframe and musical acts. Their festivals include but are not limited to Latitude Festival in Suffolk, Wireless and Community Festival in London, EDC UK in Milton Keynes and the Download Festival in Leicestershire. Their business has been growing since the late 1980s, with business booming as interest in festivals continue to climb year on year. A Program like my proposed "Festival Finder" will be very useful to sell for the End-User, as it simplifies their current process and would attract more customers.

The Current System

The current system is incredibly confusing. For most, the acts, cost and timing of each individual festival run by Festival Republic is the "make or break" decider for most customers.
I held a survey among 17 and 18 year olds in RGSHW year 13, asking what was most important to them when it came to festivals. The sample was of 60 regular festival goers, these are my results:



I also held the same survey of 60 regular festival goers from The Abbey School (Private School) in Reading (Also year 13). These are my results:

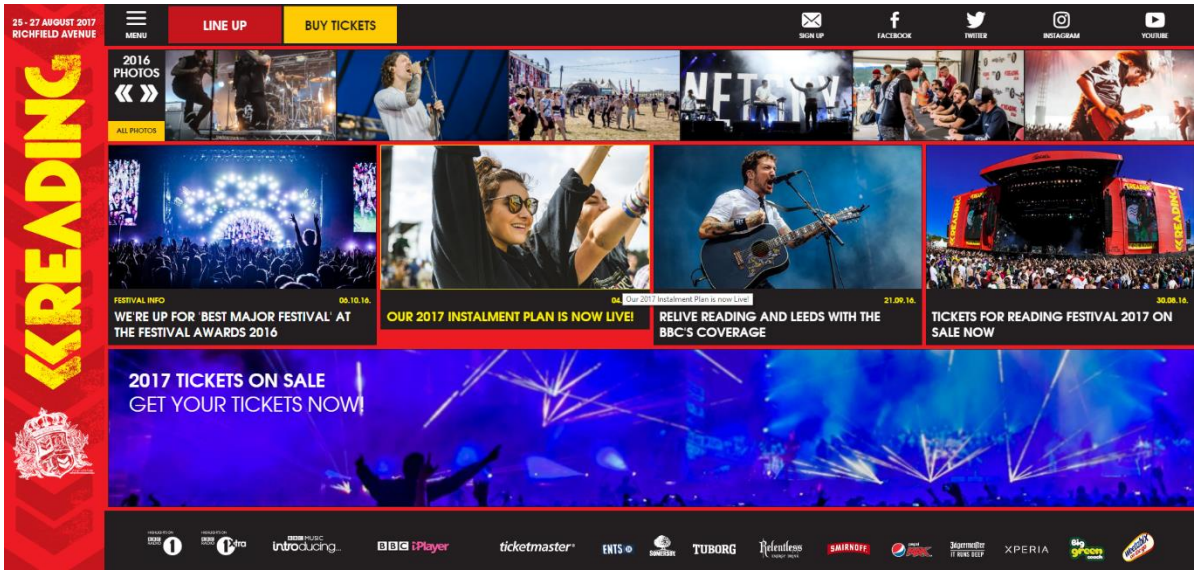


These results prove that these individual factors remain largely the deciding factors when choosing a festival to go to – the program would therefore have a large target market for Festival Republic.

Current Examples and Competitors

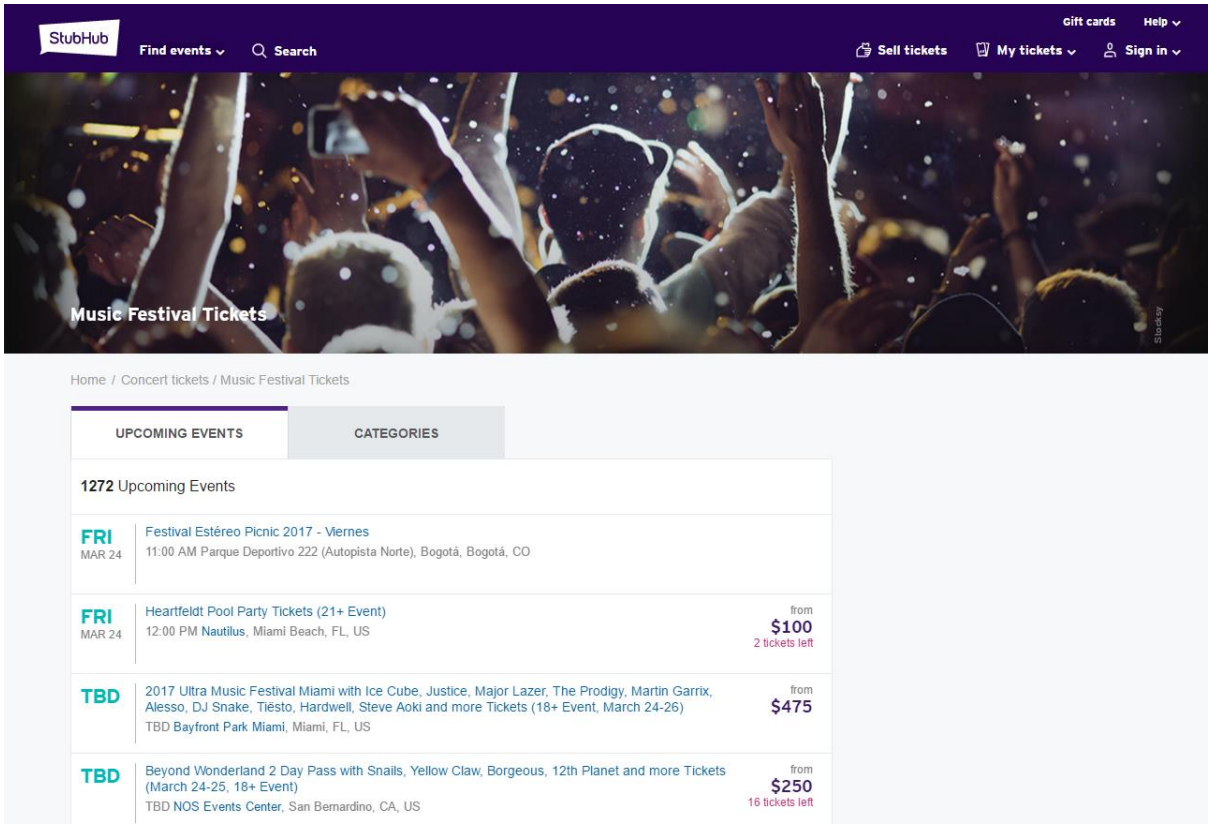
These are current examples from ticket buying/selling websites and the festival republic site itself. The festival republic doesn't yet have a system in place for buying and selling tickets directly, nor does it have an automated system to sort and lookup festivals by cost, location or acts.

Right away we have a problem. Festival Republic, despite the formatted background, immediately presents us with a very unfriendly interface for potential customers. It boasts 7 unique festivals across three countries yet fails to provide any links to buy tickets, information on acts or any actual easy sale links. For its modern target market, to sell its tickets efficiently, we will need a nice user interface with search functions so potential customers can search for the acts they want and find the tickets they would need. Other things to think about in the potential system would be searching via location, costs and perhaps social media integration to see who is "interested" or "going" via a Facebook applet link.

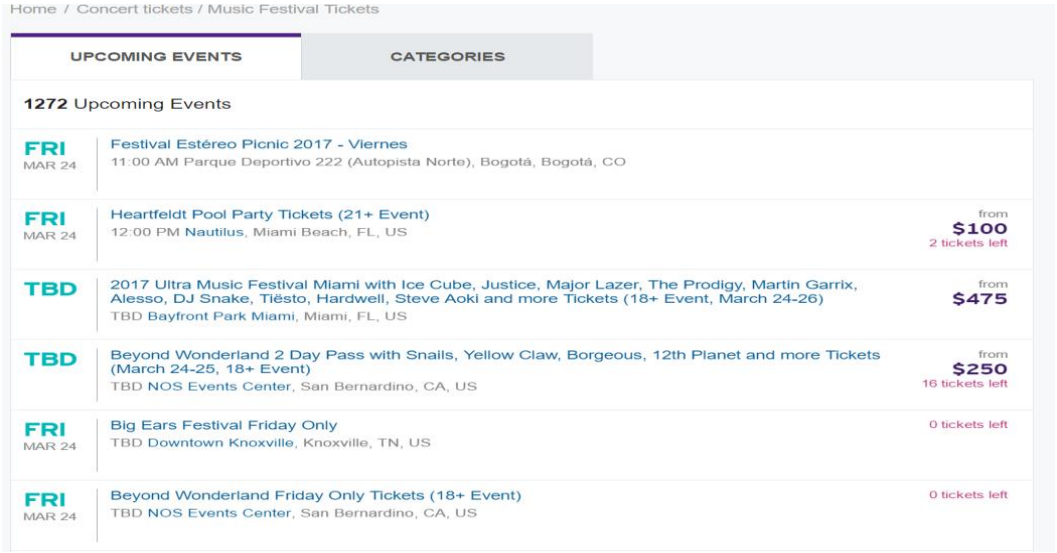


Here is a snapshot of a dedicated festival site.

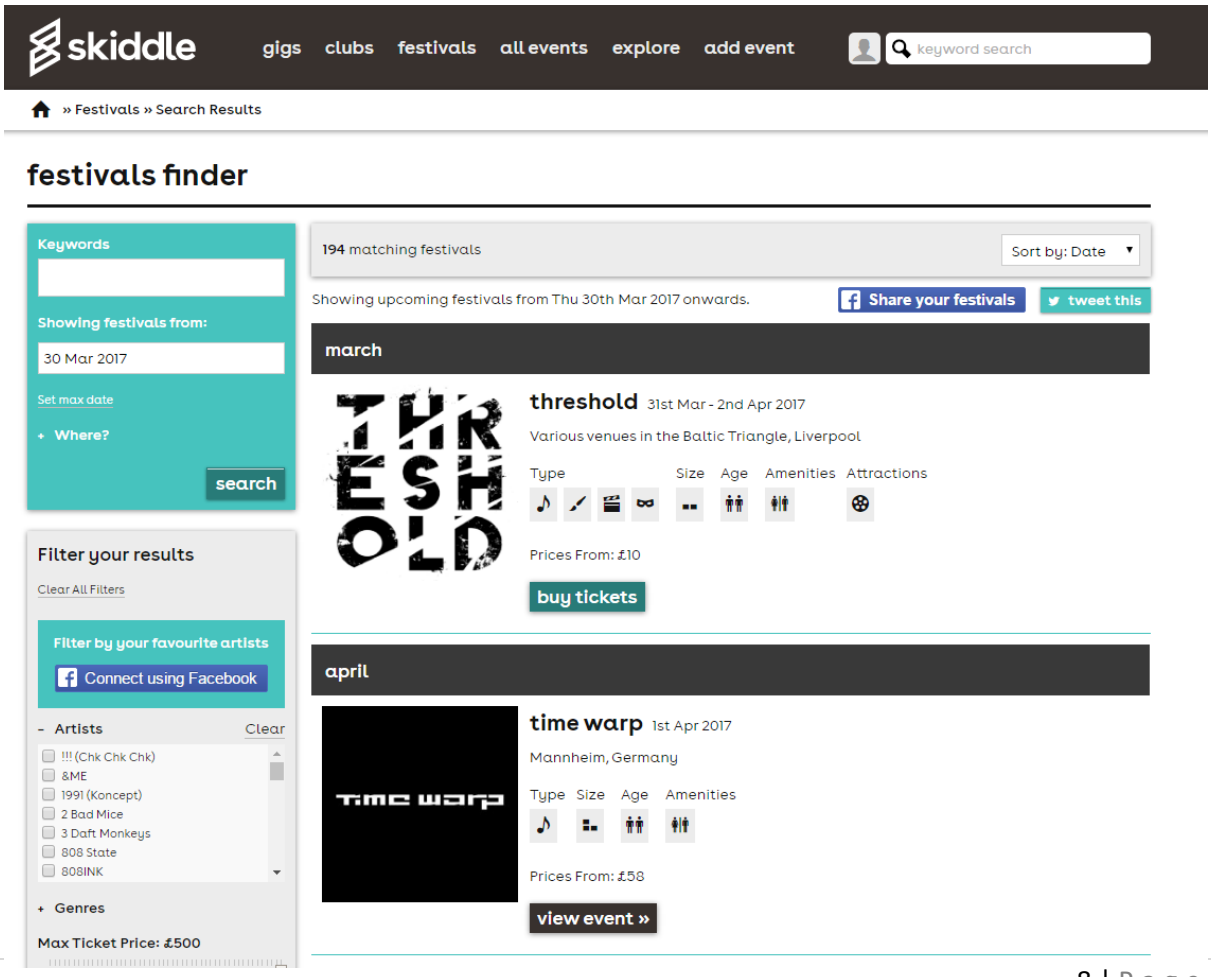
Straight away, even though we've gone through another link to a dedicated website, most of our screen is taken up by graphics and videos. For a website selling tickets at roughly £220 each, this is quite worrying. For something so expensive, the company would benefit massively from a search based engine and database system for buying and finding out information about festivals and its tickets.



This example, StubHub.com, is a music festival ticket vendor, and again we can immediately spot some problems with its hassle inducing interface. Despite a sleek and finished website we can spot something problematic about the list of festivals –



We are met with a display of mixed formats: We are given details, acts, dates and age ranges in the title line, a mix of data in each separate box and conflicting events – there are 1272 events yet immediately we are met with inconvenient ones first, events in different countries to the user, events in different countries to each other (Canada and America) and events on the same day – if the program/service displays events in this manner on the first page, the user will have a hard time understanding the format, finding relevant events, finding vendors, finding events near to them and reading the data itself: all of the data is presented at once and it isn't clear.



This final example provides a substantially more effective competitor than the previous websites, yet is still rife with problems. It has a variety of search functions, yet first provides very different festivals at the forefront, festivals in non-user specific locations, during the same timeframe and of different genres. It does not offer the option to buy tickets for both, and the Graphical User Interface is filled with confusing options. The calculation entirely depends on the continual input of the user, and so is not as accessible as it could be – my product will succeed in user accessibility and functionality where these products have fallen short.

Current Process/ Data-Flow

The Proposal

This program would plan and organise festivals, to be used by both Festival Staff and ticket companies to sell tickets, as well as the staff running the website of the festival. The staff associated with the Festival Republic organisation and perhaps other similar organisations would therefore be the target market. The user could search all festivals offered by genre, bands, location, price, and social reasons, achieving their perfect festival, and making the whole system a lot more personable and simple.

Initial Objectives

- Write a clean and efficient program, a well written program that should be easy to maintain for future code writers. Therefore, I wish to include the following paradigms –
    1. Functions – functions are sub-programs that return values. This makes for smooth and easy to understand code writing and saves resources. These are called multiple times during computation so that time and energy is saved and a program runs smoothly
    2. Modularity – Dedicated modules for different functions (eg. Login, Database Editing, output etc)
    3. Comments – Commenting and annotating my code and thought process will make my program easy to maintain for our client and easy to improve upon. This is also will help me greatly in debugging
    4. Ease of Use – A nice clean UI and easy to use product solves the problems currently faced by the company and provides a vehicle for our end user's customers to buy tickets and find out information. An easy to use program makes it accessible for all potential customers and clients.
    5. Efficiency – Clean, uncluttered, and short code ensures slow processing downtime and fast responses as well as an accessible system. It enhances maintainability also.

What makes this a problem that could be solvable and improved by various computation methods?

For a service/product so expensive, a focus should be ease of use and accessibility for our end user's client(s). Therefore, a search based lookup system for tickets, bands and festivals would be incredibly useful. The computational aspect of this is straightforward; in an age where cars are being driven by artificial intelligence, personal assistants are being replaced by voice-activated machine learning algorithms and all our shopping is personally tailored based on location, browsing history and interests, any service selling a product, e.g. a festival ticket, should be as accessible as possible.

When selling a product, the customer, which could very well be a disapproving parent, needs to know all the facts. The key demographic of music festivals is the age 16-25 bracket and our end-user's final product needs to reflect this. Therefore, we would need an integrated search function with full backend database compatibility for accessibility and general functionality. I would also include an account based interface, as people buying tickets would need a user/customer based level of permissions and means to buy the product itself and our end-user would need a way to moderate and maintain this, with administrator level permissions. It could also solve affordability and act clashing problems, with a sophisticated computational aspect ensuring the customer has the most information possible with automated information dynamically basing options for seeing their favourite music acts based upon their interests and if it clashes with pre-bought tickets. This would

enable customers to access a more inclusive all around booking type system, while maintaining the simple yet extremely sellable concept of a music festival. It could even host advertisements to create more revenue for our end-user. It's an extremely lucrative and sophisticated product which will enable our end-user to manifest forms of revenue with a low process heavy system, if implicated correctly. It will revolutionise ticket buying systems by cutting off third party vendors and saving money, whilst selling more tickets overall.

To provide relevant data, the program would need to provide and output information based upon the data that the user inputs. The program would act abstractly: all necessary data would be removed whilst the user based inputs, their location, price range and the dates would be kept and used to provide a useful output for the user – justifying the need for the problem to be solved via computational methods. Any additional complexity would not affect the output as that would be solely based on the relevant stored information in comparison to the data the user has input into the program.

The program can carry out the problem in a maximum of polynomial time, as this is the maximum time complexity of any function or command in the program – therefore the program is not intractable. I've made the assumption that the postcodes for the festivals provided are the exact locations, the festivals are listed under the headliners only and the price is used as GBP.

My platform would be the best way to carry out this process for users – this way the program can calculate the data simultaneously for many festivals and saves results for later use for users. It can also be used to purchase tickets – something pen, paper or other means could not do without the use of a ticket vendor. The program's main unique selling point would be simplifying the entire process of finding a festival, optimising the process of searching and finding out information in order to reduce the time taken, making my program vastly superior to manual research by pen and paper, or even querying via many different websites. Using my platform conglomerates all of the festival data relevant to the user for full, efficient analysis of available festivals, and simplifiers the process of finding a festival to purchase tickets to for consumers.

I chose the python platform, along with the main use of Object-Orientated Programming and the PyQt Graphical User Interface library and SQL based databases as they are familiar, there is a lot of support for the platform and therefore will be easier to debug and understand for the next owner of the code, as well as being accessible on multiple platforms. The wide range of libraries and compatibility with other languages, modules and functionality makes the python programming language a versatile platform for my program. It has compatibility with the Google Maps API, JavaScript, HTML, JavaScript Object Notation as well as many other programming languages, however these languages are significant as I can incorporate them in Distance Matrix calculations, web views for my PyQt Graphical User Interface and many other functions. Also, the large amount of libraries for public use, such as "smtplib", a library for connecting with Send Mail Transfer Protocol in order to send emails, or "re", a library for recognising regular expressions, make python an even more versatile platform and the choice for the port of my program.

Limitations of the Proposed Solution

Given the constraint of the timeframe of this project, my personal skillset, the number of workers and the means at my disposal, there are some features I will not be able to implement in the due timeframe for a project of this scale – these features could be added by the next holder of the source code or at a later date, but this project will not include these features and will be limited to the ones we implement. The features that are conceptually captivating but ultimately extensive and impossible to implement in this timeframe are:

- Social-Media integration – One implementation could be to link a Facebook account with the user account and synchronise the friend feed – display the events that friends are interested in, "like", are watching or going to. Besides a festival profile could like the friends of the user that are planning to go to this even through the festival event page.
- Media Player Functionality – This could be implemented through a mini-media-player to show a taster of some of the acts associated with the festival. The could be connected to Spotify or YouTube to showcase some of the songs that the artist performing at the particular festival has released.
- Buying the Tickets Directly Through the program – Due to pricing and actual ticket licensing issues, the program will only be able to sell tickets through third parties – however in later additions I would like to improve on this limitation through acquiring rights to sell tickets to festivals directly through the program. However, the program still provides quick access to the ticket vendor directly through the profile aspect, providing a link to buy tickets in plain view, for when a user has decided on which festival tickets to purchase.

First Meeting

My first meeting with my initial client and end-user, **Festival Republic,** involved Seymour Parsons, a senior member of Festival Republic and Gareth Cutbill and Johnathon Skelton, two sales representatives. These were my initial queries and issues, based in a Question and Answer format in a focus group type meeting.

What are your initial worries or things you are looking for in a search-based festival finder system?

"Our main worries would of course be customer appeal – the interface of your program would have to look the part"

"Ease of use would be essential – it would have to be easy to navigate"

Do you think your current system could use some improvement?

"I believe that our current system could use some improvement, especially with user integration and in particular the ticket selling process"

"Maybe ease of use and general GUI looks and usability could be improved"

Could my idea evolve into a product that fixes these problems?

"*If your product focusses on the ideas and objectives you discussed then yes"*

Would a program with a GUI, web integration and database compatibility fit your idea specifications?

"These ideas would fit our ideal program to sell tickets and further our profit margin for our products and services"

Link to Signatures of the End-Users (Bibliography)

Objectives and Product Specifications (Hardware and Software)

Usability

1. The program will provide an interface between the user, a festival ticket buyer, to purchase tickets and find out information about festivals.
2. The program should link to the appropriate ticket vendor for the festival selected.
3. It will provide search/ selection tools that allow the user to narrow down their search by different protocols, for example, location and cost.
4. It should involve full database integration.
5. It provides a graphical user interface for log-on, then provides access to the festival program:
    a. The log-on should have full database compatibility. Usernames and Passwords need to be linked.
    b. The system should store essential information on current users, such as links to orders, and information on the festivals themselves.
    c. The program should provide a registering service for new users, where they can set up a username and password and create an account to use the service.
    d. The log-on screen must be passed by logging in in order to gain access to the product.
    e. The program should have a "Forgot Password" service, either through email or security question, for password recovery.
    f. The programs forgot password service should provide a random number to reset the password to, update the user's table accordingly and email them their forgotten password in a correct format – eg. An Official "Festival Finder – Forgot Password" style email.
    g. To use Regular Expressions functionality in order to regulate username and password form. For example, maintain a form for passwords to be between 8 and 20 characters, including numbers and different cases.
    h. To provide a service that allows a user to delete their account and all records of it if necessary.
    i. The program should encrypt all passwords accordingly, for example using a MD5 encryption when checking, submitting and changing passwords.
6. The Program should provide a suitable menu interface to link all the features together.
7. The programs should show evidence of fetching the current users details and greeting them, e.g. "Welcome back, (Username)"
8. The program should provide a means to navigate the program itself.
    a. The program should provide an easy to use Graphical User Interface which can navigate from one function to another.
    b. The program should provide a menu after the process of logging on to navigate between different search functions – cost, location finder etc.
    c. The program should provide easy navigation buttons to traverse the service – Close, Back, Logout etc.
    d. The program should provide buttons for all the possible needs the user would want to use.
9. The program should provide an accurate depiction of the end-user, to ensure maximum product effectiveness, tailored to our target audience. (In the form of logos, perhaps tailored information for tables)

Performance

10. Provide a permission based system:
    a. Admin users should have access to the users database from inside the program.
    b. Admin users should be able to edit the permissions levels of other uses.
    c. Admin users should be able to delete users from the database.
11. The program should display data from a Database in a table view format:
    a. This format should be able to be sorted via certain parameters – for example with the festivals via cheapest cost to most expensive – providing two options for both day ticket type and the full ticket type.
    b. This data should be output to be viewed by the user in a table model or otherwise for best viewing of information.
12. Provide service to view the "profile" of different festivals – provide the option of all festivals to be chosen via a menu – perhaps a dropdown list design – and then display the overview of that festival.
    a. Profile would fetch the name of the festival and display it as the title of the window
    b. Profile would also fetch the cost in GBP.
    c. Profile would fetch the start and end dates.
    d. Festival would display the date range on a calendar view.
    e. Profile would display all this data to the user in a suitable selection of widgets for the GUI .
13. The program should also provide an interface for location services.
    a. Looking up a postcode, an origin, to return a formatted address.
    b. Using Festival Postcode stored in database to return formatted addresses
    c. Finding the distance between the origin and the festival addresses using a Google Maps API.
    d. Dropping all of these locations in markers onto a map of the nearest locations.
    e. Displaying all of this information found for the user in a clean format on the GUI.
14. The Program would supply a means to directly view the dates of festivals.
    a. Provide a selection of festivals.
    b. Out the name of the festival.
    c. Output the start and end dates of this festival
    d. Display these dates on a calendar widget.
15. The program should accurately fetch data from the database with SQL queries – for example fetching all postcodes of festivals etc.
16. The Program should allow the user to narrow down their search via music acts performing at certain festivals.
    a. This should fetch and display all the performing acts.
    b. Allow the user to search these acts specifically with a search bar for their selection or favourite.
    c. The Program should then display the festivals for which this act is performing.
    d. It should provide the names of all of the multiple festivals, if applicable.
17. The Program should have the functionality to perform Create, Read, Update, Delete (CRUD) functionality with the databases in real time, this should include:
    a. **C**reate – creating a new entry into our database. This could be employed via the Register form, administrator-level edits for the festival databases (if we expand this program to festivals beyond the Festival Republic brand) or for any ticket purchases
    b. **R**ead – read the data from the database. This is incorporated with the viewing of any data from the databases, such as when the Festivals are researched based on information in their databases.

c. **U**pdate – Updating tables in the databases and committing the changes would be necessary to amend account information, change the location of festivals and update festival acts each year.
d. **D**elete – Deleting data from the database would be necessary to delete unnecessary date, amend address information and delete user accounts. (For example when a user would like to cancel or delete an account from our service, we should construct a window and option for that, which executes the suitable query to drop the user account from all the tables.

Reliability

18. To Validate all available inputs – whether they suit the form the program needs, or there is any date at all.
19. Using Regular Expressions to validate the username and password strength upon sign up
    a. Username should be checked for matching regular expressions for basic checks – ie a length between 6-15, only alphanumeric characters or underscores.
    b. Password should be checked for matching regular expressions for basic password strength (7-15 characters long, at least one uppercase letter, one lowercase letter and a number.)
20. Validate inputs for matching passwords in the register form.
21. Validate inputs for valid email addresses, usernames already in use etc.
22. Validate for empty entries without any inputs at all.
23. Make sure that an admin can't edit their own user data.
24. Make sure that a postcode is valid using regular expressions to validate it.

Maintainability

25. At all points in the program where the code isn't easy to understand or there is a loop, selection or procedure, use comprehensive comments to demonstrate for the next maintainer of the program what is involved and what happens at different points, so they can maintain the program accordingly.
26. Meaningful variable names have been used, along with Hungarian notation to specify the usual/initial filetype of variables or data structures.
27. Consistency in code writing, keeping methods as short as possible and making separate modules for aspects of the code will keep it easy to maintain for the next user.

At the point of reflection on our objectives, we will review whether we have met these through testing and marking them off as applicable. Using tests for correct input and output signals, we will determine whether we have met the objectives of our end-user, and we will meet with them and discuss the implications of our results.

Research for Product Development Needs

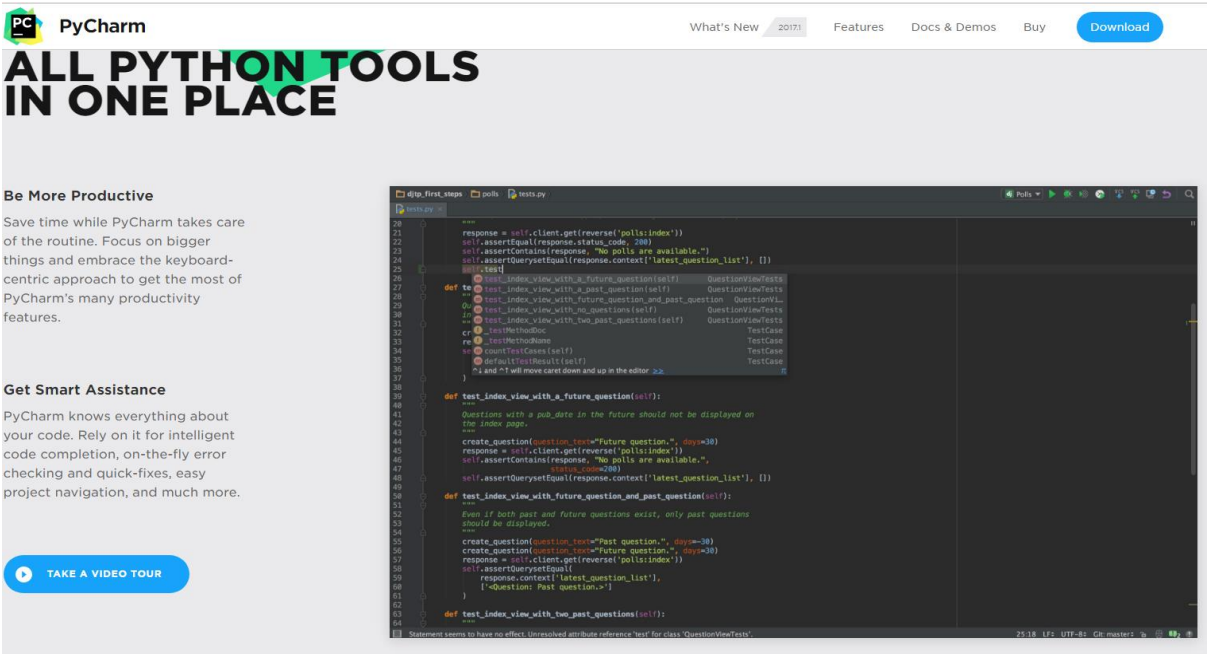
For this particular project, I knew I had several particular needs that would require a particular set of libraries and commands. After much research, I came to two clear solutions for the easiest development of my particular product – the JetBrains Python IDE "Pycharm" and the python package WinPython, as the latter contains all the PyQt libraries I required along with the Qt Designer package itself, and Pycharm provided a centralised way to program and display both my data source, code, multiple different modules, text files and folders all at once whilst making sure I keep my code

neat and orderly. I also came to the conclusion that I would need map functionality – I would need to apply for a Google Maps API key in order to use their programming matrix.

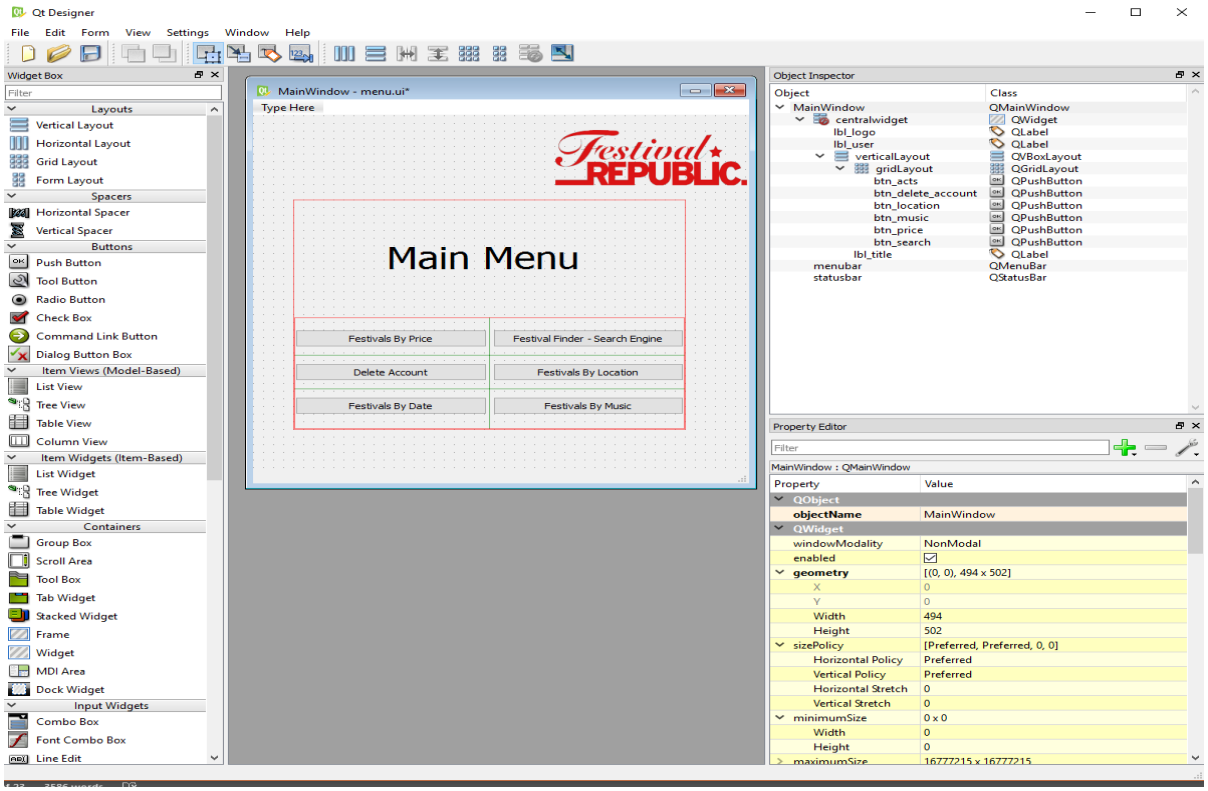
Development Needs – IDE and Packages (Pycharm, WinPython and Qt Designer)

Here is the package I used – it provided a wide array of packages and plugins that I required for my project to function.

Here is the IDE I decided on – due to the centralised view of all the data and files at once it would be the most applicable IDE to my problem.

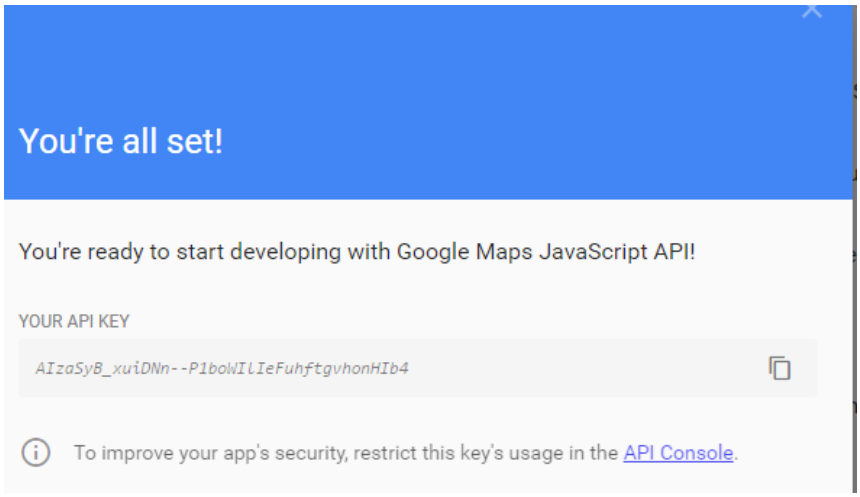


In order to design the GUI, I chose the Qt-Designer package to customise and code the GUI's front end whilst handling the back end of the GUI on the python side. This particular product seamlessly works with the python language, in an Object Orientated Code manner. In my program I used different classes to signify and code different GUI windows.



Requesting an API key from Google

One of the sections of my program would offer festival results based on location and distance from the user. For the section of my program that incorporates Google Maps to find distance to festivals from the user's current location, I will need to integrate the Google Maps API in my project. Here is a copy of my request:



Product Requirements (Software and Hardware Requirements)

The system requirements for my service are as follows, due to the nature of PyCharm and the associated programs behind the product.

- Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
- 1 GB RAM minimum
- 2 GB RAM recommended
- 1400x900 minimum screen resolution (Due to the size of the largest window
- Python 2.4 or higher, Jython, PyPy or IronPython (Preferably Pycharm as this includes the other requirements – packages such as QtDesigner.
- The QtDesigner framework for python.
- Fully Functioning Keyboard and Mouse
- Intel Core 2 Duo processor or better – a clock rate above 1Ghz is preferable.
- A Working Internet Connection – the program requires connection to Google Maps to run.

Design

Problem Decomposition

Breaking the Problem Down into different manageable parts

1. Creating the database structure in SQL-Lite
2. Creating the festivals database
    a. Researching the relevant data
    b. Adding this data to the database itself (festID, Name, PostCode, StartDate, EndDate, DayCost, FullCost)
    c. Creating a primary key (festID)
    d. Making festID both unique and autoincremented
3. Creating the users database
    a. Adding in some initial data to test log on before adding more with a register function of the program.
4. Creating the acts database
    a. Researching which acts are playing which festivals.
    b. Adding this data to the database (Act Name, festId)
    c. Making festID a foreign key and joining it to festivals festID column
5. Designing the GUI of the separate windows for the program:
    a. Designing, creating, designating, and aligning the widgets for the Log in Window.
    b. Designing, creating, designating, and aligning the widgets for the Menu Window.
    c. Designing, creating, designating, and aligning the widgets for the Admin Menu Window.
    d. Designing, creating, designating, and aligning the widgets for the Register an Account Window.
    e. Designing, creating, designating, and aligning the widgets for the Forgot Password Window.
    f. Designing, creating, designating, and aligning the widgets for the Buy Tickets Window.
    g. Designing, creating, designating, and aligning the widgets for the Sort By Artist Window.
    h. Designing, creating, designating, and aligning the widgets for Date Tracking Window.
    i. Designing, creating, designating, and aligning the widgets for the Sort By Cost Window.
    j. Designing, creating, designating, and aligning the widgets for the Delete Account Window.
    k. Designing, creating, designating, and aligning the widgets for the Alert (Ask for Postcode) Window.
    l. Designing, creating, designating, and aligning the widgets for the Location finder and distance measurer Window.
    m. Designing, creating, designating, and aligning the widgets for the Profile Window.
6. Coding for the separate windows of the program:
    a. Programming the operations, functions, widgets, and calculations behind the Log In window.
    b. Programming the operations, functions, widgets, and calculations behind the Register window.

c. Programming the operations, functions, widgets, and calculations behind the Forgot Password window.
d. Programming the operations, functions, widgets, and calculations behind the Location window.
e. Programming the operations, functions, widgets, and calculations behind the Alert (Find your postcode) window.
f. Programming the operations, functions, widgets, and calculations behind Delete Account window.
g. Programming the operations, functions, widgets, and calculations behind the Admin Menu window.
h. Programming the operations, functions, widgets, and calculations behind the Date Finder window.
i. Programming the operations, functions, widgets, and calculations behind the Festival Profile window.
j. Programming the operations, functions, widgets, and calculations behind the Sort By Cost window.
k. Programming the operations, functions, widgets, and calculations behind the Find Acts at Festival window.
l. Programming the operations, functions, widgets, and calculations behind Buy Tickets window.
m. Programming the operations, functions, widgets, and calculations behind Main Menu Window.

7. Carry out validation on all code that involves input if necessary and not already validated.

Problem Decomposition – Diagrams

main.tbl_festivals | main.tbl_users | festivalFinder.py | googlemap.py

7 rows

<Filter criteria>

	userID	user	pass	email	perm_lvl
1	1	a	0cc175b9c0f1b6a831c399e269772661	majesticseabass@gmail.com	2
2	2	admin	6f2457b608622214f753c8522bf63a38	cyberhylian@gmail.com	1
3	5	complex	0d91a55ac531df50ede54f22b23ed9ec	philnsjspam@talktalk.net	1
4	8	jdsalinger	e7d8993bf6de68a2d4e3a8a24e51f4fd	ltmattwelly@gmail.com	1
5	9	asdfasdf	7e274f7903305885b210ead4b62557fe	a@a.com	1
6	10	jessica	d794fccc1926643b90944533d7baf0e2	jess@4818.com	1
7	11	George	30d2e8942cca845bc66236b4de28ae07	georgewellingtonwork@gmail.com	1

main.tbl_festivals | main.tbl_acts | main.tbl_users | festivalFinder.py | googlemap.py

6 rows

Tab-se...d (TSV)

<Filter criteria>

	festID ▲1	Name	Postcode	StartDate	EndDate	Cost	DayCost
1	1	Reading Festival	RG1 8EQ	25/08	27/08	213	72
2	2	Leeds Festival	LS23 6ND	25/08	27/08	213	72
3	3	Latitude Festival	NR34 8AQ	13/07	17/07	197.5	84.5
4	4	Download Festival	DE74 2RP	09/07	11/07	205	83
5	5	Wireless Festival	N4 1EE	08/07	10/07	210	62
6	6	Community Festival	N4 1EF	01/07	01/07	40.3	40.3

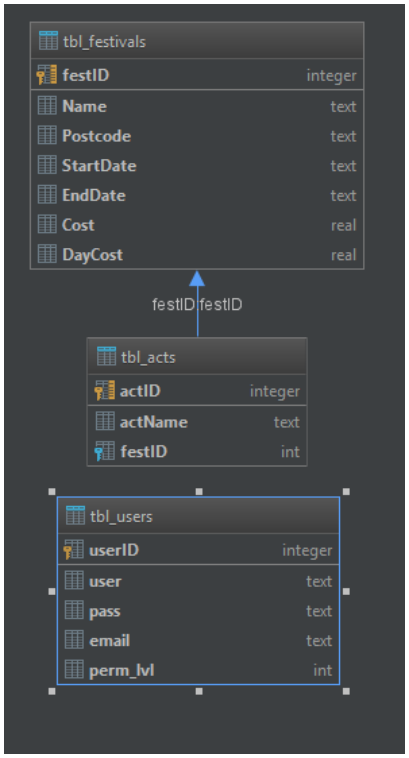
Above are the different tables once made with information entered, below is one more table for the acts and the ERD diagram for the tables and the data types for each column.

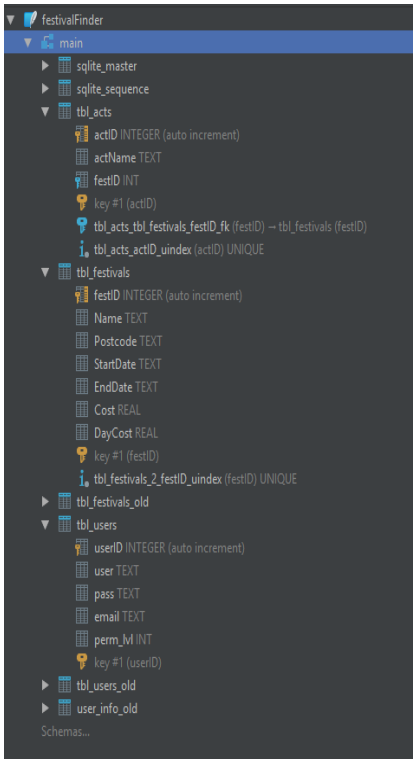
main.tbl_festivals | main.tbl_acts | main.tbl_users | festivalFin

33 rows

<Filter criteria>

	actID	actName	festID
1	1	Eminem	1
2	2	Two Door Cinema Club	6
3	3	Catfish and the Bottlemen	6
4	4	The Wombats	6
5	5	Slaves	6
6	6	Nothing But Thieves	6
7	7	Eminem	2
8	8	Chance the Rapper	5
9	9	Bryson Tiller	5
10	10	G-Eazy	5
11	11	Fetty Wap	5
12	12	Skepta	5
13	13	Travis Scott	5
14	14	Rae Sremmurd	5
15	15	The Weeknd	5
16	16	Nas	5
17	17	Tory Lanez	5
18	18	The 1975	3
19	19	Mumford and Sons	3
20	20	Baaba Maal	3
21	21	Glass Animals	3
22	22	Fleet Foxes	3
23	23	John Cale	3
24	24	Goldfrapp	3
25	25	Kasabian	2
26	26	Kasabian	1
27	27	Muse	1
28	28	Muse	2
29	29	Vant	1
30	30	Vant	2
31	31	System of a Down	4
32	32	Biffy Clyro	4
33	33	Aerosmith	4





Pseudocode

Login class/window.

```
START
class FestivalLogin # uses object# wdw_login ##
    new btn_login
    new btn_close
    new btn_forgot
    new btn_register
    new inp_user
    new inp_password
    new out_txt_msg

    public PROCEDURE new()
        super.new()
        if super.btn_forgot.pressed() then
            forgot()
        elseif super.btn_register.pressed() then
            register()
        elseif super.btn_login.pressed() then
            login()
        elseif super.btn_close.pressed() then
            HALT
        endif

    private PROCEDURE login() inherits wdw_login #private as details are sensitive #
        str_user = inp_user.text
        str_pass = inp_password.text
        str_hashed_pass = str_pass.hash(md5)
        connection = openRead("tbl_users")
        users = connection.read_table()
        for i=0 to (LENGTH(users) - 1)
            if users[i].user = str_user and users[i].pass = str_pass then
                result = users[i].user + users[i].pass + users[i].perm_lvl
            endif
        next i
        if LENGTH(result) > 0 then
            FestivalMenu.fillgui(wdw_menu, str_user )
            FestivalDelete.fillgui( wdw_delete)
            if result[2] == 2 then  #checks if perm_lvl from tbl_users is admin level #
                wdw_menu.int_perm = 1
                wdw_admin.show()
            else
                wdw_menu.show()
            endif
            wdw_login.hide()
        else
            out_txt_msg.setText("Wrong username or password")
        endif
        connection.close()

    public PROCEDURE register() inherits wdw_login
        wdw_register.show()
        wdw_login.hide()

    public PROCEDURE forgot() inherits wdw_login
        wdw_forgot.show()
        wdw_login.hide()

HALT
```

Register class/window:

```
START
class FestivalRegister # uses object# wdw_register ##
    new btn_submit
    new btn_close
    new inp_user
    new inp_password
    new inp_passw2
    new inp_email
    new out_txt_msg

    public PROCEDURE new()
        super.new()
        if super.btn_submit.pressed() then
            submit()
        elseif super.btn_close.pressed() then
            HALT

    private PROCEDURE submit() inherits wdw_register #private as details are sensitive #
        str_user = inp_user.text
        str_pass = inp_password.text
        str_pass2 = inp_passw2.text
        str_email = inp_email.text
        str_hashed_pass = str_pass.hash(md5)
        connection = openRead("tbl_users")
        users = connection.read_table()
        for i=0 to ( LENGTH(users) - 1)
            if users[i].email = str_email then
                result = users[i].user + users[i].pass +  users[i].email + users[i].perm_lvl
            endif
        next i
        if LENGTH(result) > 0 then
            out_txt_msg.setText("email already in use")
        else
            if str_pass matches regex(password_pattern) and str_user matches regex(user_pattern):
            #if the password and user matches the regex form#
            # user starts with alphanumeric, can only contain alphanumeric or underscore with length 4-12#
            # password is (6-12 chars, at least one lower,upper and number and no specials) #
                if str_pass = str_pass2 then
                    out_txt_msg.setText("password match - account created")
                    connection(Insert into tbl_users user, pass email values (str_user, str_pass, str_email))
                    connection.commit()
                else
                    out_txt_msg.setText("passwords don't match")
                endif
            else
                out_txt_msg.setText("Username or password don't match regular form")
            endif
        connection.close()

HALT
```

Forgot Password class/window.

```
START
class FestivalForgot # uses object# wdw_forgot ##
    new btn_submit
    new btn_close
    new inp_email
    new out_txt_msg

    public PROCEDURE new()
        super.new()
        if super.btn_submit.pressed() then
            submit()
        elseif super.btn_close.pressed() then
            HALT
        endif

    private PROCEDURE submit() inherits wdw_forgot #private as details are sensitive #
        str_email = inp_email.text
        connection = openRead("tbl_users")
        users = connection.read_write_table()
        for i=0 to ( LENGTH(users) - 1)
            if users[i].email = str_email then
                result = users[i].email
            endif
        next i
        if LENGTH(result) > 0 then
            out_txt_msg.setText("Valid Email ")
            int_new_pass = RANDOM_INTEGER_BETWEEN(1000, 1000000)
            str_hashed_pass = STRING(int_new_pass).hash(md5)
            for i=0 to ( LENGTH(users) - 1)
                if users[i].email = str_email then #updates password#
                    users[i].pass = str_hashed_pass
                    str_name = users[i].name
                endif
            next i
            emailpass.send( str_email , str_name , int_new_pass )
        else
            out_txt_msg.setText("no email registered")
        endif
        connection.close()

HALT

```

Location class/window.

```
START
class FestivalLocation # uses object# wdw_location ##
    new combobox_festivals
    new btn_buy
    new btn_back
    new out_title_festival_name
    new web_map
    new out_origin
    new out_destinations

    public PROCEDURE new()
        super.new()
        if super.get_distance.pressed() then
            wdw_alert.show()
            wdw_location.hide()
        elseif super.btn_back.pressed() then
            wdw_location.hide()
            wdw_home.show()
        endif
        connection = openRead("tbl_festivals")
        festivals = connection.read_table()
        for i=0 to LENGTH(festivals)
            postcodes = postcodes + festivals[i].postcode
            names = names + festivals[i].nam
        next i
        for i=0 to LENGTH(Postcodes)
            locations.append(names[i], postcodes[i]
        next i
        html = googlemap.plot(locations, len(locations), 0) #sends my googlemap plotter a locations array, along with
        how many locations there are and a signal to if the last location is home location (0 means its just a festival #
        web_map.setHtml(html)

    public PROCEDURE alert inherits wdw_location
        wdw_alert.show() # alert calls back to get_distance with a users postcode #
        wdw_location.hide()

    public PROCEDURE get_distance(str_home_postcode) inherits wdw_location
        origin = str_home_postcode
        addresses_and_distances = request_geocache_distance_from_origin(origin, postcodes)
        out_origin.setText(addresses_and_distances.originaddress)
        out_destinations.setText(addresses_and_distances.destination_address_and_distance)
        locations = address_and_distances.addresses
        html = googlemap.plot(locations, len(locations), 1) # as array locations includes home, we include is home
        as a value as one, so the map will plot a perimeter around it #

HALT
```

Alert class/window.

```
START
class FestivalAlert # uses object# wdw_alert ##
    new btn_ok
    new btn_back
    new inp_postcode

    public PROCEDURE new()
        super.new()
        if super.ok.pressed() then
            ok()
        elseif super.btn_back.pressed() then
            wdw_alert.hide()
            wdw_location.show()
        endif

    public PROCEDURE ok inherits wdw_alert
        str_home = inp_postcode.text()
        wdw_alert.hide()
        wdw_location.show()
        FestivalLocation.get_distance( str_home)

HALT
```

Delete Account class/window.

```
START
class FestivalDelete # uses object# wdw_delete ##
    new btn_delete
    new btn_back
    new inp_artist
    new out_acts
    new out_msg

    public PROCEDURE new()
        super.new()
        if super.btn_delete.pressed() then
            delete()
        elseif super.btn_back.pressed() then
            wdw_delete.hide()
            wdw_home.show()
        endif

    public PROCEDURE delete inherits wdw_delete
        connection = openRead("tbl_users")
        users = connection.read_table()
        str_user = wdw_login.user
        str_pass = wdw_login.pass
        for i=0 to LENGTH(users)
            if users[i].user == str_user and users[i].pass = str_pass then
                remove(users[i])
            endif
        next i
        connection.close()
        out_festivals.setText("Account registered to" + str_user + "deleted.")

HALT
```

Admin Menu Password class/window.

```
START
class FestivalAdmin # uses object# wdw_admin ##
    new btn_back
    new btn_delete
    new btn_toggle_admin
    new table_widget_users

    public PROCEDURE new()
        super.new()
        refresh()
        if super.btn_menu.pressed() then
            menu()
        elseif super.btn_delete.pressed() then
            delete()
        elseif super.btn_admin.pressed() then
            admin()
        elseif super.table_widget_users.pressed() then
            select_user()
        elseif
            HALT

    public PROCEDURE select_user() inherits wdw_admin
        refresh()
        selected_user = row.pressed()
        return selected_user

    private PROCEDURE delete() inherits wdw_admin #private as details are sensitive #
        connection = openRead("tbl_users")
        users = connection.read_table()
        for i=0 to ( LENGTH(users) - 1)
            if users[i].user = selected_user[1] then
                remove(users[i])
            endif
        next i
        refresh()
        connection.close()

    private PROCEDURE admin() inherits wdw_admin #private as details are sensitive #
        connection = openRead("tbl_users")
        if selected_user[4] == 1 then
            perm_lvl_new = int(selected_user[4]) + 1
        else
            perm_lvl_new = int(selected_user[4]) - 1
        users = connection.read_table()
        for i=0 to ( LENGTH(users) - 1)
            if users[i].user = selected_user[1] then
                users[i].perm_lvl = perm_lvl_new
            endif
        next i
        selected_user = ""
        refresh()
        connection.close()

    public PROCEDURE menu() inherits wdw_admin
        wdw_admin.hide()
        wdw_menu.show()

    private PROCEDURE refresh() inherits wdw_admin
        connection = openRead("tbl_users")
        users = connection.read_table()
        for i=0 to LENGTH(users): # for every column in table #
            for j=0 to LENGTH(users[0]): # for every row in table #
                table_widget_users.setitem(users[i][j]) #sets each item in order#
            next j
        next i
        connection.close()
        selected_user = ""

HALT
```

Date class/window.

```
START
class FestivalDate # uses object# wdw_date ##
    new combobox_festivals
    new str_fest_name
    new btn_back
    new cal_calendar
    new out_message


    public PROCEDURE new()
        super.new()
        refresh()
        connection = openRead("tbl_festivals")
        festivals = connection.read_table()
        for i=0 to LENGTH(festivals)
            names = names + festivals[i]
        next i
        for i in names:
            combobox_festivals.additem(i)
        next i
        if super.combobox_festivals.pressed() then
            str_fest_name = combobox_festivals.currentText()
            fill_gui()
        elseif super.btn_back.pressed() then
            back()
        endif

    public PROCEDURE back() inherits wdw_date
        wdw_date.hide()
        wdw_menu.show()

    public PROCEDURE fill_gui() inherits wdw_date
        connection = openRead("tbl_festivals")
        festivals = connection.read_table()
        for i=0 to LENGTH(festivals)
            if festivals[i].Name = str_fest_name then
                datetime_end_date = festivals[i].EndDate
                datetime_start_date = festivals[i].StartDate
            endif
            cal_calendar.setDate(datetime_start_date)
            out_message.setText("Dates from " , datetime_start_date, "to ", datetime_end_date)
            connection.close()

HALT
```

Profile class/window.

```
START
class FestivalProfile # uses object# wdw_profile ##
    new combobox_festivals
    new btn_buy
    new btn_back
    new out_title_festival_name
    new cal_calendar
    new out_cost
    new out_daycost
    new out_address
    new out_acts
    new out_dates


    public PROCEDURE new()
        super.new()
        if super.btn_buy.pressed() then
            wdw_buy.show()
            wdw_profile.hide()
        elseif super.btn_back.pressed() then
            wdw_profile.hide()
            wdw_menu.show()
        endif

    public PROCEDURE fill_gui(str_festival_name) inherits wdw_profile
        out_title_festival_name = str_festival_name
        connection = openRead("tbl_festivals")
        festivals = connection.read_table()
        for i=0 to LENGTH(festivals)
            if festivals[i].Name = str_festival_name then
                results = festivals[i]
            endif
        next i
        datetime_start_date = results[3]
        datetime_end_date = results[4]
        str_postcode = results[2]
        str_cost = results[5]
        str_daycost = results[6]
        str_festival_id = results[0]
        cal_calendar.setDate(datetime_start_date)
        address = request_geocache_post_address_format(str_postcode)
        out_address.setText(address)
        out_daycost.setText(daycost)
        out_cost.setText(cost)
        connection.close()
        connection = openRead("tbl_acts")
        acts = connection.read_table()
        for i=0 to LENGTH(acts)
            if acts[i].festID = str_festival_id then
                str_acts = str_acts + acts[i].Name
            endif
        out_acts.setText(str_acts)
        out_dates.setText("Dates from " , datetime_start_date, "to ", datetime_end_date)

HALT
```

Cost class/window.

```
START
class FestivalCost # uses object# wdw_cost ##
    new btn_day
    new btn_back
    new btn_full
    new table_widget_festivals

    public PROCEDURE new()
        super.new()
        if super.day.pressed() then
            day()
        elseif super.full.pressed() then
            full()
        elseif super.btn_back.pressed() then
            wdw_cost.hide()
            wdw_home.show()
        endif

    public PROCEDURE full inherits wdw_cost
        connection = openRead("tbl_festivals")
        unordered = connection.read_table()
        highest = 0
        for i=0 to LENGTH(unordered)
            cost = cost + unordered[i].cost
        next i
        cost.sort_by_size()
        list_festivals_ordered = []
        for j=0 to LENGTH(cost)
            for i=0 to LENGTH(unordered)
                if unordered[i].cost == cost[j]
                    list_festivals_ordered.append(unordered[i]) # fetches cost for day ticket and orderes that list #
                endif
            next i
        next j
        for i in range LENGTH(list_festivals_ordered): # for every column in table #
            for j in range LENGTH(list_festivals_ordered[0]): # for every row in table #
                table_widget_festivals.setitem(list_festivals_ordered[i][j]) #sets each item in order#
            next j
        next i
        connection.close()

    public PROCEDURE day inherits wdw_cost
        connection = openRead("tbl_festivals")
        unordered = connection.read_table()
        highest = 0
        for i=0 to LENGTH(unordered)
            cost.append(unordered[i].daycost)
        next i
        cost.sort_by_size()
        list_festivals_ordered = []
        for j=0 to LENGTH(cost)
            for i=0 to LENGTH(unordered)
                if unordered[i].daycost == cost[j]
                    list_festivals_ordered.append(unordered[i]) # fetches cost for day ticket and orderes that list #
                endif
            next i
        next j
        for i in range LENGTH(list_festivals_ordered): # for every column in table #
            for j in range LENGTH(list_festivals_ordered[0]): # for every row in table #
                table_widget_festivals.setitem(list_festivals_ordered[i][j]) #sets each item in order#
            next j
        next i
        connection.close()

HALT
```

Music class/window

```
START
class FestivalMusic # uses object# wdw_music ##
    new btn_go
    new btn_back
    new inp_artist
    new out_acts
    new out_festivals

    public PROCEDURE new()
        super.new()
        if super.btn_go.pressed() then
            search()
        elseif super.btn_back.pressed() then
            wdw_music.hide()
            wdw_home.show()
        endif
        connection = openRead("tbl_acts")
        acts = connection.read_table()
        for i in acts
            display = display + newline + i
        next i
        out_acts.setText(display)
        connection.close() # sets the possible acts to be shown in the output acts box #

    public PROCEDURE search inherits wdw_music
        str_artist = inp_artist.text()
        connection = openRead("tbl_acts")
        acts = connection.read_table()
        for i=0 to LENGTH(acts)
            if acts[i].name == str_artist then
                int_id = acts[i].festID
            endif
        next i
        connection.close()
        connection = openRead("tbl_festivals")
        festivals = connection.read_table()
        for i=0 to LENGTH(festivals)
            if festivals[i].festID = int_id then
                display = display + newline + festivals[i].Name
            endif
        next i
        out_festivals.setText(display)
        connection.close()

HALT
```

Buy Tickets class/window.

```
START
class FestivalBuy # uses object# wdw_buy ##
    new btn_back
    new web_tickets


    public PROCEDURE new() inherits wdw_buy
        super.new()
        if super.btn_back.pressed() then
            wdw_buy.hide()
            wdw_home.show()
        endif
        web_tickets.setUrl("http://www.ticketmaster.co.uk/search?tm_link=tm_header_search&user_input=&q="
                           + wdw_location.fest_name)

HALT
```

Main Menu class/window.

```
START
class FestivalMenu # uses object# wdw_menu ##
    new btn_logout
    new btn_admin
    new btn_location
    new btn_price
    new btn_delete
    new btn_date
    new btn_acts
    new combobox_dropdown_festivals
    new out_txt_msg

    public PROCEDURE new()
        super.new()
        if super.btn_location.pressed() then
            wdw_menu.hide()
            wdw_location.show()
        elseif super.btn_admin.pressed() then
            if int_perm == 1 then
                wdw_menu.hide()
                wdw_admin.show()
            else
                out_txt_msg.setText("Admin Menu for Admin level users only")
        elseif super.btn_price.pressed() then
            wdw_menu.hide()
            wdw_price.show()
        elseif super.btn_delete.pressed() then
            wdw_menu.hide()
            wdw_delete.show()
        elseif super.btn_date.pressed() then
            wdw_menu.hide()
            wdw_date.show()
        elseif super.btn_acts.pressed() then
            wdw_menu.hide()
            wdw_acts.show()
        elseif super.combobox_dropdown_festivals.pressed() then
            str_fest_name = combobox_dropdown_festivals.currentText()
            FestivalProfile.fill_gui(str_fest_name)
            wdw_menu.hide()
            wdw_profile.show()
        elseif super.btn_logout.pressed() then
            HALT
        endif
        int_perm = 0 #permission default at initialisation is 0, changed to 1 if admin at login point #

HALT
```

The other modules of the program were not written in object orientated code. They are both single functions that serve one purpose for the program – they carry out single functions, plotting a map in html and sending an email with python's SMTP library.

Email module pseudocode:

```
START
FUNCTION send(recipient, name, reset_password)
    user = #email username#
    password = #email password of automated account #
    to = [recipient[
    body = " Dear " + name "," + newline +
            "This is an automated message - your new password is" + reset_password
    msg = "Festival Finder: Reset your Password" # the subject# + newline + newline + body # the body #
    try
        mailserver.connect(gmail.com)
        server.login(user, password)
        server.sendmail(user, to, msg)
        server.close()
    except
        print "Something went wrong"
HALT
```

Google Mapper pseudocode:

```
START

FUNCTION plot (locations, ishome)
    # This is HTML Code that is returned to FestivalFinder.py #
    code = """

    array_locations = [locations]   # this is transferred from the festivalfinder module,
                                      in the format of [Name, [Latitude, Longitude]]
                                      for each different location to be plotted #

    Map = new GoogleMaps.Map()
    Map.SetCentre("England")
    Map.SetType("Hybrid")
    Map.SetTypeControl("False")
    Map.SetScaleControl("False")
    Map.SetStreetViewControl("False")

    Marker_Desription = new GoogleMaps.InfoWindow()
    for i in range(0, LENGTH(array_locations) + ishome)
        Marker = new GoogleMaps.Maker()
        Marker.SetAnimation("bounce")
        Marker.SetPosition(locations[i][1]) # sets position to the lat/long of current location being set #
        first_letter = locations[i][0][0]   # fetches the first letter of the name of the festival #
        Marker.SetLabel(first_letter)       # makes the marker have the first letter of its festival #
        Marker.SetMap(Map)
    if ishome == -1 then
        HomeMarker = new GoogleMaps.Maker()
        HomeMarker.SetAnimation("bounce")
        HomeMarker.SetPosition(locations[ LENGTH(array_locations) - 1][1])  # sets position to the lat/long
        HomeMarker.SetMap(Map)
        HomeMarker.SetIcon(GreenMarker)

        fortymiles = new GoogleMaps.Circle() # sets a forty mile radius around the homemarker #
        fortymiles.setCentre(HomeMarker)
        fortymiles.setOpacity(0.1)
        fortymiles.setfillcolour(#7ec0ee#)
        fortymiles.setradius("40mi")
        fortymiles.setmap(Map)
    endif

    if marker.pressed() or HomeMarker.pressed() then # Shows a descriptive name of festival for #
        Marker_Desription.setText(locations[i][0])   # each marker pressed #
    endif

    """
    return code

HALT
```

Explain and Justify the Structure of the Solution

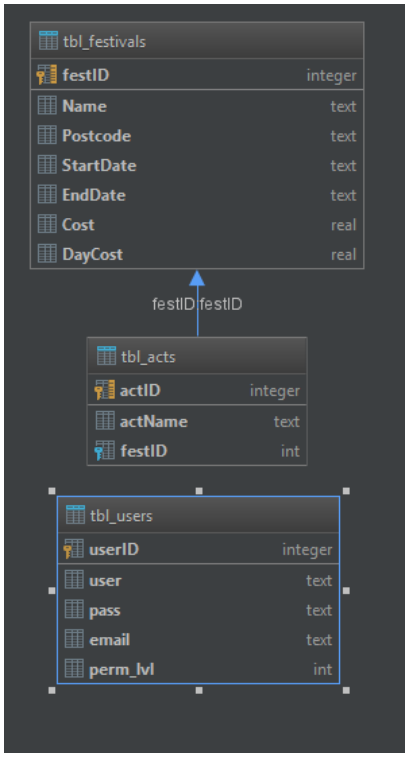
Algorithms

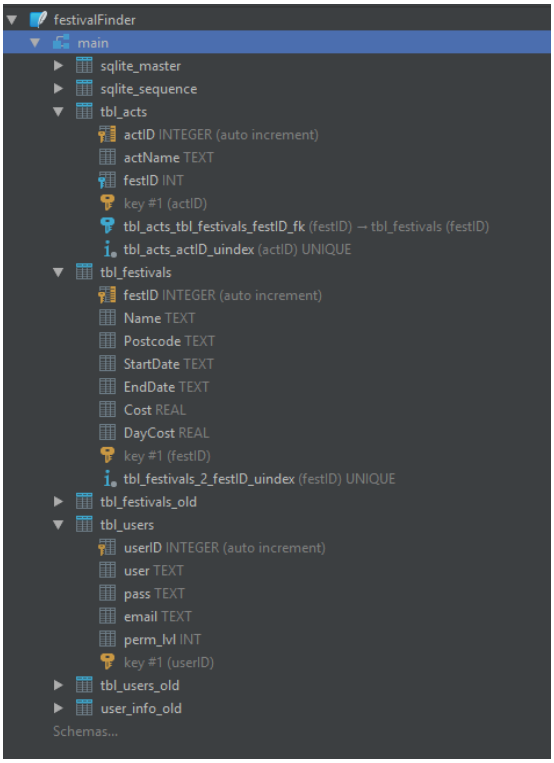
Algorithms to be used —

1. Shortest Path Algorithm (Google Maps Location Distance Finder)
2. Navigating a 3D Array (Array of Locations, which contain a Name, and an array of coordinates)
3. Navigating a 2D Array
4. Hashing Algorithms (For Password adding and matching)
5. Regular Expressions (Checking passwords/ usernames during registration for strength)
6. SQL Queries (Create, Read, Update, Delete)
7. Email – Accessing and sending emails using SMTP and associated libraries via connecting to the smtp server.
8. Google Maps API plotting
9. GUI control (Widgets etc)
10. String Concatenation, splitting, selection.
11. Casting variables
12. Filling a table widget with nested for loops
13. JSON reading/using/climbing
14. Geocaching requests
15. Inner Join Queries – Finding related tables and outputting the data.
16. Calculating distance user is willing to travel and displaying that on a map format in a perimeter circle. (using a constant for google radius mile unit)
17. Regular Expressions (Using the UK Government Cabinet's formula for regular postcode format to enforce rules)
18. Sorting data via alphabetically or numerically (Sort By Queries)
19. Type Casting
20. Filling a Table Widget from information in a 2D Array

UML Diagrams

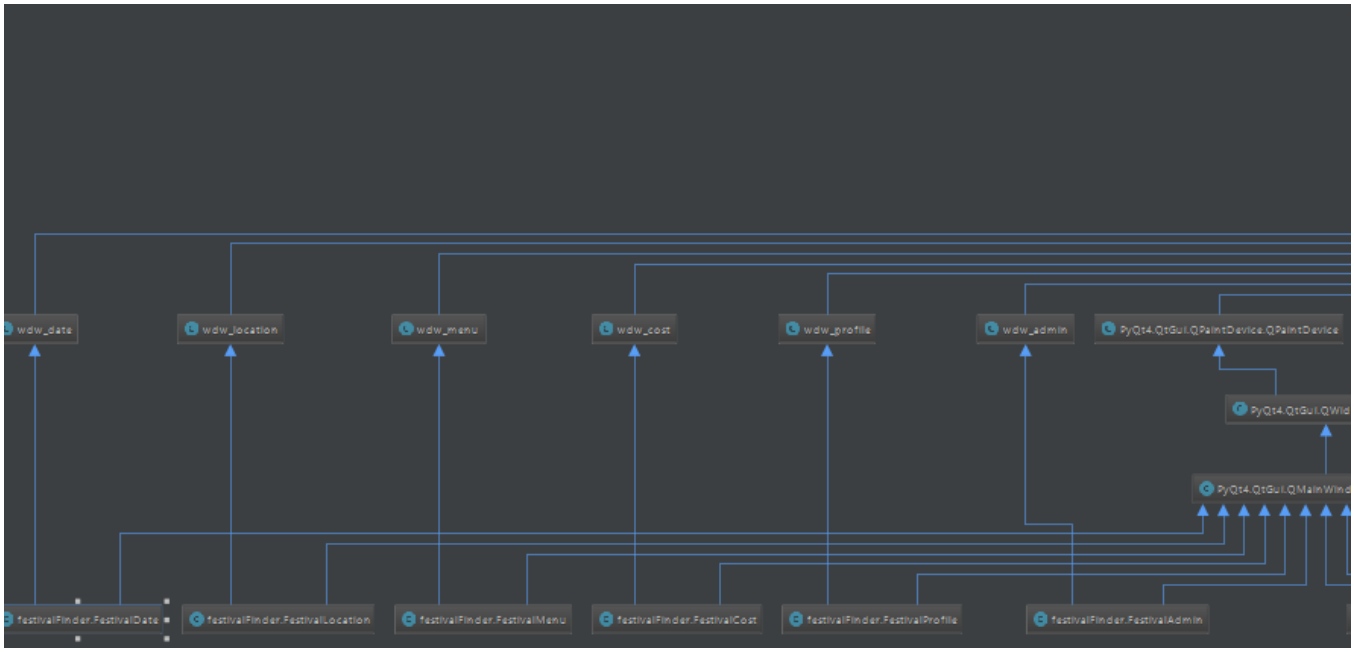
Overview of Tables



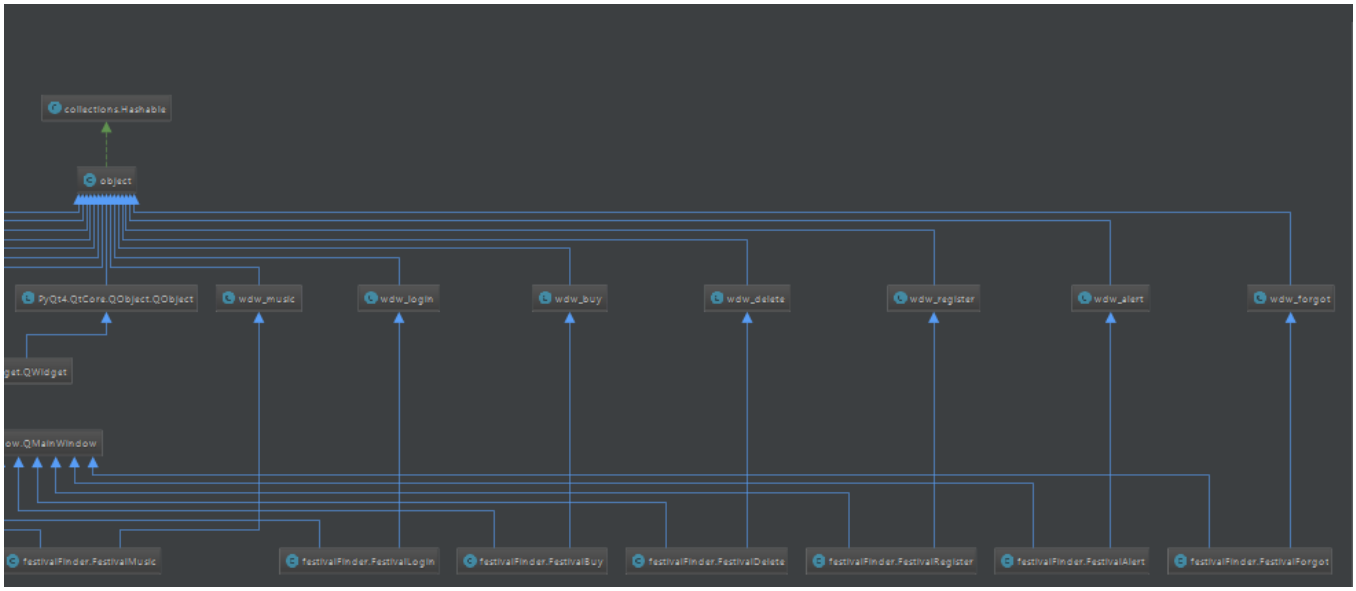


Overview of Classes (Without Methods)

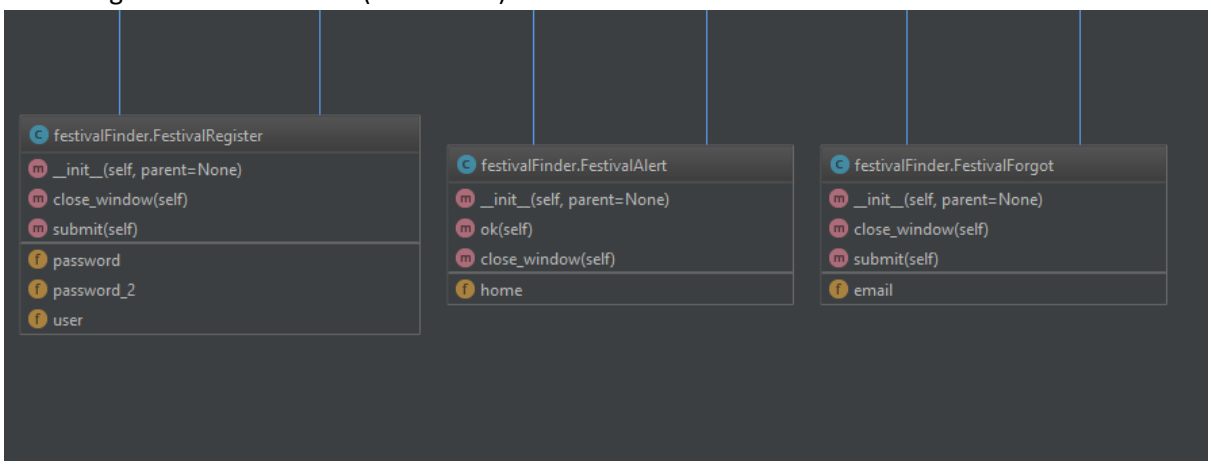
1st Side

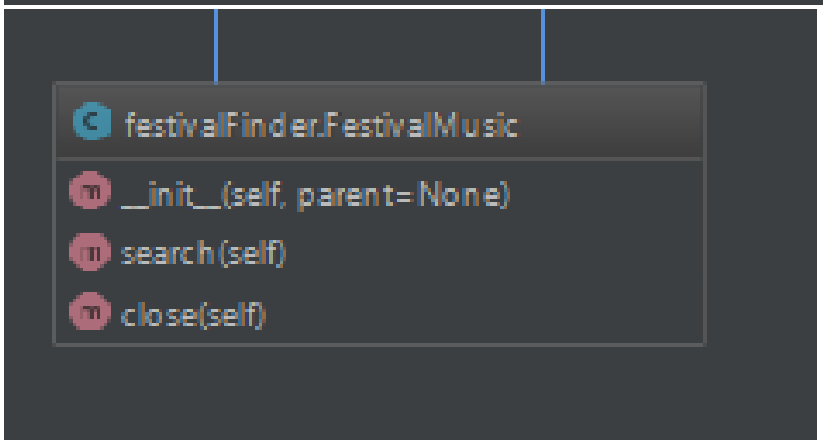


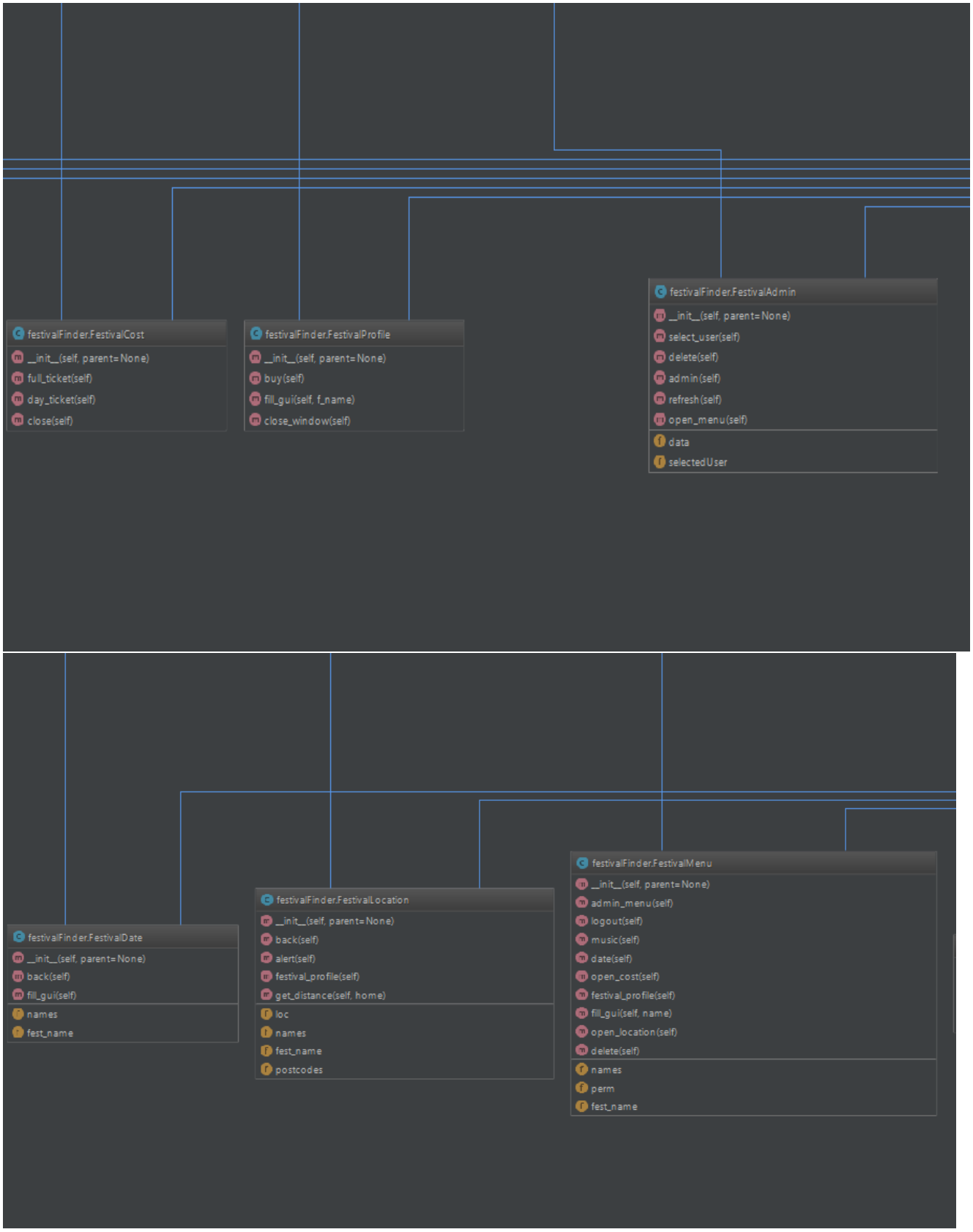
2nd Side (Without Methods)

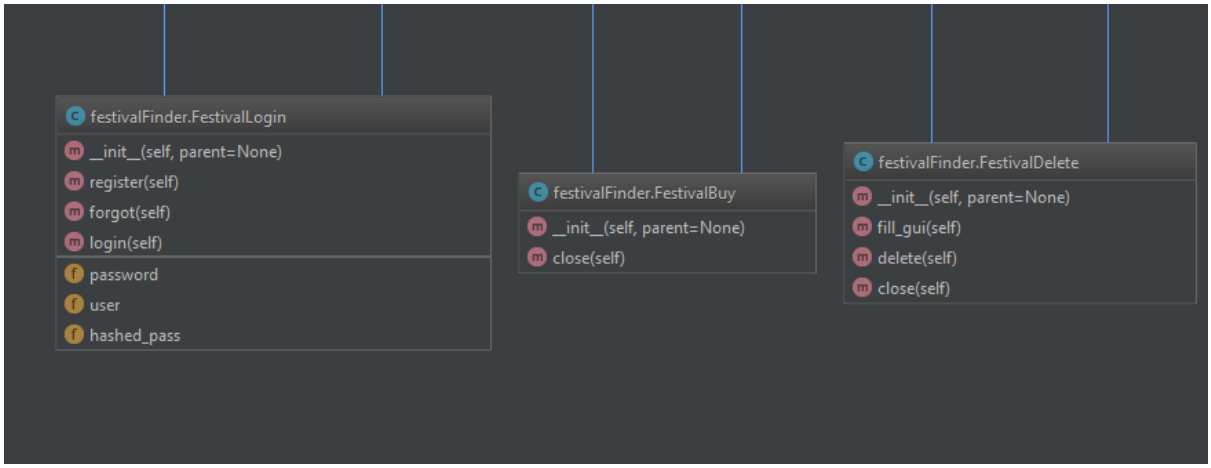


Class Diagrams with Methods (Zoomed in)



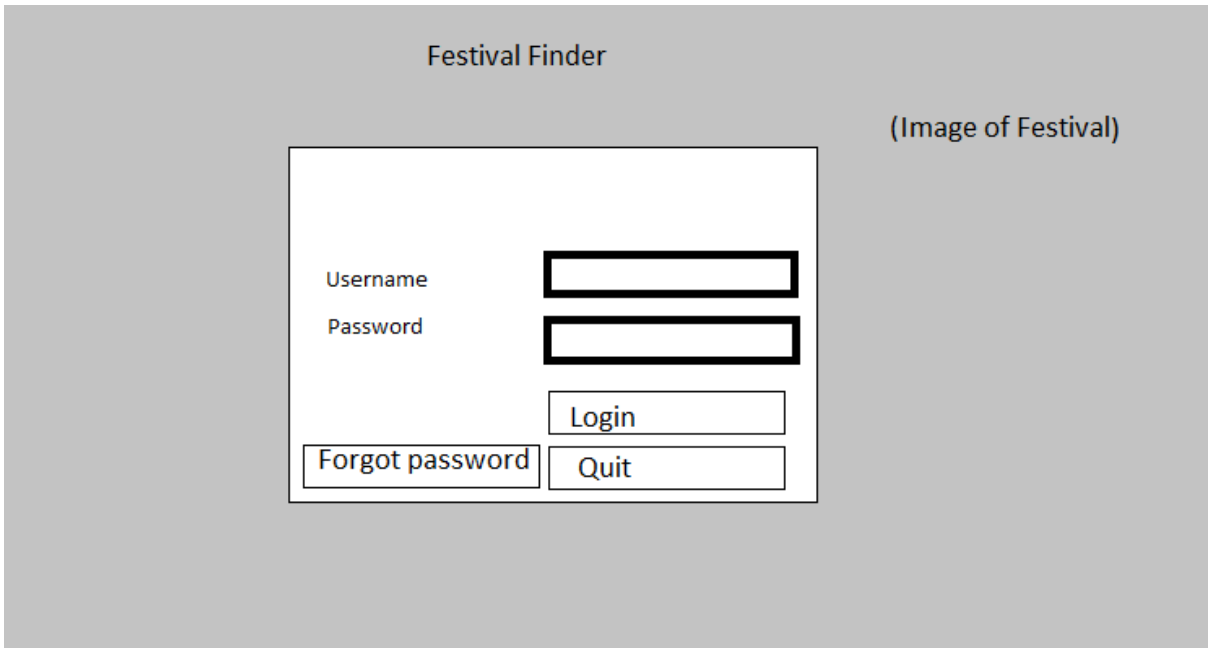



festivalFinder.FestivalCost
__init__(self, parent=None)
full_ticket(self)
day_ticket(self)
close(self)
festivalFinder.FestivalProfile
__init__(self, parent=None)
buy(self)
fill_gui(self, f_name)
close_window(self)
festivalFinder.FestivalAdmin
__init__(self, parent=None)
select_user(self)
delete(self)
admin(self)
refresh(self)
open_menu(self)
data
selectedUser
festivalFinder.FestivalDate
__init__(self, parent=None)
back(self)
fill_gui(self)
names
fest_name
festivalFinder.FestivalLocation
__init__(self, parent=None)
back(self)
alert(self)
festival_profile(self)
get_distance(self, home)
loc
names
fest_name
postcodes
festivalFinder.FestivalMenu
__init__(self, parent=None)
admin_menu(self)
logout(self)
music(self)
date(self)
open_cost(self)
festival_profile(self)
fill_gui(self, name)
open_location(self)
delete(self)
names
perm
fest_name

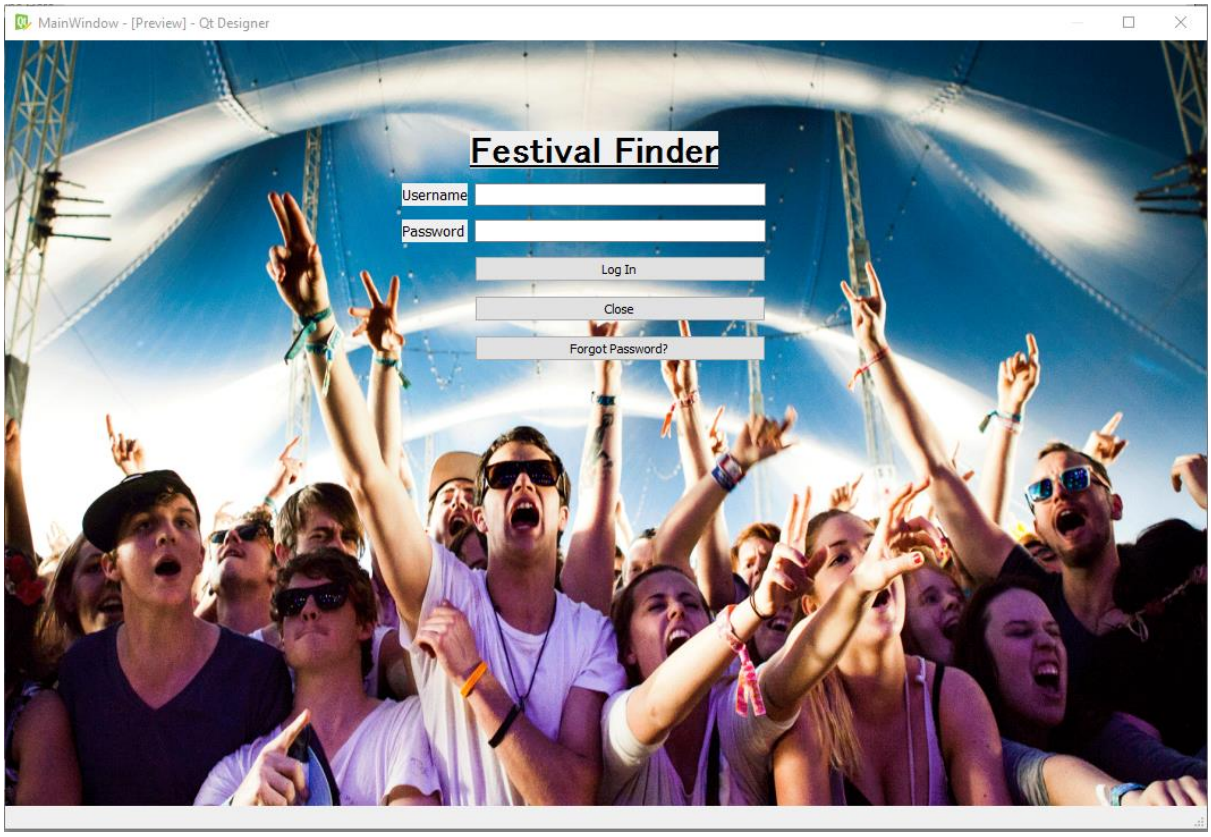

festivalFinder.FestivalLogin
__init__(self, parent=None)
register(self)
forgot(self)
login(self)
password
user
hashed_pass
festivalFinder.FestivalBuy
__init__(self, parent=None)
close(self)
festivalFinder.FestivalDelete
__init__(self, parent=None)
fill_gui(self)
delete(self)
close(self)

Describe usability features to be included in the solution

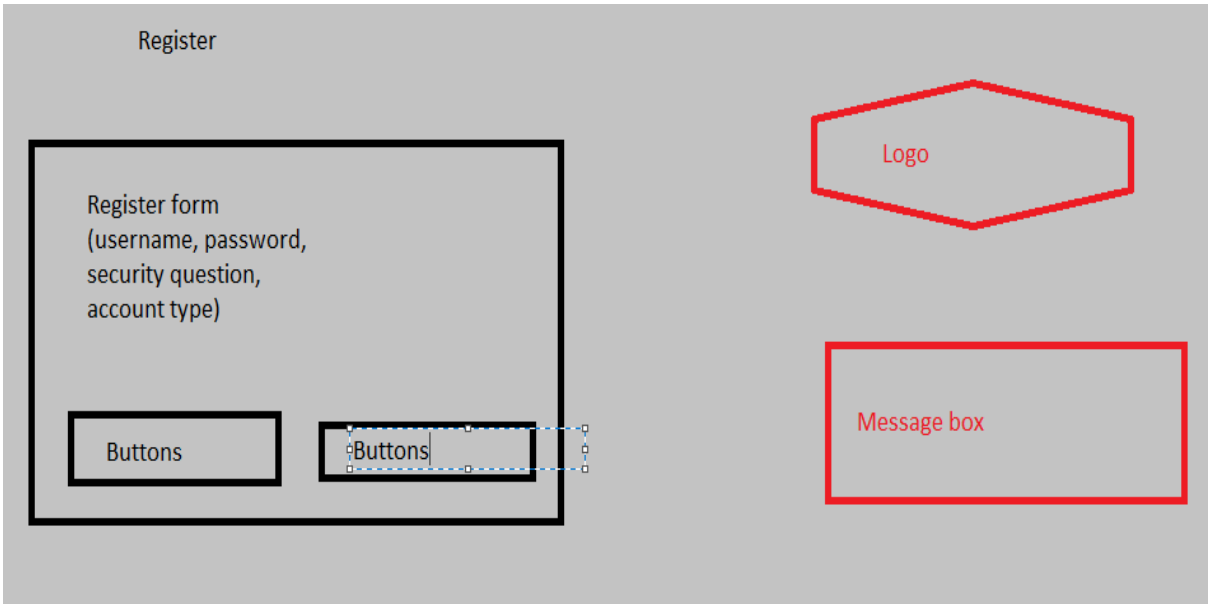
Our End-User drew up an initial idea of the design on Microsoft paint in our first meeting for the Graphical User Interface for the log-in page:



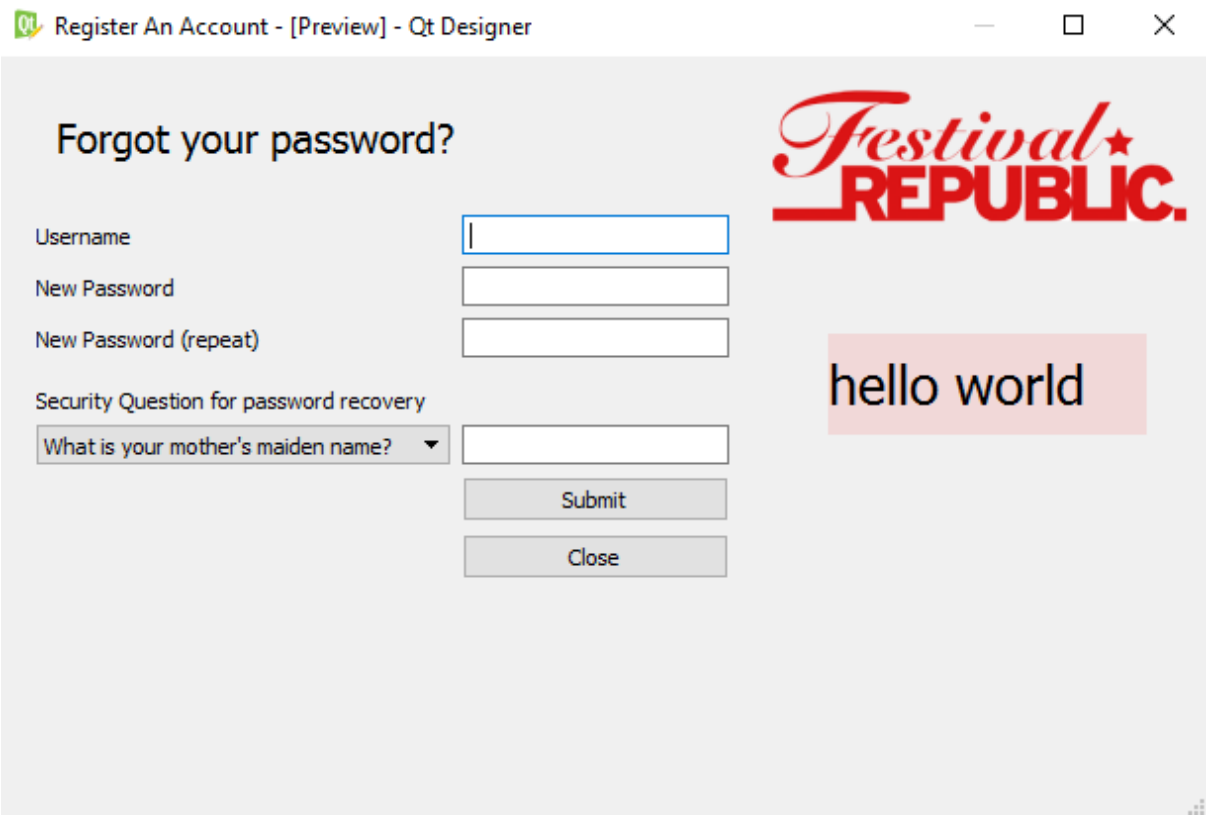
This is my initial design of the Graphical User Interface for the log-in page:



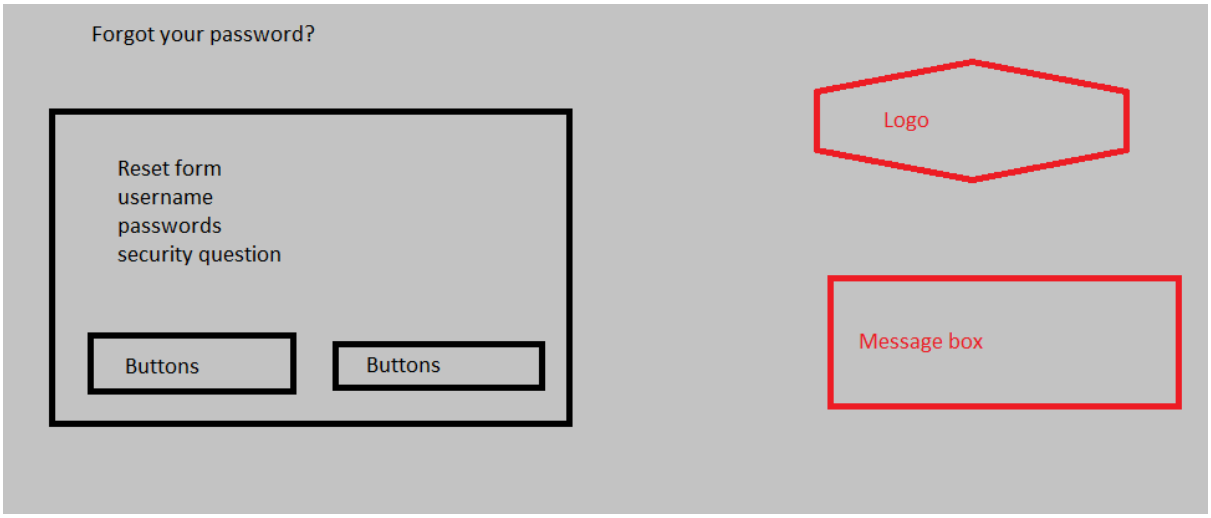
Our End-User drew up an initial idea of the design on Microsoft paint in our first meeting for the Graphical User Interface for the register page:



Here is the initial design/ first draft of the register page:

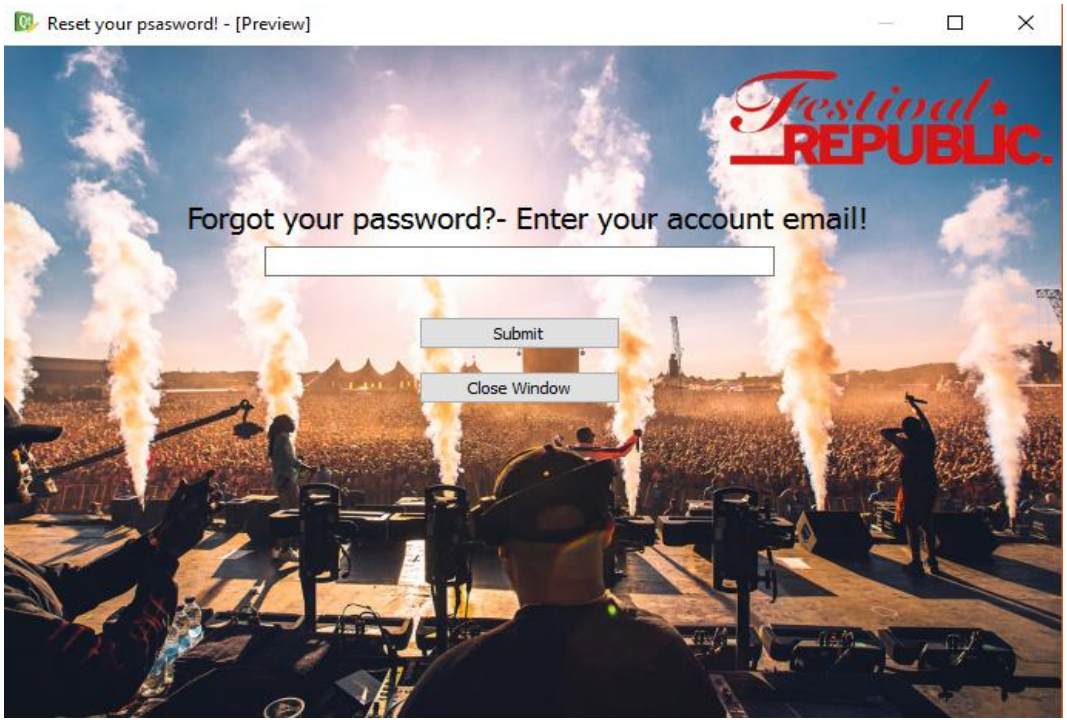


Like our register form, our End-User drew up an initial idea of the design on Microsoft paint in our first meeting for the Graphical User Interface for the forgot passwordpage:



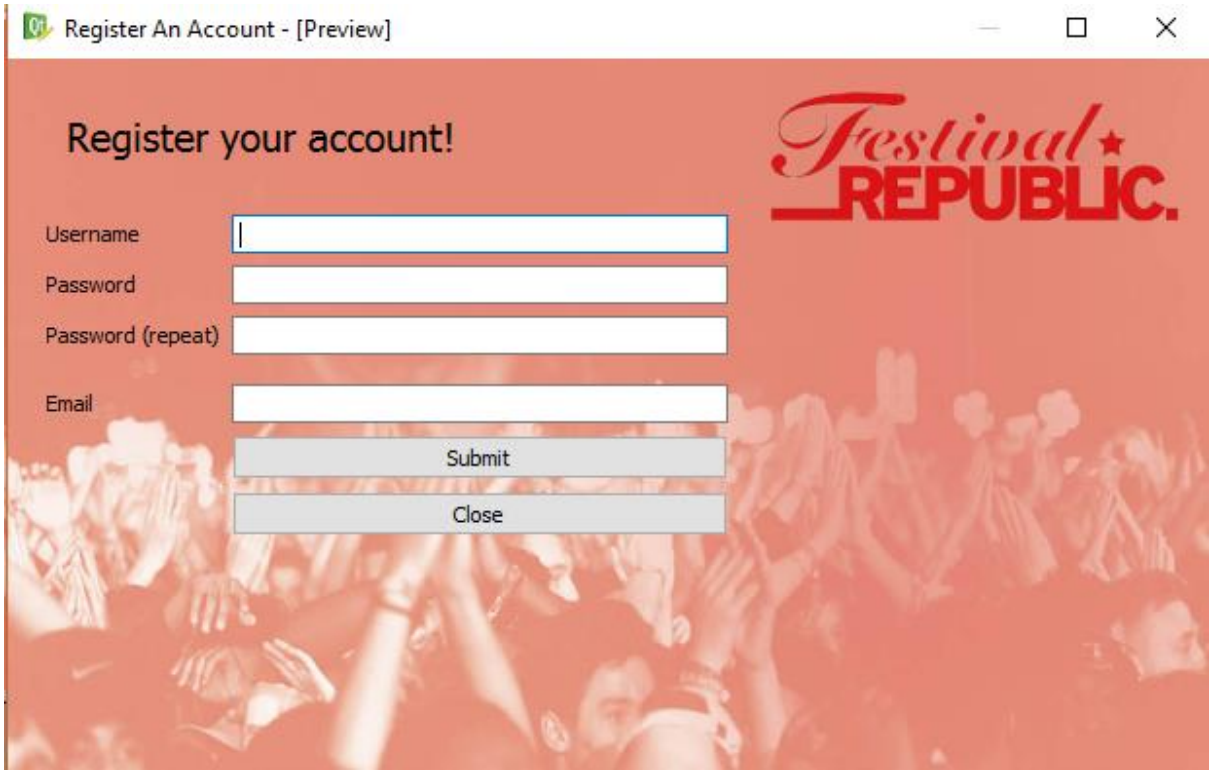
And here is the similar design for our forgot password form on our first draft:

This register form, along with the forgot password button, was then turned into two separate finished windows – Register and Forgot Password:



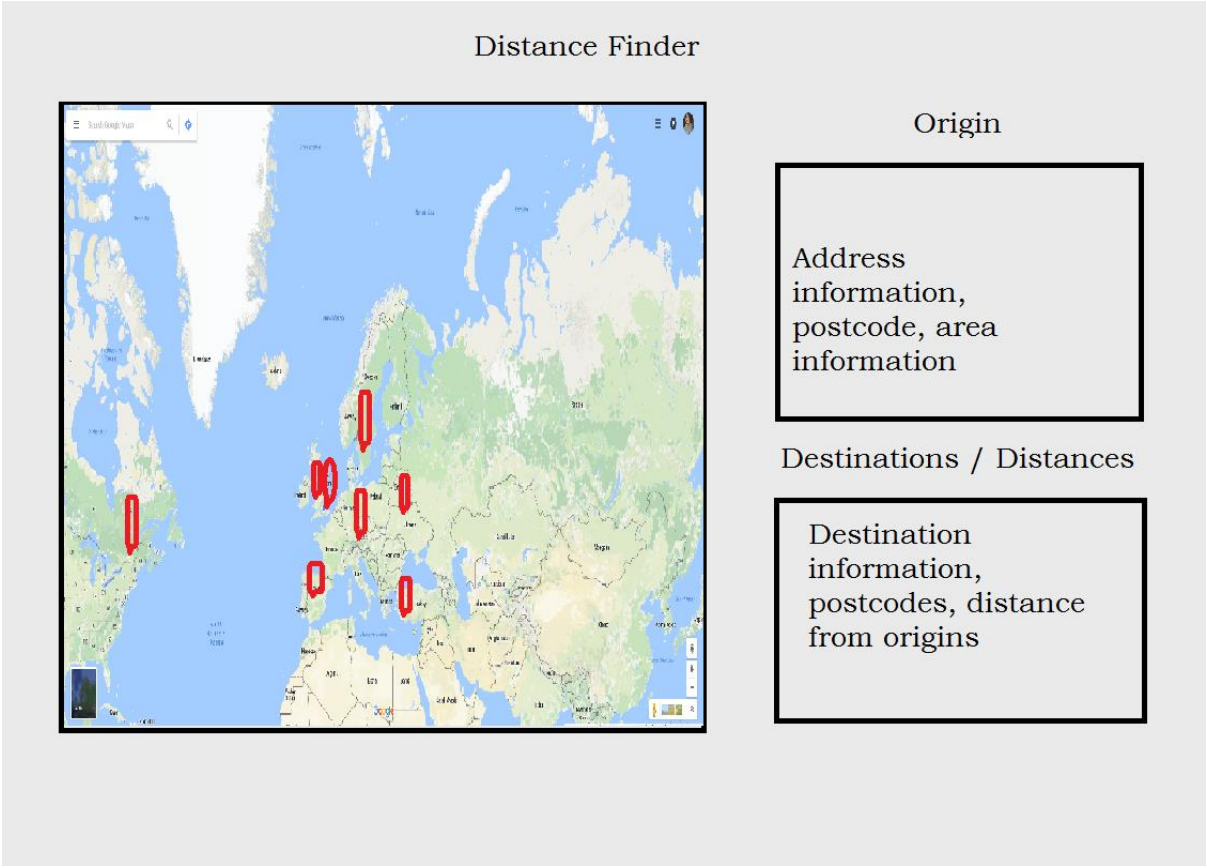
This window serves as now the user accounts are connected with email, the "Forgot your password?" function simply requires a valid email address of a user account, and it will reset the password and send the email.

The register window was then converted into this form:



This simply requires an email, username and password, as password recovery is serviced through email and administrator privileges are handled through existing administrators.

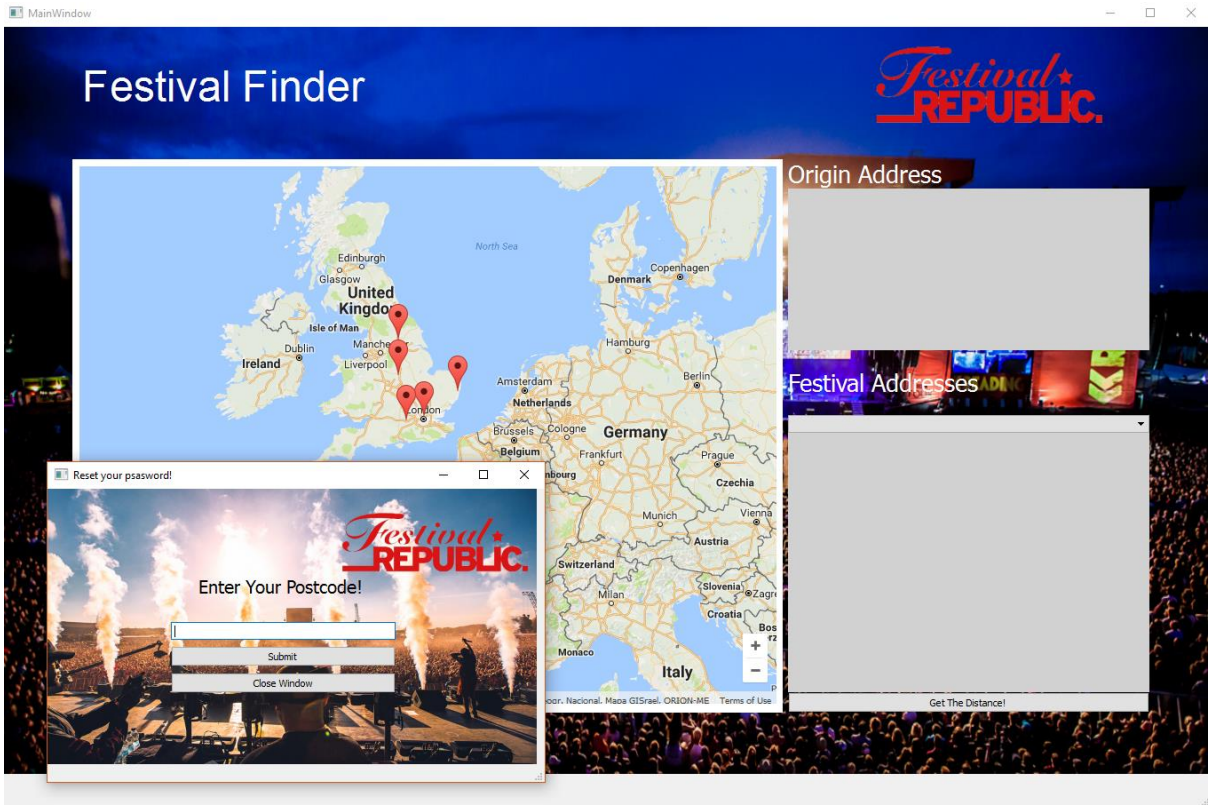
Our end-user also drew up designs for a possible map-like screen for finding closest festivals – an initial design to be worked upon but containing the essentials, a working map with multiple markers for festivals and a "home" location, as well as a panel for the "home" and destinations (festivals) addresses and the associated distances in miles, showing and explaining the closest festival.



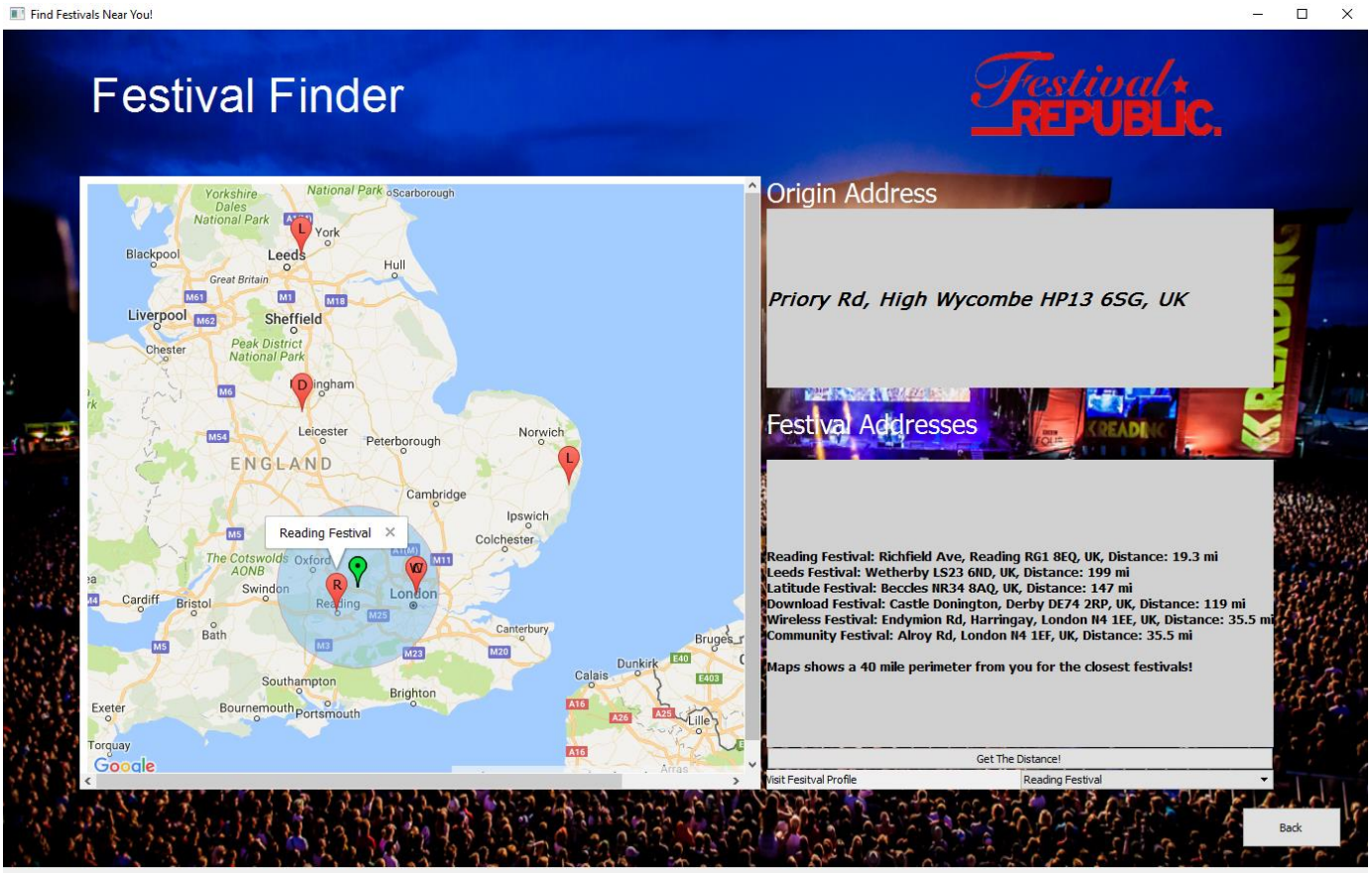
From this I developed a more polished initial interface, using the end-users logo, multiple textboxes and a combo-box for selection of particular festivals, multiple textboxes for text display (address details for both the "home" address and the addresses of the festivals), a button for entry of a home postcode or address line, which results in a separate pop—up window for information to enter into. I used the existing Google Maps API, a free, open-source solution for solving problems related to the google maps interface. For the addresses, I used a URL query, passing in the addresses entered by the user and the festival addresses read in upon program initialisation from the database. The URL queries returned JSON formatted distance matrices and address details, providing extensive address detail from a simple postcode and also distance from the home marker in miles. To plot the map, I used the latitude and longitude coordinates returned in the JSON output from the address details and passed each coordinate in in a list form into a JavaScript program I wrote in string format, in a separate, imported python module. The JavaScript code plots each destination by their latitude and longitude coordinates until the list of locations is empty, this is only altered when the origin postcode is appended and the map is then updated.

The initial design here includes the map specified with all the features described by the end user, whilst the second screenshot depicts the updated GUI after the popup window has had the origin

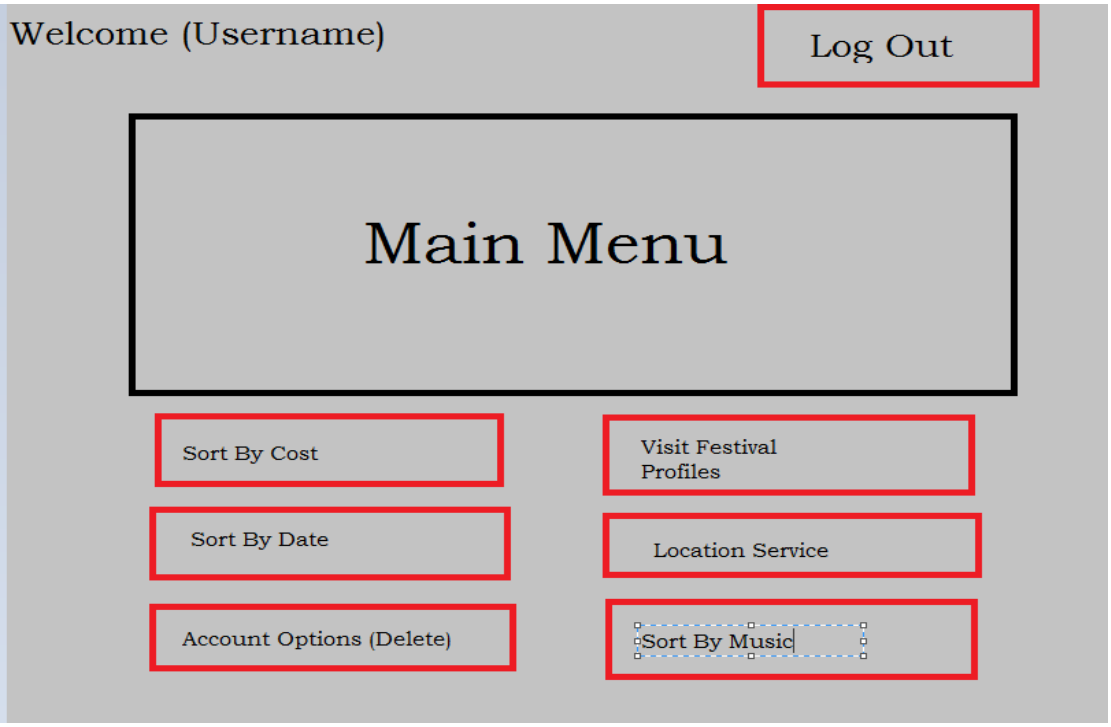
address/postcode inputted into it. Both the Origin Address and the addresses stored in the festivals table are geocached and used with the Google Distance Matrix, providing extensive address details from postcodes and the distances from the origin itself.



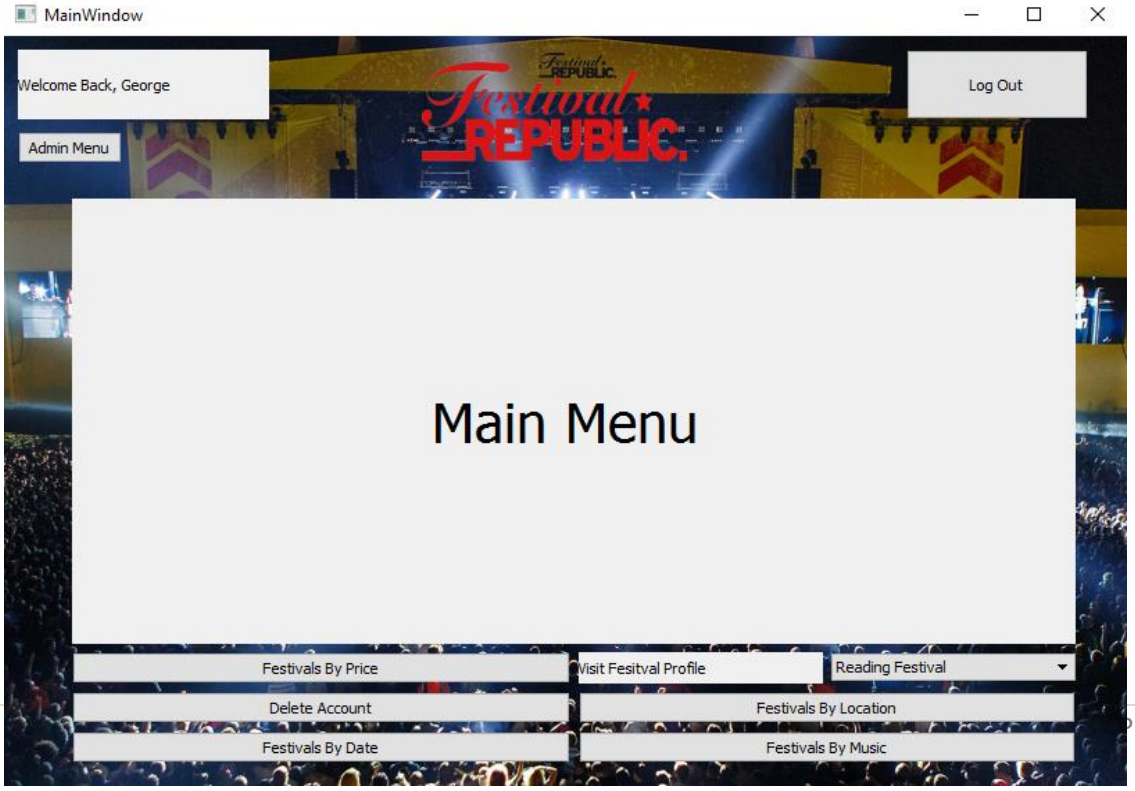
The second GUI, with the address information output, along with distances and an updated map with a home marker and 40 mile perimeter circle.



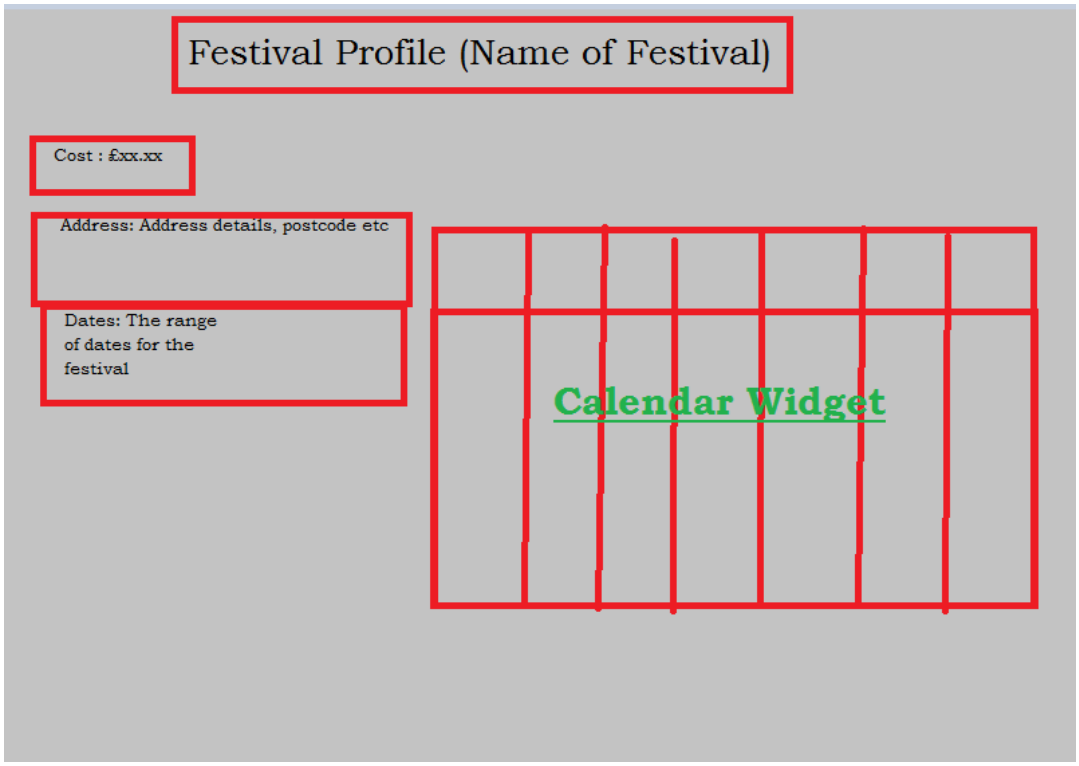
The end-user specified a template for an initial idea for user navigation – a clear main menu to navigate the program with – the menu has to navigate to the separate tabs – either back to the log in menu via "Log Out" or to the Cost, Date, Account Options, Festival Profiles, Location Service, Music tabs via the different Push Buttons. We Chose the push button option purely for simplicity. It also specified a personal welcome message.



This is the product we developed to suit the end-users needs. It welcomes the user with the use of the user—specific username and provides an array of choices for separate windows, to log out and to navigate the program accordingly.

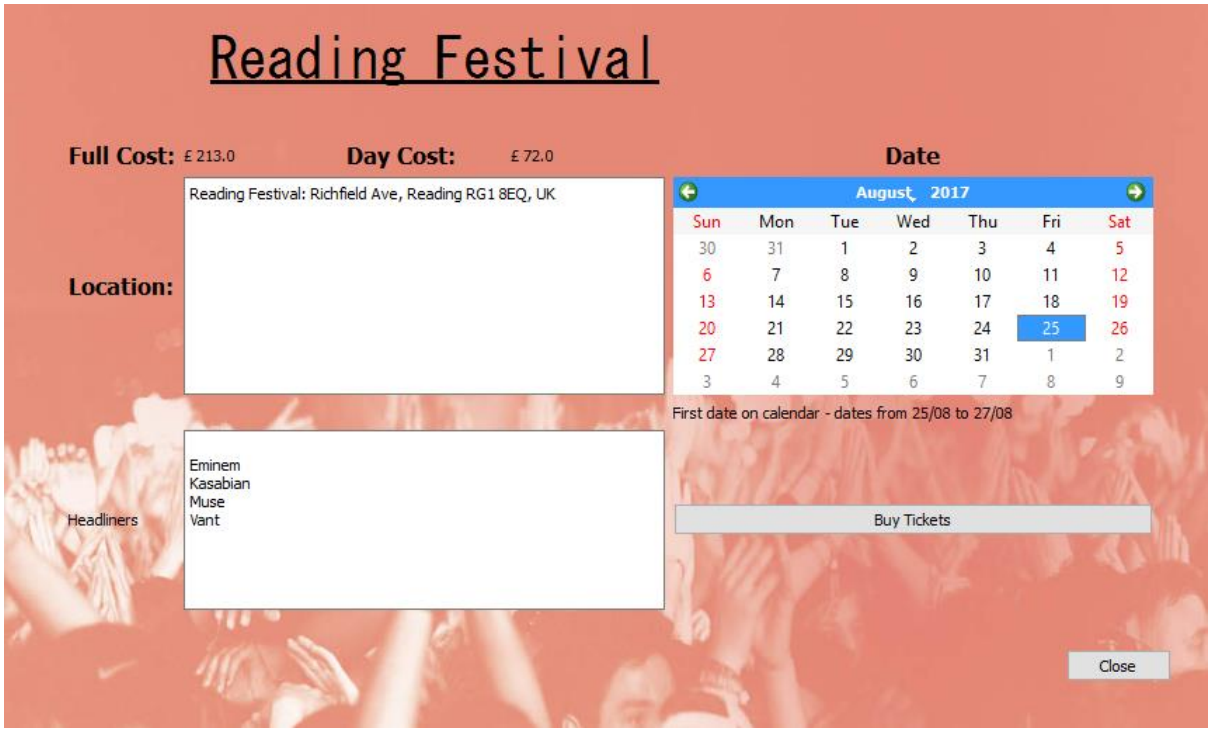


Like our register form, our End-User drew up an initial idea of the design on Microsoft paint in our first meeting for the Graphical User Interface for the profile page:

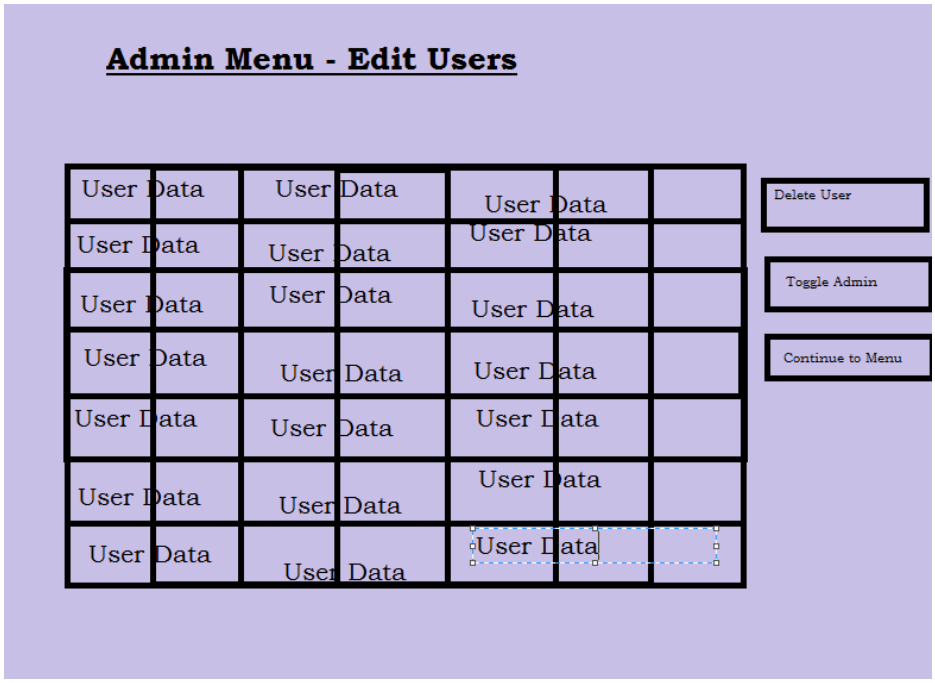


We provided a profile page that output the required fields, cost, day ticket cost, location and dates as well as a link to buy tickets for the specific festival. We also provided a display of all the headlining

acts for the festival itself.

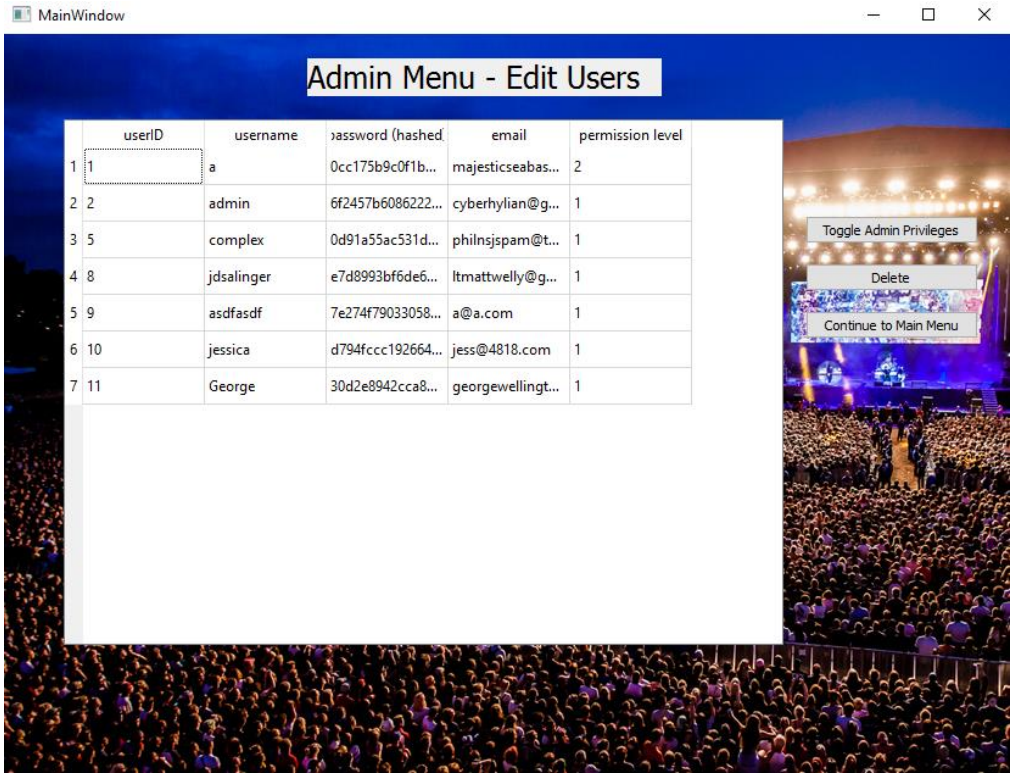


Our user also specified a special, unique menu which has restricted access – only admin users with a permissions level of 2 can access it. They suggested a table, title and multiple buttons and drew this up in MSPaint:



After viewing the end-users initial planned design, I incorporated a Qlabel for the title, as labels are the best for static text view, some QPushButtons for simple calling of functions – Delete, Toggle Administrator Privileges and Main Menu and a Table Widget for my database view, as I found that

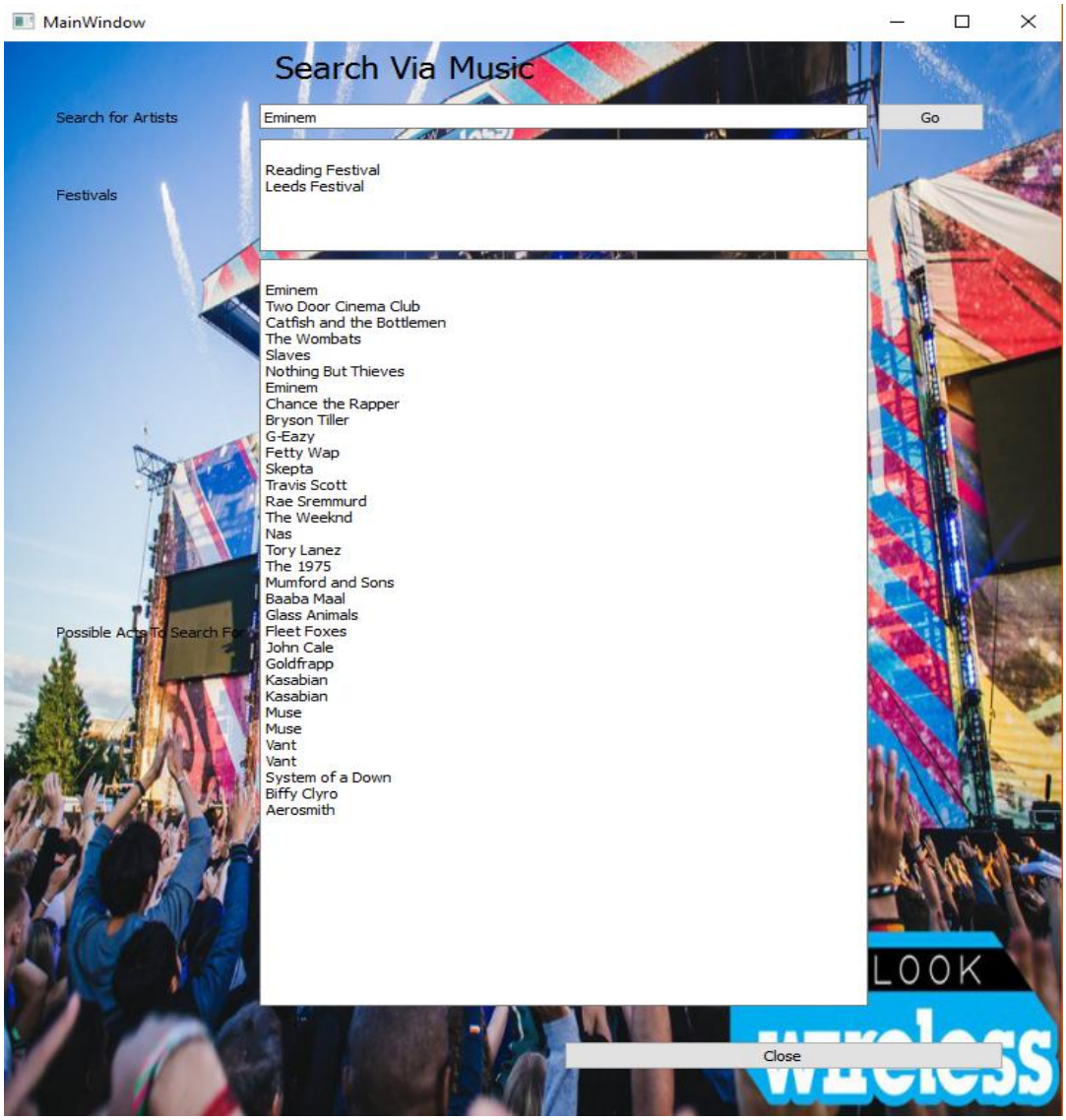
outputting data and refreshing the database could be done with fewer lines of code with a table widget versus a table view.



Like our profile form, our End-User drew up an initial idea of the design on Microsoft paint in our first meeting for the Graphical User Interface for the page where the user can find the festival by artist:

The design includes a group of labels for the title, artist, festivals they are playing, available headliners – I will use the QLabel object for this as it is the most versatile for displaying simple text in a window and for editing quickly in the backend side of the program. I used a QTextBrowser for Festivals and Acts playing as they are the best for displaying multiple lines of output text to print from the program – I will fetch each database result for this and iteratively add it to a string "display" to put into the QTextBrowser. I used a QLineEdit object to take in the artist name that the user would input as it is best suited to receiving input and this input will be allocated to a variable once the "Go" button is activated. Most predictably, I used QPushButtons for the "Go" and "Close" buttons as these are best suited to custom-text, single purpose push button processes.

Below is the design I came up with to satisfy the needs of the end-user – it contains a line edit for artist input, push buttons for "Go" and "Close", labels for the title and descriptions and text browsers for the larger output displays. The window itself takes in the users artist via input and fetches the festivals for output associated with that artist. It also displays all the potential artists to choose from in the lower text browser.



Justification for Widgets Used

QLabel – QLabel were used for most static and in rare cases dynamic text, as they were the best objects for text display. They are simple and lightweight and easy to implement for short lines of text.

QTextBrowser – QTextBrowser were used for most outputs of a large amount of data, usually several lines – they are larger than a standard object and provide a clear, visible display for information output on a large scale

QCalendar – This was used as the only object for display of graphical dates in a calendar format.

QLineEdit – QLineEdit was used for short amount of input, as it is lightweight and easy for the user to manage their information input by typing.

QPushButton – Push buttons were used for simple calling of functions within the class as no radio or check buttons were needed at any point – they are simple and only served the purpose they were required to.

QWebView – This widget was either used for displaying program generated HTML or a link to a URL. This was the only object capable of this function and therefore was used.

Key Variables / Date Structures / Classes

Google Maps Module

Name	Data Type	Description
locations	Array	The array of locations to plot, includes names and longitude and latitude coordinates.
counter	Integer	Is the length of the array locations, and ergo is the amount of iterations needed to plot.
Ishome	Integer	Signifies whether the last coordinate is the home coordinate, and thus whether there will be a forty mile radius and a different coloured marker.
code	String	The html code to be returned to festivalfinder.py.

Email Module

Name	Data Type	Description
user	String	The string form of the email address login username for sending emails.
password	String	The string form of the email address login password for sending emails.
to	String	Takes the form of the recipient address in string form.
body	String	Takes the form of the body of the email to be sent in string form.
Msg	String	Takes the message as a whole before it is submitted to the email server in the format that is required.

The upcoming classes use Hungarian notation by denoting each variable name by "datatype_variablename". I have omitted the Data Type column accordingly.

- **Btn_ denotes a QPushButton.**
- **Lbl_ denotes a QLabel**
- **Dropdown_ denotes a QComboBox, a dropdown widget of QtDesinger**
- **Tbl_ or db_ denotes a QTablewidget**
- **Cal_ or calendar denotes a QCalendarWidet**
- **Txt_ denotes a large amount of information output, usually a QTextBrowser**
- **Entry_ denotes a QLineedit**

Some common QWidgets that are not unique to any specific window are

Lbl_title – a Widget for the title of the page.

Lbl_bg – a widget that holds the pixmap for the image used in the background of the page

Lbl_msg/ txt_msg – a widget used to display information to the user eg. (Wrong password)

Btn_close – a push button used to return to the previous page, logout, or close the program.

Lbl_logo – a label holding the pixmap of the end-user's logo.

Class FestivalLogin

Name	Description
Str_password	Takes on the password value in string format.
Str_user	Takes on the username value in string format
Str_hashed_pass	Takes on the string value of the hashed password in md5 encryption.
Btn_login	The submit button used for logging in. (Submit, Enter etc)
Btn_forgot	Links to the forgot window
Btn_register	Links to the register window
Entry_user/entry_pass	A QLineEdit used for entering the username and password

Class FestivalRegister

Name	Description
Str_password	Takes on the password value in string format.
Str_user	Takes on the username value in string format
Str_hashed_pass	Takes on the string value of the hashed password in md5 encryption.
Str_password2	Takes on the password value for checking a second attempt (makes sure the user enters the password they intended
Reg_user	The Regular expressions for the username field, specifies that it has to start with an alphanumeric character, can only contain alphanumeric characters and underscores and be in a length of 4-12 characters.
Reg_pass	The Regular Expressions used for validating the password field. It must be within 6 and 12 characters, contain at least one uppercase character, one lowercase character and no special characters.
Btn_submit	Submits the input information by the user for work by the functions.

Class FestivalForgot

Name	Description
Btn_submit	Submits the input information by the user for work by the functions.
Str_email	The string format of the email – it is checked in the users table for existence.
New_password	The randomly generated integer that the password is reset to for the user with the email used to reset the account.

Class FestivalAdmin

Name	Description
Btn_menu	Continues to the main menu when pushed.
Btn_admin	Toggles the admin privileges of the currently selected user in the table.
Btn_delete	Deletes the currently selected user from tbl_users.
Db_table	The table widget displaying tbl_users for the admin.
Str_user	The variable that stores the user currently logged in – this ensures that that users details aren't fetched and therefore a user can't delete oneself.
Str_selectedUser	Takes the string form of the currently clicked user on the table widget. (From tbl_users)
Data	Holds the tbl_users data apart from the current user.

Class FesitvalDate

Name	Description
List_names	Holds all the names of the festivals from tbl_festivals.
Dropdown_fest	A combobox menu filled with festival names for selection.
Str_fest_name	The currently selected festival name from dropdown_fest
Str_Start_date	Holds the start date of the festival in string format.
Str_End_date	Holds the end date of the festival in string format.
Lbl_dates	Displays the start and end date for the user
Calendar	A calendar widget to show the start date.

Class FestivalProfile

Name	Description
Btn_buy	Links to the buy tickets window (Guesses the url to a ticket vendor)
Festival_name	The title of the GUI to display the currently selected festival's name.
Display_address	The formatted postal address of the festival's postcode.
Day_month	The date of start date split to be formatted.
Lbl_cost	Shows the Full weekend cost for the current festival.
Lbl_daycost	Shows the day ticket cost for the current festival.
Calendar	Displays the start date for the current festival.
List_acts	A list of all acts for the current selected festival.

Class FestivalLocation

Name	Description
List_names	List of all the festival names gathered from tbl_festivals.
List_postcodes	List of all the festival postcodes gathered from tbl_festivals.
Str_fest_name	The currently selected festival from the dropdown menu (links to festivalprofile)
List_coordinates	The latitude and longitude values fetched by a geocache google maps query with list_postcodes.
loc	A concatenated array of both the festival names followed by a nested list of the festival coordinates.
Str_Html	The returned value in string format of calling googlemap.plot(). Googlemap returns a string form of html with the newly inserted locations to plot all the festivals and the user's home location.
Btn_getDistance	Asks for a users postcode to calculate distance.
Dropdown_fest	A dropdown menu of festivals for quick navigation to festivalprofile.

Class FestivalAlert

Name	Description
Str_Home	A variable to take in a home postcode from an entrybox.
Entry_postcode	A means of postcode input to be assigned to str_home.
Regex_postcode	A regular expression formula issued by the UK Cabinet office to recognise postcodes. It checks and validates postcode input.
Btn_ok	Submits the postcode for checking.

Class FestivalCost

Name	Description
Btn_full	A request to display the cost for a full ticket at every festival, sorting all festivals by low – high.
Btn_day	A request to display the cost for a day ticket at every festival, sorting all festivals by low – high.
Tbl_data	A table widget for displaying this data.
Data	A variable for the fetchall result of the SQL queries.

FestivalMusic

Name	Description
Btn_go	Submits the users data for entry (The artist)
Btn_overview	Displays an overview of all festivals and their artists.
Txt_acts	Displays all possible acts.
Txt_festivals	Displays the festivals that entered acts are playing at.

Festival Delete

Name	Description
Btn_delete	Deletes the currently signed in users account.

Festival Menu

Name	Description
List_names	Names of all the festivals for the dropdown_fest combobox to display
Dropdown_Fest	Links to the currently selected festival in Festival profile.
Admin	Links admin users to the admin menu.
Btn_location	Links to the location window
Btn_price	Links to the cost window
Btn_delete	Links to the delete window
Btn_date	Links the the date window
Btn_acts	Links to the music window
Int_perm	Signifies the permission of the user – admin level means that int_perm is 1.

Afterword of Variables – Variables of Different Scopes

Throughout, these variables have not been attributed to their scope – global, local, object etc. The variables that appear for each class are assigned to the class they are in in a "self.variablename" form, local variables are just repeated unimportant variables such as "result" that take the returned values from SQL queries etc, global variables are something I set out purposefully to avoid – which I only include as the "debug_mode" variable to signify which debug mode the program is in and therefore has to be used throughout the whole program.

SQL Tables

The SQL tables are organised into two main tables and two tables used for relational data on the music played at festivals.

They are:

1. "tbl_festivals" – used to store data on festivals used by the program.
2. "tbl_users" – used to store extensive data on the users of the program.
3. "tbl_acts" – used to hold information on music acts.
4. "tbl_genres" – used to hold genre information on the acts.

FESTIVALS TABLE

Name	Type	Special Constraints
festID	Integer	Primary Key / Autoincrement
Name	Text	/
Postcode	Text	/
StartDate	Text	/
EndDate	Text	/
Cost	Real	/
DayCost	Real	/

ACTS TABLE

Name	Type	Special Constraints
actID	Integer	Primary Key / Autoincrement
actName	Text	/
festID	INT	Foreign key on tbl_acts.festID = tbl_festivals.festID

GENRES TABLE

Name	Type	Special Constraints
genreID	Integer	Primary Key / Autoincrement
Genre	Text	/
actID	Integer	Foreign key on tbl_acts.actID = tbl_genres.actID

USERS TABLE

Name	Type	Special Constraints
userID	Integer	Primary Key / Autoincrement
User	Text	/
Pass	Text	/
Email	Text	Not null
Perm_lvl	Int	Default 1, Not Null

Below is the setup data and constraints for “tbl_festivals”:

```
CREATE TABLE "tbl_festivals"
(
    festID INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT,
    Postcode TEXT,
    StartDate TEXT,
    EndDate TEXT,
    Cost REAL,
    DayCost REAL
);
CREATE UNIQUE INDEX tbl_festivals_2_festID_uindex ON "tbl_festivals" (festID)
```

Below is the setup data and constraints for “tbl_acts”:

```
CREATE TABLE tbl_acts
(
    actID INTEGER PRIMARY KEY AUTOINCREMENT,
    actName TEXT,
    festID INT, genreID INT NULL,
    CONSTRAINT tbl_acts_tbl_festivals_festID_fk FOREIGN KEY (festID) REFERENCES tbl_festivals (festID)
);
CREATE UNIQUE INDEX tbl_acts_actID_uindex ON tbl_acts (actID)
```

Below is the setup data and constraints for “tbl_users”:

```
CREATE TABLE tbl_users
(
    userID INTEGER PRIMARY KEY AUTOINCREMENT,
    user TEXT,
    pass TEXT
, email TEXT NULL, perm_lvl INT DEFAULT 1 NULL)
```

Below is the setup data and constraints for “tbl_genres”:

```
CREATE TABLE "tbl_genres"
(
    genreID INTEGER PRIMARY KEY AUTOINCREMENT,
    Genre TEXT,
    CONSTRAINT tbl_genres_tbl_acts_genreID_fk FOREIGN KEY (genreID) REFERENCES tbl_acts (genreID)
);
CREATE UNIQUE INDEX tbl_genres_genreID_uindex ON "tbl_genres" (genreID)
```

This is the sample data for users:

	userID	user	pass	email	perm_lvl
1	1	a	0cc175b9c0f1b6a831c399e269772661	majesticseabass@gmail.com	2
2	2	admin	6f2457b608622214f753c8522bf63a38	cyberhylian@gmail.com	1
3	5	complex	0d91a55ac531df50ede54f22b23ed9ec	philnsjspam@talktalk.net	1
4	8	jdsalinger	e7d8993bf6de68a2d4e3a8a24e51f4fd	ltmattwelly@gmail.com	1
5	10	jessica	db1a0dee243af63d8db3703ab7d80e53	jess@4818.com	1
6	11	George	30d2e8942cca845bc66236b4de28ae07	georgewellingtonwork@gmail.com	1

This is the sample data for festivals:

	festID	Name	Postcode	StartDate	EndDate	Cost	DayCost
1	1	Reading Festival	RG1 8EQ	25/08	27/08	213	72
2	2	Leeds Festival	LS23 6ND	25/08	27/08	213	72
3	3	Latitude Festival	NR34 8AQ	13/07	17/07	197.5	84.5
4	4	Download Festival	DE74 2RP	09/07	11/07	205	83
5	5	Wireless Festival	N4 1EE	08/07	10/07	210	62
6	6	Community Festival	N4 1EF	01/07	01/07	40.3	40.3

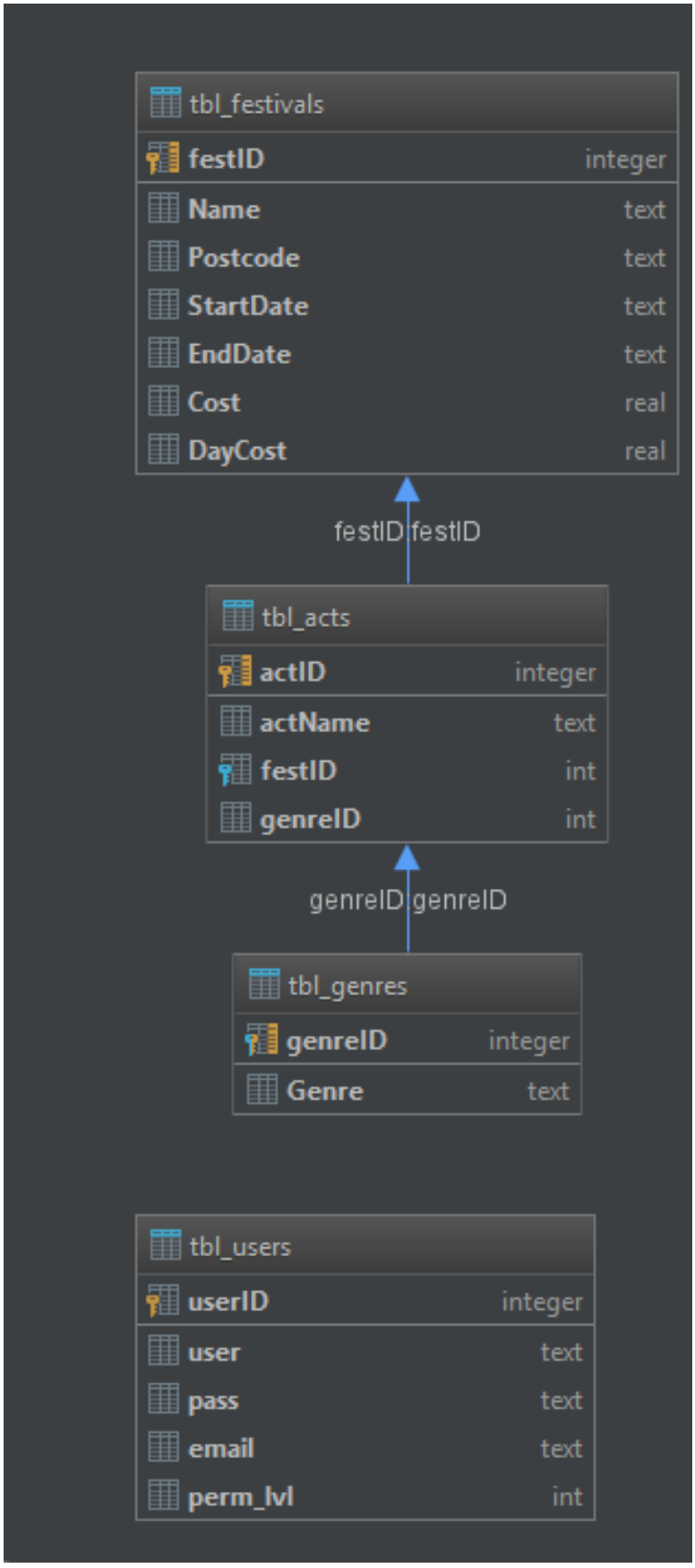
This is the sample data for genres:

	genreID	Genre
1	1	Rock
2	2	Pop
3	3	Rap
4	4	Alternative
5	5	Grime
6	6	Punk
7	7	Indie
8	8	Metal
9	9	R&B
10	10	Hip-Hop
11	11	Electronic

This is the sample data for the acts:

	actID	actName	festID	genreID
1	1	Eminem	1	3
2	2	Two Door Cinema Club	6	4
3	3	Catfish and the Bottlemen	6	4
4	4	The Wombats	6	2
5	5	Slaves	6	6
6	6	Nothing But Thieves	6	1
7	7	Eminem	2	3
8	8	Chance the Rapper	5	3
9	9	Bryson Tiller	5	9
10	10	G-Eazy	5	3
11	11	Fetty Wap	5	3
12	12	Skepta	5	5
13	13	Travis Scott	5	10
14	14	Rae Sremmurd	5	10
15	15	The Weeknd	5	2
16	16	Nas	5	3
17	17	Tory Lanez	5	3
18	18	The 1975	3	7
19	19	Mumford and Sons	3	7
20	20	Baaba Maal	3	3
21	21	Glass Animals	3	4
22	22	Fleet Foxes	3	7
23	23	John Cale	3	1
24	24	Goldfrapp	3	11
25	25	Kasabian	2	4
26	26	Kasabian	1	4
27	27	Muse	1	1
28	28	Muse	2	1
29	29	Vant	1	1
30	30	Vant	2	1
31	31	System of a Down	4	8
32	32	Biffy Clyro	4	1
33	33	Aerosmith	4	1

This is the ERD for the tables, including names and data types.


tbl_festivals
festID integer
Name text
Postcode text
StartDate text
EndDate text
Cost real
DayCost real
festID festID
tbl_acts
actID integer
actName text
festID int
genreID int
genreID genreID
tbl_genres
genreID integer
Genre text
tbl_users
userID integer
user text
pass text
email text
perm_lvl int

Examples of Validation Pseudocode

```
# pseudocode for checking input via regular expressions #
if input regex.matches(regex_pattern) then
    submit()
else
    print("Please enter something that adheres to strength rules")
endif

#pseudocode for checking if input is there at all #
input = super.inp_item.getText()
input2 = = super.inp_item2.getText()
input3 = super.inp_item3.getText()
if input OR input2 OR input3 == "" then
    print("Please complete all fields")
endif

#pseudocode for checking for existing user (example) #
str_user = inp_user.text
str_pass = inp_password.text
str_pass2 = inp_passw2.text
str_email = inp_email.text
str_hashed_pass = str_pass.hash(md5)
connection = openRead("tbl_users")
users = connection.read_table()
for i=0 to ( LENGTH(users) - 1)
    if users[i].email = str_email then
        result = users[i]
    elseif users[i].user = str_user then
        result = users[i]
    endif
if LENGTH(result) > 0:
    print("email or username already in use")
else
    submit()
endif
```

For some examples I've included my testing for empty inputs, regular expression matching and checking whether email or usernames already exist, another section I've included is testing whether the user input maximum distance they would like to travel is actually an integer (made difficult as its stored as a string:

More Validation types: (length, type, format, range):

```
#type check #
input int_max_distance
if int_max_distance is integer then
    max_distance()
elseif int_max_distance is float then
    int_max_distance = int_max_distance.round()
    max_distance()
else
    print("please enter a integer value")

#range check #
input distance
if distance > 0 and distance < 150 then
    distance()
else
    print("enter between the correct range, 0 and 150")

# length check #
input username
input password
if LENGTH(username) > 0 and LENGTH(password) > 0 then
    if  LENGTH(username) < 20 and LENGTH(password) < 20 then
        submit(username, password)
    else
        print("username and password must be less than 20 characters")
else
    print("usernames and passwords must be filled")

# format check (needed for postcodes #
if input regex.matches(regex_pattern) then
    submit()
else
    print("Please enter something that adheres to strength rules")
endif
```

The Approach to Testing

Test Plan

With each class (one class for each screen used), I will test that it provides results for each function it is supposed to provide successfully, see if it accepts/declines acceptable and unacceptable data accordingly with validation – test that stressing it will not make it crash (ie The user getting to the level where they can crash the GUI with excessive button pressing or requests). Any failed tests will be immediately met with debugging and fixing, which will be documented.

Festival Login Class

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test
1	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.
2	**Test for data entry**	Test for nothing entered	The program must reject/ alert user to blank data entries and accept data entry.
3	**Logging in with correct user information.**	Logging in with incorrect user information.	The program must login for the correct information and not allow entry for incorrect information.
4	**Logging in with/without administrator privileges.**	Logging in with/without administrator privileges.	The program must pass to the administrator table for administrators and pass to the main menu for non-administrator users.

Festival Register Class

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test
5	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.
6	**Test for data entry**	Test for nothing entered	The program must reject/ alert user to blank data entries and accept data entry.
7	**Test the regular expressions for the password field for conformity.**	Test the regular expressions for the password field for inconformity.	The program must reject inconformity and accept conforming inputs.
8	**Test the regular expressions for the**	Test the regular expressions for the	The program must reject inconformity and accept conforming inputs.

	username field for conformity.	username field for inconformity.	
9	**Test to try and set up an account with an unused email.**	Test to try and set up an account with a used email.	The program must reject a new user with an already used email and accept one with an unused email.

Festival Forgot Class

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test
10	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.
11	**Test for data entry**	Test for nothing entered	The program must reject/ alert user to blank data entries and accept data entry.
12	**Test a valid user email.**	Test an invalid user email.	The program must send emails out to valid emails and reject invalid user emails.

Festival Admin

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test
13	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.
14	**Test that the delete function works**	n/a	Test that the delete function works
15	**Test all data is displayed**	Try and delete own account	The program must display all the data and not allow users to delete their own account.
16	**Test that the toggle admin privilege feature works**	n/a	Test that the toggle admin privilege feature works

Festival Date

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test

17	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.
18	**Test for correct output (List of Festivals)**	n/a	**Test for correct output (List of Festivals)**
19	**Tests must return correct date for each festival**	n/a	Tests must return correct date for each festival

Festival Profile

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test
20	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.
21	**Links to the correct Ticket Vendor URL**	n/a	**Links to the correct Ticket Vendor URL**
22	**Shows the correct data for the selected festival**	n/a	Shows the correct data for the selected festival

Festival Location

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test
23	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.
24	**Successfully Plots the user's point of origin**	n/a	Successfully Plots the user's point of origin
25	**Successfully Calculates Distances**	-	Successfully Calculates Distances
26	**Displays the correct data**	-	Displays the correct locations of all the festivals

Festival Alert

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test
27	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.

28	**Passing in the correct location to festival Location**	n/a	Passing in the correct location to festival Location
29	**Test the regular expressions for the postcode field for conformity.**	Test the regular expressions for the postcode field for inconformity.	The program must reject inconformity and accept conforming inputs.

Festival Cost

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test
30	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.
31	**Displays data from a database**	n/a	Displays data from a database
32	**Displays the correct data for each button.**	-	Displays the correct data for each button.

Festival Music

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test
33	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.
34	**Test for data entry**	Test for nothing entered	The program must reject/ alert user to blank data entries and accept data entry.
35	**Inputting the correct artist (An existing artist)**	Inputting the incorrect artist (An non-existence artist)	Yielding results for correct input
36	**Displaying the correct data**	/	Displaying the correct data

Festival Delete

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test
37	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.
39	**Deleting the account that's currently logged in.**	/	Deleting the account that's currently logged in.

Festival Buy

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test
39	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.
40	**Displaying the correct data**	/	Displaying the correct data

Festival Menu

Success Criteria Number	Test (Positive)	Test (Destructive)	Required Result from Test
41	**Test the buttons for correct links (correct windows and functions etc)**	Stress test the GUI (Trying to create a crash via spam pressing buttons etc)	The program must stand up to stress test and link to the correct functions/classes.
42	**All the links link to the correct windows.**	/	All the links link to the correct windows.
43	**Only admin level users can access the admin menu via the main menu**	/	Only admin level users can access the admin menu via the main menu

Extra Success Criteria

Success Criteria Number	Required Result from Test
44	The program must be suitable for all computers that run on the Windows OS
45	The program must provide full information on all artists performing at the festivals.
46	The program must provide an adequate user interface for finding information about festivals
47	The program must provide information relevant to all users
48	All of the GUI is acceptable and is in keeping with the end-user's general look

The program's functions can be tested as the program is going along however full user data and across the board functionality can only be tested once the program is fully complete.

Development

Iterative Development Process

Login Screen –

The login screen was naturally the first problem I set out to develop a solution to in the program, with it being the first screen that the user will use. I started first with bits of boilerplate code for Object Orientated Programming solutions with pyqt to display a log-in window.

```
import sys, os
from PyQt3 import QtCore, QtGui, uic

form_class = uic.loadUiType("login.ui")[0]  # Load the UI

class MyWindowClass(QtGui.QMainWindow, form_class):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)


####################
#Boilerplate code - always same unless have multiple modules/windows in a program

app = QtGui.QApplication(sys.argv)
myWindow = MyWindowClass(None)
myWindow.show()
app.exec_()
```

This is the first attempt at some boilerplate code and next came stitching together some log-in functions:

```
    def login(self):
        self.user = self.entry_user.text()
        self.password = self.entry_pass.text()
        print(self.user, self.password)
        print(real_user, real_password)
        if self.user == real_user:
            if self.password == real_password:
                self.wdw_home.show()
                self.hide()
            else:
                self.txt_msg.setText("Wrong password")
        else:
            self.txt_msg.setText("Wrong username")
```

I then refined this solution by accessing the users database and trying to find both a username and password match – this allows the user to pass if the length of a search query for both a username and a password is met with something other than 0 – in other words, if a search for a username and password that matches both of them yields a result:

```
def login(self):
    self.user = self.entry_user.text()
    self.password = self.entry_pass.text()
    con = lite.connect("festivalFinder")
    result = con.execute("SELECT * FROM tbl_users WHERE user_name = ? AND pass = ?", (self.user, self.password))
    if int(len(result.fetchall()) > 0):
        self.wdw_home.show()
        self.hide()
    else:
        self.txt_msg.setText("Wrong username or password")
    con.close()
```

This code continued to develop, including writing more code for the register and forgot windows that it would link to -

```
class MyWindowClass(QtGui.QMainWindow, form_class):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        # QString = type("")
        self.load_data()
        self.user = ""
        self.password = ""
        self.setupUi(self)
        self.wdw_home = FestivalHome()
        self.wdw_register = FestivalRegister()
        self.btn_login.clicked.connect(self.login)
        self.btn_close.clicked.connect(self.close)# binds to the buttons
        self.btn_forgot.clicked.connect(self.forgot)
        self.btn_register.clicked.connect(self.register)

    def register(self):
        self.wdw_register.show()
        self.hide()

    def close(self):
        sys.exit(app.exec_())

    def forgot(self):
        print("hello world")

    def load_data(self):
        con = lite.connect('login')
        cur = con.cursor()
        u ='SELECT user_name from tbl_users'
        cur.execute(u)
        usernames = list(cur.fetchall())
        p = 'SELECT pass from tbl_users'
        cur.execute(p)
        passes = list(cur.fetchall())
        for i in range(0, (len(passes))):
            print(usernames[i])
            print(passes[i])

    def login(self):
```

```
    def login(self):
        self.user = self.entry_user.text()
        self.password = self.entry_pass.text()
        print(self.user, self.password)
        print(real_user, real_password)
        if self.user == real_user:
            if self.password == real_password:
                self.wdw_home.show()
                self.hide()
            else:
                self.txt_msg.setText("Wrong password")
        else:
            self.txt_msg.setText("Wrong username")


class FestivalRegister(QtGui.QMainWindow, wdw_register):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.worldhello()
        self.setupUi(self)

    def worldhello(self):
        print("helloworldRegister")


class FestivalHome(QtGui.QMainWindow, wdw_home):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.helloworld()
        self.setupUi(self)

    def helloworld(self):
        print("helloworldRegister")


app = QtGui.QApplication(sys.argv)
myWindow = MyWindowClass(None)
myWindow.show()

app.exec_()
```

This then evolves to a full-fleshed out login class later in the development process –

```
21      class FestivalLogin(QtGui.QMainWindow, wdw_login):
22          def __init__(self, parent=None):
23              QtGui.QMainWindow.__init__(self, parent)
24              self.user = ""
25              self.password = ""
26              self.setupUi(self)
27              self.btn_login.clicked.connect(self.login)
28              self.btn_close.clicked.connect(self.close)  # binds to the buttons
29              self.btn_forgot.clicked.connect(self.forgot)
30              self.btn_register.clicked.connect(self.register)
31
32          def register(self):
33              wdw_register.show()
34              self.hide()
35
36          def forgot(self):
37              wdw_forgot.show()
38              self.hide()
39
40          def login(self):
41              self.user = self.entry_user.text()
42              self.password = self.entry_pass.text()
43              con = lite.connect("festivalfinder")
44              hashed_pass = (hashlib.md5(self.password.encode()).hexdigest())
45              result = con.execute("SELECT * FROM tbl_users WHERE user = ? AND pass = ?",
46                                   (self.user, hashed_pass))
47              if int(len(result.fetchall()) > 0):
48                  FestivalMenu.fill_gui(wdw_menu, self.user)
49                  wdw_menu.show()
50                  self.hide()
51              else:
52                  self.txt_msg.setText("Wrong username or password")
53              con.close()
54
```

Which is further refined with defining administrator permissions, filling the GUI of other windows and further validation and comments for future maintainers of the code. As the data of the password is now hashed, a major change, this has to change the way the password is compared to the data in the table:

```
class FestivalLogin(QtGui.QMainWindow, wdw_login):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.str_user = ""  # setting up user variables
        self.str_password = ""
        self.str_hashed_pass = ""
        self.setupUi(self)
        self.btn_login.clicked.connect(self.login)
        self.btn_close.clicked.connect(self.close)  # binds to the buttons (clicking calls the function mentioned)
        self.btn_forgot.clicked.connect(self.forgot)
        self.btn_register.clicked.connect(self.register)

    def register(self):
        wdw_register.show()
        self.hide()

    def forgot(self):
        wdw_forgot.show()
        self.hide()

    def login(self):
        self.str_user = self.entry_user.text()  # takes in the values in the user, password entryboxes and sets them
        wdw_admin.str_user = self.str_user  # assigns a user for the admin window to know who not to show
        self.str_password = self.entry_pass.text()  # to a string variable value
        if self.str_password or self.str_user != "":
            con = lite.connect("festivalFinder")  # connects to the database
            self.str_hashed_pass = (hashlib.md5(self.str_password.encode()).hexdigest())  # hashes the password with md5
            result = con.execute("SELECT * FROM tbl_users WHERE user = ? AND pass = ?",
                                 (self.str_user, self.str_hashed_pass)).fetchall()  # searches for a user with the user
            # name and hashed password identical to the one entered
            debug(result)
            if len(result) > 0:  # if the search yields a user
                FestivalMenu.fill_gui(wdw_menu, self.str_user)
                FestivalDelete.fill_gui(wdw_delete)
                if result[0][4] == 2:  # checks if permission level is 2 or 1, 2 being admin
                    wdw_menu.int_perm = 1
                    wdw_admin.show()
                else:
                    wdw_menu.show()
                self.hide()
            else:  # search yielding no user, therefore wrong username or password
                self.txt_msg.setText("Wrong username or password")
            con.close()
        else:
            self.txt_msg.setText("data must be entered")
```

During the development of the Login window, I was also working on two closely related windows, as they all provide the first point at which the user uses in the program – login, register and forgot password.

For the register window however, it was clear I had hit a dead-end during development – how would I regulate the information entered – test the strength of passwords, test for input, test for strength of usernames or already existing accounts? I resolved eventually to use the hugely popular tool "Regular Expressions" to check for exact matches of string criteria.

Before Regular expressions – I could only enforce character/string length here, however when I implemented regular expressions I could enforce multiple rules at once.

Before

```
class FestivalRegister(QtGui.QMainWindow, wdw_register):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.worldhello()
        self.setupUi(self)
        self.btn_submit.clicked.connect(self.submit)
        self.btn_close.clicked.connect(self.close)  # binds to the buttons

    def close(self):
        wdw_login.show()
        self.hide()

    def submit(self):
        self.user = self.entry_username.text()
        self.passw = self.entry_password.text()
        self.passw2 = self.entry_password2.text()
        if len(self.user) < 5:
            self.lbl_msg.setText("username must be more than 5 chars in length")
        else:
            self.lbl_msg.setText("username accepted")
            if len(self.passw) > 6:
                if self.passw != self.passw2:
                    self.lbl_msg.setText("passwords don't match")
                else:
                    self.lbl_msg.setText("passwords and usernames ok")
            else:
                self.lbl_msg.setText("passwords must be more than 6 chars")

    def worldhello(self):
        print("helloworldRegister")
```

After (First Implementation of Regular Expressions)

```
class FestivalRegister(QtGui.QMainWindow, wdw_register):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        print("Register Window opened")
        self.setupUi(self)
        self.user = ""
        self.passw = ""
        self.passw2 = ""
        self.btn_submit.clicked.connect(self.submit)
        self.btn_close.clicked.connect(self.closewin)  # binds to the buttons

    def closewin(self):
        wdw_login.show()
        self.hide()

    def submit(self):
        self.user = self.entry_username.text()
        self.passw = self.entry_password.text()
        self.passw2 = self.entry_password2.text()
        reg_user = r"^[a-zA-Z0-9][a-zA-Z0-9_]{4,12}$"
        # starts with alphanumeric, can only contain alphanumeric or underscore with length 4-12
        reg_pass = r"(?!^[0-9]*$)(?!^[a-zA-Z]*$)^([a-zA-Z0-9]{6,12})$"
        # (6-12 chars, at least one lower,upper and number and no specials)
        if re.match(reg_pass, self.passw) and re.match(reg_user, self.user):
            self.lbl_msg.setText("success")
            if self.passw == self.passw2:
                self.lbl_msg.setText("passwords successfully match")
            else:
                self.lbl_msg.setText("passwords don't match")
        else:
            self.lbl_msg.setText("password or username or pass doesn't match form")
```

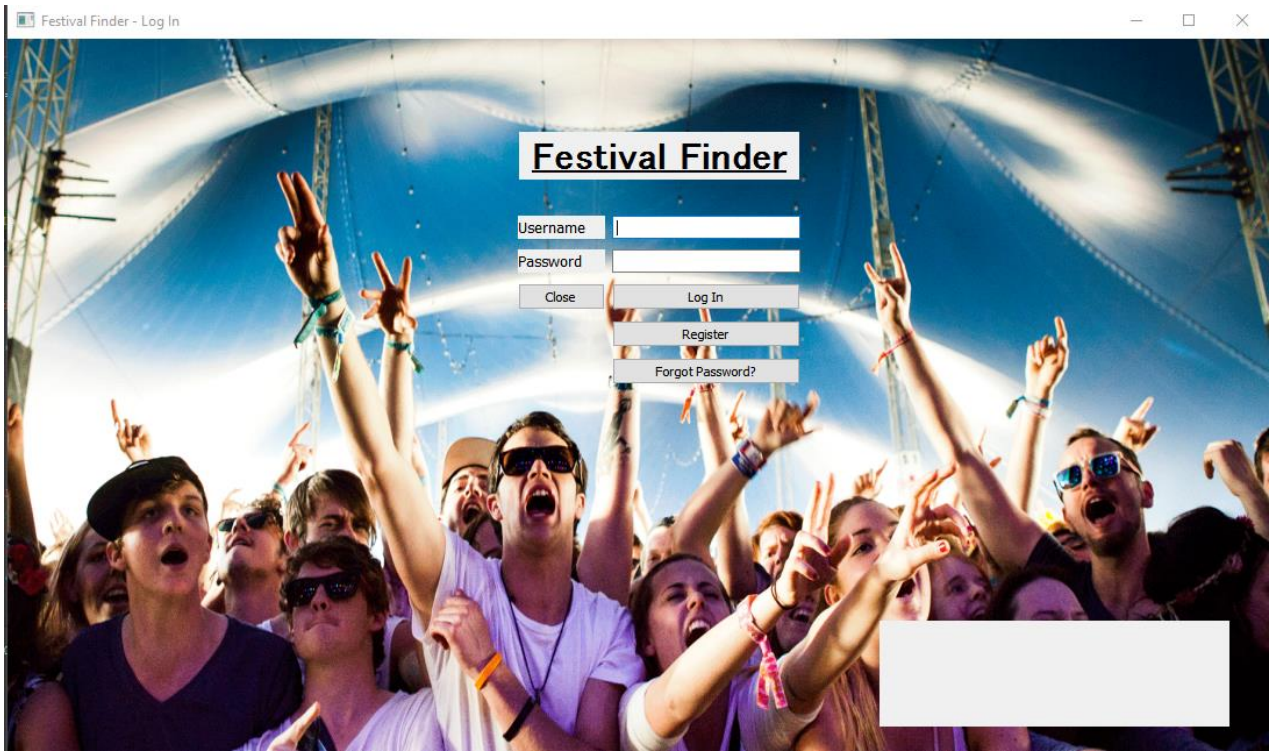
I further developed this section of code by not allowing blank fields or new users to be set up with existing user data, such as emails or username (This forms the final form of Register) –

```
def submit(self):
    self.str_user = self.entry_username.text()  # takes in the items in the entries as variables
    self.str_password = self.entry_password.text()
    self.str_password_2 = self.entry_password2.text()
    str_email = self.entry_email.text()
    debug(self.str_password)
    if self.str_password or self.str_password_2 or self.str_user or str_email != "":
        reg_user = r"^[a-zA-Z0-9][a-zA-Z0-9_]{4,12}$"
        # starts with alphanumeric, can only contain alphanumeric or underscore with length 4-12
        reg_pass = r"(?!^[0-9]*$)(?!^[a-zA-Z]*$)^([a-zA-Z0-9]{6,12})$"
        # (6-12 chars, at least one lower,upper and number and no specials)
        con = lite.connect("festivalFinder")  # connects to the database
        result = con.execute("SELECT * FROM tbl_users WHERE email = ?", (str_email,)).fetchall()
        existing_user = con.execute("SELECT * FROM tbl_users WHERE user = ?", (self.str_user,)).fetchall()
        debug(result)
        if len(result) > 0:  # if there is already an existing user with that email (search yields)
            self.lbl_msg.setText("Email already in use, try another")
        elif len(existing_user) > 0:  # if there is an existing user with that username
            self.lbl_msg.setText("Username already in use, try another")
        else:
            if re.match(reg_pass, self.str_password) and re.match(reg_user, self.str_user):
                # checks if the variables user and pass match the regex criteria above
                self.lbl_msg.setText("success")
                if self.str_password == self.str_password_2:
                    # if the passwords are the same on both attempts
                    self.lbl_msg.setText("passwords successfully match - Account Created")
                    hashed_pass = (hashlib.md5(self.str_password.encode())).hexdigest()
                    # hashes the password
                    con.execute("INSERT INTO tbl_users (user, pass, email) VALUES (?,?,?)",
                                (self.str_user, hashed_pass, str_email))
                    # creates a new user
                    con.commit()
                else:
                    self.lbl_msg.setText("passwords don't match")
            else:
                self.lbl_msg.setText("password or username or pass doesn't match form")
        con.close()
    else:
        self.lbl_msg.setText("fields must not be left blank")
```

Testing to Inform Development (Initial Tests of some Classes)

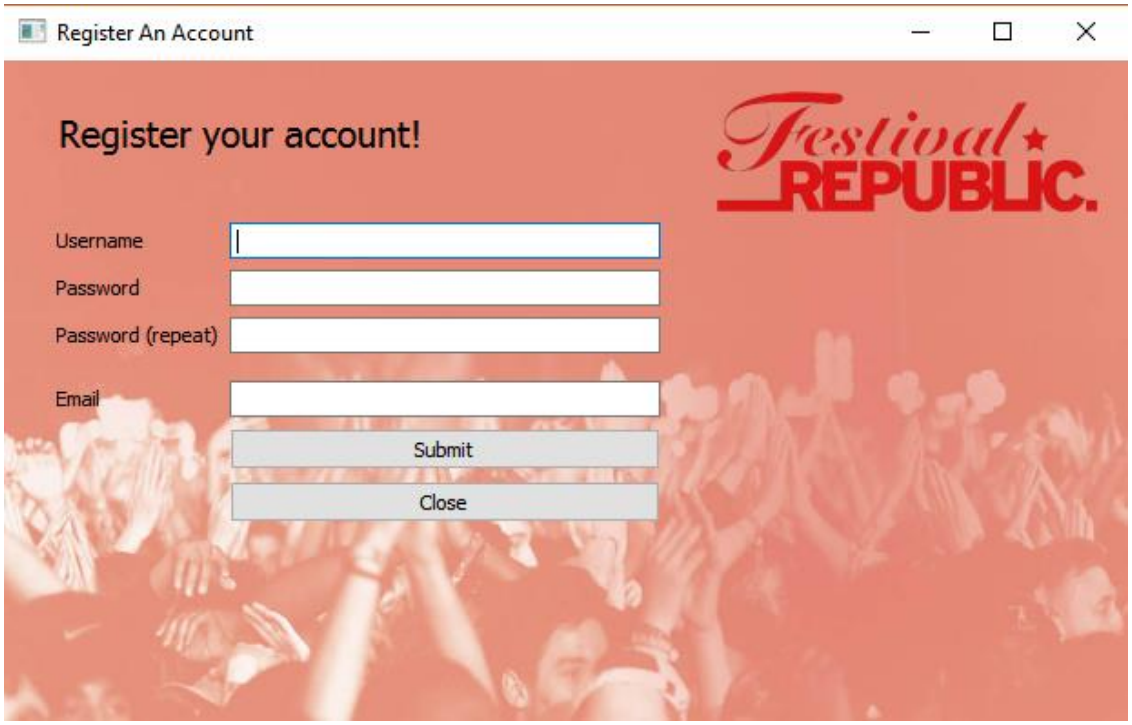
Login Testing

The login screen had several links – the buttons all linked to separate windows:

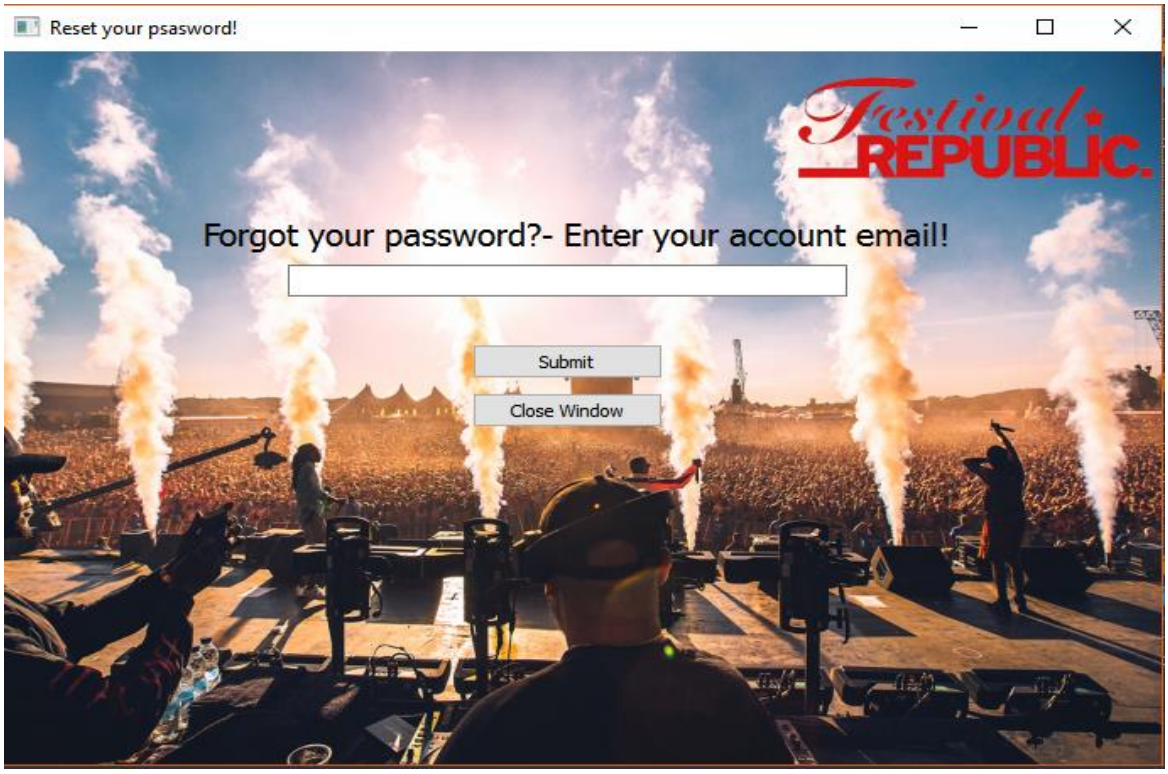


First we test that the buttons link to the correct windows –

The Register Button successfully links to the Register window:



The forgot button links to the forgot window –



When stress testing the original GUI, repeatedly pressing the buttons yielded no crashes – I ensured that input buttons that would be allowed to repeatedly be pressed did little processing and therefore would not induce a crash.

Logging in with user data (logging in with hashed information) –

I use two user accounts, an admin level (user:complex, password:Imaginary1) and a normal level (user:George, password:Wellyb00t).

user	pass	email
a	0cc175b9c0f1b6a831c399e269772661	majesticseabass@gmail.com
admin	6f2457b608622214f753c8522bf63a38	cyberhylian@gmail.com
complex	0d91a55ac531df50ede54f22b23ed9ec	philnsjspam@talktalk.net
jdsalinger	e7d8993bf6de68a2d4e3a8a24e51f4fd	ltmattwelly@gmail.com
jessica	db1a0dee243af63d8db3703ab7d80e53	jess@4818.com
George	30d2e8942cca845bc66236b4de28ae07	georgewellingtonwork@gmail.com

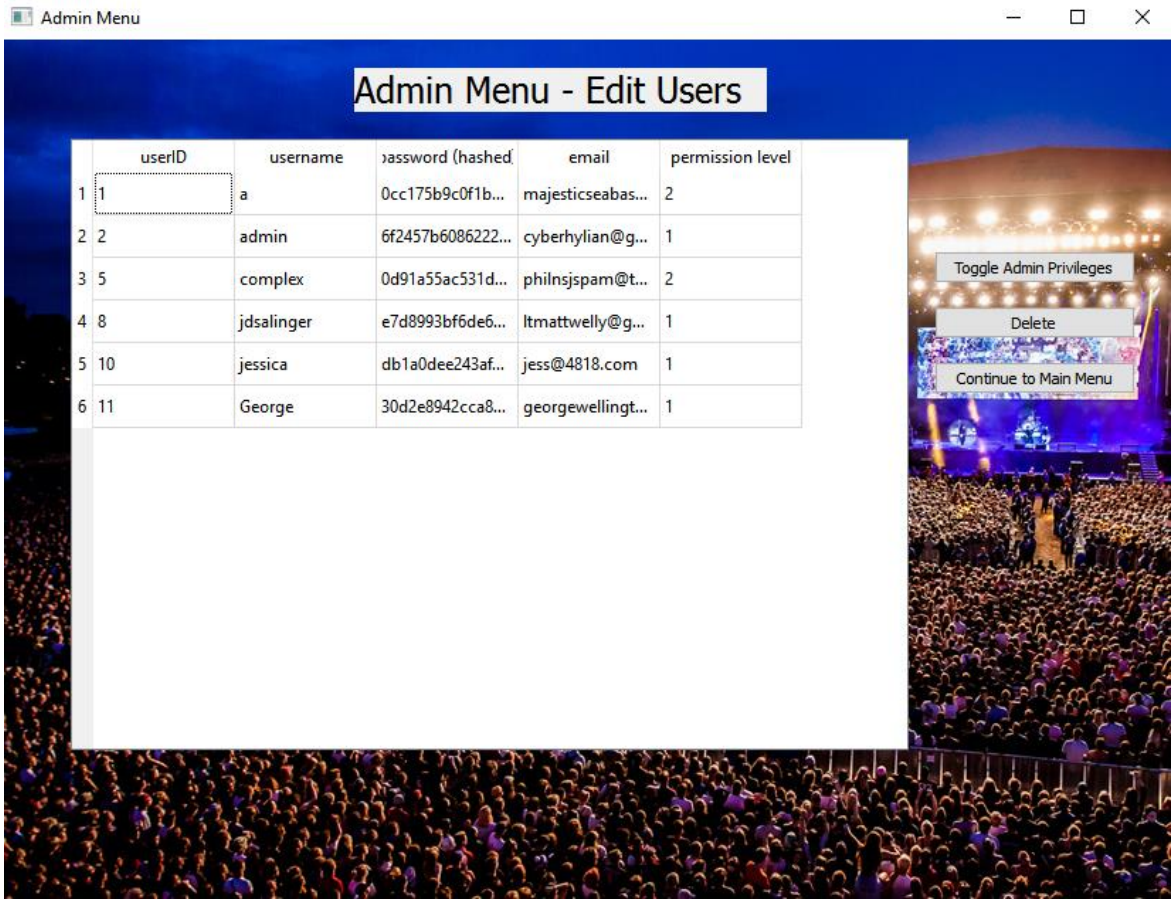
I unhashed the associated pass entries for the two accounts to demonstrate the information -

```
0d91a55ac531df50ede54f22b23ed9ec MD5 : Imaginary1
```

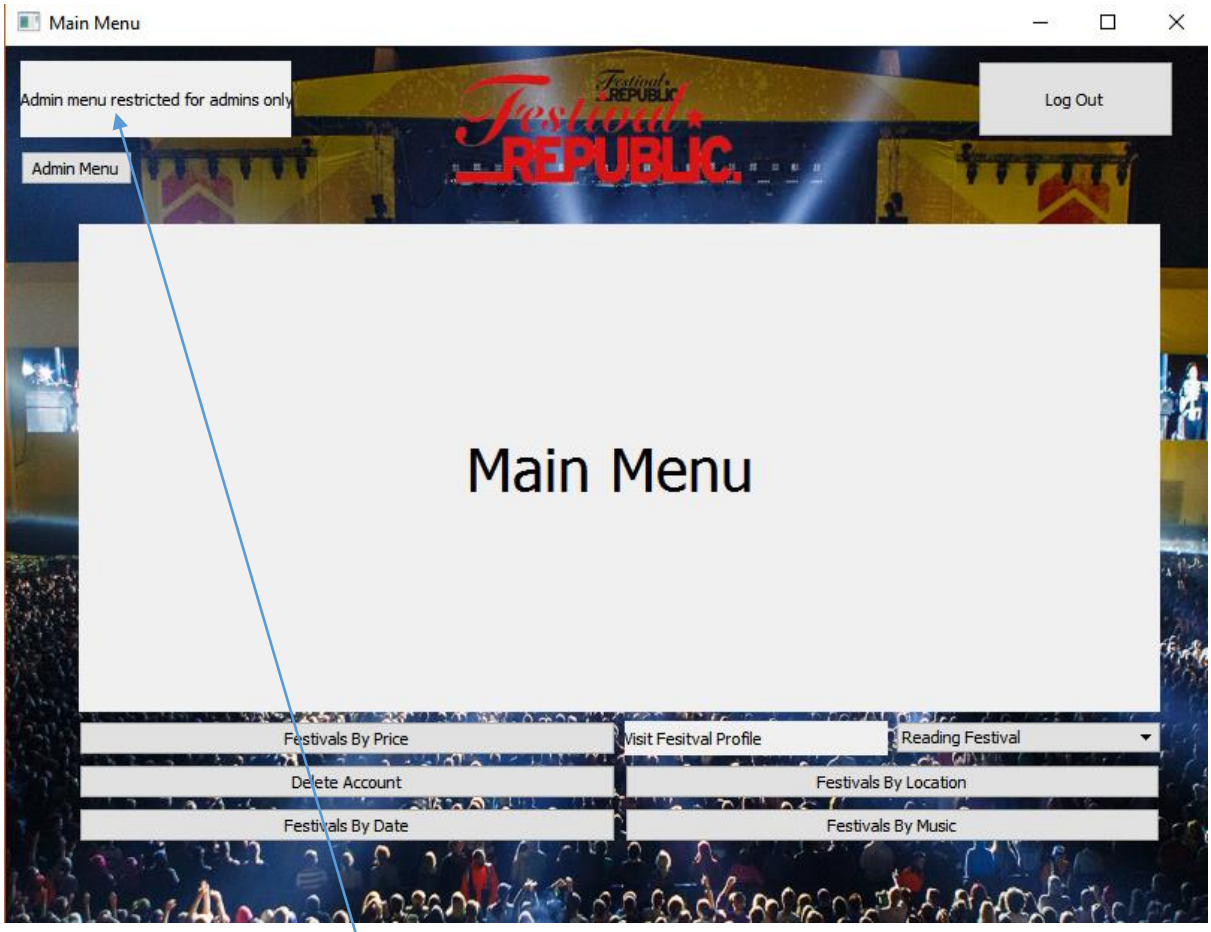
```
30d2e8942cca845bc66236b4de28ae07 MD5 : Wellyb00t
```

Logging in with incorrect information yields no login –

Logging in with administrator level username and password yields a direct link to the administrator menu:



Logging in with normal user level information only yields the main menu, and when the administrator button is pressed the user is informed they cannot access it –



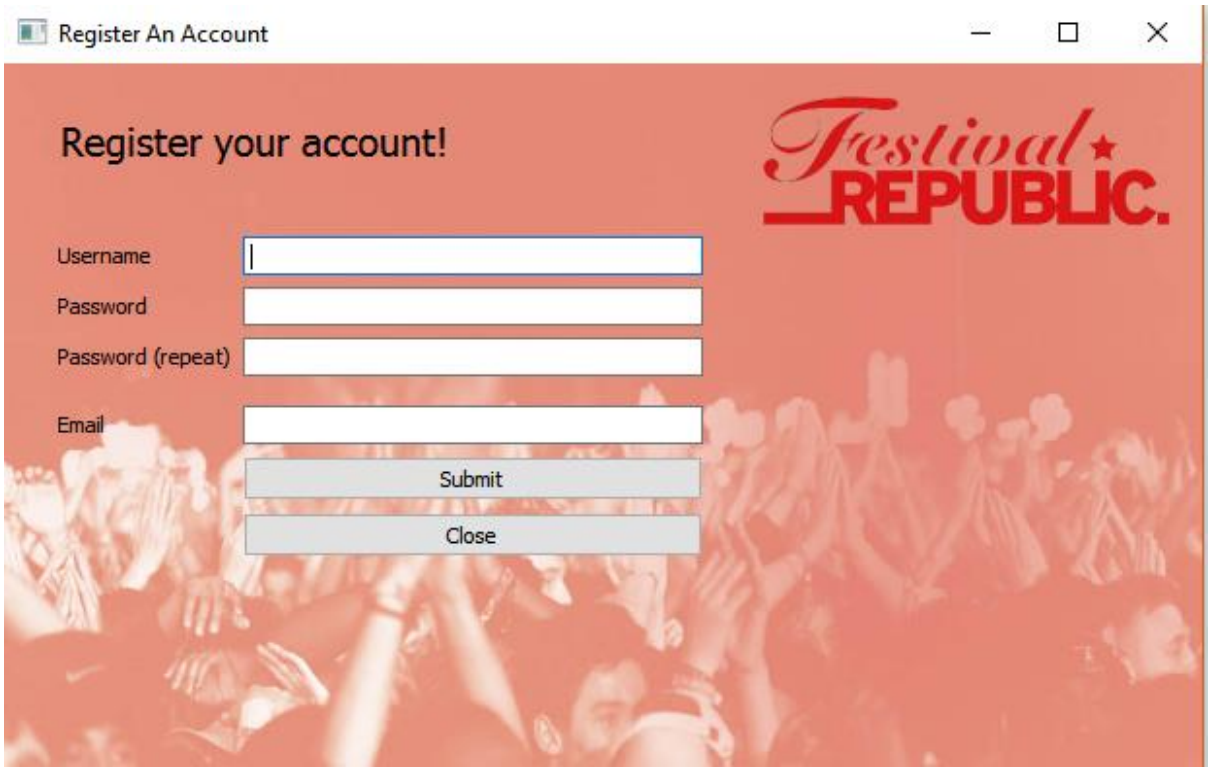
When the administrator button is clicked, the message box tells the user that "Admin menu restricted for admins only"

Testing with a correct username and incorrect password does not allow access either:

And with a correct password and non-recorded username it also blocks access ("a" is a recorded password for the account with the username "a" for testing purposes):

Register Class Testing

This window's close button successfully links back to the logon screen –



Also, stress testing the available buttons that do not change window (btn_submit or just "Submit" on the GUI) goes straight to one line of code- asking users to enter information – and therefore does not require enough processing to make the program crash.

Entering a username and password immediately checks for strength, to better illustrate this I tested it outside the program (using the regex involved in the form):

Regular Expression Testing –

```
Enter passwordpseudo
invalid
Enter passwordPSEUDO
invalid
Enter password12pseudo
invalid
Enter password12PSEUDO
invalid
Enter passwordpseudo12
invalid
Enter passwordPSEUDO12
invalid
Enter password1Pseudo02
matches
Enter passwordpseud12dOAB
matches
Enter passwordOABasdf123
matches
Enter passwordoabasf123
invalid
Enter password
```

```
import re
regex = r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]{6,16}$"
while 1 == 1:
    test = input("Enter password")
    if re.match(regex, test):
        print("matches")
    else:
        print("invalid")
```

I developed a separate program specifically for testing the regex queries I use in the program. The program itself asked repeatedly for input and when given, outputs "invalid" if it does not match and "match" if it does.

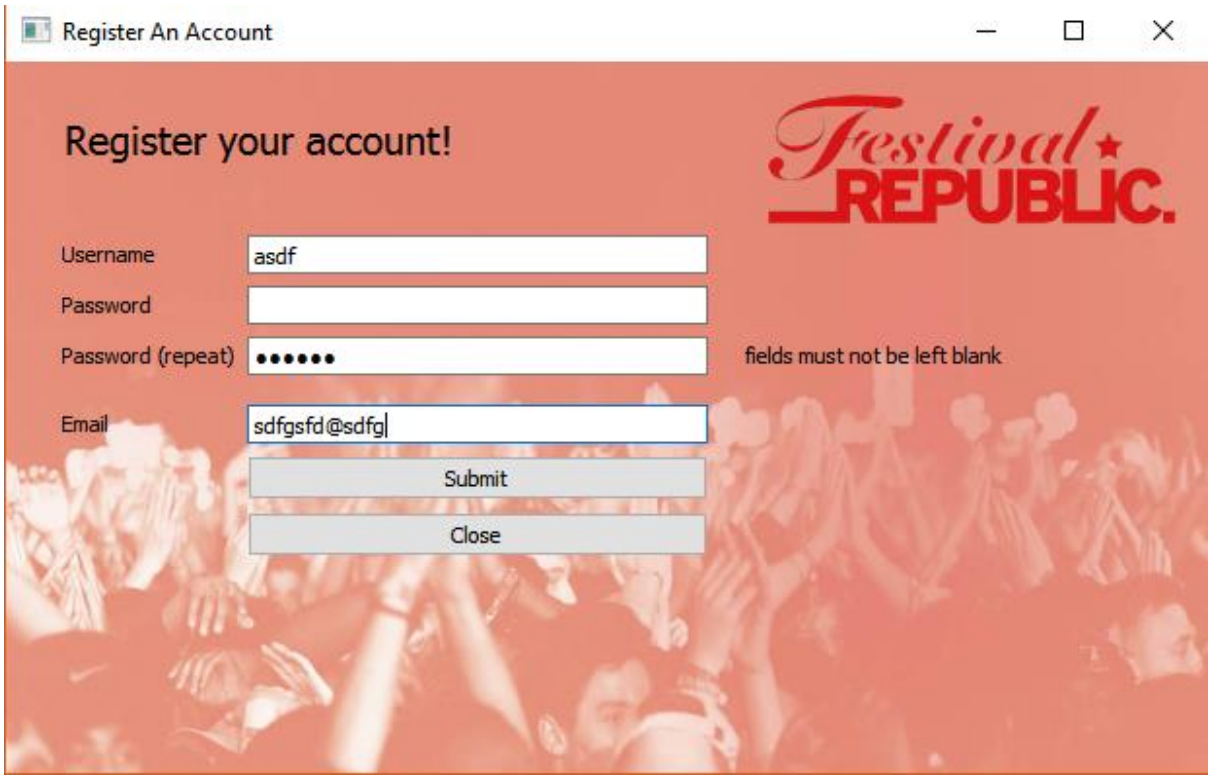
```
Enter usernamepseudo
matches
Enter usernamepseu
invalid
Enter usernamepseudo
matches
Enter usernamepseud
matches
Enter usernameabaABABA
matches
Enter usernameasdf_sadf
matches
Enter username_sadfsdaf
invalid
Enter username_asdf
invalid
Enter usernamesadfas_
matches
Enter username$%$fasdfsd
invalid
Enter username"£$"£$sadf
invalid
Enter username123sandy
matches
```

```python
import re
regex = r"^[a-zA-Z0-9][a-zA-Z0-9_]{4,12}$"
while 1 == 1:
    test = input("Enter username")
    if re.match(regex, test):
        print("matches")
    else:
        print("invalid")
```

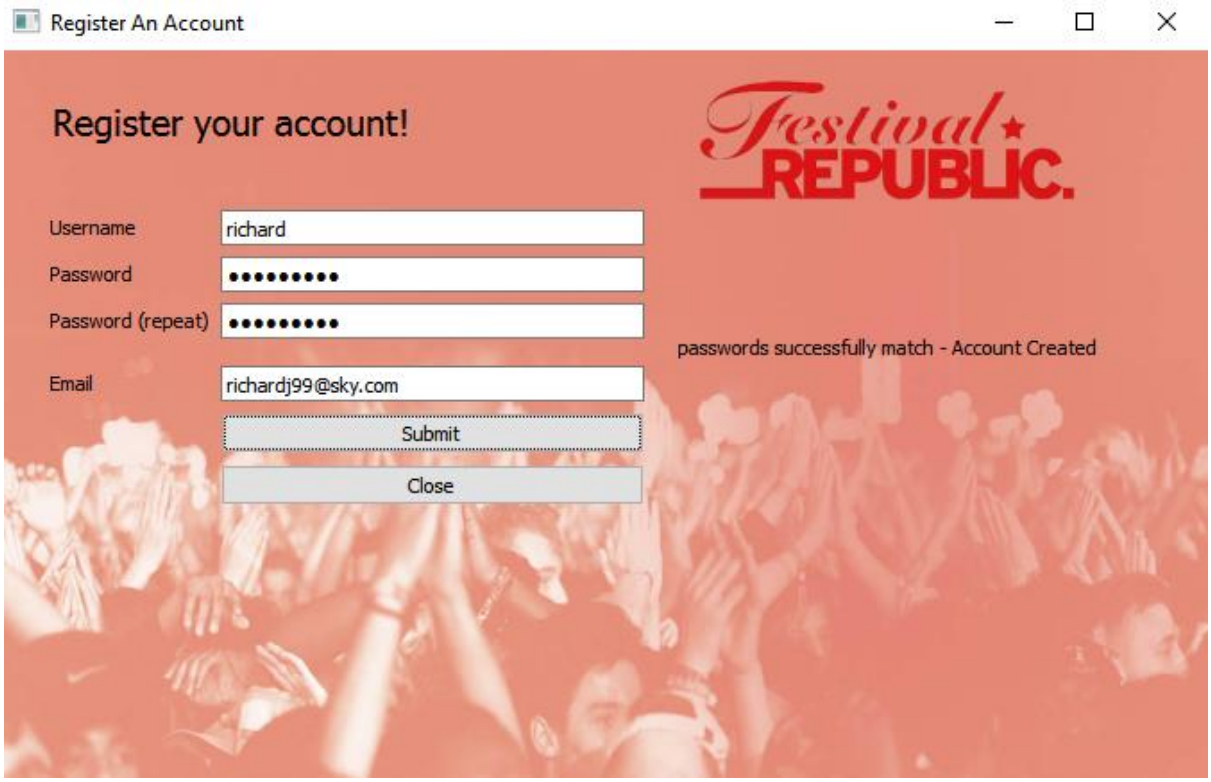
I developed a separate program specifically for testing the regex queries I use in the program. The program itself asked repeatedly for input and when given, outputs “invalid” if it does not match and “match” if it does.

Any username or password not matching the strength criteria of the regex will be rejected by the program and looks like this:

Any empty input box left in the register window when the submit button is pressed will cause the program to ask for fields not to be left blank:

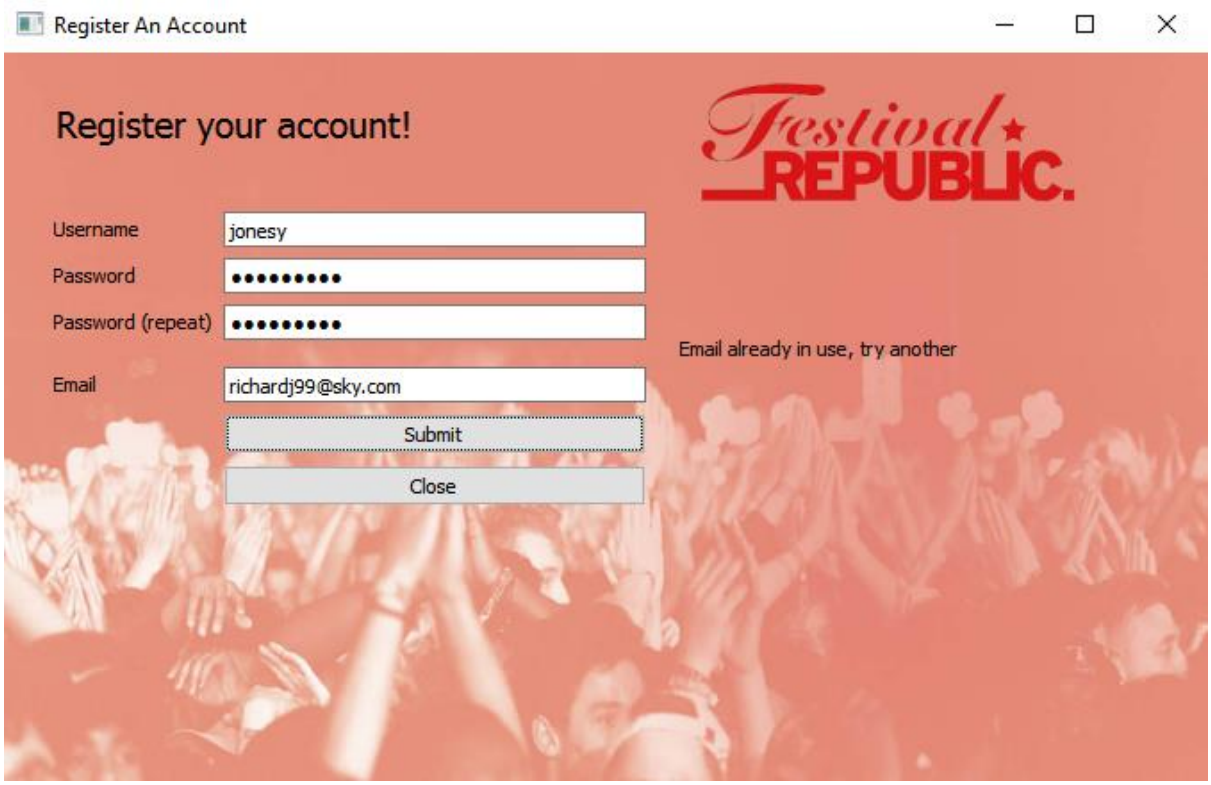


Using a non-used username and non-used email and conforming username and password fields would yield a new account:

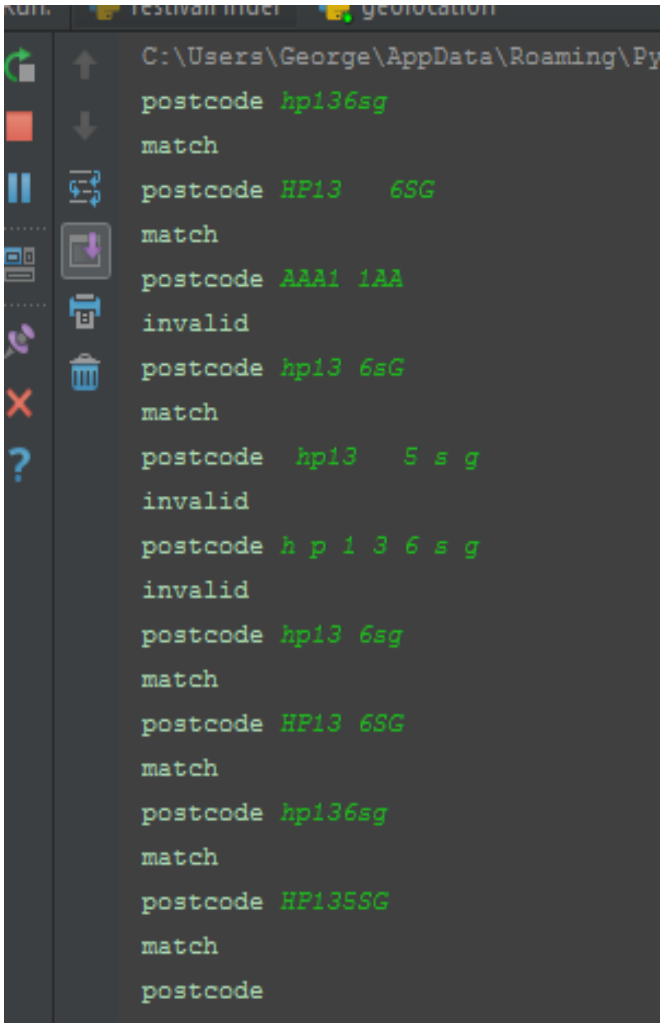


Whereas a new account with a username that is in use would create this error:

And trying to create an account with an email that is already is in use with another user would output this:



Testing Regex out of main program



I developed a separate program specifically for testing the regex queries I use in the program. The program itself asked repeatedly for input and when given, outputs “invalid” if it does not match and “match” if it does.

```
# Testing the regex for postcodes
import re
regex_postcode = "^(([gG][iI][rR] {0,}0[aA]{2})|((([a-pr-uwyzA-PR-UWYZ][a-hk-yA-HK-Y]?[0-9][0-9]?)|" \
                 "(([a-pr-uwyzA-PR-UWYZ][0-9][a-hjkstuwA-HJKSTUW])|([a-pr-uwyzA-PR-UWYZ][a-hk-yA-HK-Y][0-9]" \
                 "[abehmnprv-yABEHMNPRV-Y]))) {0,}[0-9][abd-hjlnp-uw-zABD-HJLNP-UW-Z]{2}))$"
while 1 == 1:
    post = input("Enter postcode ")
    if re.match(regex_postcode, post):
        print("Postcode matches")
    else:
        print("Postcode is invalid")
```

Fully Finished, Commented Code of the Final Program (Proof of Programming Constructs eg. Debugger, commenting, inner joins)

```
# Festival Finder Program // Year 13 Coursework
# George Wellington RGSHW (2016/2017)
import re
import sys
import emailpass
import sqlite3 as lite
import requests
import googlemap
import hashlib
import PyQt4
import random
import ctypes
from PyQt4 import QtGui, uic, QtCore


debug_mode = 0
wdw_login = uic.loadUiType('festLogin.ui')[0]  # Load the UI files and sets them to the wdw_ objects
wdw_location = uic.loadUiType('location.ui')[0]  # wdw = window
wdw_register = uic.loadUiType('register.ui')[0]
wdw_forgot = uic.loadUiType('forgot.ui')[0]
wdw_alert = uic.loadUiType('postalert.ui')[0]
wdw_profile = uic.loadUiType('profile.ui')[0]
wdw_menu = uic.loadUiType('menu.ui')[0]
wdw_cost = uic.loadUiType('cost.ui')[0]
wdw_delete = uic.loadUiType('delete.ui')[0]
wdw_date = uic.loadUiType('date.ui')[0]
wdw_music = uic.loadUiType('music.ui')[0]
wdw_buy = uic.loadUiType('buy.ui')[0]
wdw_admin = uic.loadUiType('admin.ui')[0]
```

^ Import statements of used libraries and setting up the GUI windows for PyQT (Start of Code)

v End of code, calling the different classes to the GUI widget windows and a debugging function for use of debugger (if debug mode is active, anything passed into debugger is printed)

```
def debug(test):
    if debug_mode == 1:
        print(test)

app = QtGui.QApplication(sys.argv)
wdw_menu = FestivalMenu()
wdw_admin = FestivalAdmin()
wdw_buy = FestivalBuy()
wdw_music = FestivalMusic()
wdw_date = FestivalDate()
wdw_delete = FestivalDelete()
wdw_cost = FestivalCost()
wdw_alert = FestivalAlert()
wdw_location = FestivalLocation()
wdw_register = FestivalRegister()
wdw_profile = FestivalProfile()
wdw_forgot = FestivalForgot()
wdw_login = FestivalLogin(None)
wdw_login.show()
app.exec_()
```

Ie. Passing in result is a way of testing the returned value of that query as if debugging mode is set to one, the result is printed

This also involves passing parameters and variables of different scopes

```
result = con.execute("SELECT * FROM tbl_users WHERE email = ?", (str_email,)).fetchall()
existing_user = con.execute("SELECT * FROM tbl_users WHERE user = ?", (self.str_user,)).fetchall()
debug(result)
```

```
if debug_mode == 1:
    print("window: ", window, "variable: ", test)
```

This provides a printout of the window and the variable itself – meaning I can inspect the finished code through its variable and understand which class it's being used in (window is passed in as self)

```
window:  <__main__.FestivalLocation object at 0x05108620> variable:
    <!DOCTYPE html>
    <html>
    <head>
      <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
      <title>Google Maps Multiple Markers</title>
      <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyA3rveo5Ju7hrpCb-MGdjaf
    </head>
    <body>
      <div id="map" style="width: 785px; height: 605px;"></div>

      <script type="text/javascript">
        var locations = [[['Reading Festival', {'lat': 51.4636729, 'lng': -0.98492499999999

        ];

        var map = new google.maps.Map(document.getElementById('map'), {
          zoom: 5,
          center: new google.maps.LatLng(51,0.1),
          mapTypeId: google.maps.MapTypeId.hybrid,
          mapTypeControl: false,
          scaleControl: false,
          streetViewControl: false
        });

        var infowindow = new google.maps.InfoWindow();

        var marker, i;
        for (i = 0; i < 6; i++) {
          marker = new google.maps.Marker({
            animation: google.maps.Animation.DROP,
            position: locations[0][i][1],
            label: locations[0][i][0].charAt(0),
            map: map,
          });
        if (0 == -1)
         var marker_home = new google.maps.Marker({
            animation: google.maps.Animation.DROP,
            position: locations[0][5][1],
            icon : 'http://maps.google.com/mapfiles/ms/icons/green-dot.png',
            map: map,
         })
         var cityCircle = new google.maps.Circle({
              strokeColor: '#FF0000',
              strokeOpacity: 0.1,
              strokeWeight: 0.5,
```

```
window:  <__main__.FestivalLocation object at 0x05108620> variable:  [['Reading Festival', {'lat': 51.4636729, 'lng': -0.9849249999999999}], ['Leeds Festiv
window:  <__main__.FestivalLocation object at 0x05108620> variable:  [('RG1 8EQ',), ('LS23 6ND',), ('NR34 8AQ',), ('DE74 2RP',), ('N4 1EE',), ('N4 1EF',)]
```

```
CONSTANT_GOOGLE_MILE = 1562.5  # the value for which google uses in circles to represent a mile radius

def plot(locations, counter, ishome, radius):
    # function takes in the array locations, counter (which corresponds to the length of locations and therefore is
    # the amount of times the plotting function is looped. The function returns the formatted string with the correct
    # array and formatting.
    code = """
    <!DOCTYPE html>
    <html>
    <head>
      <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
      <title>Google Maps Multiple Markers</title>
      <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyA3rvec5Ju7hrpCb-MGdjafEIM5wD7u8M8"></script>
    </head>
    <body>
      <div id="map" style="width: 785px; height: 605px;"></div>

      <script type="text/javascript">
        var locations = [%s

        ];

        var map = new google.maps.Map(document.getElementById('map'), {
          zoom: 5,
          center: new google.maps.LatLng(51,0.1),
          mapTypeId: google.maps.MapTypeId.hybrid,
          mapTypeControl: false,
          scaleControl: false,
          streetViewControl: false
        });

        var infowindow = new google.maps.InfoWindow();

        var marker, i;
        for (i = 0; i < %s; i++) {
          marker = new google.maps.Marker({
            animation: google.maps.Animation.DROP,
            position: locations[0][i][1],
            label: locations[0][i][0].charAt(0),
            map: map,
          });
        if (%s == -1)
         var marker_home = new google.maps.Marker({
            animation: google.maps.Animation.DROP,
            position: locations[0][%s][1],
            icon : 'http://maps.google.com/mapfiles/ms/icons/green-dot.png',
            map: map,
         })
         var cityCircle = new google.maps.Circle({
              strokeColor: '#FF0000',
              strokeOpacity: 0.1,
              strokeWeight: 0.5,
              fillColor: '#7ec0ee',
              fillOpacity: 0.11,
              map: map,
              center: locations[0][%s][1],
              radius: (%s * %s)

         });

         google.maps.event.addListener(marker, 'click', (function(marker, i) {
           return function() {
             infowindow.setContent(locations[0][i][0]);
             infowindow.open(map, marker);
           }
         })(marker, i));

        }
      </script>
    </body>
    </html>
    """ % (locations, counter + ishome, ishome, counter - 1, counter - 1, CONSTANT_GOOGLE_MILE, radius)
    return code
```

This javascript string involves a snippet with a capitalised constant used, is an example of a function (returns the string code) whereas the other parts of my program and mostly procedures and also a massive example of parameter passing and inserting into strings using %s.

```
class FestivalLogin(QtGui.QMainWindow, wdw_login):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.str_user = ""  # setting up user variables
        self.str_password = ""
        self.str_hashed_pass = ""
        self.setupUi(self)
        self.btn_login.clicked.connect(self.login)
        self.btn_close.clicked.connect(self.close)  # binds to the buttons (clicking calls the function mentioned)
        self.btn_forgot.clicked.connect(self.forgot)
        self.btn_register.clicked.connect(self.register)

    def register(self):
        wdw_register.show()
        self.hide()

    def forgot(self):
        wdw_forgot.show()
        self.hide()

    def login(self):
        self.str_user = self.entry_user.text()  # takes in the values in the user, password entryboxes and sets them
        wdw_admin.str_user = self.str_user  # assigns a user for the admin window to know who not to show
        self.str_password = self.entry_pass.text()  # to a string variable value
        if self.str_password or self.str_user != "":
            con = lite.connect("festivalFinder")  # connects to the database
            self.str_hashed_pass = (hashlib.md5(self.str_password.encode()).hexdigest())  # hashes the password with md5
            result = con.execute("SELECT * FROM tbl_users WHERE user = ? AND pass = ?",
                                 (self.str_user, self.str_hashed_pass)).fetchall()  # searches for a user with the user
            # name and hashed password identical to the one entered
            debug(result)
            if len(result) > 0:  # if the search yields a user
                FestivalMenu.fill_gui(wdw_menu, self.str_user)
                FestivalDelete.fill_gui(wdw_delete)
                if result[0][4] == 2:  # checks if permission level is 2 or 1, 2 being admin
                    wdw_menu.int_perm = 1
                    wdw_admin.show()
                else:
                    wdw_menu.show()
                self.hide()
            else:  # search yielding no user, therefore wrong username or password
                self.txt_msg.setText("Wrong username or password")
            con.close()
        else:
            self.txt_msg.setText("data must be entered")
```

This section involves different permissions levels, inheritance, multiple inheritance, logging in and out with hashed data, polymorphism, procedures, SQL parameter queries, Hungarian notation and variables of different scopes.

```
class FestivalRegister(QtGui.QMainWindow, wdw_register):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.str_user = ""
        self.str_password = ""
        self.str_password_2 = ""
        self.btn_submit.clicked.connect(self.submit)
        self.btn_close.clicked.connect(self.close_window)  # binds to the buttons (clicking calls the function)

    def close_window(self):
        wdw_login.show()
        self.hide()
```

```
def submit(self):
    self.str_user = self.entry_username.text()  # takes in the items in the entries as variables
    self.str_password = self.entry_password.text()
    self.str_password_2 = self.entry_password2.text()
    str_email = self.entry_email.text()
    debug(self.str_password)
    if self.str_password or self.str_password_2 or self.str_user or str_email != "":
        reg_user = r"^[a-zA-Z0-9][a-zA-Z0-9_]{4,12}$"
        # starts with alphanumeric, can only contain alphanumeric or underscore with length 4-12
        reg_pass = r"(?!^[0-9]*$)(?!^[a-zA-Z]*$)^([a-zA-Z0-9]{6,12})$"
        # (6-12 chars, at least one lower,upper and number and no specials)
        con = lite.connect("festivalFinder")  # connects to the database
        result = con.execute("SELECT * FROM tbl_users WHERE email = ?", (str_email,)).fetchall()
        existing_user = con.execute("SELECT * FROM tbl_users WHERE user = ?", (self.str_user,)).fetchall()
        debug(result)
        if len(result) > 0:  # if there is already an existing user with that email (search yields)
            self.lbl_msg.setText("Email already in use, try another")
        elif len(existing_user) > 0:  # if there is an existing user with that username
            self.lbl_msg.setText("Username already in use, try another")
        else:
            if re.match(reg_pass, self.str_password) and re.match(reg_user, self.str_user):
                # checks if the variables user and pass match the regex criteria above
                self.lbl_msg.setText("success")
                if self.str_password == self.str_password_2:
                    # if the passwords are the same on both attempts
                    self.lbl_msg.setText("passwords successfully match - Account Created")
                    hashed_pass = (hashlib.md5(self.str_password.encode())).hexdigest()
                    # hashes the password
                    con.execute("INSERT INTO tbl_users (user, pass, email) VALUES (?,?,?)",
                                (self.str_user, hashed_pass, str_email))
                    # creates a new user
                    con.commit()
                else:
                    self.lbl_msg.setText("passwords don't match")
            else:
                self.lbl_msg.setText("password or username or pass doesn't match form")
        con.close()
    else:
        self.lbl_msg.setText("fields must not be left blank")
```

As visible, all of my code has comments for the future maintainer of the code itself to provide a monologue and description of the functionality of the program.

This section also uses regular expressions to check passwords for strength

```
class FestivalForgot(QtGui.QMainWindow, wdw_forgot):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.btn_submit.clicked.connect(self.submit)
        self.btn_close.clicked.connect(self.close_window)  # binds to the buttons
        self.str_email = ""

    def close_window(self):
        wdw_login.show()
        self.hide()

    def submit(self):
        self.str_email = self.entry_email.text()  # takes in the items in the entries as variables
        con = lite.connect("festivalFinder")
        result = con.execute("SELECT * FROM tbl_users WHERE email = ?", (self.str_email,)).fetchall()
        # checks for a matching email
        if len(result) > 0:  # if there is a registered email
            self.lbl_msg.setText("Valid email, reset password for '" + result[0][1] + "' sent to " + self.str_email)
            # result[0][1] yields the username field of the sql query, therefore the clients name
            new_password = str(random.randint(1000, 1000000))  # the reset password is random between 1000 and 1000000
            hash_new_pass = (hashlib.md5(new_password.encode())).hexdigest()
            # hashes the newly randomized number into a password but sends an email with the unhashed version as
            # hashing occurs in the backend
            con.execute("UPDATE tbl_users SET pass = ? WHERE email=?", (hash_new_pass, self.str_email))
            con.commit()
            emailpass.send(self.str_email, result[0][1], new_password)
        else:
            self.lbl_msg.setText("invalid email, no user registered with this email")
        con.close()
```

This section includes SQL Parameter Queries here

This section also includes hashing as seen in the log on menu, but adds a reset password in its hashed form into the database.

This part has examples of imported modules being used

It also involves variables of different scopes.

```
class FestivalAdmin(QtGui.QMainWindow, wdw_admin):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.btn_menu.clicked.connect(self.open_menu)
        self.btn_admin.clicked.connect(self.admin)
        self.btn_delete.clicked.connect(self.delete)
        self.db_table.clicked.connect(self.select_user)
        self.data = ""
        self.str_user = ""  # needs the username of the current user as you cant edit your own account
        self.str_selectedUser = ""
        self.refresh()  # refresh refreshes the table, so calling it at the start loads the table

    def select_user(self):
        self.refresh()
        row_details = []
        table_row_details = self.db_table.selectionModel().selectedRows()
        for index in table_row_details:  # goes through each row and assigns variables for the index of the row for use
            row = index.row()
            row_details = self.data[row]  # gets the currently selected user info
        self.str_selectedUser = row_details

    def delete(self):
        con = lite.connect("festivalFinder")
        con.execute("DELETE FROM tbl_users WHERE userID=?", (self.str_selectedUser[0],))
        con.commit()  # deletes the current user from the database, then refreshes the table
        con.close()
        self.refresh()

    def admin(self):
        con = lite.connect("festivalFinder")  # connects to the user database, function then inverts the admin
        if self.str_selectedUser[4] == 1:     # permissions of the current user that's been selected.
            new_perm = int(self.str_selectedUser[4]) + 1
        else:
            new_perm = int(self.str_selectedUser[4]) - 1
        con.execute("UPDATE tbl_users SET perm_lvl = ? WHERE userId=?", (new_perm, self.str_selectedUser[0]))
        con.commit()  # commits any changes to the table
        con.close()
        self.refresh()

    def refresh(self):
```

```
    def refresh(self):
        con = lite.connect("festivalFinder")
        self.data = con.execute("SELECT * FROM tbl_users WHERE user != ?;", (self.str_user,)).fetchall()
        self.db_table.setColumnCount(len(self.data[0]))
        self.db_table.setRowCount(len(self.data))
        for i in range(len(self.data)):  # length of self.data is the number of rows in the table
            for j in range(len(self.data[0])):  # length of self.data refers the number of columns in each entry
                self.db_table.setItem(i, j, QtGui.QTableWidgetItem(str(self.data[i][j])))
                #  for every single item in the table, row by row, column by column, it sets each individual item
                #  this way in turn it can both clear each row and fill the table in one fell swoop.
        con.close()
        self.str_selectedUser = ""  # resets the selected user

    def open_menu(self):
        self.hide()
        wdw_menu.show()
```

This involves obviously the implementation of administrator privileges and actions, as well as commenting and SQL parameters, Hungarian notation etc.

```
class FestivalDate(QtGui.QMainWindow, wdw_date):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        con = lite.connect("festivalFinder")
        self.list_names = con.execute("SELECT Name FROM tbl_festivals;").fetchall()
        for i in self.list_names:  # populates the dropdown menu with festival names to chose from
            self.dropdown_fest.addItem(str(i[0]))
        con.close()
        self.dropdown_fest.activated.connect(self.fill_gui)
        self.str_fest_name = self.dropdown_fest.currentText()
        self.btn_back.clicked.connect(self.back)

    def back(self):
        wdw_menu.show()
        self.hide()

    def fill_gui(self):
        self.str_fest_name = self.dropdown_fest.currentText()  # gets the currently selected festival name
        con = lite.connect("festivalFinder")
        start_date = con.execute("SELECT StartDate FROM tbl_festivals WHERE Name=?;", (self.str_fest_name,)).fetchall()
        end_date = con.execute("SELECT EndDate FROM tbl_festivals WHERE Name=?;", (self.str_fest_name,)).fetchall()
        day_month = start_date[0][0].split("/")
        # plots the current start date of the festival selected
        self.calendar.setSelectedDate(PyQt4.QtCore.QDate(2017, int(day_month[1]), int(day_month[0])))
        self.lbl_dates.setText("First date on calendar - dates from " + start_date[0][0] + " to " + end_date[0][0])
        con.close()
```

```
class FestivalDelete(QtGui.QMainWindow, wdw_delete):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.btn_close.clicked.connect(self.close)
        self.btn_delete.clicked.connect(self.delete)

    def fill_gui(self):
        self.lbl_personal.setText("Are you sure? This will delete the account registered to: " + wdw_login.str_user)

    def delete(self):
        con = lite.connect("festivalFinder")
        con.execute("DELETE FROM tbl_users WHERE user = ? AND pass = ?",  # deletes the currently logged in user
                    (wdw_login.str_user, wdw_login.str_hashed_pass))
        con.commit()
        con.close()
        self.lbl_personal.setText("Account registered to " + wdw_login.str_user + "deleted.")

    def close(self):
        wdw_menu.show()
        self.hide()


class FestivalBuy(QtGui.QMainWindow, wdw_buy):  # displays the ticket vendors website
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.btn_close.clicked.connect(self.close)

    def close(self):
        self.hide()
        wdw_profile.show()
```

```
class FestivalProfile(QtGui.QMainWindow, wdw_profile):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.btn_close.clicked.connect(self.close_window)
        self.btn_buy.clicked.connect(self.buy)

    def buy(self):
        wdw_buy.show()
        self.hide()

    def fill_gui(self, f_name):
        self.festival_name.setText(f_name)
        wdw_buy.web_buy.setUrl(PyQt4.QtCore.QUrl("http://www.ticketmaster.co.uk/search?tm_link=tm_header_search"
                                                 "&user_input=&q=" + f_name))
        con = lite.connect("festivalFinder")
        start_date = con.execute("SELECT StartDate FROM tbl_festivals WHERE Name=?;", (f_name,)).fetchall()
        end_date = con.execute("SELECT EndDate FROM tbl_festivals WHERE Name=?;", (f_name,)).fetchall()
        cost = con.execute("SELECT Cost FROM tbl_festivals WHERE Name=?;", (f_name,)).fetchall()
        day_cost = con.execute("SELECT DayCost FROM tbl_festivals WHERE Name=?;", (f_name,)).fetchall()
        home = con.execute("SELECT Postcode FROM tbl_festivals WHERE Name=?;", (f_name,)).fetchall()
        # as all the fields are needed, I fetched all the relevant data from the festivals table, and labelled them
        # accordingly, for clarity of a select all query and referring to them as results[x][x][x]
        r = requests.get('https://maps.googleapis.com/maps/api/geocode/'
                         'json?address=%s&key=AIzaSyA3rvec5Ju7hrpCb-MGdjafEIM5wD7u8M8' % home[0][0]).json()
        display_address = (f_name + ": " + r['results'][0]['formatted_address'])
        day_month = start_date[0][0].split("/")
        # splits the date into DD and MM for the calendar widget to understand
        self.calendar.setSelectedDate(PyQt4.QtCore.QDate(2017, int(day_month[1]), int(day_month[0])))
        # puts the start date on the calendar (only one date can be viewed)
        self.lbl_cost.setText("£ " + str(cost[0][0]))
        self.lbl_daycost.setText("£ " + str(day_cost[0][0]))
        # sets the outputs to show the prices
        home = con.execute("SELECT festID FROM tbl_festivals WHERE Name=?;", (f_name,)).fetchall()
        display = ""
        list_acts = con.execute("SELECT * FROM tbl_acts WHERE festID=?;", (home[0][0],)).fetchall()
        # fetches the acts associateed with table festivals
        for i in list_acts:
            display = display + "\n" + i[1]
        self.txt_acts.setText(display)
        con.close()
        self.lbl_dates.setText("First date on calendar - dates from " + start_date[0][0] + " to " + end_date[0][0])
        self.entry_address.setText(display_address)

    def close_window(self):
        self.hide()
```

```
class FestivalLocation(QtGui.QMainWindow, wdw_location):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        list_coordinates = []
        con = lite.connect("festivalFinder")
        self.list_names = con.execute("SELECT Name FROM tbl_festivals;").fetchall()
        self.postcodes = con.execute("SELECT PostCode FROM tbl_festivals;").fetchall()
        self.str_fest_name = self.dropdown_fest.currentText()  # fills the dropdown combobox with festival names
        for i in self.list_names:
            self.dropdown_fest.addItem(str(i[0]))
        for i in self.postcodes:
            r = requests.get('https://maps.googleapis.com/maps/api/geocode/'
                             'json?address=%s&key=AIzaSyA3rvec5Ju7hrpCb-MGdjafEIM5wD7u8M8' % i).json()
            list_coordinates.append(r['results'][0]['geometry']["location"])

            # geocaches each postcode in tbl_festivals and appends the longitude and latitude coordinates into a list
        self.loc = []
        for x in range(0, len(self.list_names)):  # for every festival, append a name and coordinates
            self.loc.append([])
            self.loc[x].append(str(self.list_names[x][0]))
            self.loc[x].append(list_coordinates[x])
        str_html = googlemap.plot(self.loc, len(self.loc), 0, 0)  # plots the map with the locations and without a
        # radius circle or home point
        self.map.setHtml(str_html)
        debug(str_html)
        debug(self.loc)
        debug(self.postcodes)
        self.dropdown_fest.activated.connect(self.festival_profile)
        self.btn_getDistance.clicked.connect(self.alert)
        self.btn_back.clicked.connect(self.back)
        self.int_max_distance = ""
```

```
    def set_text(self):
        text, ok = QtGui.QInputDialog.getText(self, "QInputDialog.getText()",
                                              "Max Distance you'd want to travel (Miles) :", QtGui.QLineEdit.Normal,)
        if ok and text != '':
            try:
                int(text)
                return text
            except ValueError:
                return 0
        else:
            return 0
        # asking the user for a maximum distance they would like to travel and this plots a radius around them for
        # this distance in miles
        # also validates for only integer entry
```

```
    def back(self):
        wdw_menu.show()
        self.hide()
```

```
def alert(self):
    self.int_max_distance = self.set_text()
    wdw_alert.show()
    self.hide()

def festival_profile(self):
    self.str_fest_name = wdw_location.dropdown_fest.currentText()
    FestivalProfile.fill_gui(wdw_profile, self.str_fest_name)
    wdw_profile.show()  # links to FestivalProfile class

def get_distance(self, home):
    destinations = ""
    for i in self.postcodes:
        destinations = destinations + str(i[0]) + "|"
    text = requests.get(
        'https://maps.googleapis.com/maps/api/distancematrix/json?units=imperial&origins=%s'
        '&destinations=%s&key=AIzaSyA3rvec5Ju7hrpCb-MGdjafEIM5wD7u8M8' % (home, destinations)).json()
    r = requests.get('https://maps.googleapis.com/maps/api/geocode/'
                     'json?address=%s&key=AIzaSyA3rvec5Ju7hrpCb-MGdjafEIM5wD7u8M8' % home).json()
    self.lbl_origin.setText(text['origin_addresses'][0])
    self.loc.append([])  # appending to locations to be plotted
    self.loc[len(self.postcodes)].append(text['origin_addresses'][0])  # appending address
    self.loc[len(self.postcodes)].append(r['results'][0]['geometry']["location"])  # appending location
    str_html = googlemap.plot(self.loc, len(self.loc), -1, self.int_max_distance)
    debug(self.loc)  # debugging prints the value passed in if debug_mode is 1
    debug(len(self.loc))
    self.map.setHtml(str_html)  # sets the html to the code returned by googlemap.py
    debug(str_html)
    display = ""
    counter = 0
    for i in text['destination_addresses']:  # grabs the distance for every seperate destination to fill the display
        distance = str(text['rows'][0]["elements"][counter]["distance"]["text"])
        display = (display + "\n" + self.list_names[counter][0] + ": " + str(i) + ", Distance: " + distance)
        counter = counter + 1
    display = display + "\n" + "\n" + "Maps perimeter from you for the closest festivals!"
    wdw_location.lbl_distances.setText(display)  # fills with distance
```

```
class FestivalAlert(QtGui.QMainWindow, wdw_alert):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.str_home = ""
        self.btn_close.clicked.connect(self.close_window)
        self.btn_ok.clicked.connect(self.ok)

    def ok(self):
        regex_postcode = "^(([gG][iI][rR] {0,}0[aA]{2})|((([a-pr-uwyzA-PR-UWYZ][a-hk-yA-HK-Y]?[0-9][0-9]?)|" \
                         "(([a-pr-uwyzA-PR-UWYZ][0-9][a-hjkstuwA-HJKSTUW])|([a-pr-uwyzA-PR-UWYZ][a-hk-yA-HK-Y][0-9]" \
                         "[abehmnprv-yABEHMNPRV-Y])))) {0,}[0-9][abd-hjlnp-uw-zABD-HJLNP-UW-Z]{2}))$"
        self.str_home = self.entry_postcode.text()
        debug(self.str_home)
        if re.match(regex_postcode, self.str_home):  # checks if the input matches the regular postcode structure
            self.hide()
            wdw_location.show()
            FestivalLocation.get_distance(wdw_location, self.str_home)
        else:
            self.lbl_msg.setText("Not valid format, try AA99 9AA where a is any letter and 9 is any number")

    def close_window(self):
        self.hide()
        wdw_location.show()
```

```
class FestivalCost(QtGui.QMainWindow, wdw_cost):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.btn_full.clicked.connect(self.full_ticket)
        self.btn_day.clicked.connect(self.day_ticket)
        self.btn_close.clicked.connect(self.close)

    def full_ticket(self):
        con = lite.connect("festivalFinder")
        data = con.execute("SELECT Name, Cost FROM tbl_festivals ORDER BY Cost;").fetchall()
        self.tbl_data.setColumnCount(len(data[0]))  # sets the amount of columns equal to the columns from the sql query
        self.tbl_data.setRowCount(len(data))  # sets the amount of rows equal to the rows from the sql query
        for i in range(len(data)):  # length of self.data is the number of rows in the table
            for j in range(len(data[0])):  # length of self.data refers the number of columns in each entry
                self.tbl_data.setItem(i, j, QtGui.QTableWidgetItem(str(data[i][j])))  # sets the item
        con.close()

    def day_ticket(self):
        con = lite.connect("festivalFinder")
        data = con.execute("SELECT Name, DayCost FROM tbl_festivals ORDER BY DayCost;").fetchall()
        self.tbl_data.setColumnCount(len(data[0]))  # sets the amount of columns equal to the columns from the sql query
        self.tbl_data.setRowCount(len(data))  # sets the amount of rows equal to the rows from the sql query
        for i in range(len(data)):  # length of self.data is the number of rows in the table
            for j in range(len(data[0])):  # length of self.data refers the number of columns in each entry
                self.tbl_data.setItem(i, j, QtGui.QTableWidgetItem(str(data[i][j])))  # sets the item
        con.close()

    def close(self):
        wdw_menu.show()
        self.hide()
```

```
class FestivalMusic(QtGui.QMainWindow, wdw_music):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.btn_go.clicked.connect(self.search)
        self.btn_close.clicked.connect(self.close)
        self.btn_overview.clicked.connect(self.overview)
        display = ""
        con = lite.connect("festivalFinder")
        r = con.execute("SELECT actName FROM tbl_acts;").fetchall()
        for i in r:  # displays the act names
            display = display + "\n" + i[0]
        self.txt_acts.setText(display)
        con.close()

    def search(self):
        artist = self.entry_music.text()
        con = lite.connect("festivalFinder")
        display = ""
        r = con.execute("SELECT * FROM tbl_acts WHERE ActName=?;", (artist,)).fetchall()
        for i in r:  # checks all the festivals that the artist entered is playing at
            fest_id = i[2]
            r = con.execute("SELECT Name FROM tbl_festivals WHERE festID=?;", (fest_id,))
            display = display + "\n" + r.fetchall()[0][0]
        self.txt_festivals.setText(display)
        con.close()

    def overview(self):
        con = lite.connect("festivalFinder")
        display = ""
        result = con.execute("Select Name, tbl_acts.actName, tbl_genres.Genre FROM tbl_festivals INNER JOIN "
                             "tbl_acts on tbl_festivals.festID = tbl_acts.festID INNER JOIN tbl_genres on "
                             "tbl_genres.genreID = tbl_acts.genreID ORDER BY Name").fetchall()
        for i in result:
            display = display + "\n" + i[0] + ": " + i[1] + " (Genre: " + i[2] + ")"
        # this provides a double inner join to display a festivle name, an act that's playing and the act's genre.
        ctypes.windll.user32.MessageBoxW(0, display, "Overview of Acts", 1)
        con.close()

    def close(self):
        wdw_menu.show()
        self.hide()
```

This shows an example of nested SQL Queries, multiple inner joins, related tables, SQL parameters, comprehensive commenting, evidence of different modules (namely a ctype popup output window)

```
class FestivalBuy(QtGui.QMainWindow, wdw_buy):  # displays the ticket vendors website
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.btn_close.clicked.connect(self.close)

    def close(self):
        self.hide()
        wdw_profile.show()
```

```
class FestivalMenu(QtGui.QMainWindow, wdw_menu):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.str_fest_name = ""
        self.int_perm = 0
        con = lite.connect("festivalFinder")
        self.list_names = con.execute("SELECT Name FROM tbl_festivals;").fetchall()
        for i in self.list_names:  # fills the combobox with festival names
            self.dropdown_fest.addItem(str(i[0]))
        con.close()
        self.btn_admin.clicked.connect(self.admin_menu)
        self.btn_location.clicked.connect(self.open_location)
        self.btn_price.clicked.connect(self.open_cost)
        self.btn_delete_account.clicked.connect(self.delete)
        self.dropdown_fest.activated.connect(self.festival_profile)
        self.btn_date.clicked.connect(self.date)
        self.btn_acts.clicked.connect(self.music)
        self.btn_close.clicked.connect(self.logout)

    def admin_menu(self):
        if self.int_perm == 1:
            wdw_admin.show()
            self.hide()
        else:  # validates whether current user is admin or not
            self.lbl_user.setText("Admin menu restricted for admins only.")

    def logout(self):
        self.hide()
        wdw_login.show()

    def music(self):
        wdw_music.show()
        self.hide()

    def date(self):
        wdw_date.show()
        self.hide()

    def open_cost(self):
        wdw_cost.show()
        self.hide()
```

```
    def festival_profile(self):
        self.str_fest_name = self.dropdown_fest.currentText()
        FestivalProfile.fill_gui(wdw_profile, self.str_fest_name)
        wdw_profile.show()

    def fill_gui(self, name):
        self.lbl_user.setText("Welcome Back, " + name)

    def open_location(self):
        wdw_location.show()
        self.hide()

    def delete(self):
        wdw_delete.show()
        self.hide()


def debug(test):
    if debug_mode == 1:
        print(test)

app = QtGui.QApplication(sys.argv)
wdw_menu = FestivalMenu()
wdw_admin = FestivalAdmin()
wdw_buy = FestivalBuy()
wdw_music = FestivalMusic()
wdw_date = FestivalDate()
wdw_delete = FestivalDelete()
wdw_cost = FestivalCost()
wdw_alert = FestivalAlert()
wdw_location = FestivalLocation()
wdw_register = FestivalRegister()
wdw_profile = FestivalProfile()
wdw_forgot = FestivalForgot()
wdw_login = FestivalLogin(None)
wdw_login.show()
app.exec_()
```

A Word on Inheritance, Multi-Inheritance and Polymorphism in this design

```
class FestivalBuy(QtGui.QMainWindow, wdw_buy):  # di
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.btn_close.clicked.connect(self.close)
```

Every class in this program inherits the attributes of a QtGui.QMainWindow and also the attributes that are in the window it represents. This allows the functions to use commands for Qt and create content in the window as well as access the existing Widgets in the user-made window. For instance one can use Qt.QMainWindow to add a new table-view widget or pop-up window to display text, and all throughout the program the wdw_"window name" objects are referred to and accessed via QtGui commands. (eg btn_close.clicked refers the to QtGui function "clicked" surrounding the wdw_buy widget "btn_close".

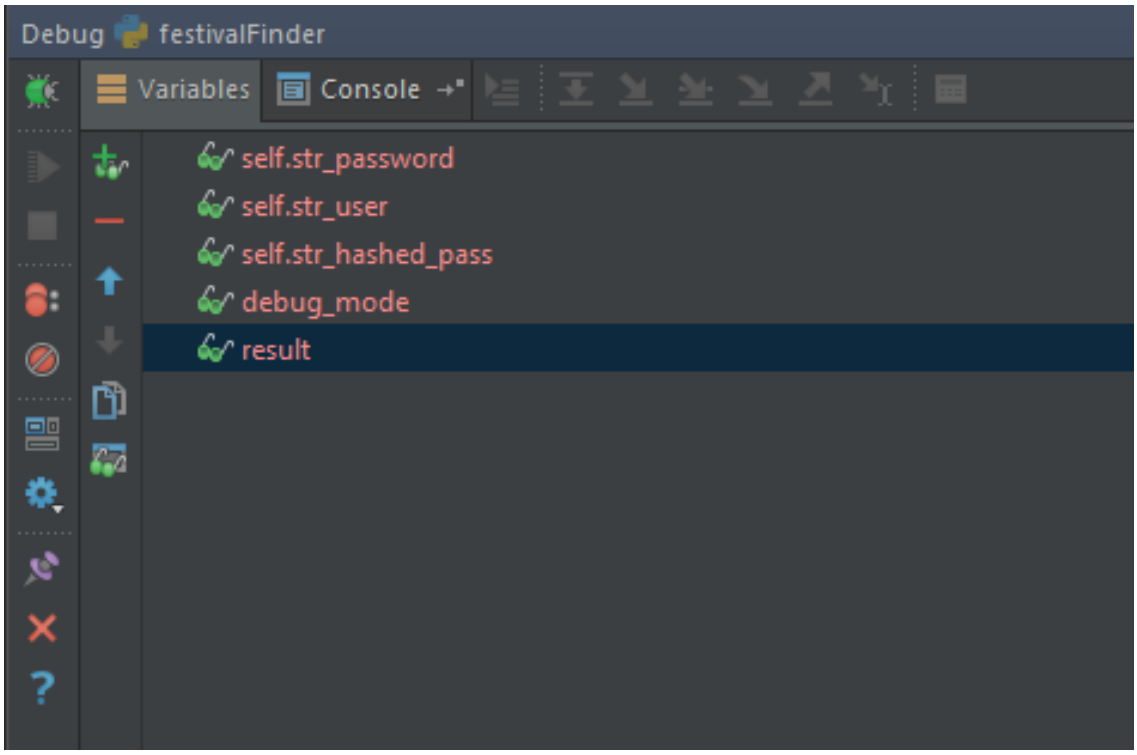
```
def send(recipient, name, reset_password):
    import smtplib
    user = 'majesticseabass@gmail.com'
    password = 'YOLODOLPHIN69'
    to = [recipient]
    body = """
            Dear """ + " " + name + """,
            This is an automated message. You have requested to reset your Festival Finder password.
            your new password is %s

    """ % reset_password
    msg = 'Subject: %s\n\n%s' % ('Festival Finder: Reset your password', body)
    print(type(msg))
    try:
        server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
        server.ehlo()
        server.login(user, password)
        server.sendmail(user, to, msg)
        print("email sent! with password "+str(reset_password))
        server.close()
    except:
        print('Something went wrong...')
```

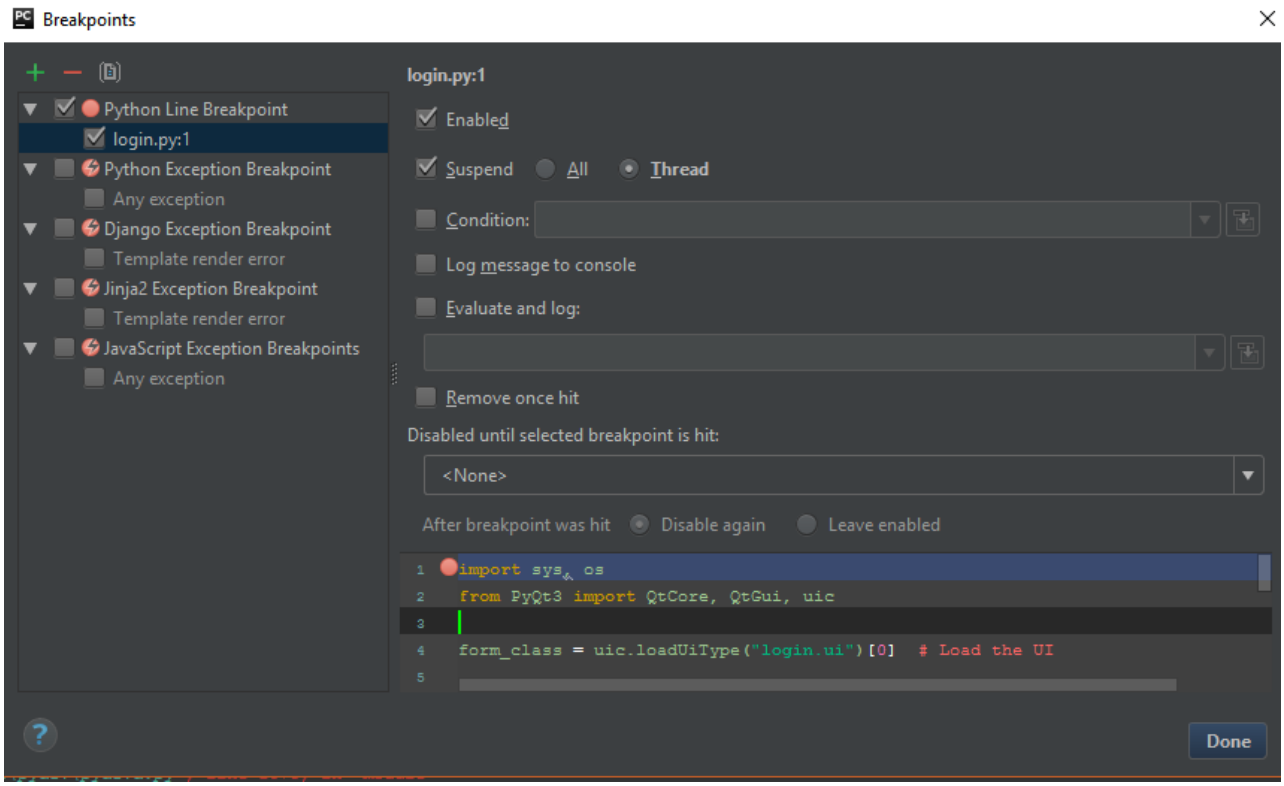
This is in emailpass.py ^ a module of my own design that sends emails for resetting passwords that is called by festivalfinder.py

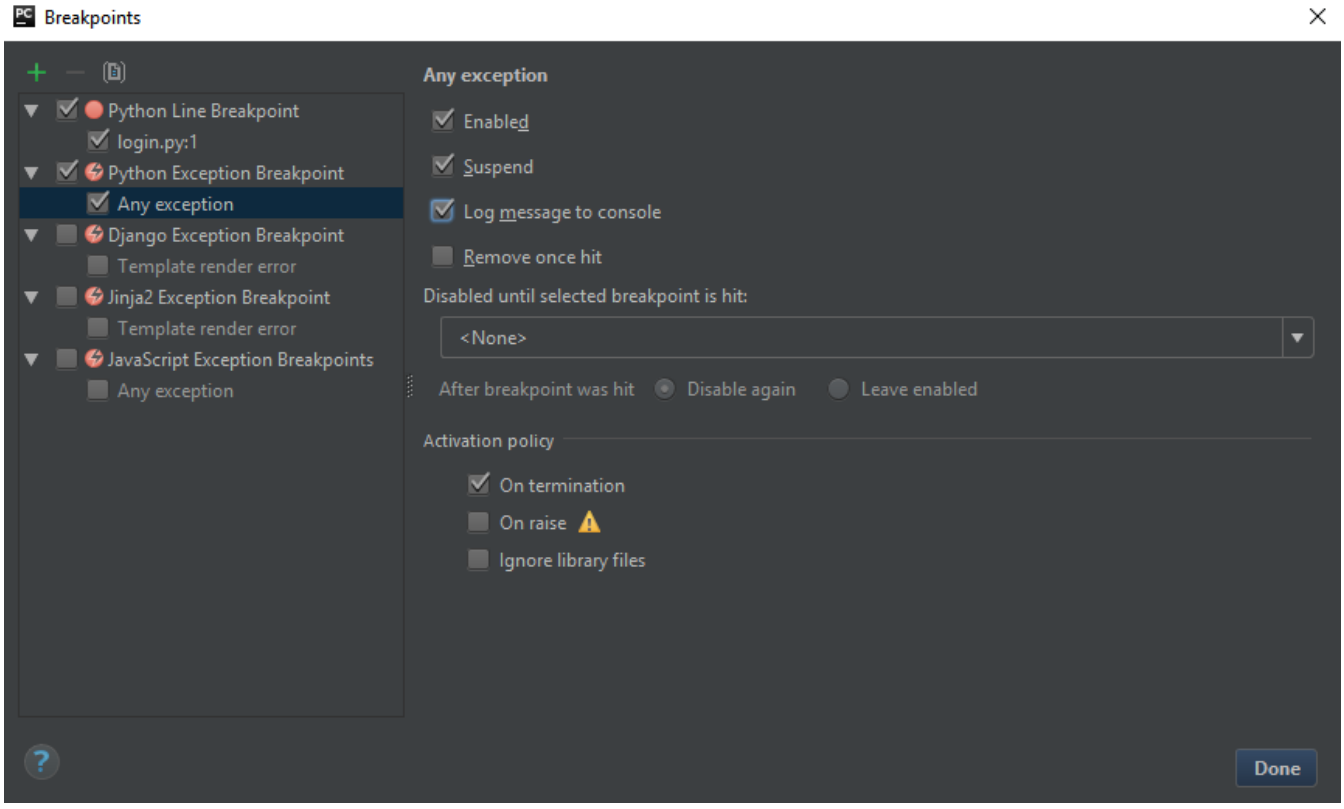
Throughout all these screenshots, explicit mentions of code have been described initially to point them out. Throughout the whole code, I used means of debugger, comprehensive commenting, parameter passing, presence and data type checks, variables of different scopes, constants will capitalised names, modules, hashed user data, administrator and normal accounts, password strength checks with regular expressions, SQL parameter queries, Inner joins and nested queries, relational databases of 3 tables (festivals, acts, genres), inheritance/multi inheritance (The properties of QtGui.QMainWindow are inherited by every class and it also inherits the widgets of every wdw_ file named by the class). The program has different examples of the back() method, sometimes closing the program and sometimes going to the previous window (Polymorphism), Hungarian notation by denoting the filetype before the name of each variable, list etc. It also involves multiple counts of validation via Regular Expressions for postcodes, usernames and passwords. I used a debugging method that prints variables I want to check during debugging mode and involves complex processing including a shortest path algorithm within the Google Maps module. It also uses a mix of procedures (ie methods that close windows) and functions (returning code in googlemap.py)

Using PyCharm Debugger, Variable Watchers and Breakpoints

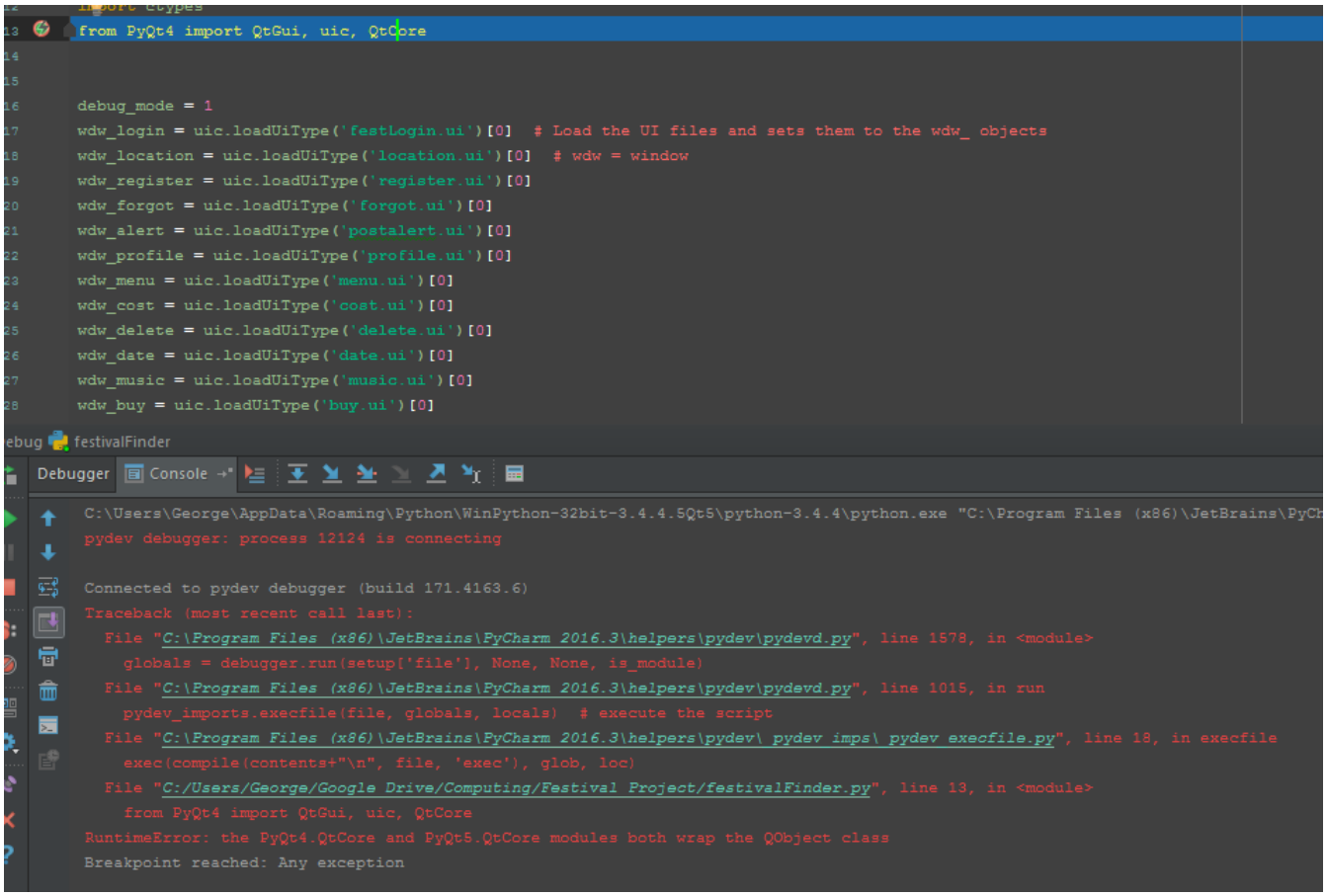
```
C:\Users\George\AppData\Roaming\Python\WinPython-32bit-3.4.4.5Qt5\python-3.4.4\python.exe "C:\Program Files (x86)\JetBrain
pydev debugger: process 17844 is connecting

Connected to pydev debugger (build 171.4163.6)
Traceback (most recent call last):
  File "C:\Program Files (x86)\JetBrains\PyCharm 2016.3\helpers\pydev\pydevd.py", line 1578, in <module>
    globals = debugger.run(setup['file'], None, None, is_module)
  File "C:\Program Files (x86)\JetBrains\PyCharm 2016.3\helpers\pydev\pydevd.py", line 1015, in run
    pydev_imports.execfile(file, globals, locals)  # execute the script
  File "C:\Program Files (x86)\JetBrains\PyCharm 2016.3\helpers\pydev\_pydev_imps\_pydev_execfile.py", line 18, in execfil
    exec(compile(contents+"\n", file, 'exec'), glob, loc)
  File "C:/Users/George/Google Drive/Computing/Festival Project/festivalFinder.py", line 13, in <module>
    from PyQt4 import QtGui, uic, QtCore
RuntimeError: the PyQt4.QtCore and PyQt5.QtCore modules both wrap the QObject class

Process finished with exit code 1
```




Breakpoints
Python Line Breakpoint
login.py:1
Python Exception Breakpoint
Any exception
Django Exception Breakpoint
Template render error
Jinja2 Exception Breakpoint
Template render error
JavaScript Exception Breakpoints
Any exception
login.py:1
Enabled
Suspend All Thread
Condition:
Log message to console
Evaluate and log:
Remove once hit
Disabled until selected breakpoint is hit:
<None>
After breakpoint was hit Disable again Leave enabled
import sys, os
from PyQt3 import QtCore, QtGui, uic
form_class = uic.loadUiType("login.ui")[0] # Load the UI
Done


Breakpoints
Python Line Breakpoint
login.py:1
Python Exception Breakpoint
Any exception
Django Exception Breakpoint
Template render error
Jinja2 Exception Breakpoint
Template render error
JavaScript Exception Breakpoints
Any exception
Any exception
Enabled
Suspend
Log message to console
Remove once hit
Disabled until selected breakpoint is hit:
<None>
After breakpoint was hit Disable again Leave enabled
Activation policy
On termination
On raise
Ignore library files
Done

I used breakpoints for exceptions –



To override this error, I shortened the amount of PyQt imports to avoid collisions.

Evaluation

Testing Success Criteria (MAIN TESTING)

A Foreword on General Tests

For each window, I tested all the functions mentioned in my success criteria but several tests remained the same throughout. Stress testing the GUI involved repeatedly clicking all the buttons to induce a crash, inputting wrong data repeatedly to induce a crash – trying any other means necessary to find the limits of the program (eg The Location window had its map repeatedly zoomed, clicked and dragged to try and induce a crash). Windows were each minimized and extended and all features were toyed with to try and influence a crash. Luckily the program stood up in 100% of the cases to destructive crash testing – in no scenario did the program successfully crash under stress of the user.

For each window, input and output were tested for correct output for any given scenario and reading in the right input for each point of data entry.

Buttons for all windows were tested for linking to the right window. This involved the current window being hidden and the directed window being shown (In the case of pop-up designated windows such as Festival Alert, the parent window was not closed).

Login Class (Criteria 1-4)

Criteria 1 - **Test the buttons for correct links (correct windows and functions etc) / Stress Testing the GUI**

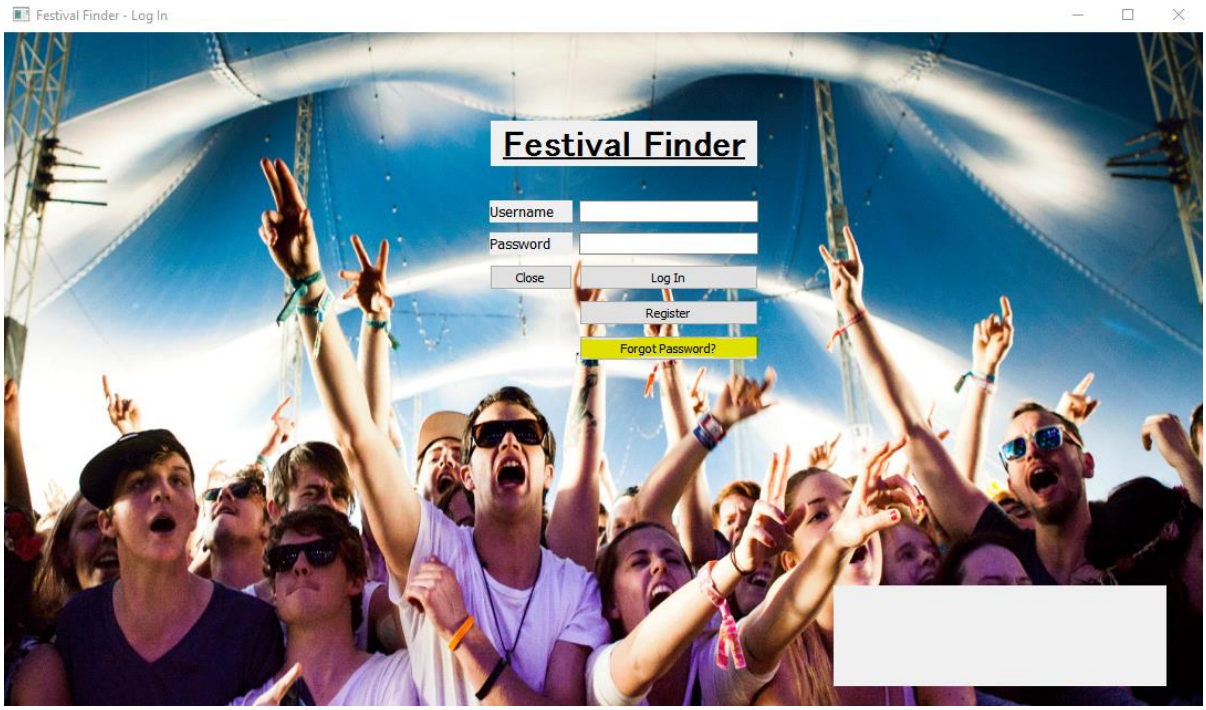
The log-on window has four different buttons, one to close the program –

Criteria 1

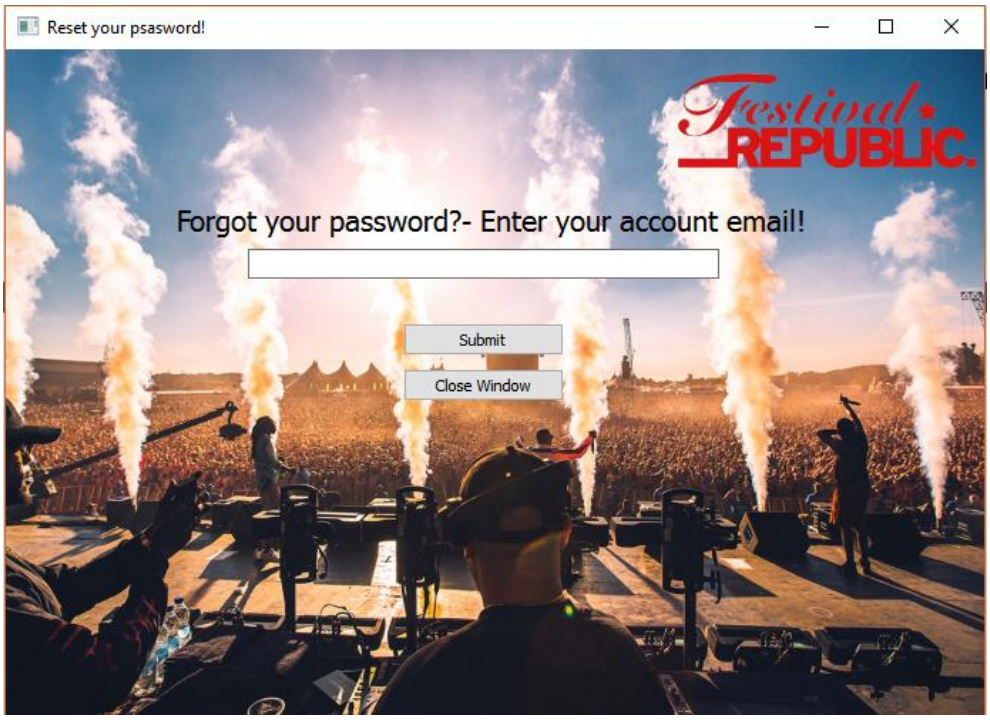
Which functions correctly – closing the program with no errors, indicated by an exit code of 0:

```
Process finished with exit code 0
```

The next button, “Forgot Password?”, links directly to the forgot window with no delay:

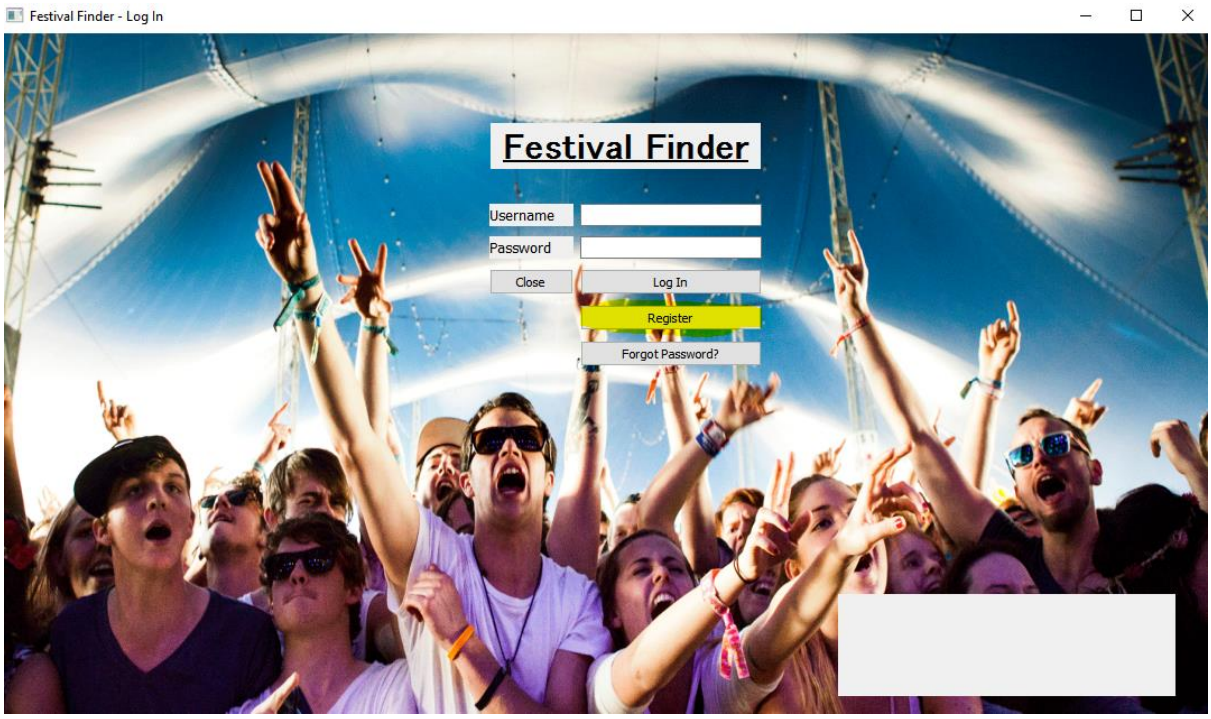


Criteria 1

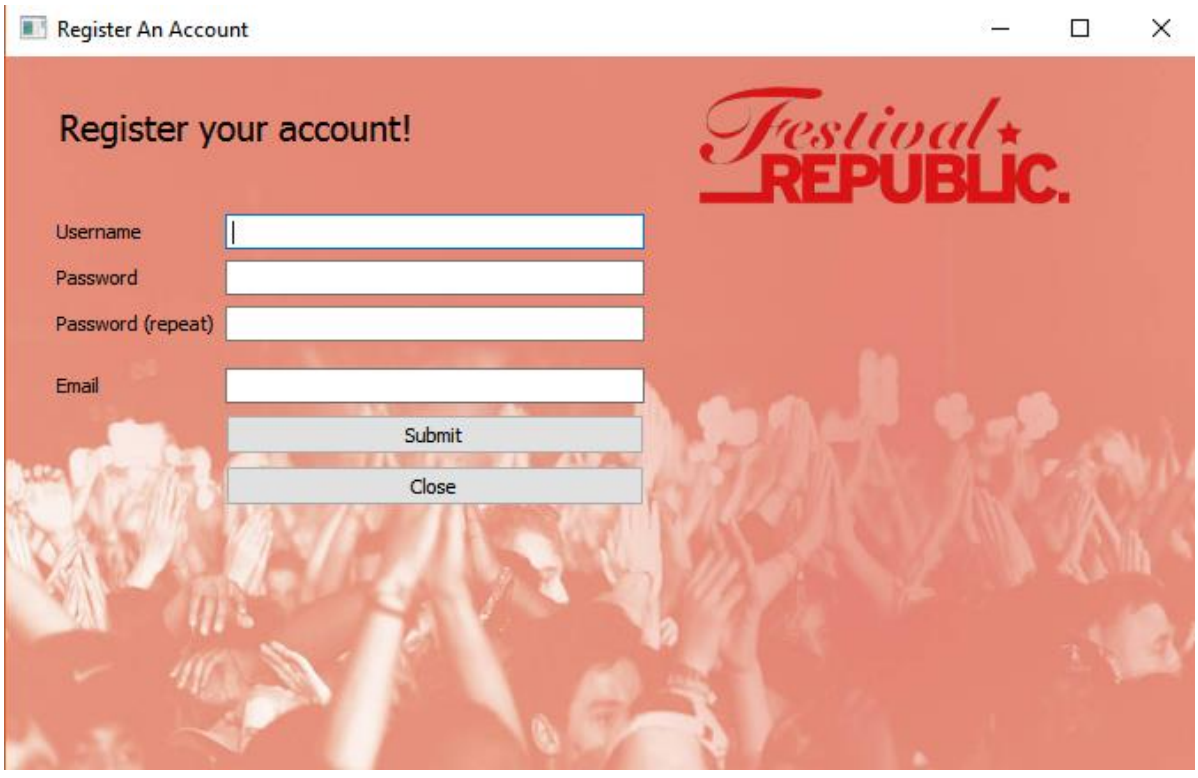


Criteria 1

The next button, Register, links correctly with no time delay to the register window –



Criteria 1



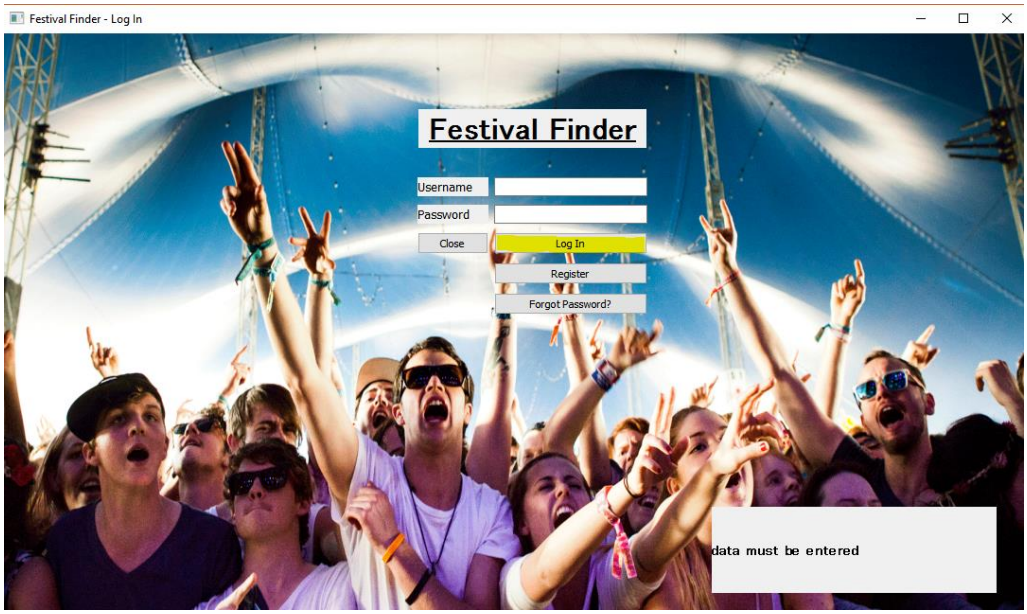
Criteria 1

And finally, the log-in button functions correctly with a correct username and password, this is tested later in criteria 3.

Stress testing the GUI had no effect with the login button, as it immediately rejects a process without correct input and therefore does not result in any complicated programming.

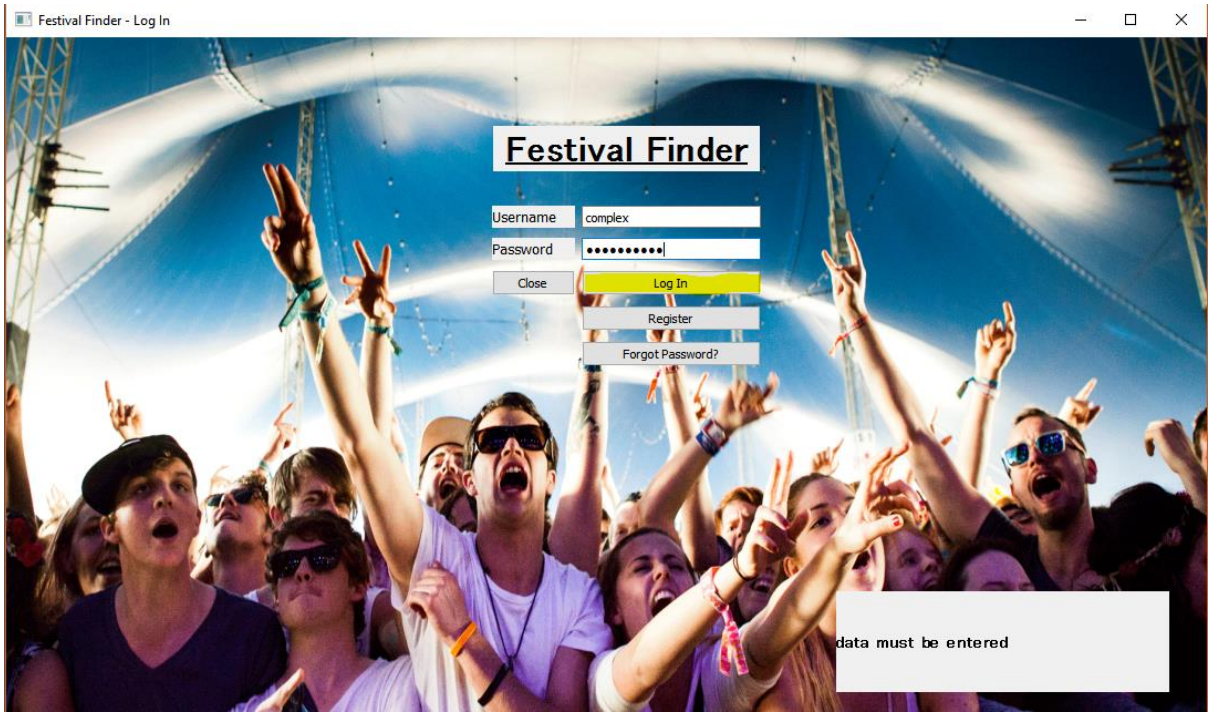
Criteria 2 - **Test for data entry / Test for nothing entered**

The program rejects input without data entry, prompting the user for more input, to fill out all the fields (The output is displayed in the “txt_msg” object -



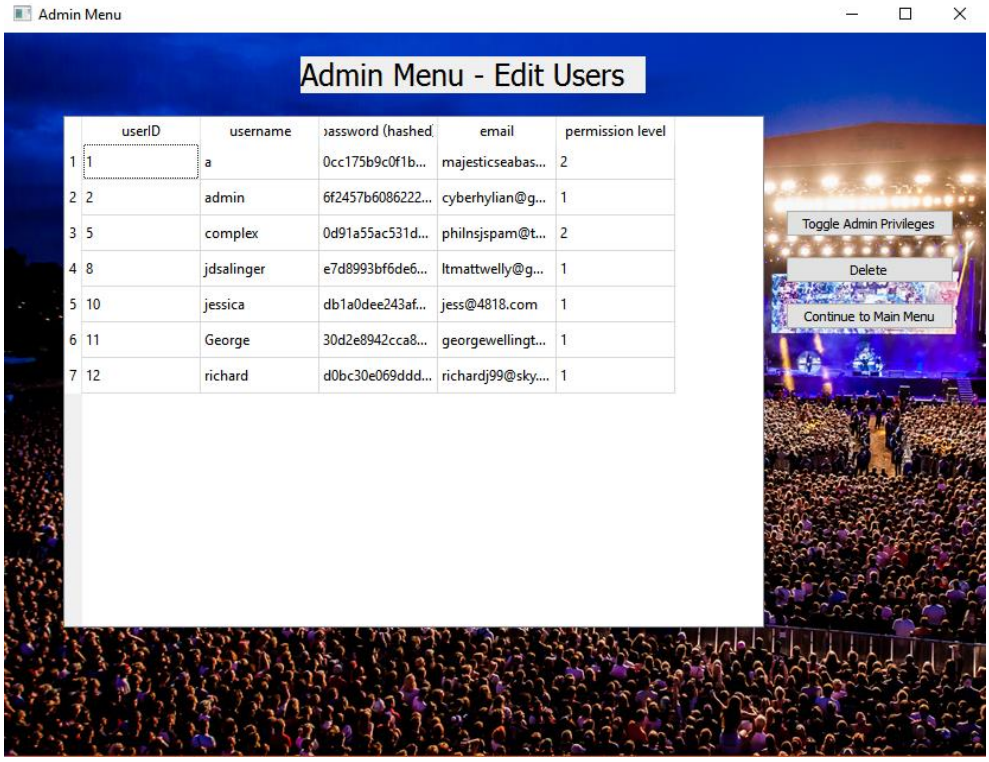
Criteria 2

Criteria 3 - **Logging in with correct/incorrect user information.**

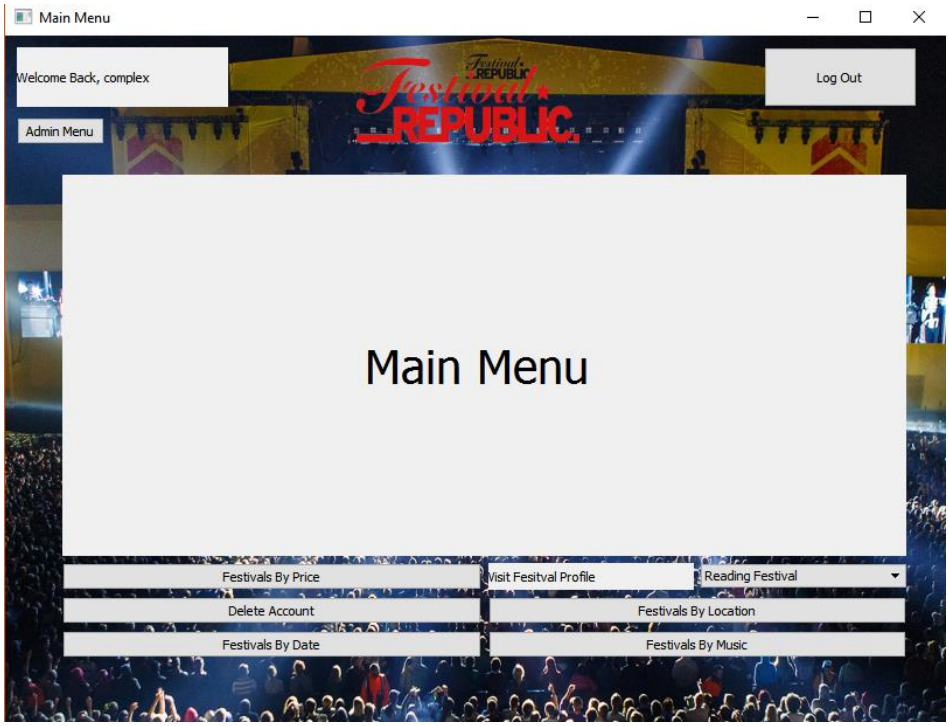


Criteria 3

Correct user information yields a link to the main menu (with the admin menu first depending on the user's permission level). For this test we used the account (username:complex, password: Imaginary1) with administrator level privileges (signified by a permission level of 2) –

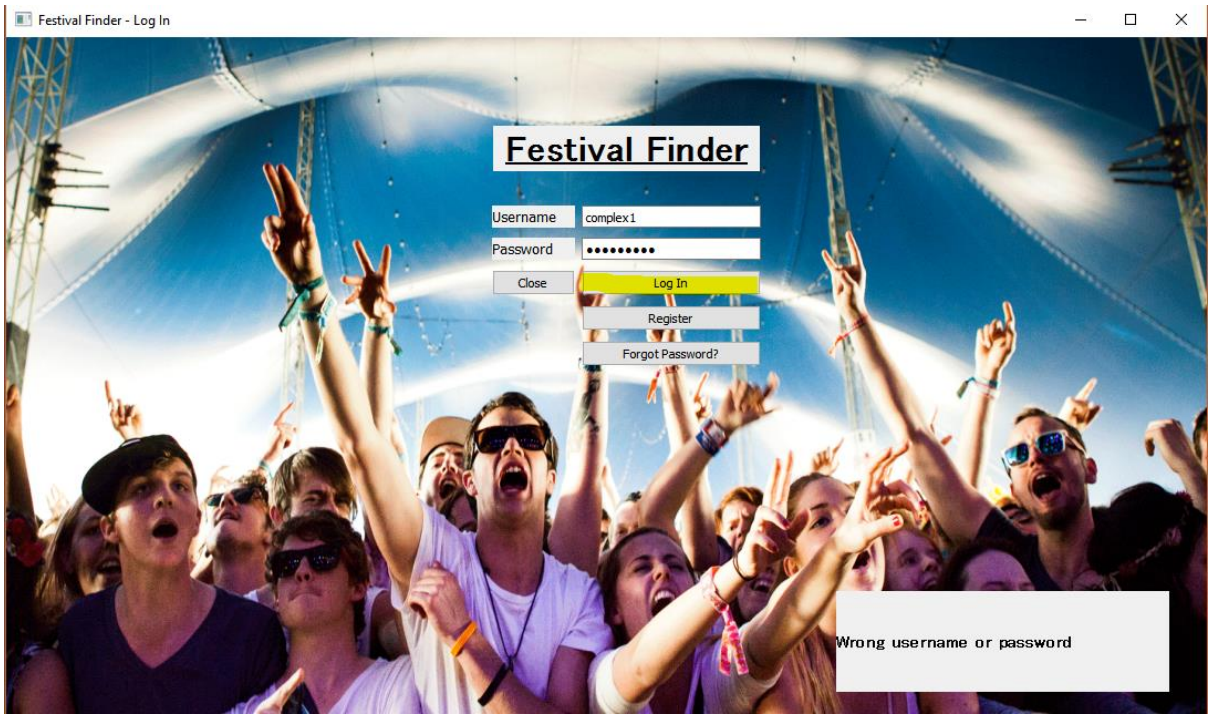


Citeria 3



Criteria 3

Incorrect information does not gain access to this menu, instead prompts the user that the information is incorrect –



Criteria 3

Also, if fields were left blank, the error message would change to prompt the user to enter more data than has already been entered.

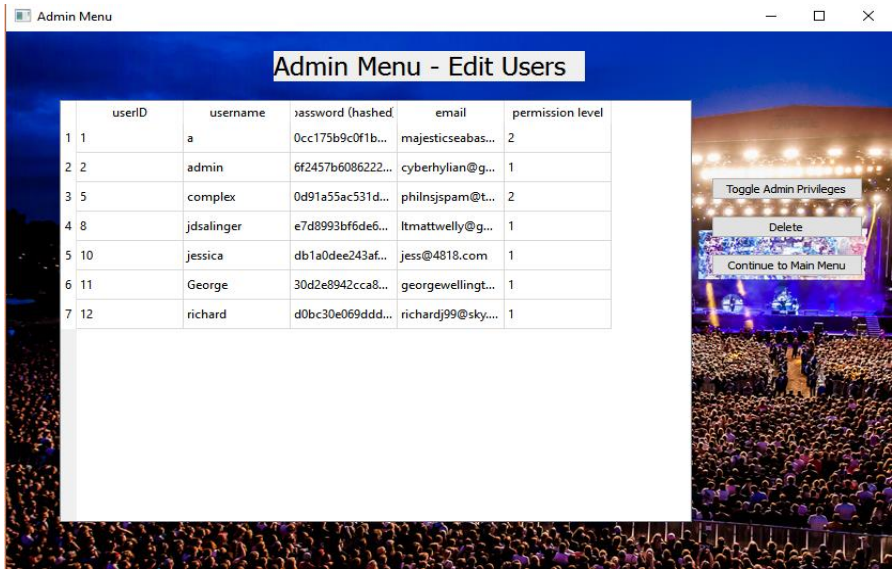
Criteria 4 – Testing Administrator permissions

Logging in with administrator information will yield the administrator menu and also allow you to access the admin menu from the main menu, which also allows you to edit permission levels and delete user accounts:

User information (username: “a”, password (unhashed): “a”, this account was used for testing due to the ease of typing only “a”, it was created in the backend, and therefore did not need to adhere to the register window’s password/username strength:

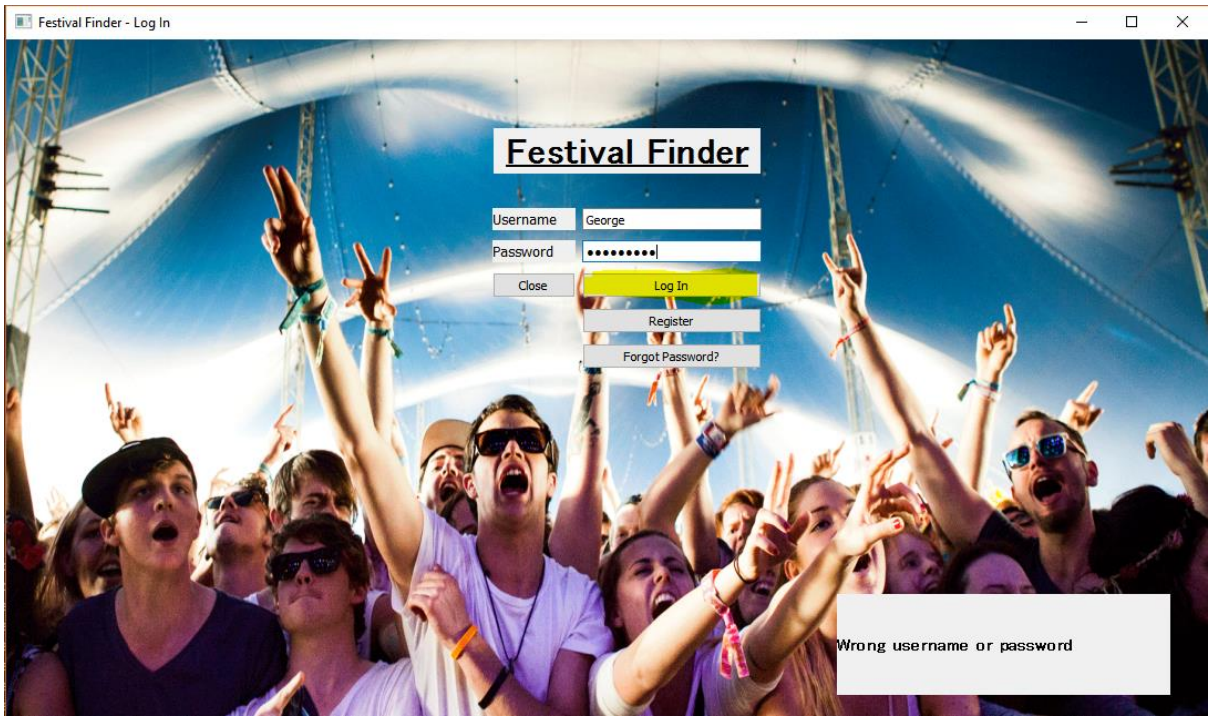
Criteria 4

This gives access to the administrator menu (Where we can see whether an account is normal or admin), thus access to delete users and toggle administrator privileges by viewing the users table-



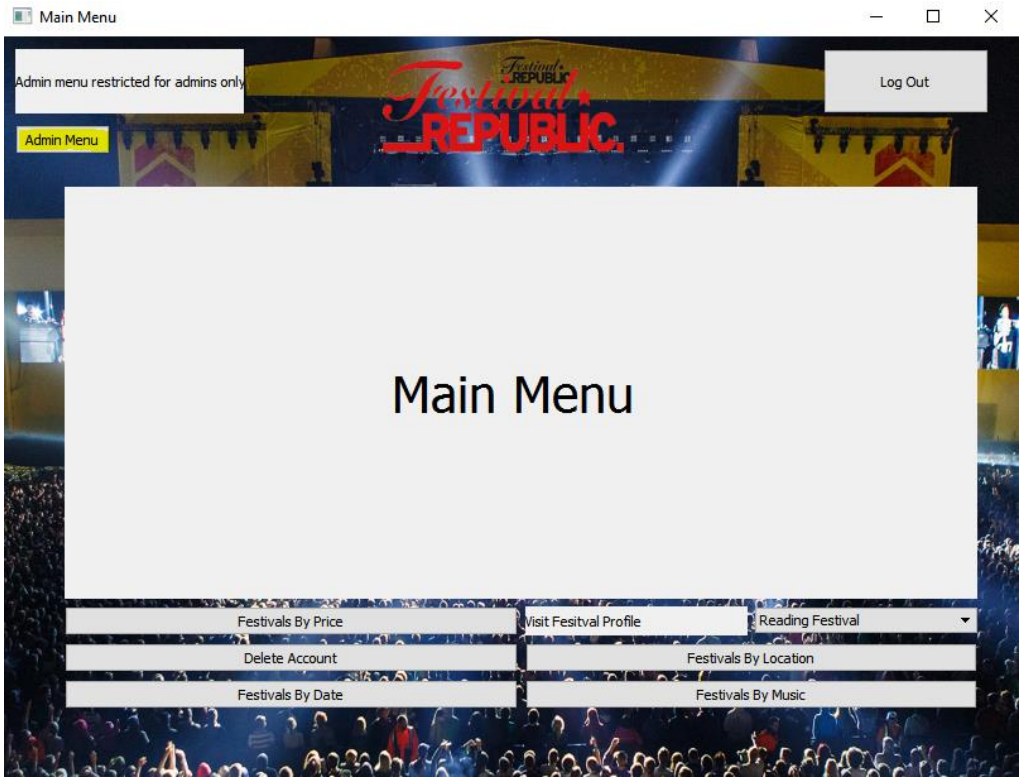
Criteria 4

Choosing a normal user from the menu with the username : “George”, unhashed password: “Wellyb00t”:



Criteria 4

This links us straight to the main menu, which we cannot use to navigate to the admin menu:



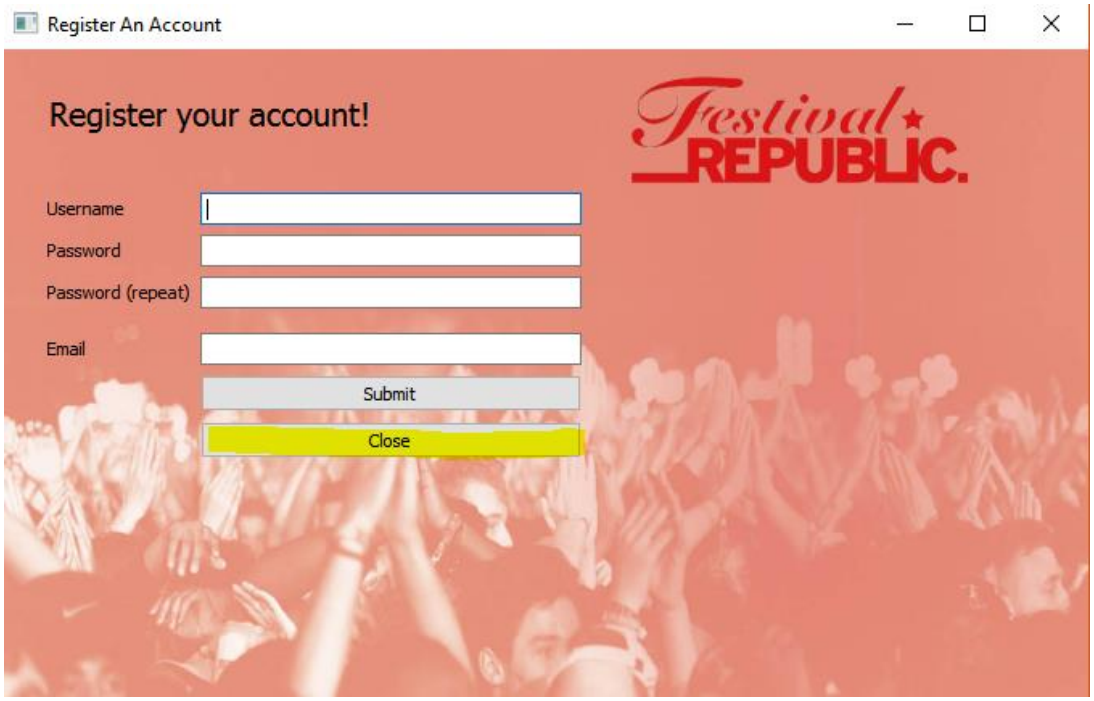
Criteria 4

And it provides an output to show us the change in user permissions at the top end of the main menu.

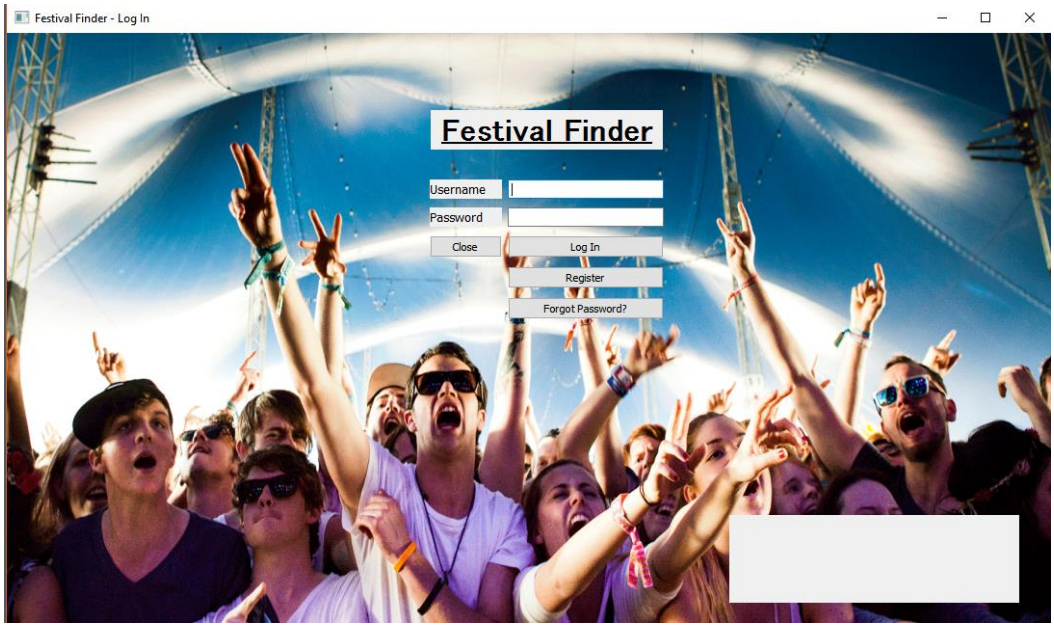
Register Class (Criteria 5-9)

Criteria 5 - **Test the buttons for correct links (correct windows and functions etc)**

Only one button in this form links to another window, the close button, which links back to the log-in interface:



Criteria 5

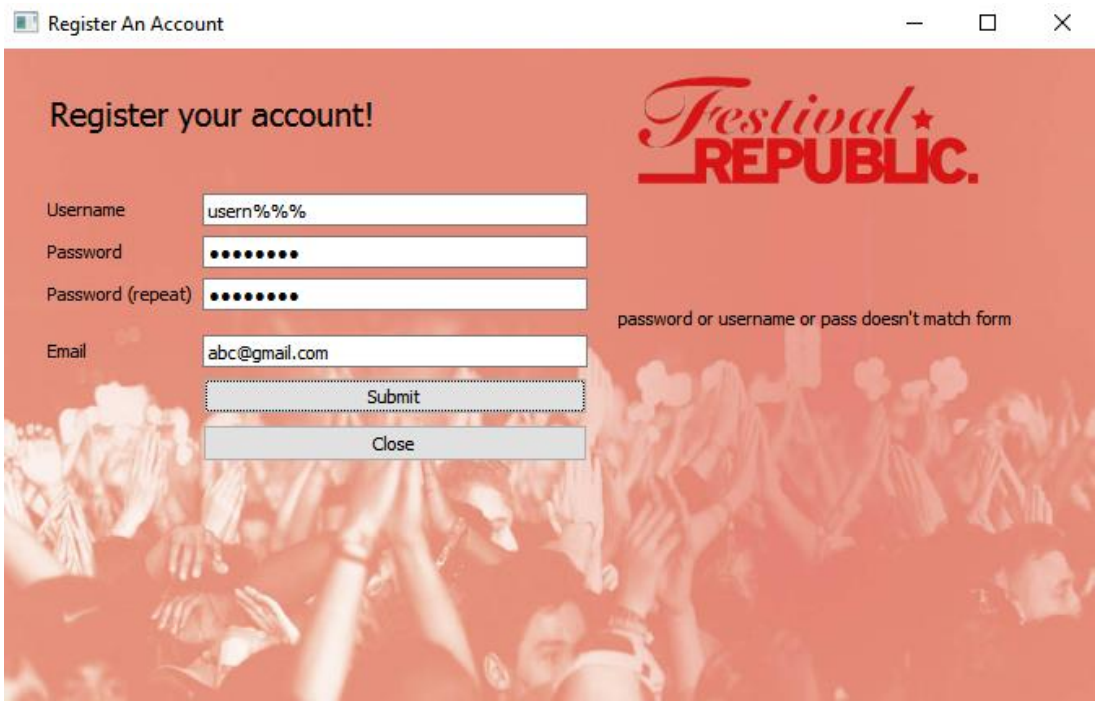


Criteria 5

Stress testing the submit button incurs no error as the code rejects any non-filled out entry immediately, rejects entries that do not conform to the strength rules or duplicate accounts.

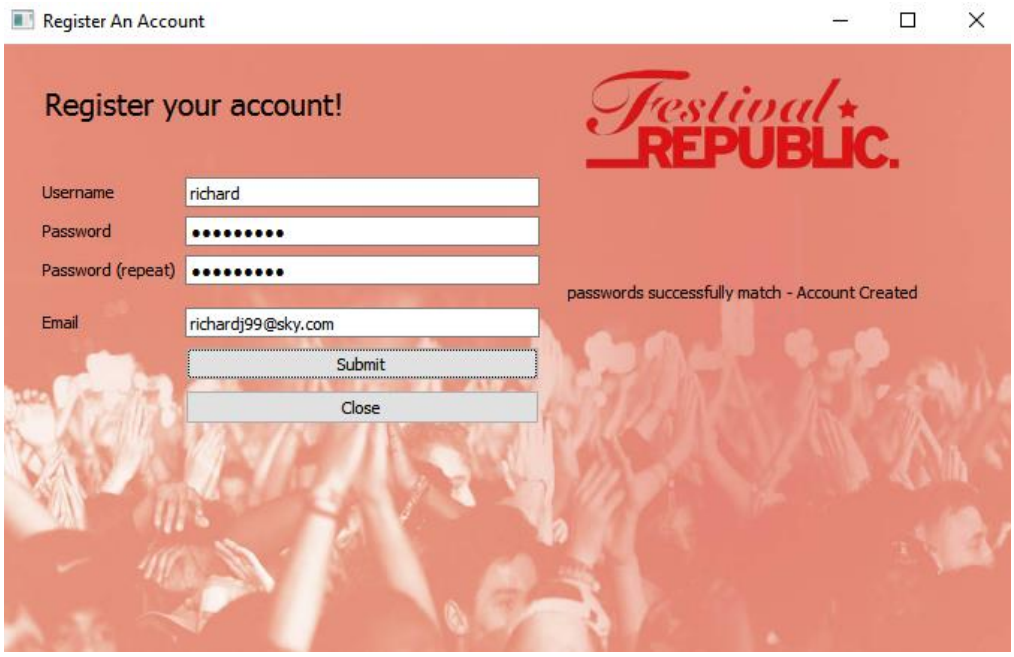
Criteria 6 - **Test for data entry/ Test for nothing entered**

Testing for data entry yields an error or a successful new account depending on the input, if it conforms to the rules and does not resemble the details of another account then it will be made:



Criteria 6

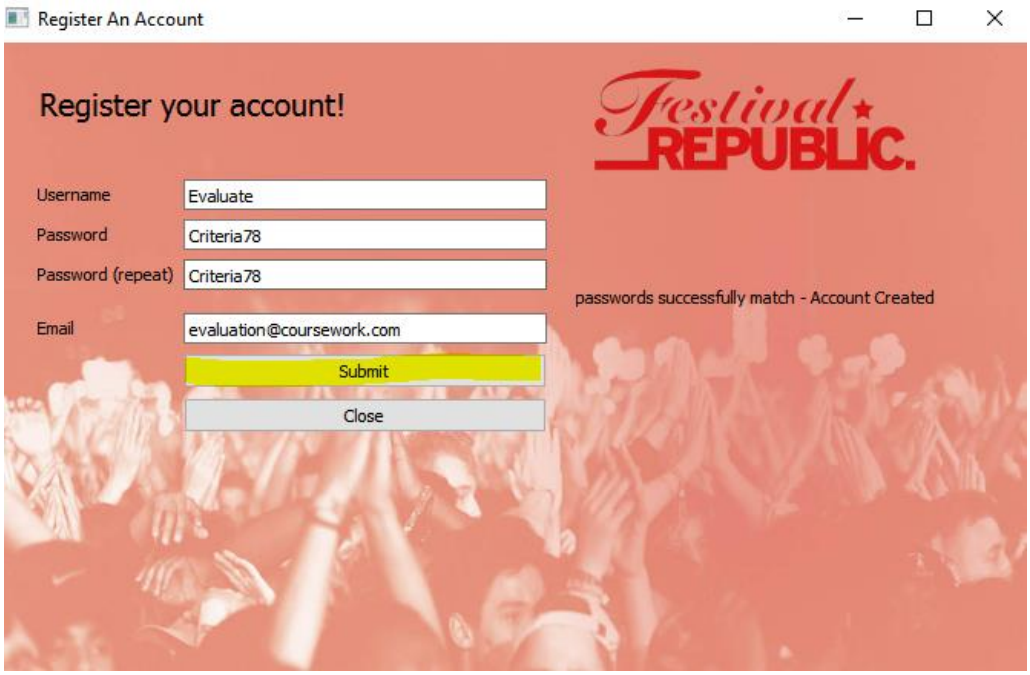
Or a successful user entry -



Criteria 6

Criteria 7 and 8 – Testing the Regex conformity of the program –

Usually the form disguises passwords, but for the sake of testing I've removed the hidden passwords-

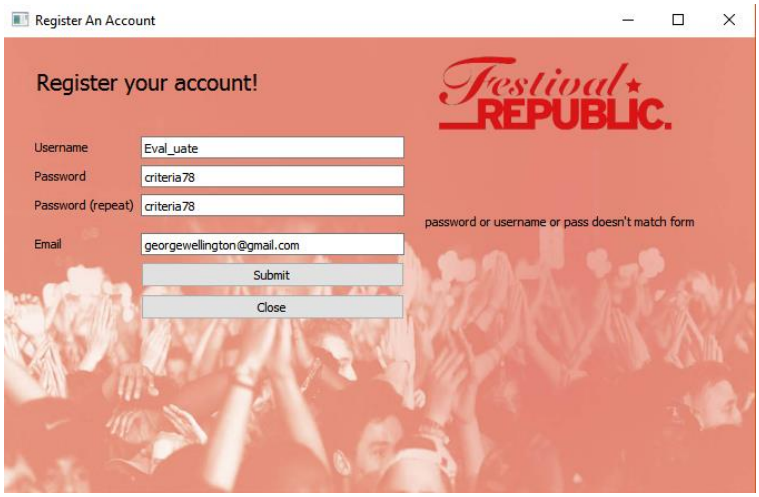


Criteria 7

The window above shows passwords and usernames conforming to the Regular Expression rules which are :

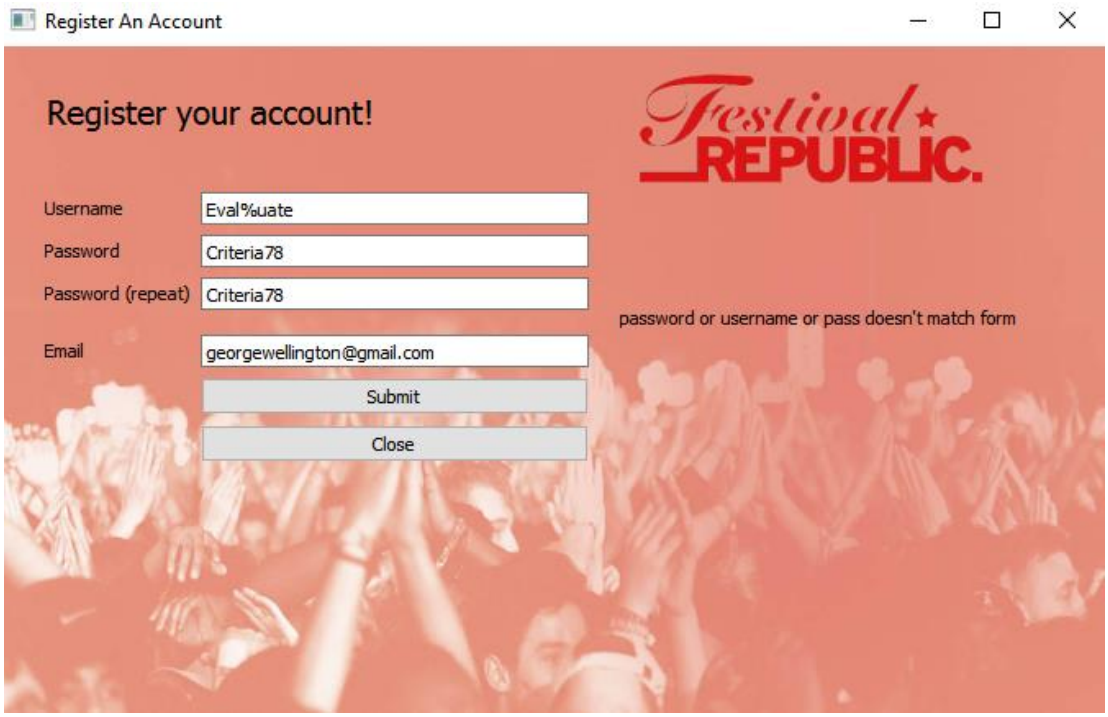
```
reg_user = r"^[a-zA-Z0-9][a-zA-Z0-9_]{4,12}$"
# starts with alphanumeric, can only contain alphanumeric or underscore with length 4-12
reg_pass = r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]{6,12}$"
# (6-12 chars, at least one lower,upper and number and no specials)
```

A window with non-conforming passwords would be rejected –



Criteria 8

As would a form inputted with a non-matching username –



The regular expression matches can be better seen here where it depicts what matches and what doesn't in a separate program:

Regular Expression Testing –

```
Enter usernamepseudo
matches
Enter usernamepseu
invalid
Enter usernamepseudo
matches
Enter usernamepseud
matches
Enter usernameabaABABA
matches
Enter usernameasdf_sadf
matches
Enter username_sadfsdaf
invalid
Enter username_asdf
invalid
Enter usernamesadfas_
matches
Enter username$%$fasdfsd
invalid
Enter username"£$"£$sadf
invalid
Enter username123sandy
matches
```

```
import re
regex = r"^[a-zA-Z0-9][a-zA-Z0-9_]{4,12}$"
while 1 == 1:
    test = input("Enter username")
    if re.match(regex, test):
        print("matches")
    else:
        print("invalid")
```

I developed a separate program specifically for testing the regex queries I use in the program. The program itself asked repeatedly for input and when given, outputs "invalid" if it does not match and "match" if it does.

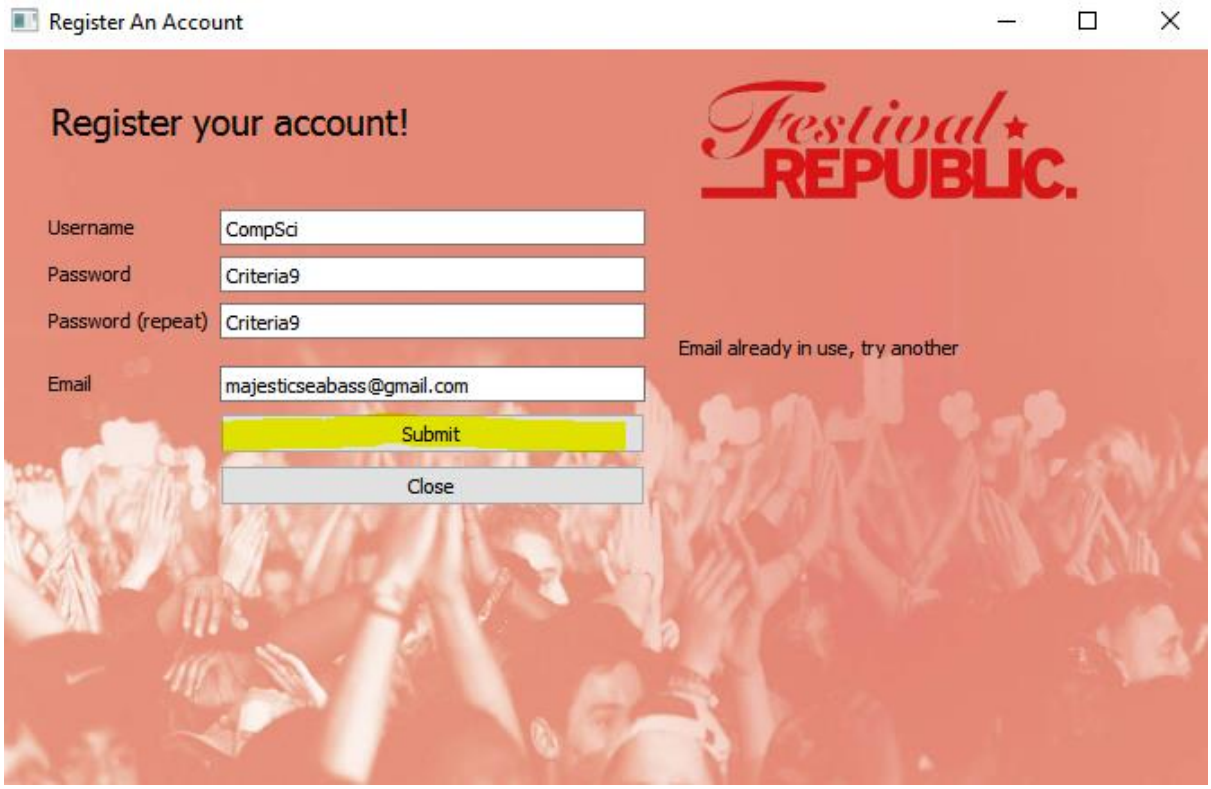
```
Enter passwordpseudo
invalid
Enter passwordPSEUDO
invalid
Enter password12pseudo
invalid
Enter password12PSEUDO
invalid
Enter passwordpseudo12
invalid
Enter passwordPSEUDO12
invalid
Enter password1Pseudo02
matches
Enter passwordpseud12d0AB
matches
Enter password0ABasdf123
matches
Enter passwordoabasf123
invalid
Enter password
```

```python
import re
regex = r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]{6,16}$"
while 1 == 1:
    test = input("Enter password")
    if re.match(regex, test):
        print("matches")
    else:
        print("invalid")
```

I developed a separate program specifically for testing the regex queries I use in the program. The program itself asked repeatedly for input and when given, outputs “invalid” if it does not match and “match” if it does.

Criteria 9 – Testing used and unused emails –

Registering an email with a used email would result in a rejection –

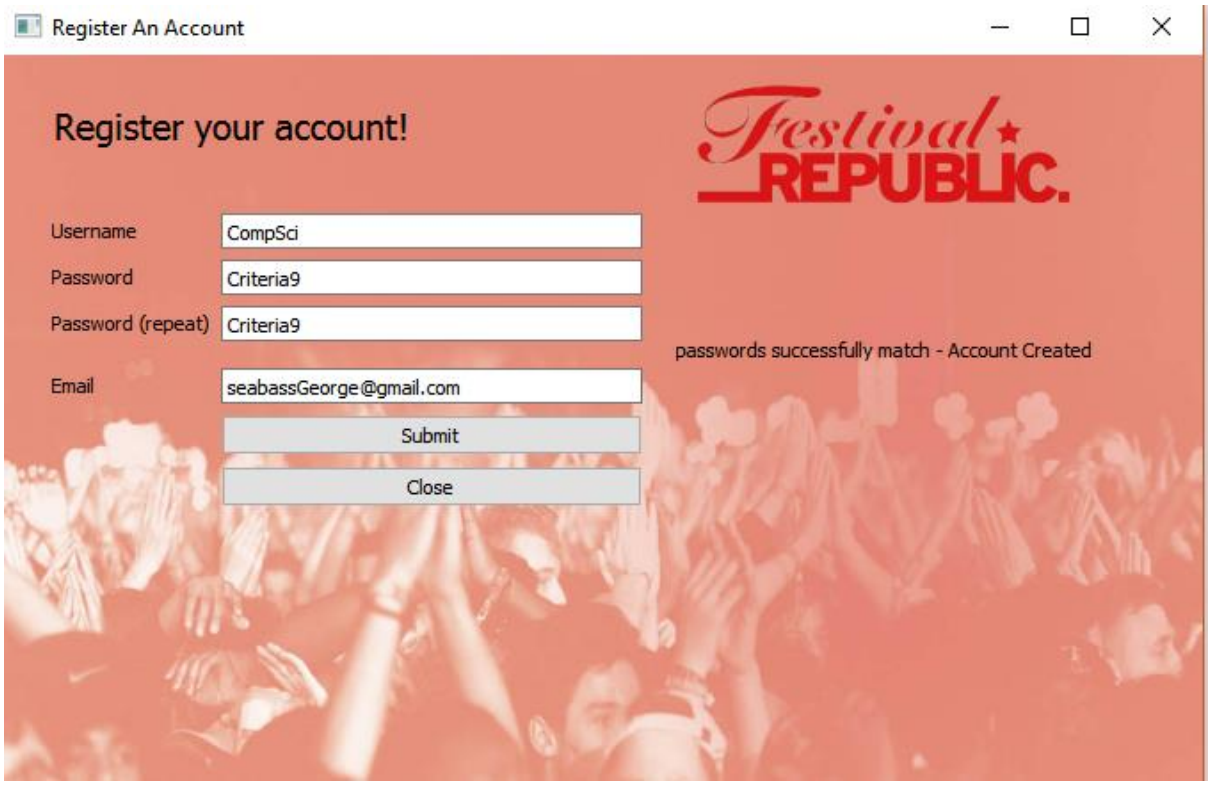


Criteria 9

The user in question:

	userID	username	password (hashed	email	permission level
1	1	a	0cc175b9c0f1b...	majesticseabas...	2
2	2	admin	6f2457b6086222...	cyberhylian@g...	1
3	5	complex	0d91a55ac531d...	philnsjspam@t...	2
4	8	jdsalinger	e7d8993bf6de6...	ltmattwelly@g...	1
5	10	jessica	db1a0dee243af...	jess@4818.com	1
6	11	George	30d2e8942cca8...	georgewellingt...	1
7	12	richard	d0bc30e069ddd...	richardj99@sky....	1

Whereas using a new email address will allow the user to be registered with the program:



Criteria 9

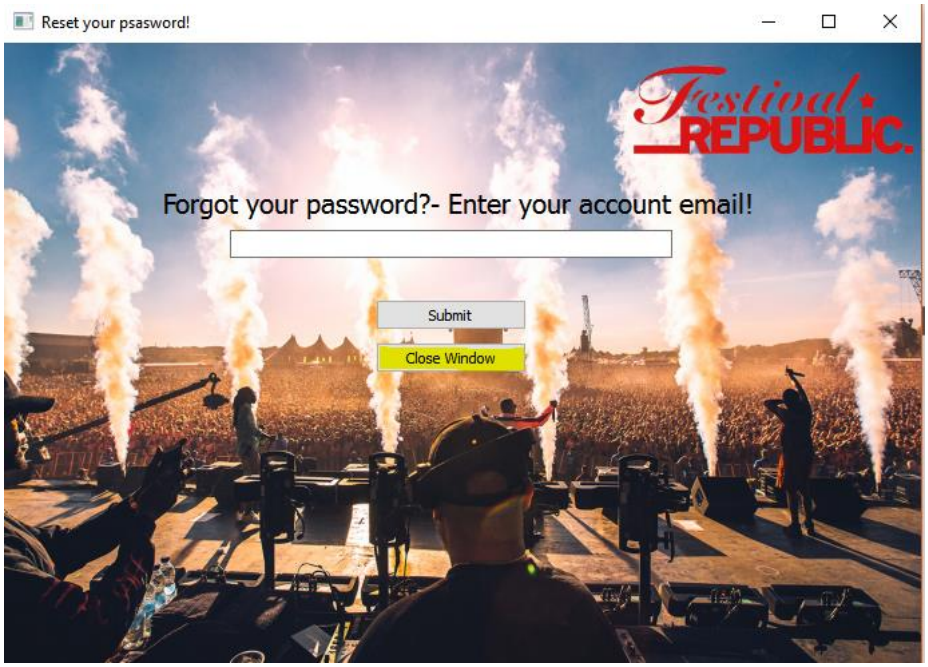
The account in question can clearly be seen now added here:

	userID	username	password (hashed	email	permission level
1	1	a	a4f647be6ad4a...	majesticseabas...	2
2	2	admin	6f2457b6086222...	cyberhylian@g...	1
3	5	complex	0d91a55ac531d...	philnsjspam@t...	2
4	8	jdsalinger	e7d8993bf6de6...	ltmattwelly@g...	1
5	10	jessica	db1a0dee243af...	jess@4818.com	1
6	11	George	30d2e8942cca8...	georgewellingt...	1
7	12	richard	d0bc30e069ddd...	richardj99@sky....	1
8	13	Evaluate	e8a63b225a5c2...	evaluation@co...	1
9	14	CompSci	d39f5e30ced67...	seabassGeorge...	1

Forgot Class (Criteria 10-12)

Criteria 10 –

Testing the links to different windows – in this case just the close button, works fine:

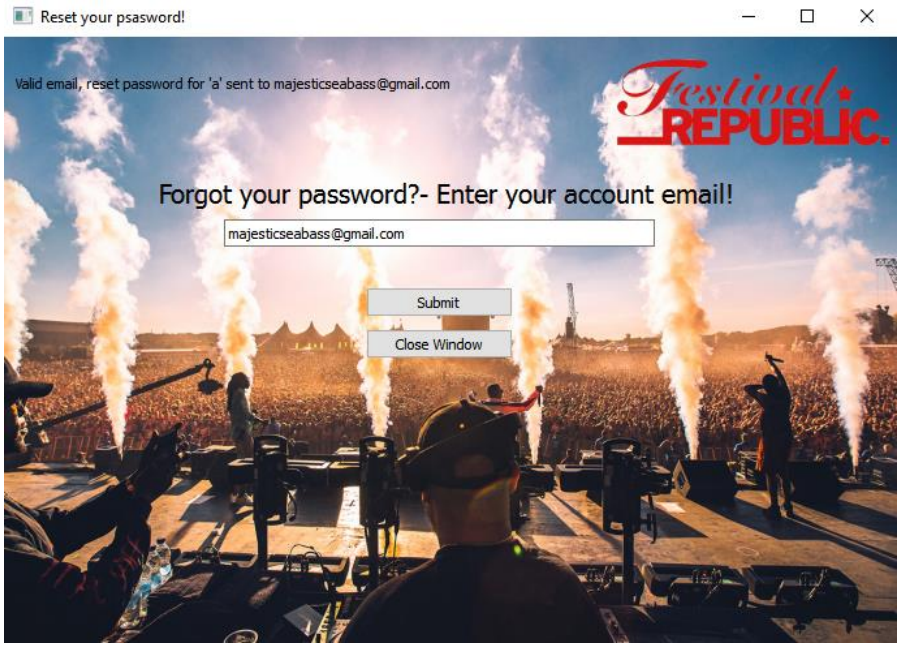


Criteria 10

Stress testing the GUI does not result in a crash, as a wrong/empty email field involves little processing and therefore does not crash the GUI.

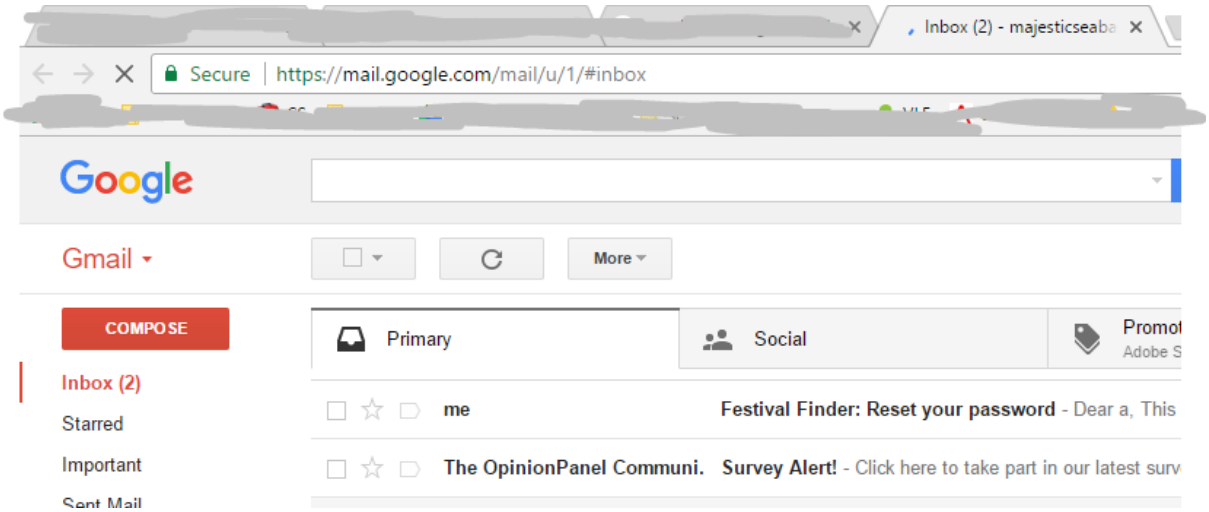
Criteria 11/12 –

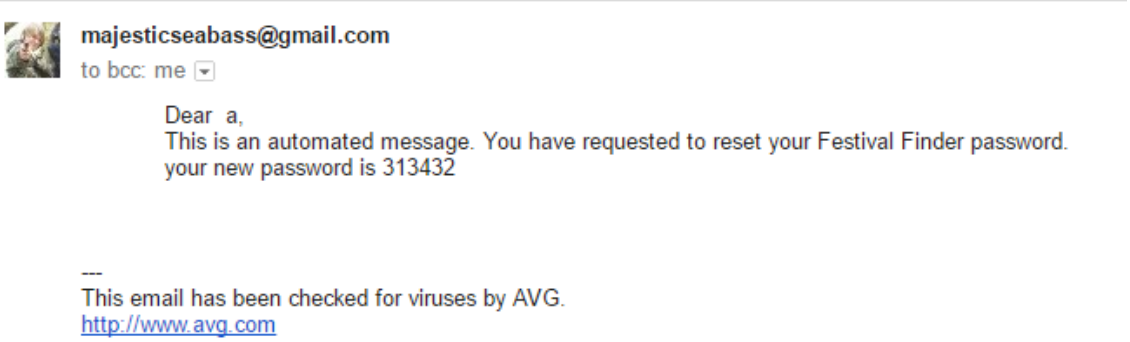
Testing for Data Entry yields a result dependent on the type of email entered, a user email will send an email:



Criteria 11

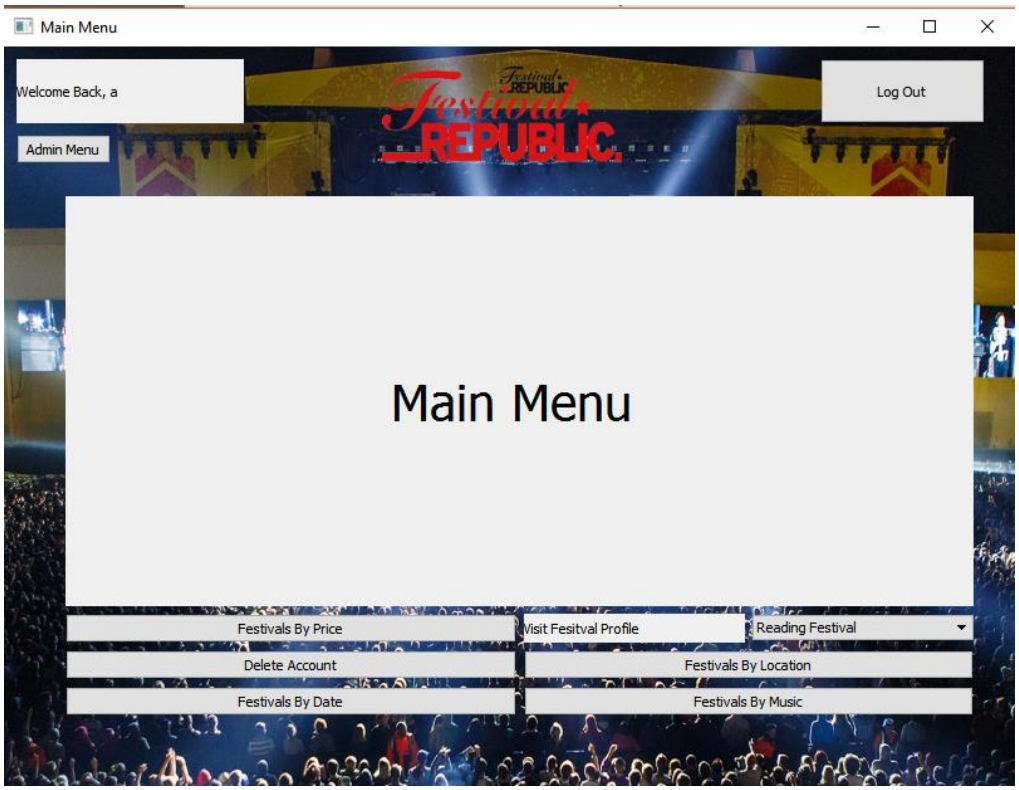
This activates the module emailpass.py – which does in fact send an email to the user with a newly reset password –





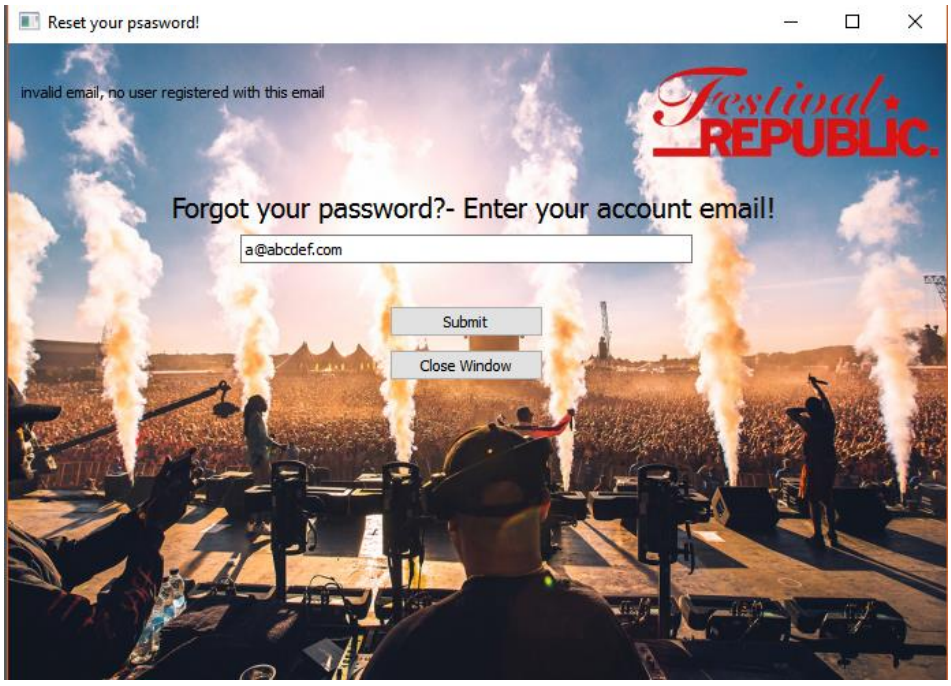
The password is assigned in the program, then hashed with md5 algorithms so it is recognised by the login process.

This password scans and lets the user in to the relevant menu:



Criteria 11

And an unused email being input will result in an error message:

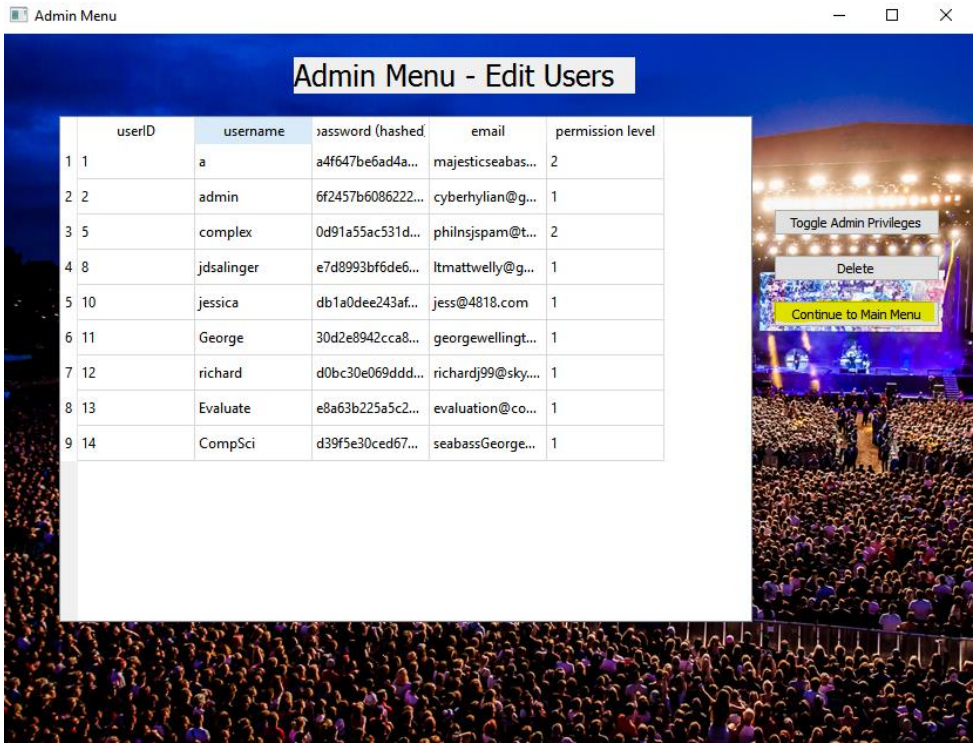


Criteria 12

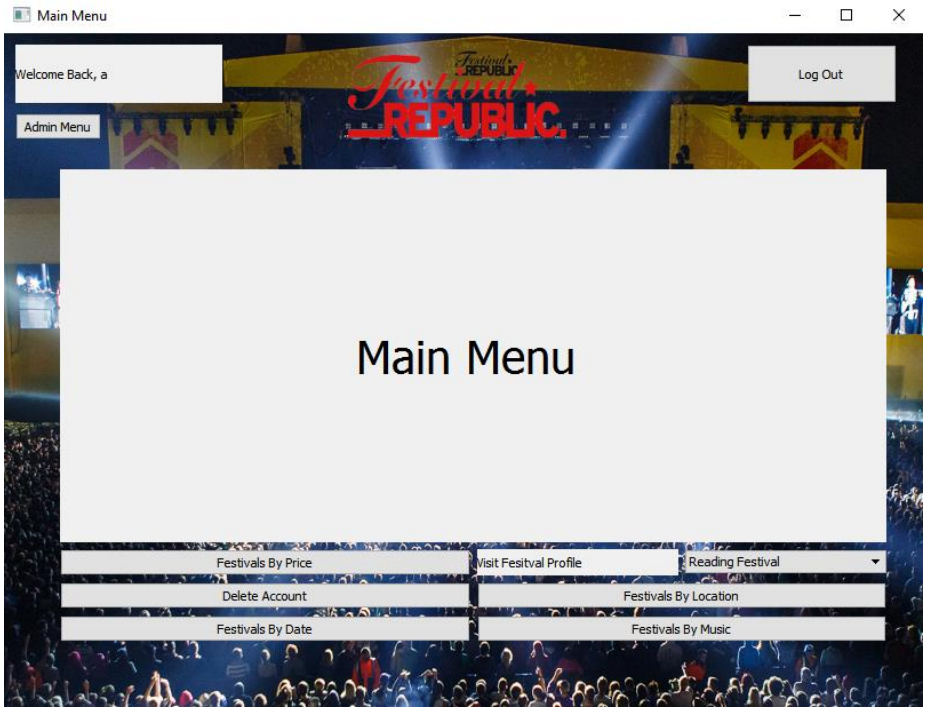
Admin Class (Criteria 13-16)

Criteria 13 – The Button links and the GUI Stress Testing

The buttons for this window link to the correct interfaces:



Criteria 13

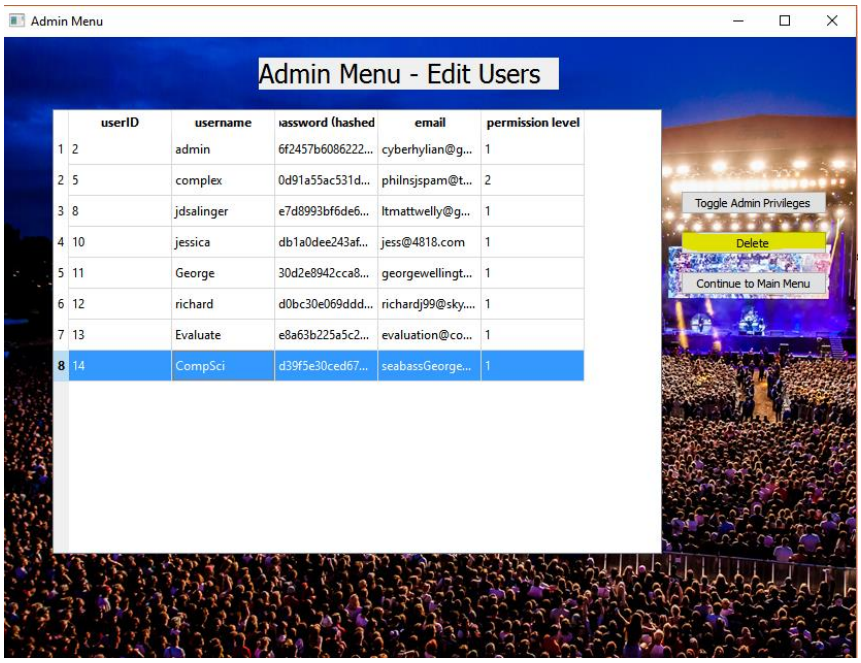


Criteria 13

Stress Testing the GUI leads to nothing substantial – The delete key only works on one account at a time and so spam clicking the button does nothing after the first press. The main menu button only

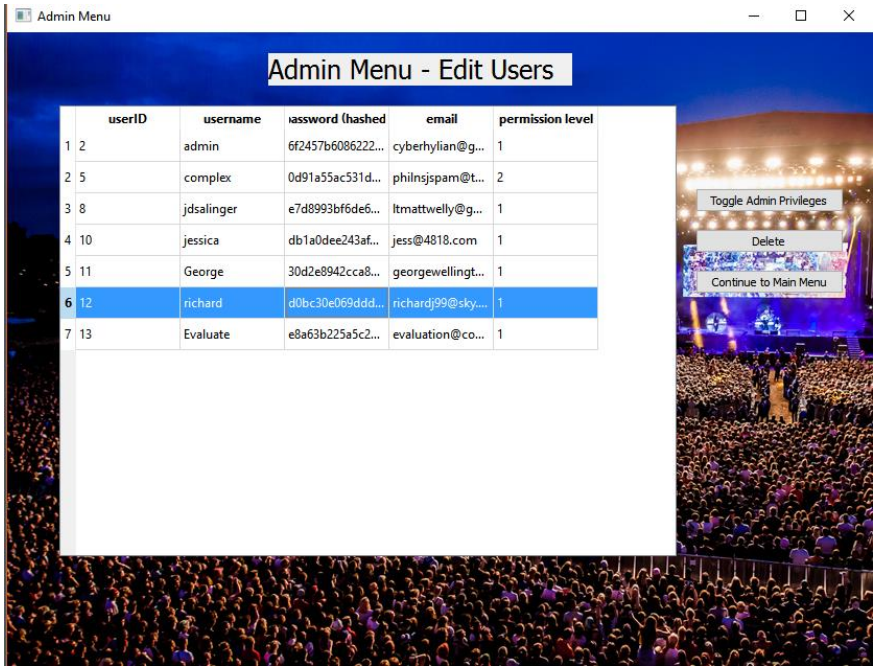
works once so therefore does not work successfully in crashing the GUI and the toggle admin privileges just toggles the privilege – it doesn't involve complex processing and so does nothing to hinder the program.

Criteria 14 - Test the delete button works –



Criteria 14

The delete function successfully deletes the row, deleting both the user and committing the changes – the table itself is automatically refreshed by the refresh() method every time something is clicked.

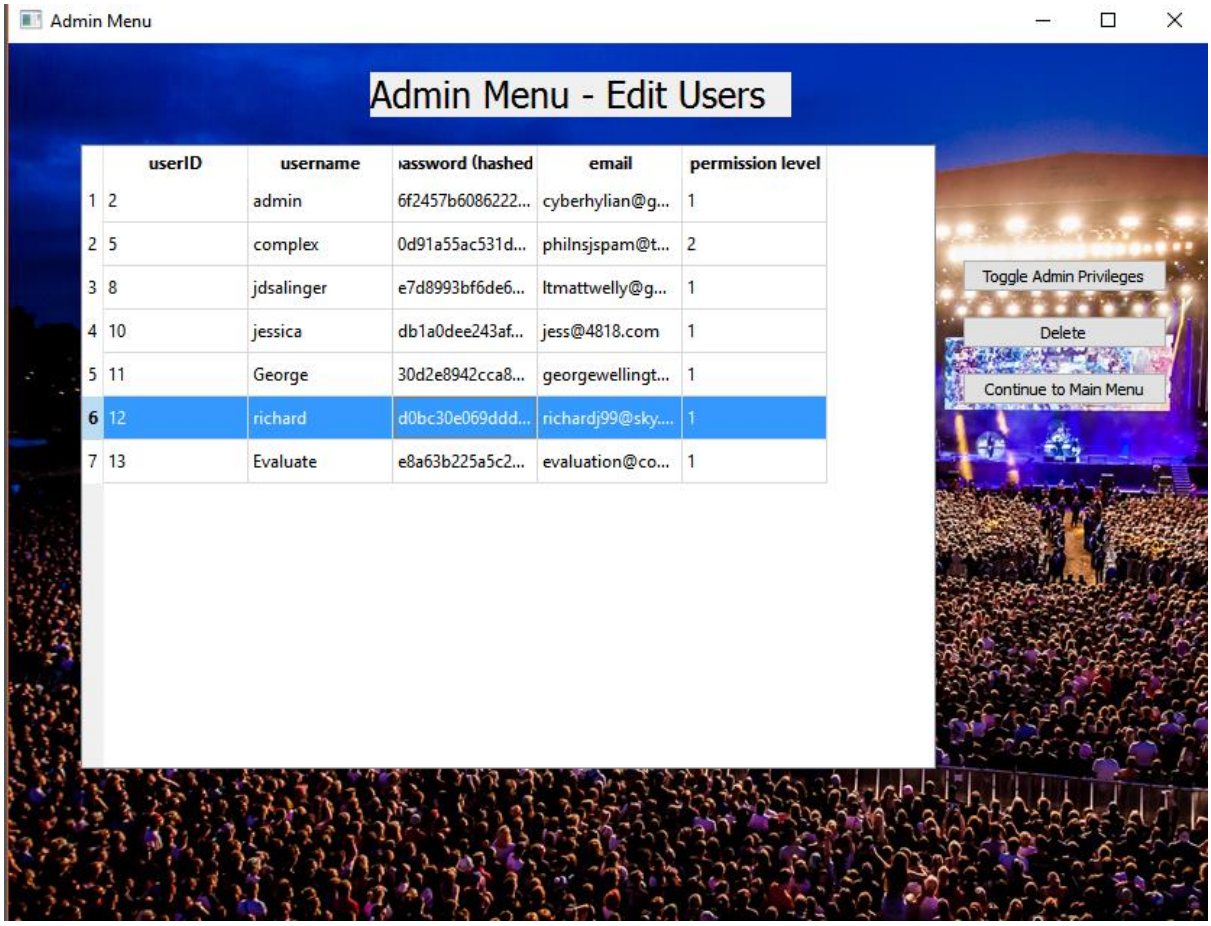


Criteria 14

The back-end no longer shows the user as the changes have been committed -

	userID	user	pass	email	perm_lvl
1	1	a	a4f647be6ad4ac0ae959985ccb362e71	majesticseabass@gmail.com	2
2	2	admin	6f2457b608622214f753c8522bf63a38	cyberhylian@gmail.com	1
3	5	complex	0d91a55ac531df50ede54f22b23ed9ec	philnsjspam@talktalk.net	2
4	8	jdsalinger	e7d8993bf6de68a2d4e3a8a24e51f4fd	ltmattwelly@gmail.com	1
5	10	jessica	db1a0dee243af63d8db3703ab7d80e53	jess@4818.com	1
6	11	George	30d2e8942cca845bc66236b4de28ae07	georgewellingtonwork@gmail.com	1
7	12	richard	d0bc30e069ddd34121be0c4a8b00231b	richardj99@sky.com	1
8	13	Evaluate	e8a63b225a5c2730367036816fd5c44d	evaluation@coursework.com	1

Criteria 15 – Data Output and Deleting your own account –



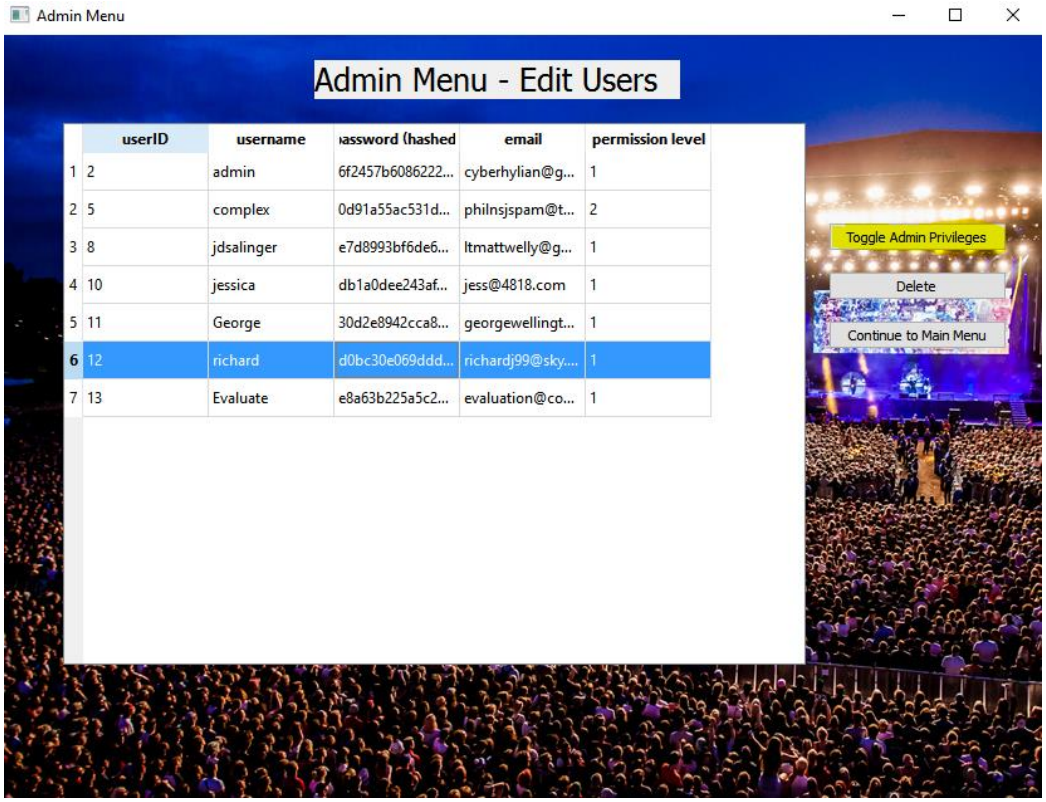
Criteria 15

The table widget clearly displays all the data in the back-end database file:

	userID	user	pass	email	perm_lvl
1	1	a	a4f647be6ad4ac0ae959985ccb362e71	majesticseabass@gmail.com	2
2	2	admin	6f2457b608622214f753c8522bf63a38	cyberhylian@gmail.com	1
3	5	complex	0d91a55ac531df50ede54f22b23ed9ec	philnsjspam@talktalk.net	2
4	8	jdsalinger	e7d8993bf6de68a2d4e3a8a24e51f4fd	ltmattwelly@gmail.com	1
5	10	jessica	db1a0dee243af63d8db3703ab7d80e53	jess@4818.com	1
6	11	George	30d2e8942cca845bc66236b4de28ae07	georgewellingtonwork@gmail.com	1
7	12	richard	d0bc30e069ddd34121be0c4a8b00231b	richardj99@sky.com	1
8	13	Evaluate	e8a63b225a5c2730367036816fd5c44d	evaluation@coursework.com	1

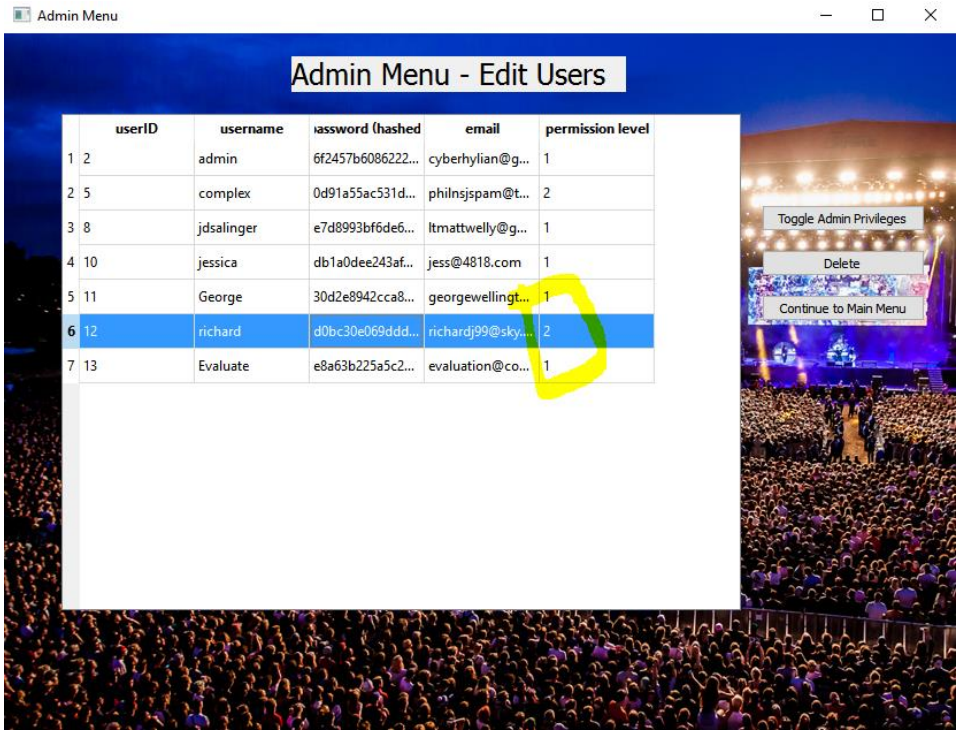
Apart from the user currently logged in (“a”) – which prevents a user deleting oneself. The table initially displays all the users and then removes the currently signed in user after any action from the user as a result of the refresh() method being called.

Criteria 16 – Testing the administrator toggling mode:



Criteria 16

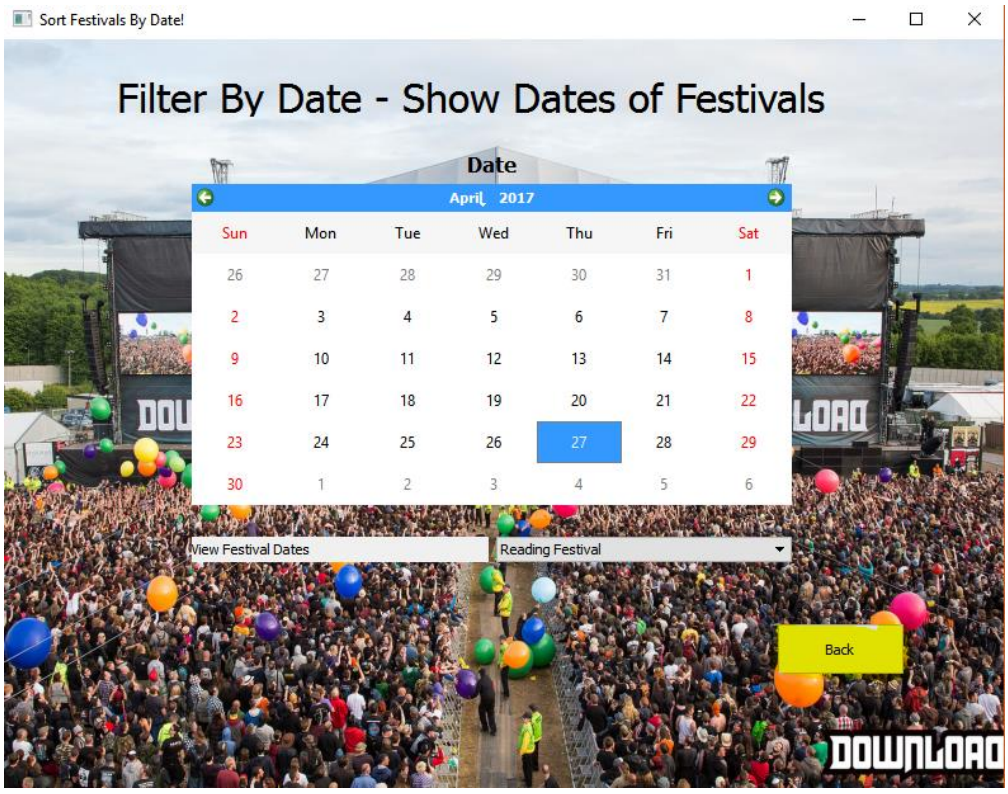
The program toggles the permission level accordingly with the currently selected user and commits the changes, changing a value of one to two and two to one. This immediately allows a new admin to resume work:



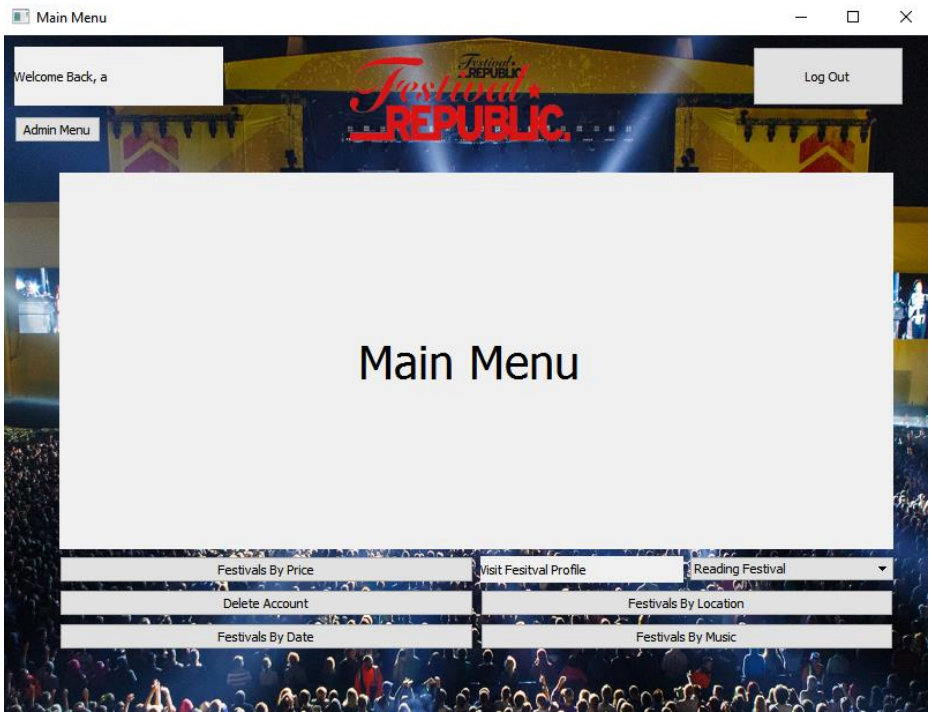
Criteria 16

Date Class (Criteria 17-19)

Testing the links to different windows – in this case just the close button, works fine:



Criteria 17

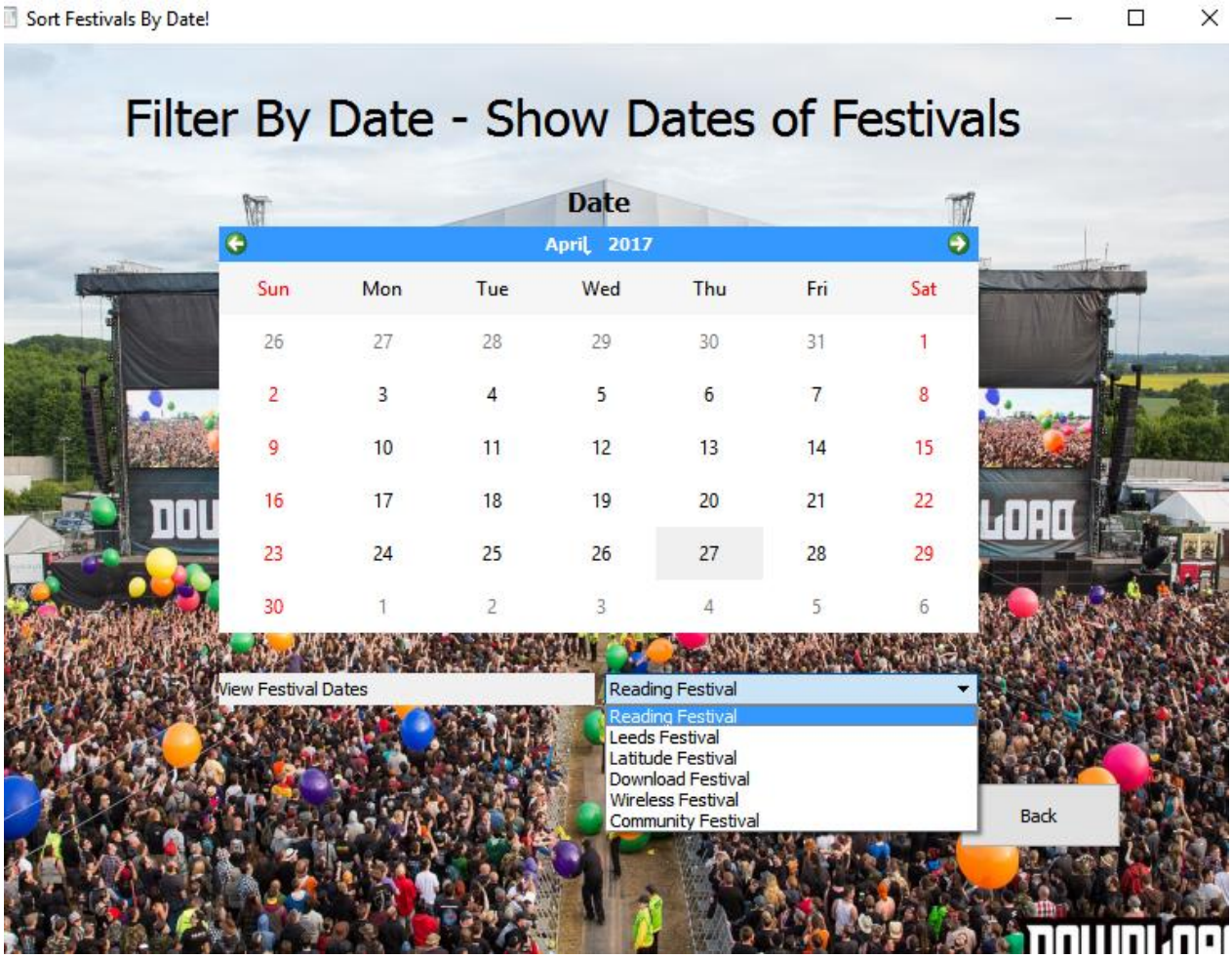


Criteria 17

Stress testing the various functions is incredibly hard to do as a dropdown menu and a back button cannot be spammed due to their nature – the program cannot be forced to crash this way.

Criteria 18 (Correct Output on Combobox):

The program correctly outputs the list of Festival choices iteratively into the combo-box:



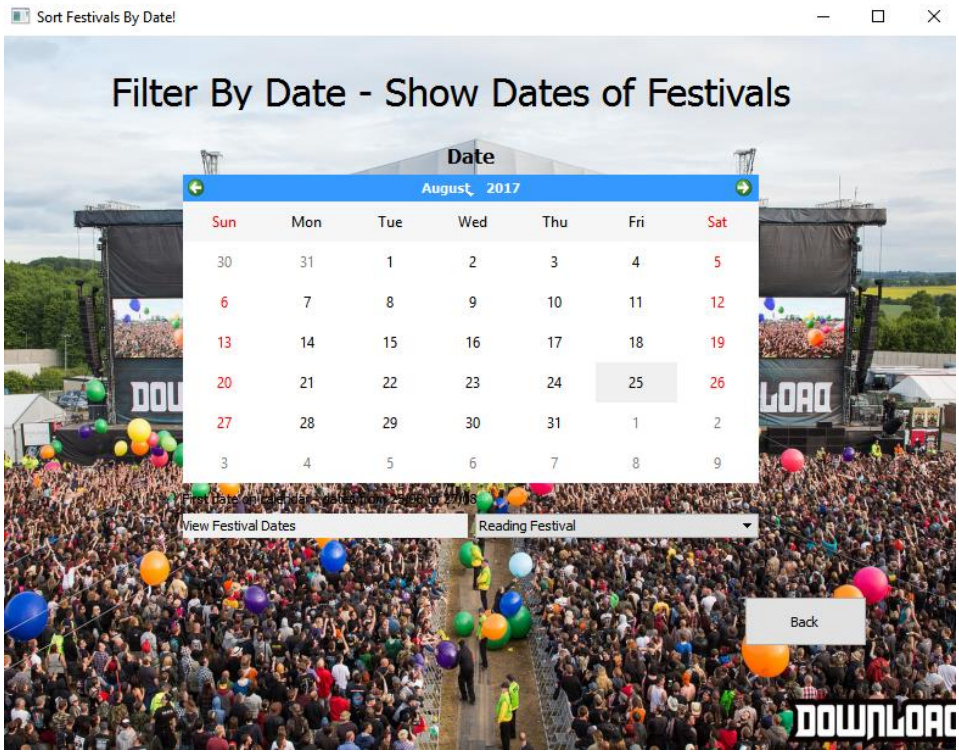
Criteria 18

	festID	Name	Postcode	StartDate	EndDate	Cost	DayCost
1	1	Reading Festival	RG1 8EQ	25/08	27/08	213	72
2	2	Leeds Festival	LS23 6ND	25/08	27/08	213	72
3	3	Latitude Festival	NR34 8AQ	13/07	17/07	197.5	84.5
4	4	Download Festival	DE74 2RP	09/07	11/07	205	83
5	5	Wireless Festival	N4 1EE	08/07	10/07	210	62
6	6	Community Festival	N4 1EF	01/07	01/07	40.3	40.3

Criteria 19 (Showing the Correct Dates for Festivals):

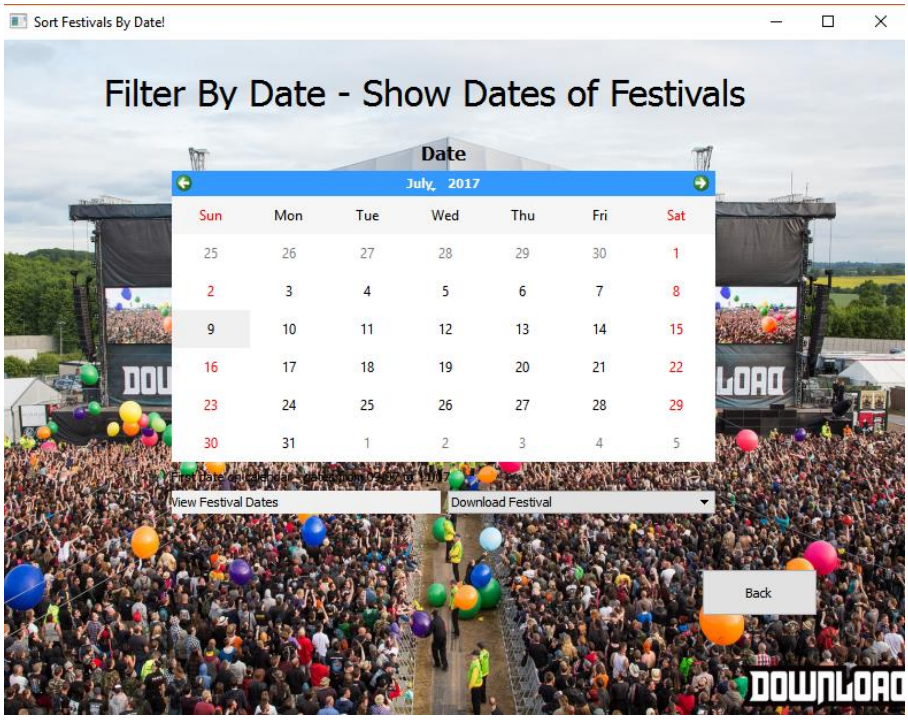
The program successfully plots each date for every separate festival on the calendar:

	festID	Name	Postcode	StartDate	EndDate	Cost	DayCost
1	1	Reading Festival	RG1 8EQ	25/08	27/08	213	72
2	2	Leeds Festival	LS23 6ND	25/08	27/08	213	72
3	3	Latitude Festival	NR34 8AQ	13/07	17/07	197.5	84.5
4	4	Download Festival	DE74 2RP	09/07	11/07	205	83
5	5	Wireless Festival	N4 1EE	08/07	10/07	210	62
6	6	Community Festival	N4 1EF	01/07	01/07	40.3	40.3



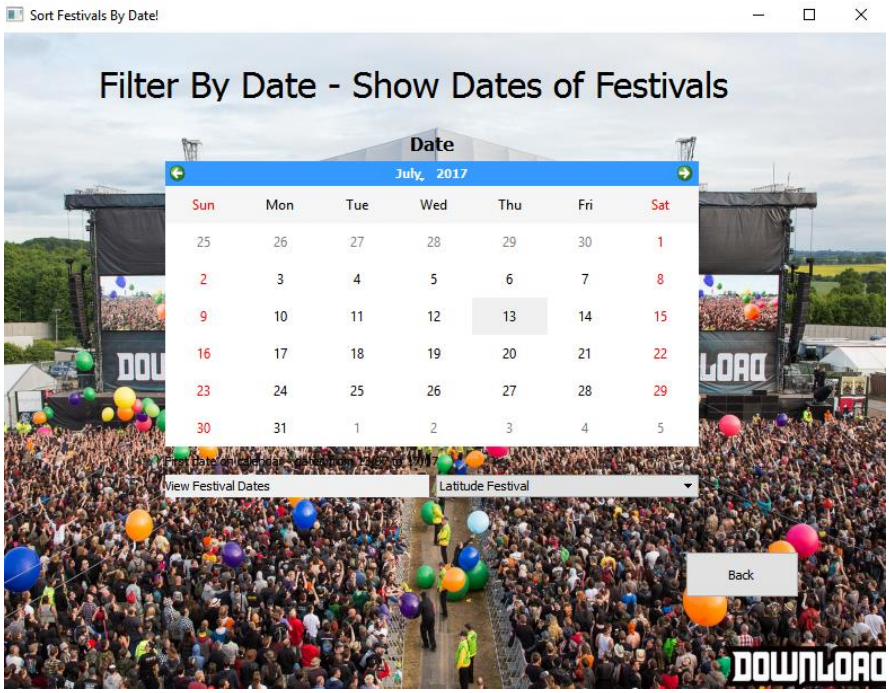
Criteria 19

Above demonstrates the setting when set to “Reading Festival”, displaying the date of the festival in question.



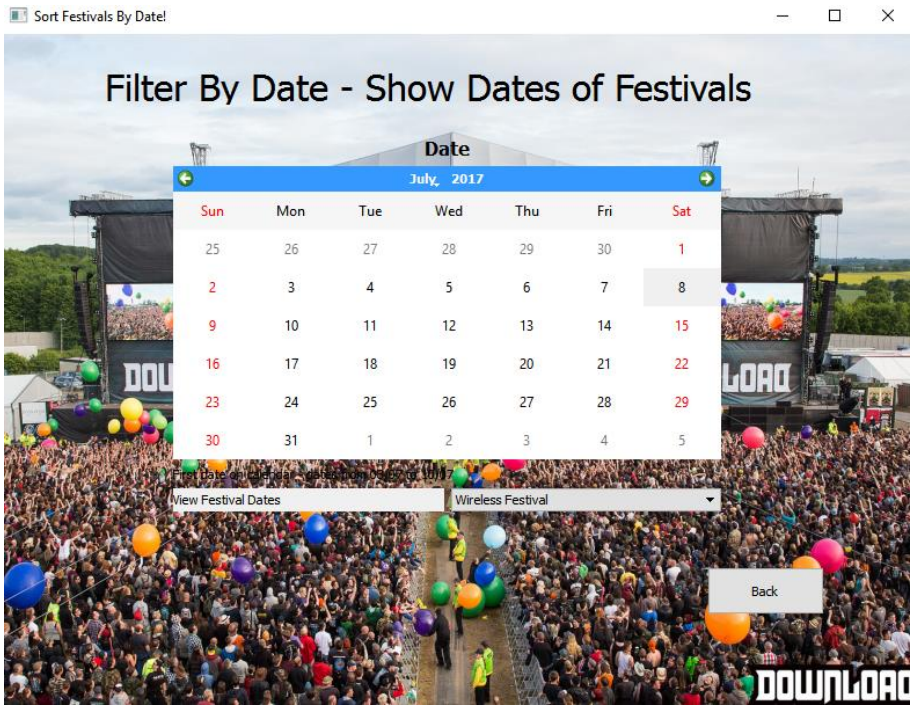
Criteria 19

Above demonstrates the setting when set to “Download Festival”, displaying the date of the festival in question.



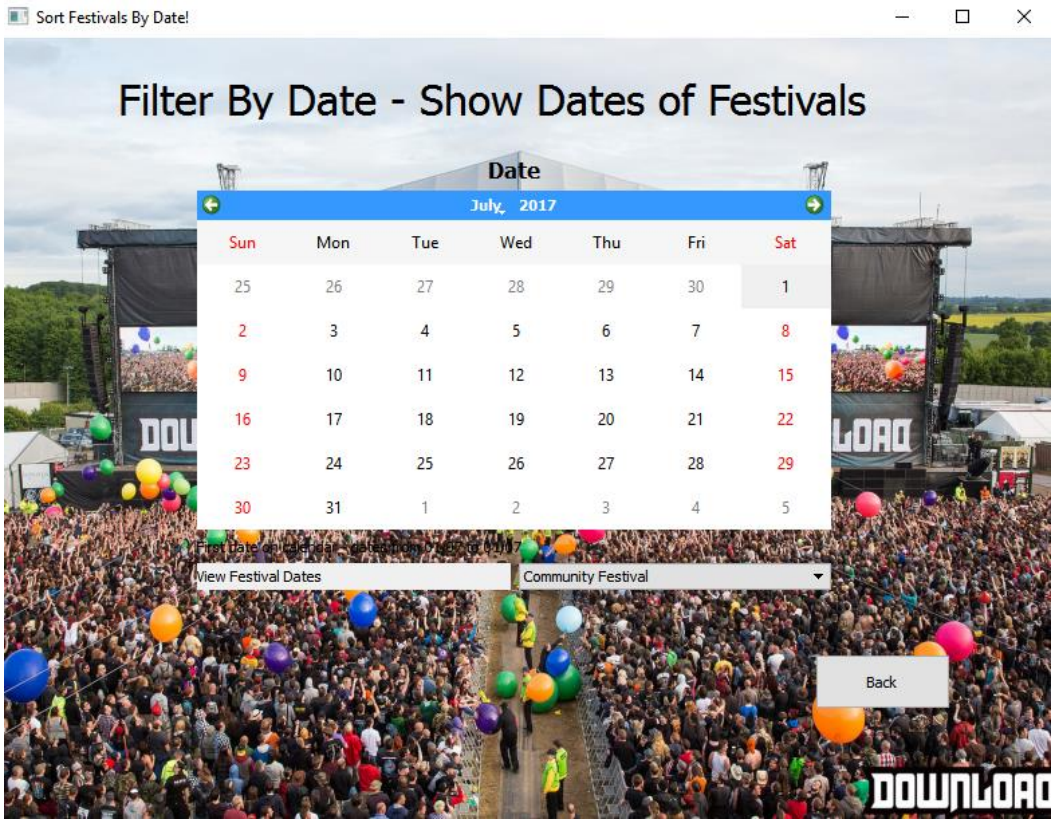
Criteria 19

Above demonstrates the setting when set to “Latitude Festival”, displaying the date of the festival in question.



Criteria 19

Above demonstrates the setting when set to “Wireless Festival”, displaying the date of the festival in question.



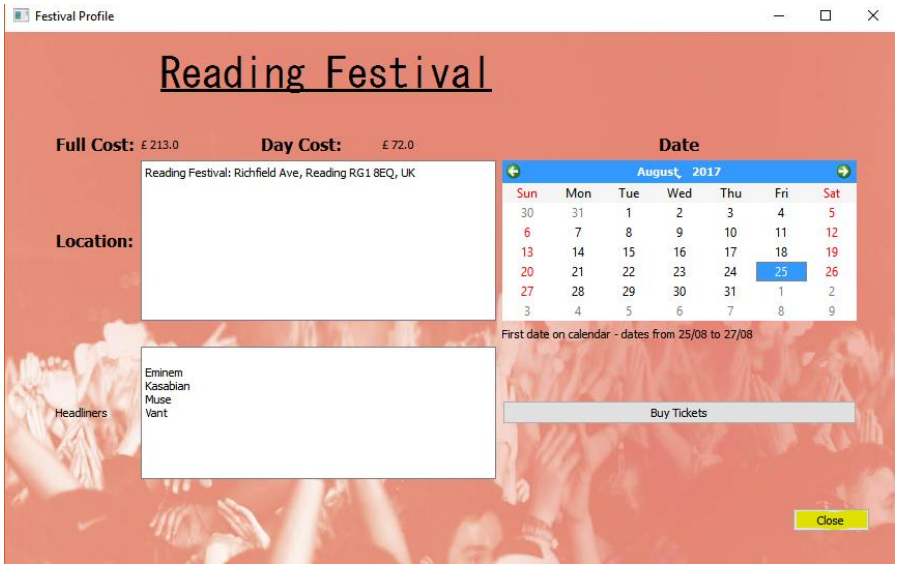
Criteria 19

Above demonstrates the setting when set to "Community Festival", displaying the date of the festival in question.

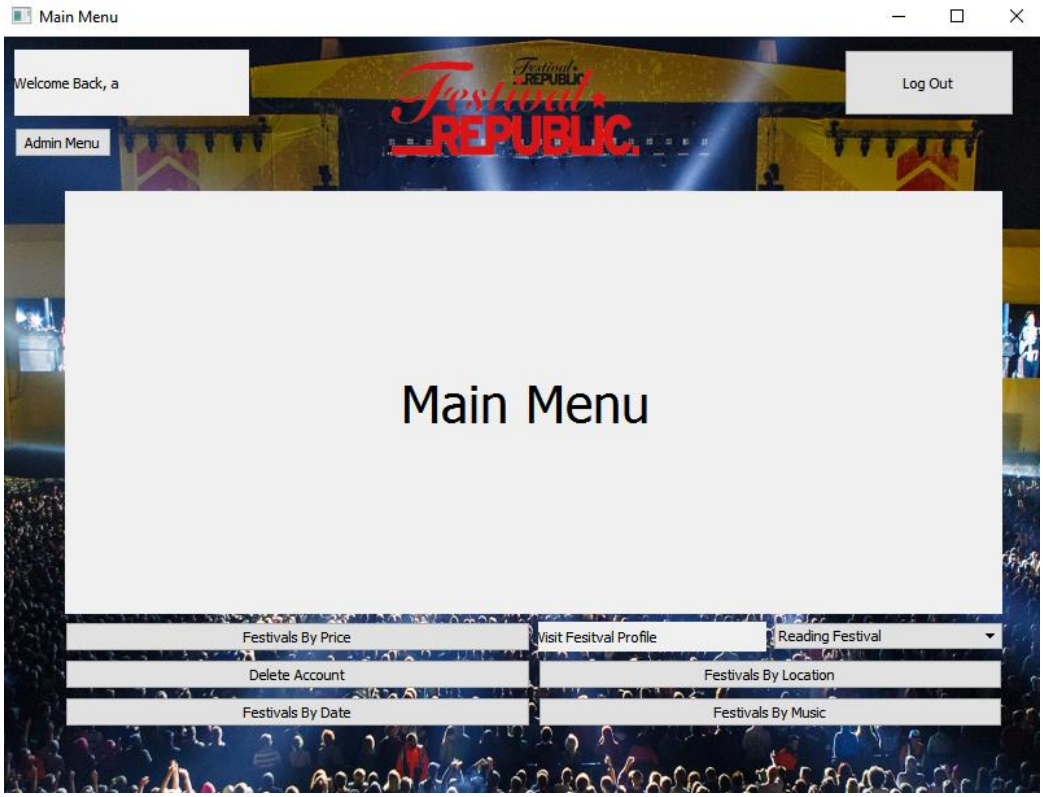
Profile Class (Criteria 20-22)

Criteria 20 -

Testing the links to different windows – in this case just the close button, works fine, returning to the main menu.:

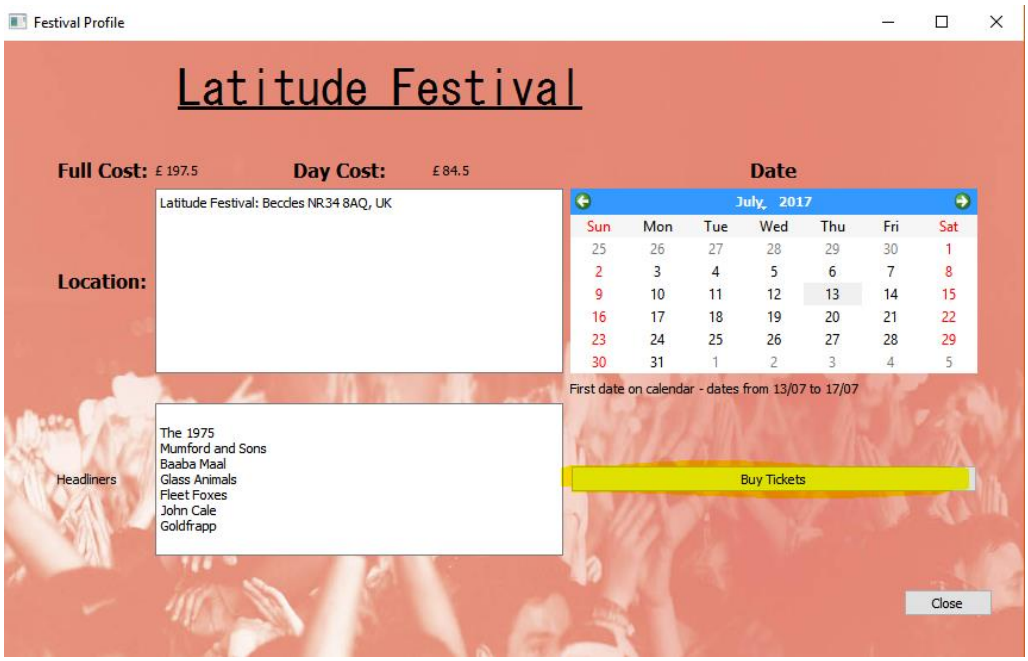


Criteria 20

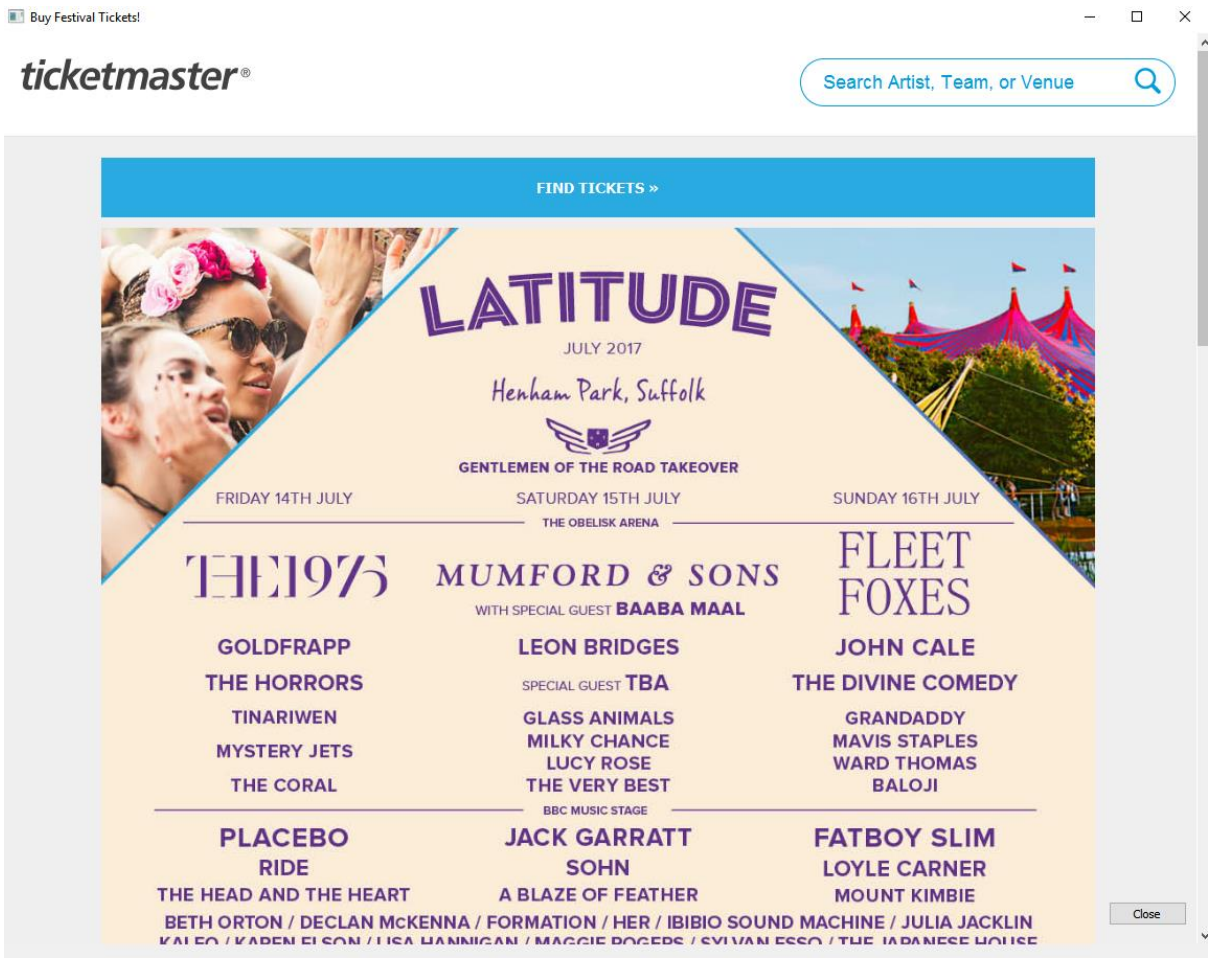


Criteria 21 – The Program links to the correct ticket vendor URL –

The program guesses the URL by concatenating the URL of a ticket vendor and the festival name. This method works wonders and works successfully for every single festival – providing a direct link to the vendor for tickets to the festival in question on first connect:

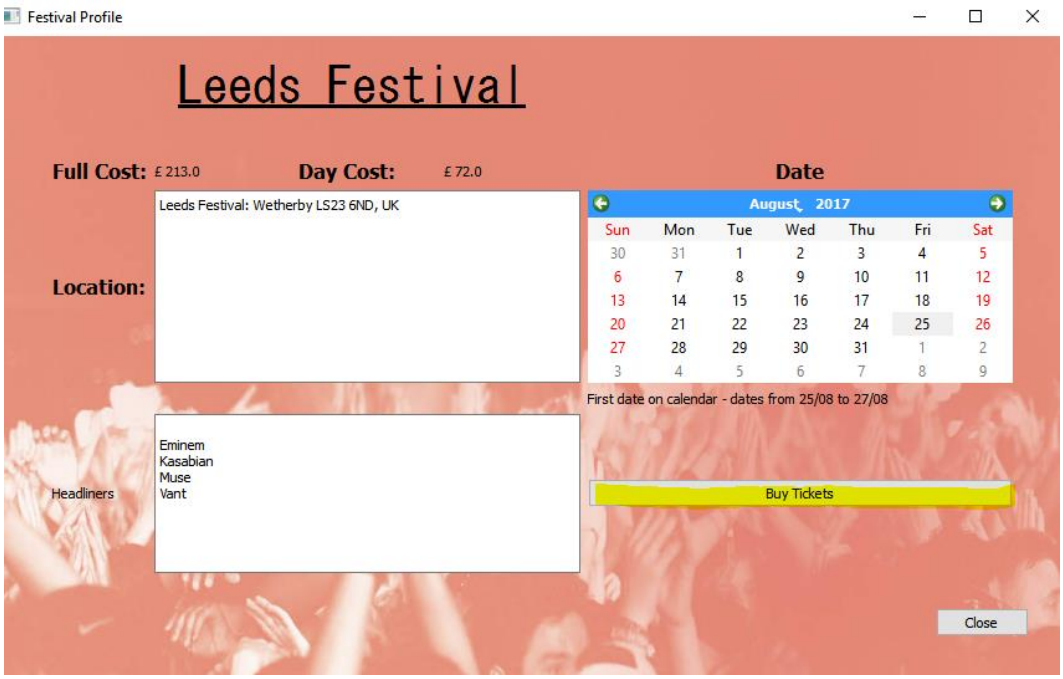


Criteria 21

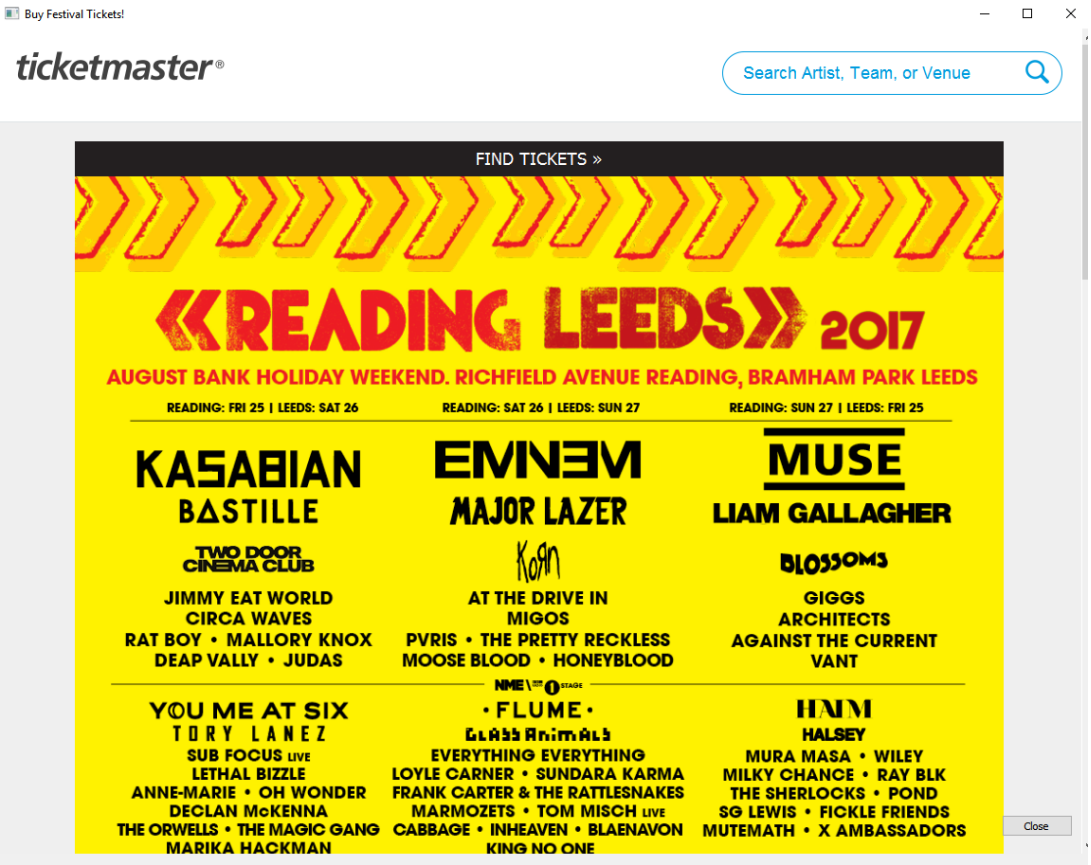


Criteria 21

Above demonstrates the working link for “Latitude Festival”



Criteria 21

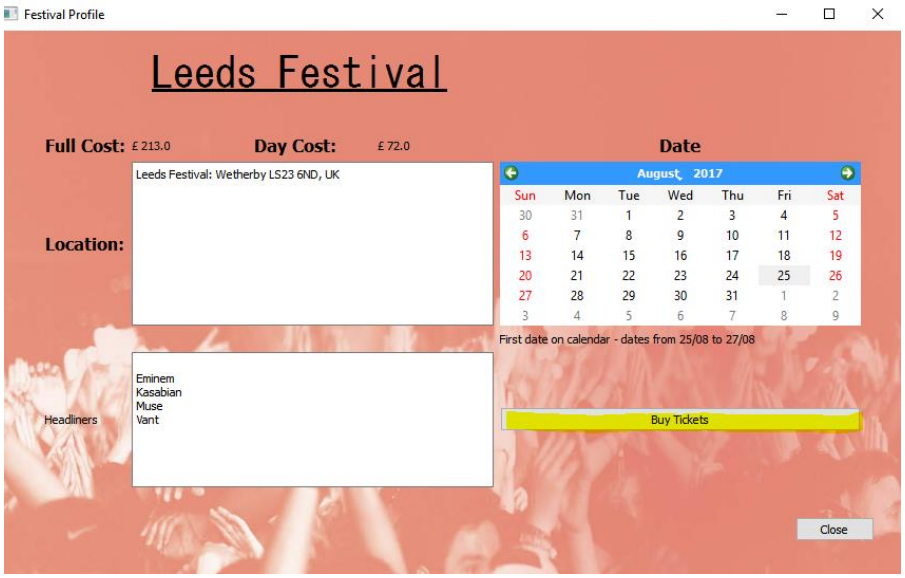


Criteria 21

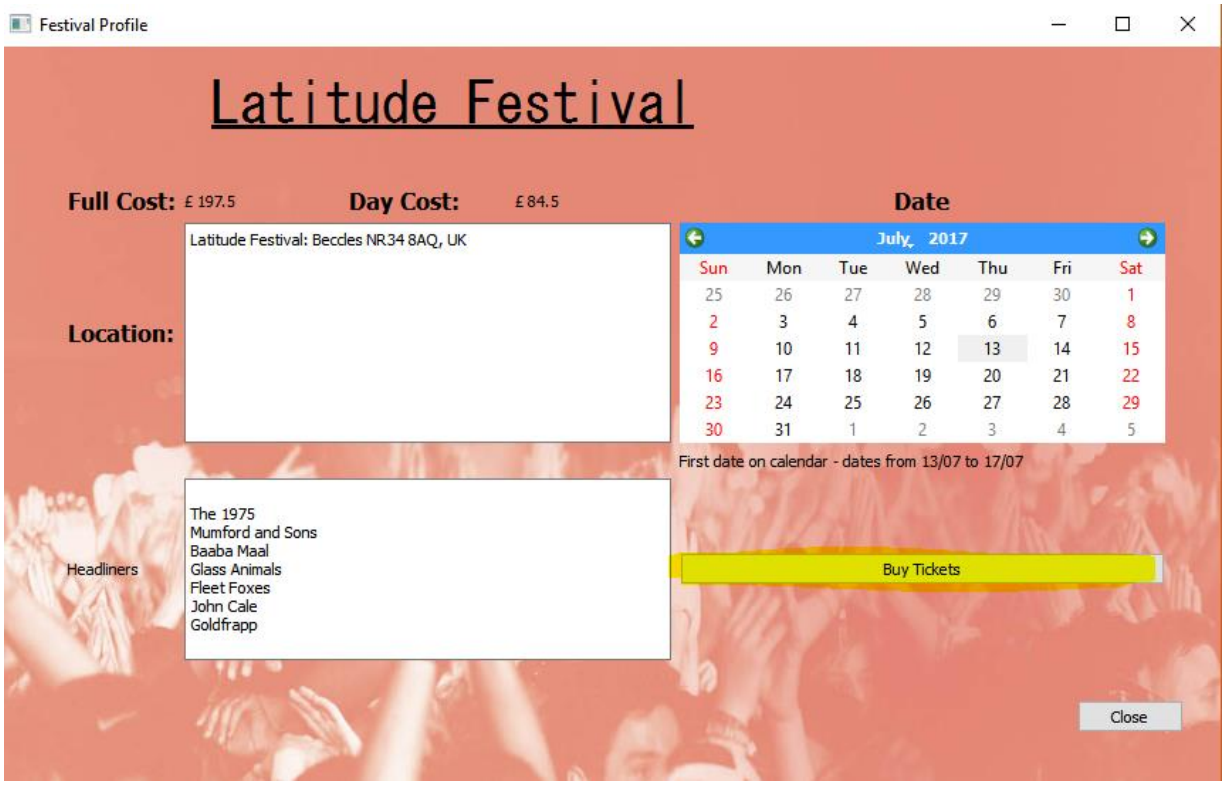
Above demonstrates the working link for “Leeds Festival”

Criteria 22 (Correctly Display all the festival information):

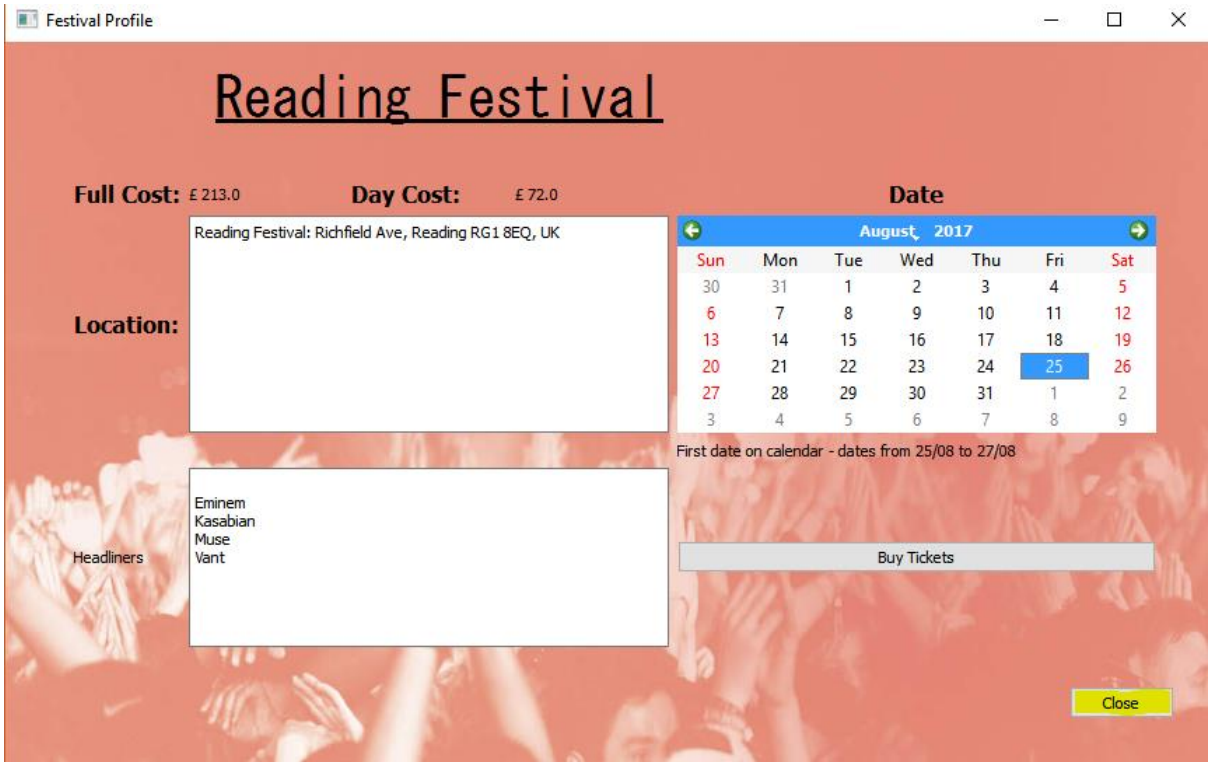
The program successfully displays the information for all the festivals:



Criteria 22

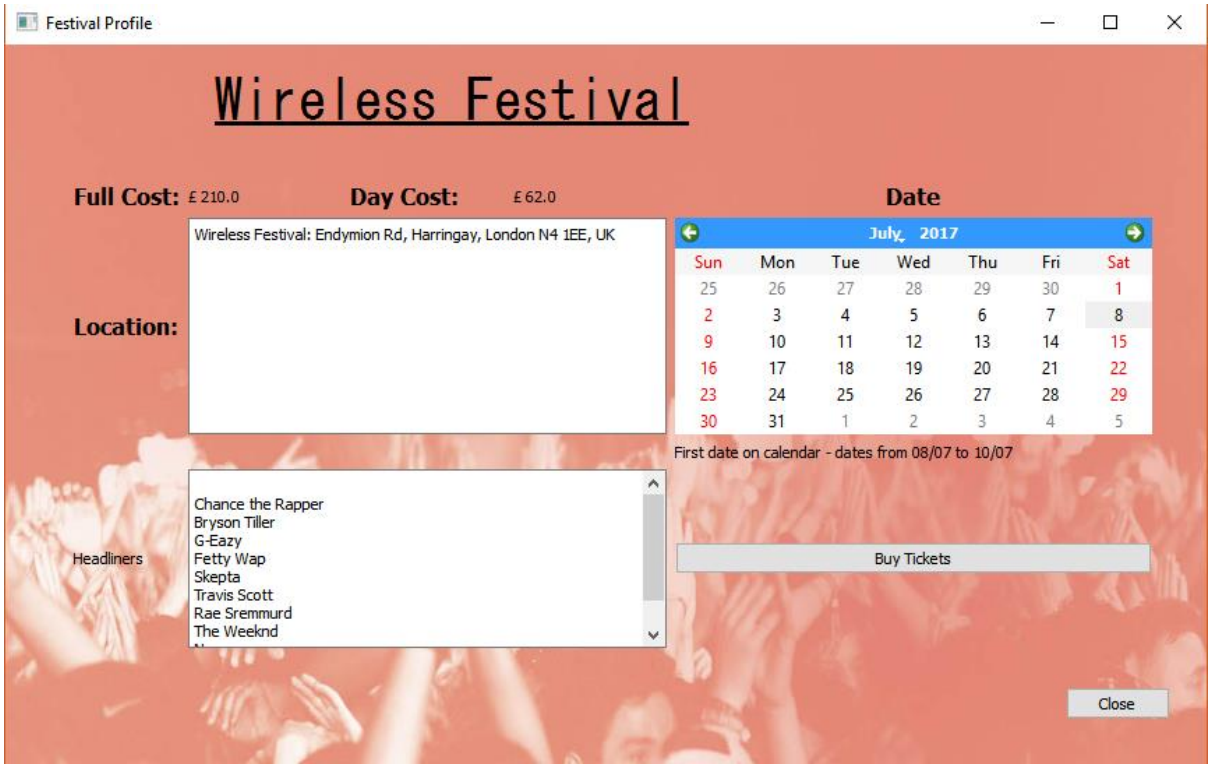


Criteria 22



Criteria 22

Criteria 22



Criteria 22

Criteria 22

	festID	Name	Postcode	StartDate	EndDate	Cost	DayCost
1	1	Reading Festival	RG1 8EQ	25/08	27/08	213	72
2	2	Leeds Festival	LS23 6ND	25/08	27/08	213	72
3	3	Latitude Festival	NR34 8AQ	13/07	17/07	197.5	84.5
4	4	Download Festival	DE74 2RP	09/07	11/07	205	83
5	5	Wireless Festival	N4 1EE	08/07	10/07	210	62
6	6	Community Festival	N4 1EF	01/07	01/07	40.3	40.3

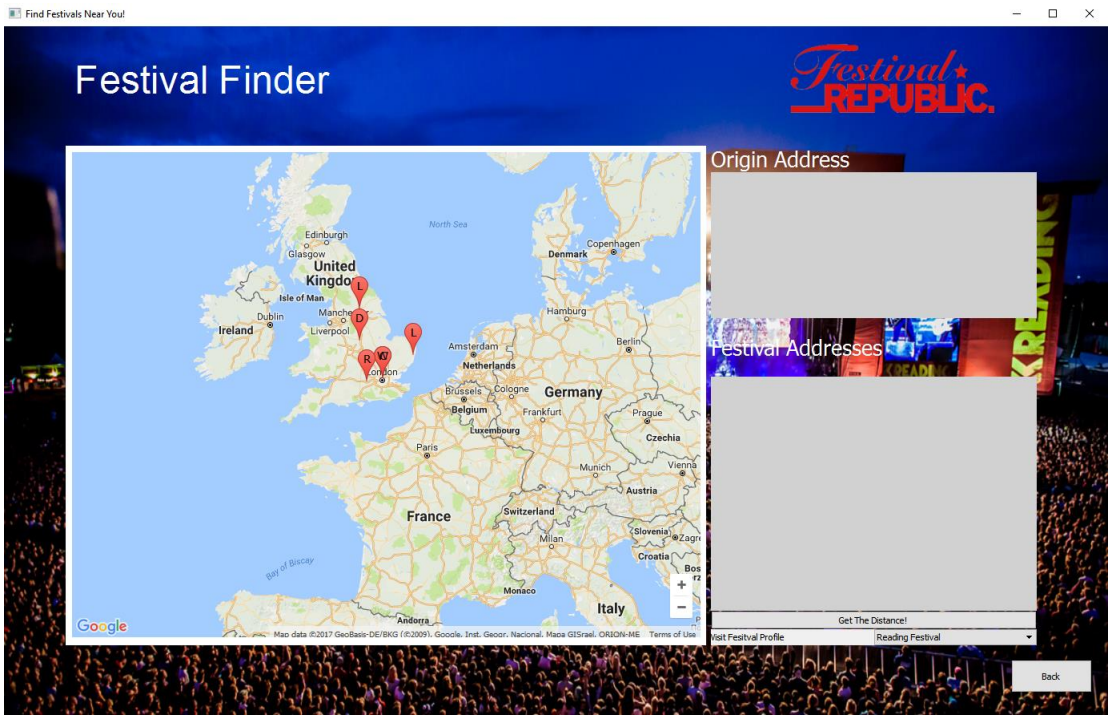
The profile Information correlates with the database and the list of acts:

Community Festival: Two Door Cinema Club (Genre: Alternative)
Community Festival: Catfish and the Bottlemen (Genre: Alternative)
Community Festival: The Wombats (Genre: Pop)
Community Festival: Slaves (Genre: Punk)
Community Festival: Nothing But Thieves (Genre: Rock)
Download Festival: System of a Down (Genre: Metal)
Download Festival: Biffy Clyro (Genre: Rock)
Download Festival: Aerosmith (Genre: Rock)
Latitude Festival: The 1975 (Genre: Indie)
Latitude Festival: Mumford and Sons (Genre: Indie)
Latitude Festival: Baaba Maal (Genre: Rap)
Latitude Festival: Glass Animals (Genre: Alternative)
Latitude Festival: Fleet Foxes (Genre: Indie)
Latitude Festival: John Cale (Genre: Rock)
Latitude Festival: Goldfrapp (Genre: Electronic)
Leeds Festival: Eminem (Genre: Rap)
Leeds Festival: Kasabian (Genre: Alternative)
Leeds Festival: Muse (Genre: Rock)
Leeds Festival: Vant (Genre: Rock)
Reading Festival: Eminem (Genre: Rap)
Reading Festival: Kasabian (Genre: Alternative)
Reading Festival: Muse (Genre: Rock)
Reading Festival: Vant (Genre: Rock)
Wireless Festival: Chance the Rapper (Genre: Rap)
Wireless Festival: Bryson Tiller (Genre: R&B)
Wireless Festival: G-Eazy (Genre: Rap)
Wireless Festival: Fetty Wap (Genre: Rap)
Wireless Festival: Skepta (Genre: Grime)
Wireless Festival: Travis Scott (Genre: Hip-Hop)
Wireless Festival: Rae Sremmurd (Genre: Hip-Hop)
Wireless Festival: The Weeknd (Genre: Pop)
Wireless Festival: Nas (Genre: Rap)
Wireless Festival: Tory Lanez (Genre: Rap)

The profile window therefore does show all the correct output that its designed for, grabbing acts names, dates, costs, addresses, names and ticket vendors for each festival.

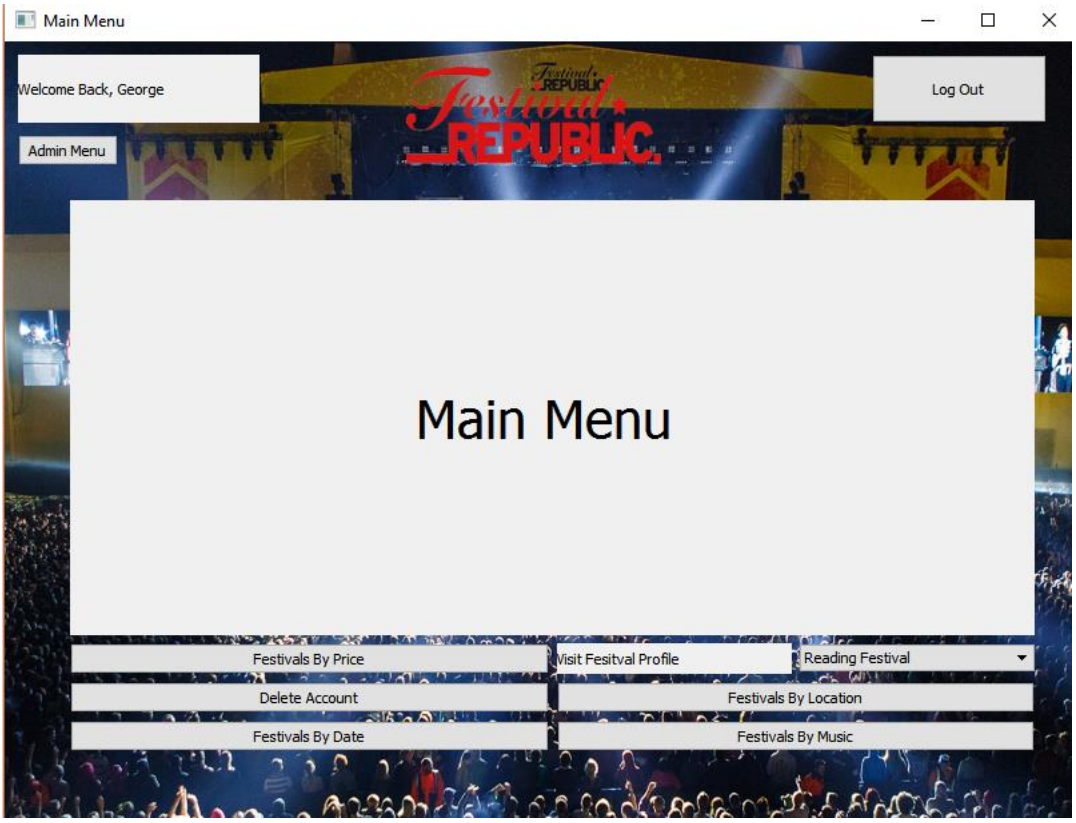
Location & Alert Classes (Criteria 23-29)

Criteria 23 (Buttons Links and Stress Testing)



Criteria 23

The back button works perfectly, connecting it to the main menu:

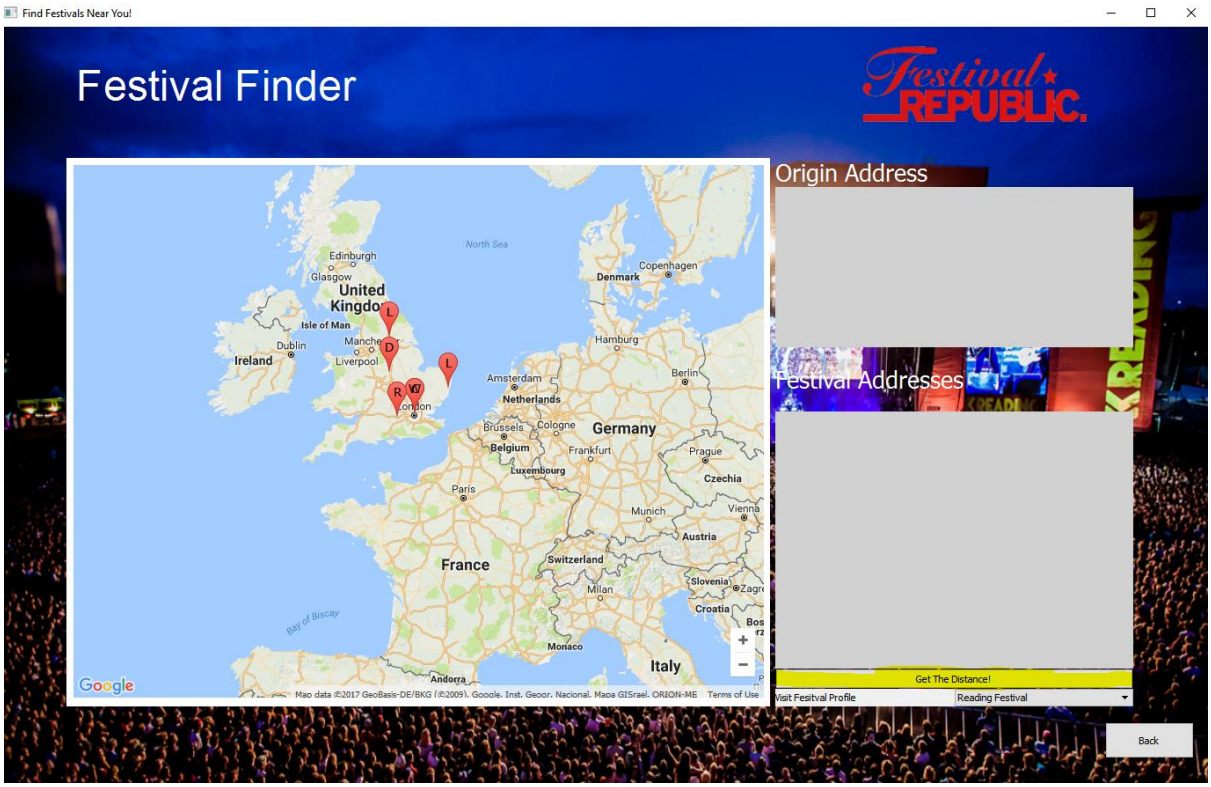


Criteria 23

The GUI cannot be crashed via repeated button pressing – the back and "Get The Distance" buttons work only once before a window pops up.

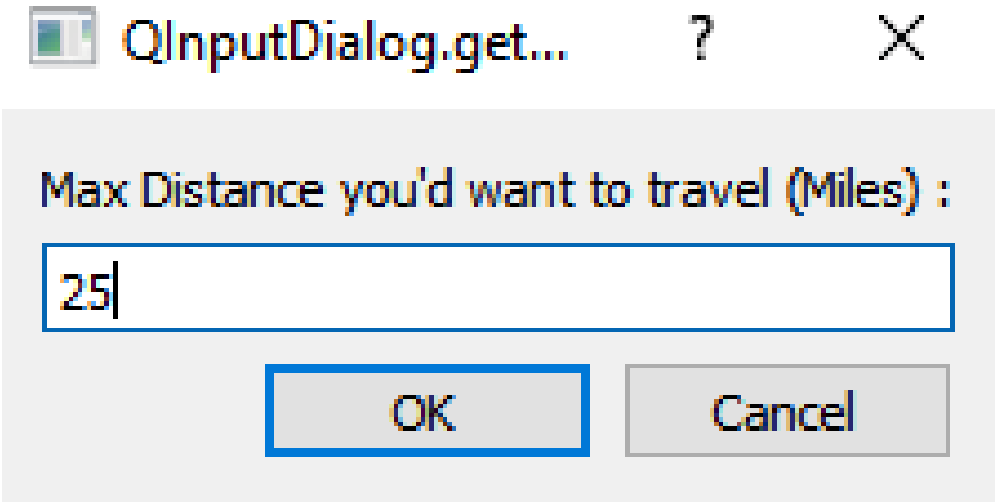
Criteria 24, 25, 26, 27, 28, 29 (Testing the correct plotting of a user's location, Postcode conformity and Postcode passing as well as linking the windows to each other)

When the "Get The distance" button is pressed, the user is asked for input:



Criteria 24

First for the maximum distance they would like to travel (Via the QInputBox)

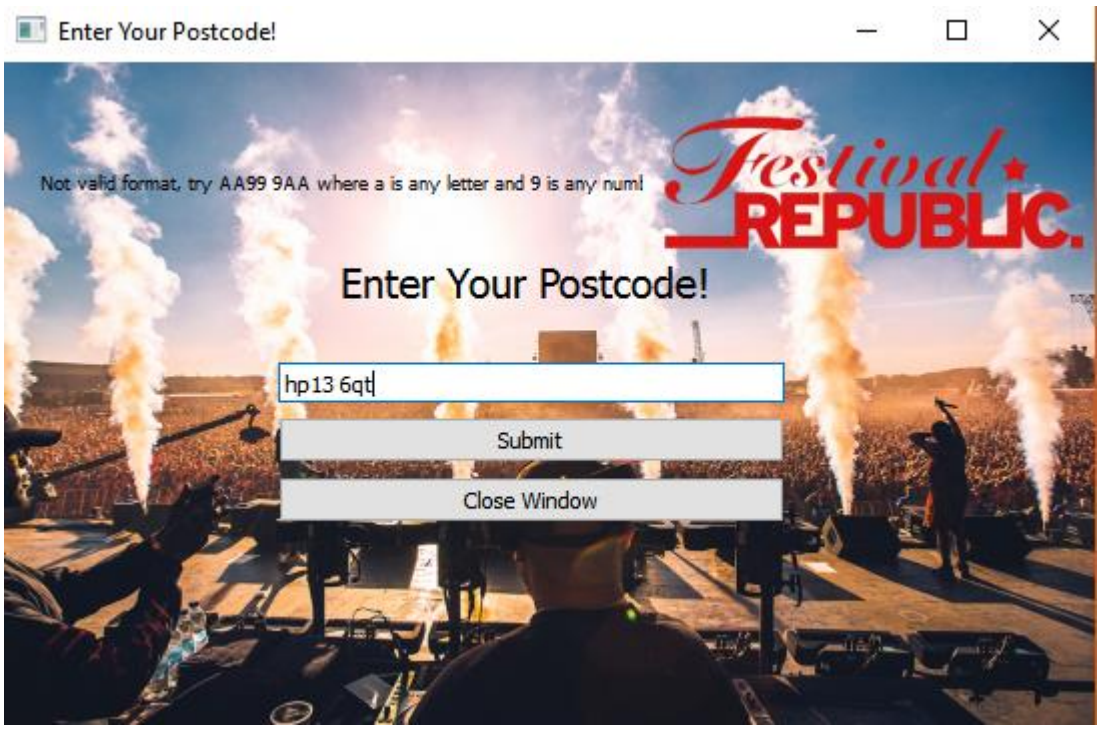


Second by the postcode box (Alert Window):

This window rejects non-conforming postcodes:

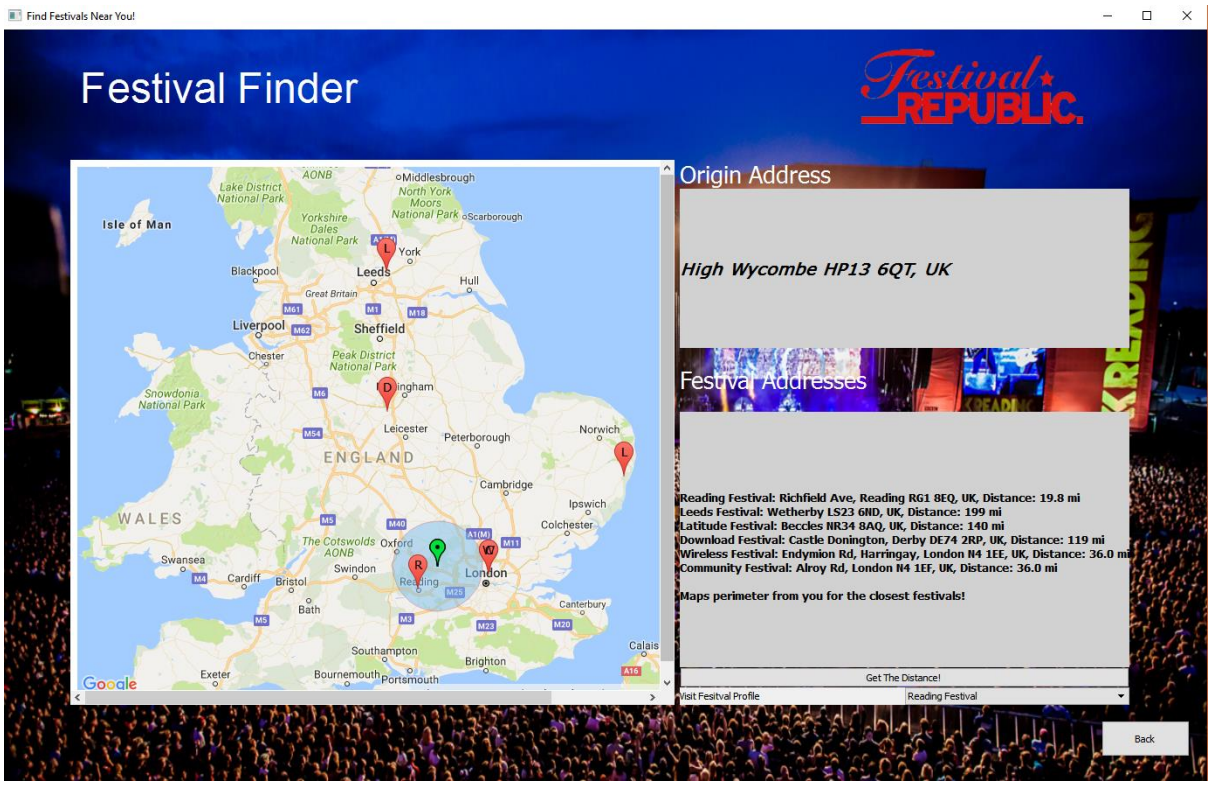
Criteria 25

And accepts normal postcodes:



Criteria 26

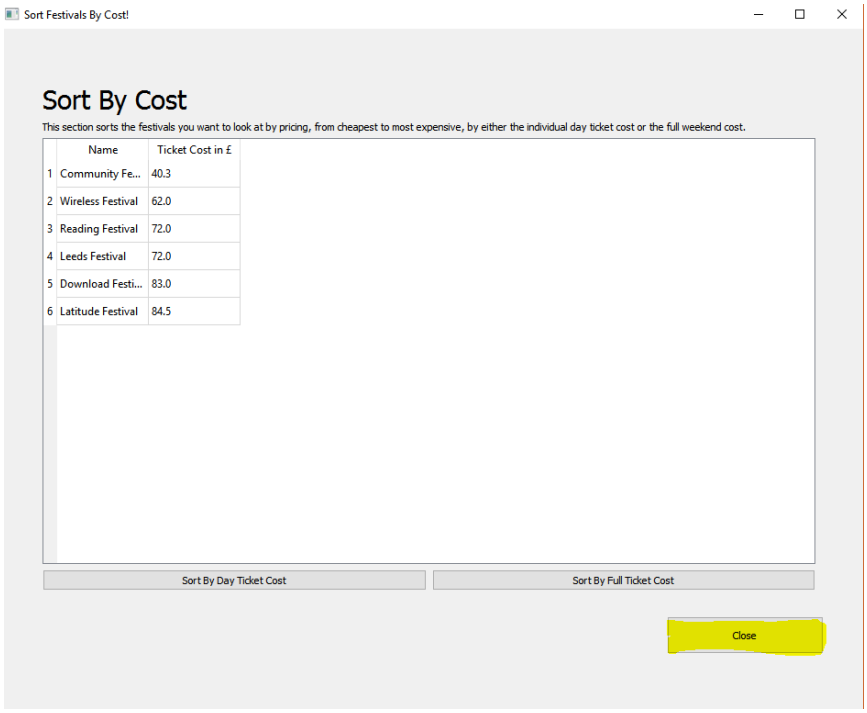
This includes a different coloured home marker, Distances to the festivals and a radius for the maximum user distance:



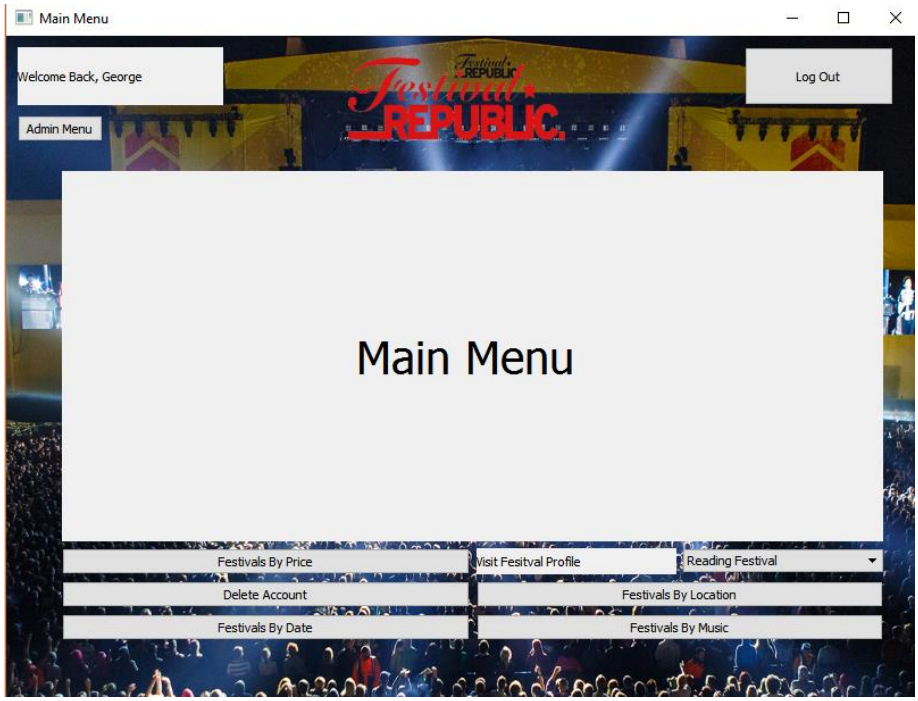
Criteria 27, 28, 29

Cost Class (30 -32)

The Cost class successfully links to the main menu window via the close button:



Criteria 30



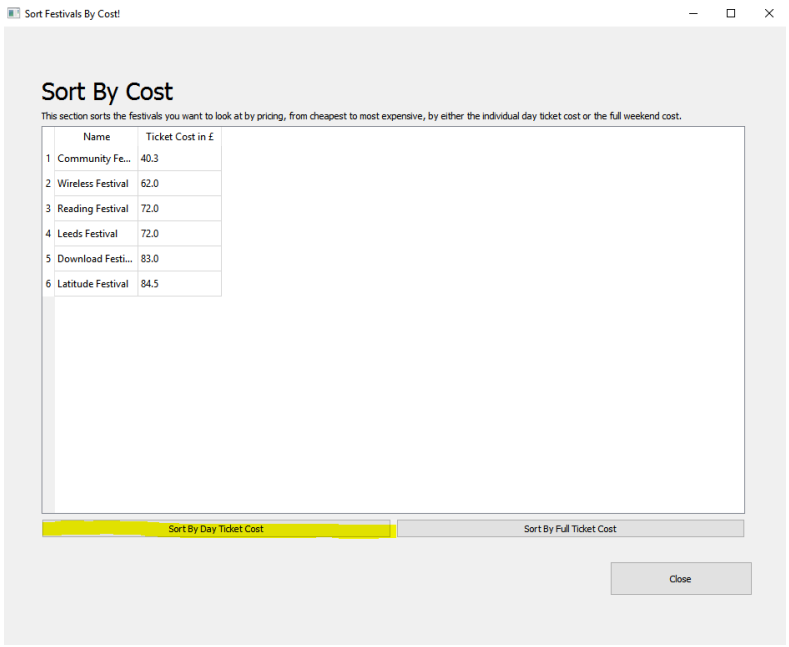
Criteria 30

And the buttons do not create any complex calculations so therefore do not stress/crash the program at all during stress testing

Criteria 31, 32 – Displays correct data from the database:

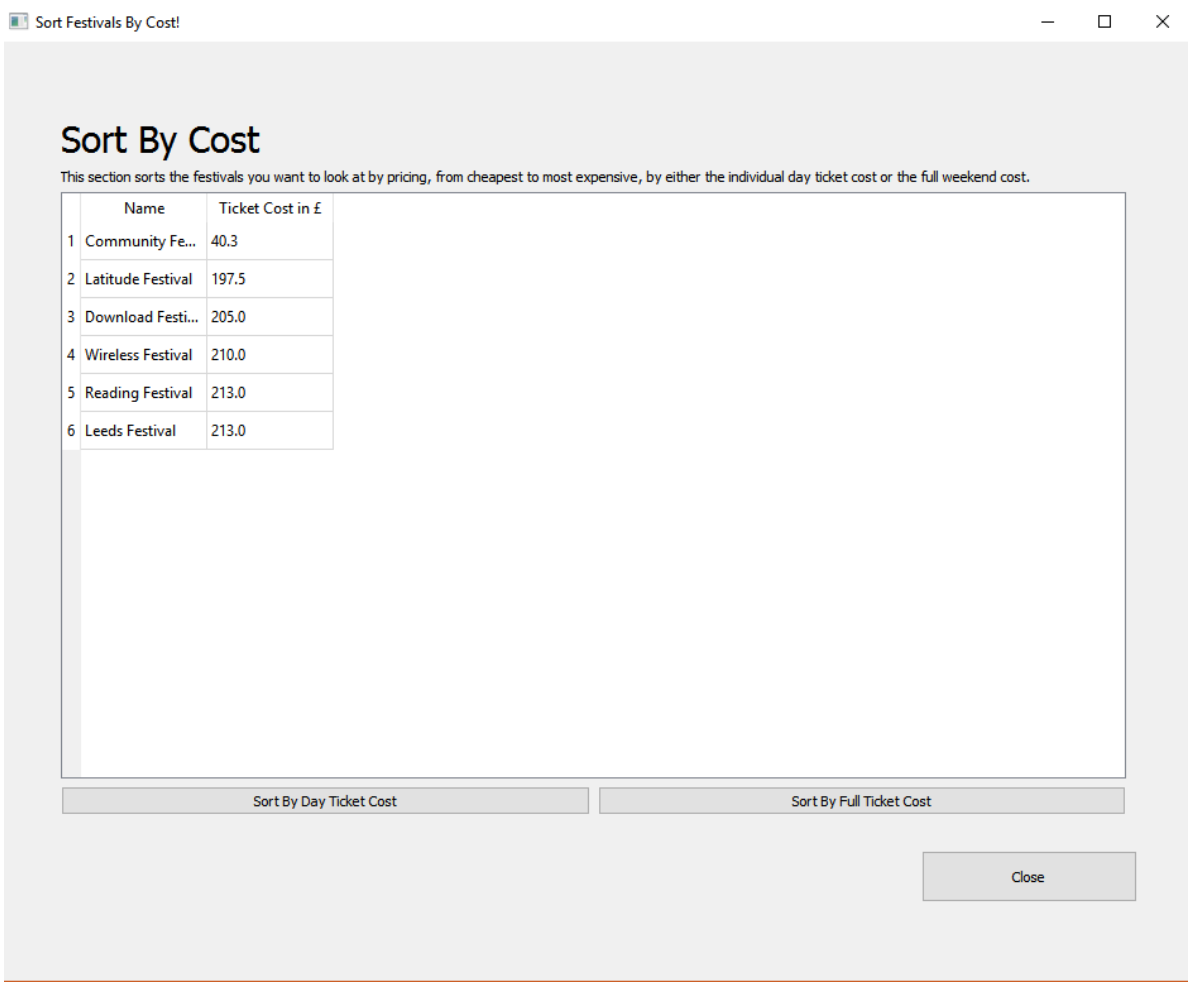
The data sorts correctly for each button:

	festID	Name	Postcode	StartDate	EndDate	Cost	DayCost
1	1	Reading Festival	RG1 8EQ	25/08	27/08	213	72
2	2	Leeds Festival	LS23 6ND	25/08	27/08	213	72
3	3	Latitude Festival	NR34 8AQ	13/07	17/07	197.5	84.5
4	4	Download Festival	DE74 2RP	09/07	11/07	205	83
5	5	Wireless Festival	N4 1EE	08/07	10/07	210	62
6	6	Community Festival	N4 1EF	01/07	01/07	40.3	40.3



Criteria 31

	festID	Name	Postcode	StartDate	EndDate	Cost	DayCost
1	1	Reading Festival	RG1 8EQ	25/08	27/08	213	72
2	2	Leeds Festival	LS23 6ND	25/08	27/08	213	72
3	3	Latitude Festival	NR34 8AQ	13/07	17/07	197.5	84.5
4	4	Download Festival	DE74 2RP	09/07	11/07	205	83
5	5	Wireless Festival	N4 1EE	08/07	10/07	210	62
6	6	Community Festival	N4 1EF	01/07	01/07	40.3	40.3

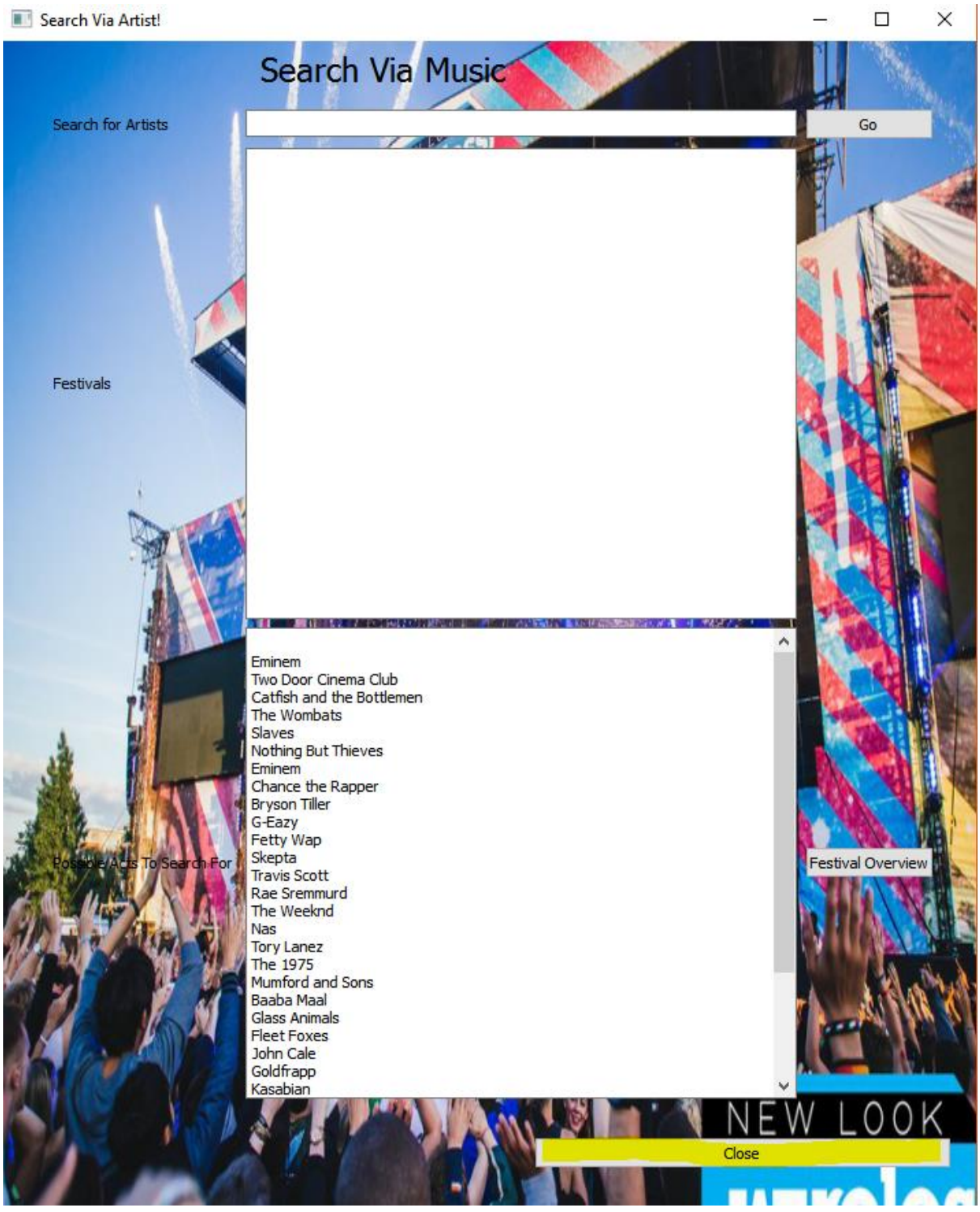


Criteria 32

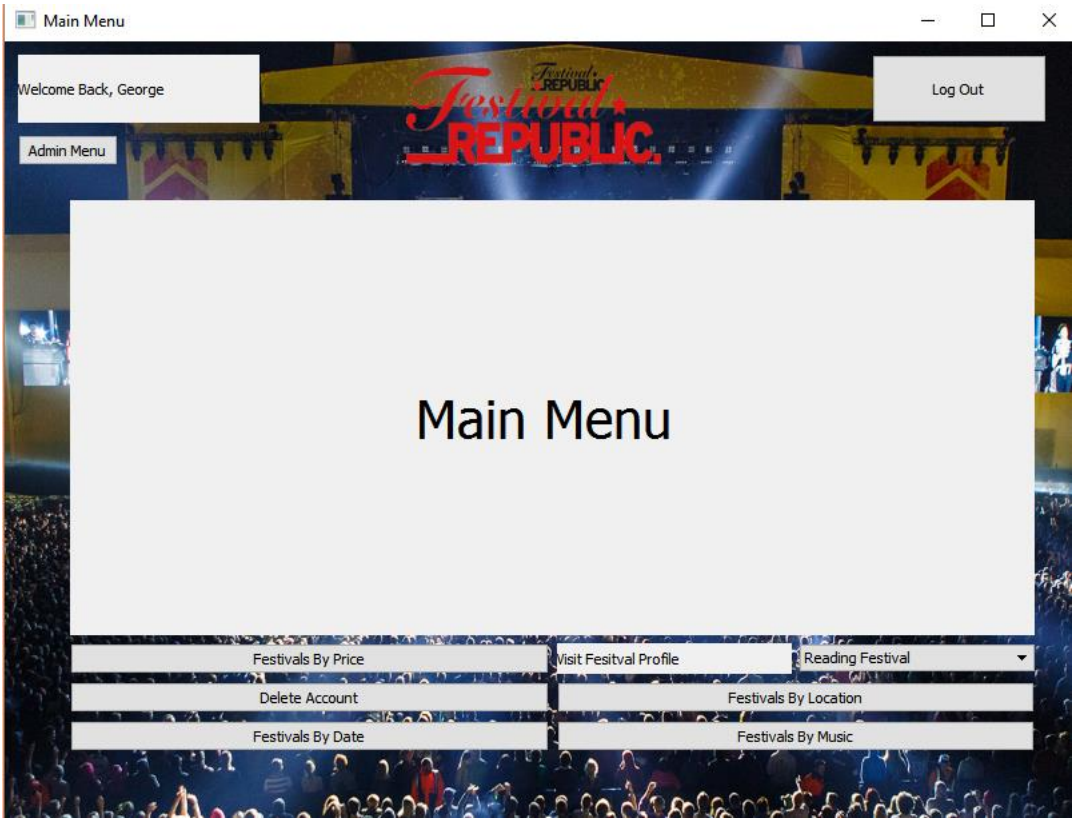
	festID	Name	Postcode	StartDate	EndDate	Cost	DayCost
1	1	Reading Festival	RG1 8EQ	25/08	27/08	213	72
2	2	Leeds Festival	LS23 6ND	25/08	27/08	213	72
3	3	Latitude Festival	NR34 8AQ	13/07	17/07	197.5	84.5
4	4	Download Festival	DE74 2RP	09/07	11/07	205	83
5	5	Wireless Festival	N4 1EE	08/07	10/07	210	62
6	6	Community Festival	N4 1EF	01/07	01/07	40.3	40.3

Music Class (Criteria 33-36)

The Music class' back button work successfully –

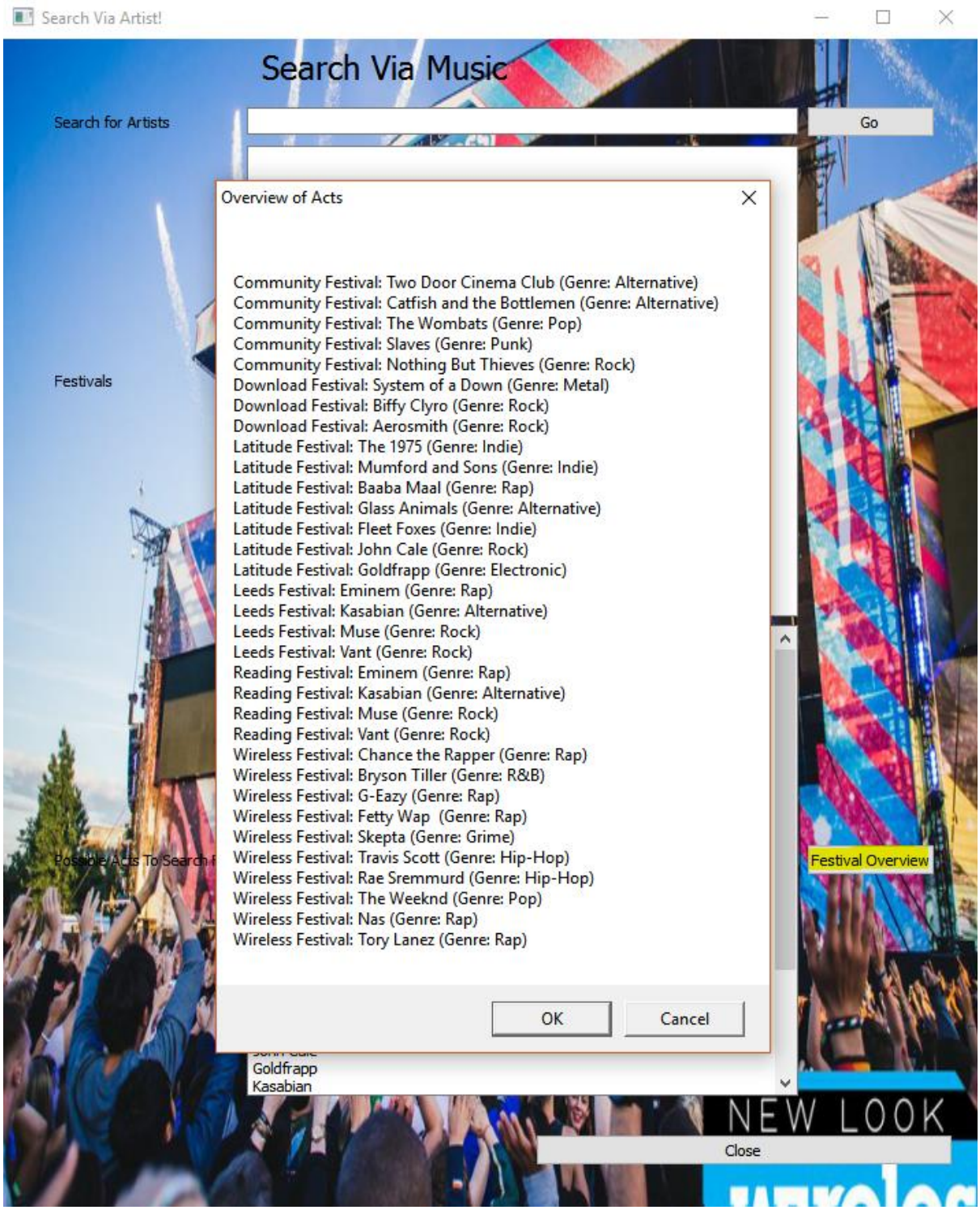


Criteria 33


Main Menu
Welcome Back, George
Log Out
Admin Menu
Main Menu
Festivals By Price
Visit Fesitval Profile
Reading Festival
Delete Account
Festivals By Location
Festivals By Date
Festivals By Music

Criteria 33

As well as its overview button –

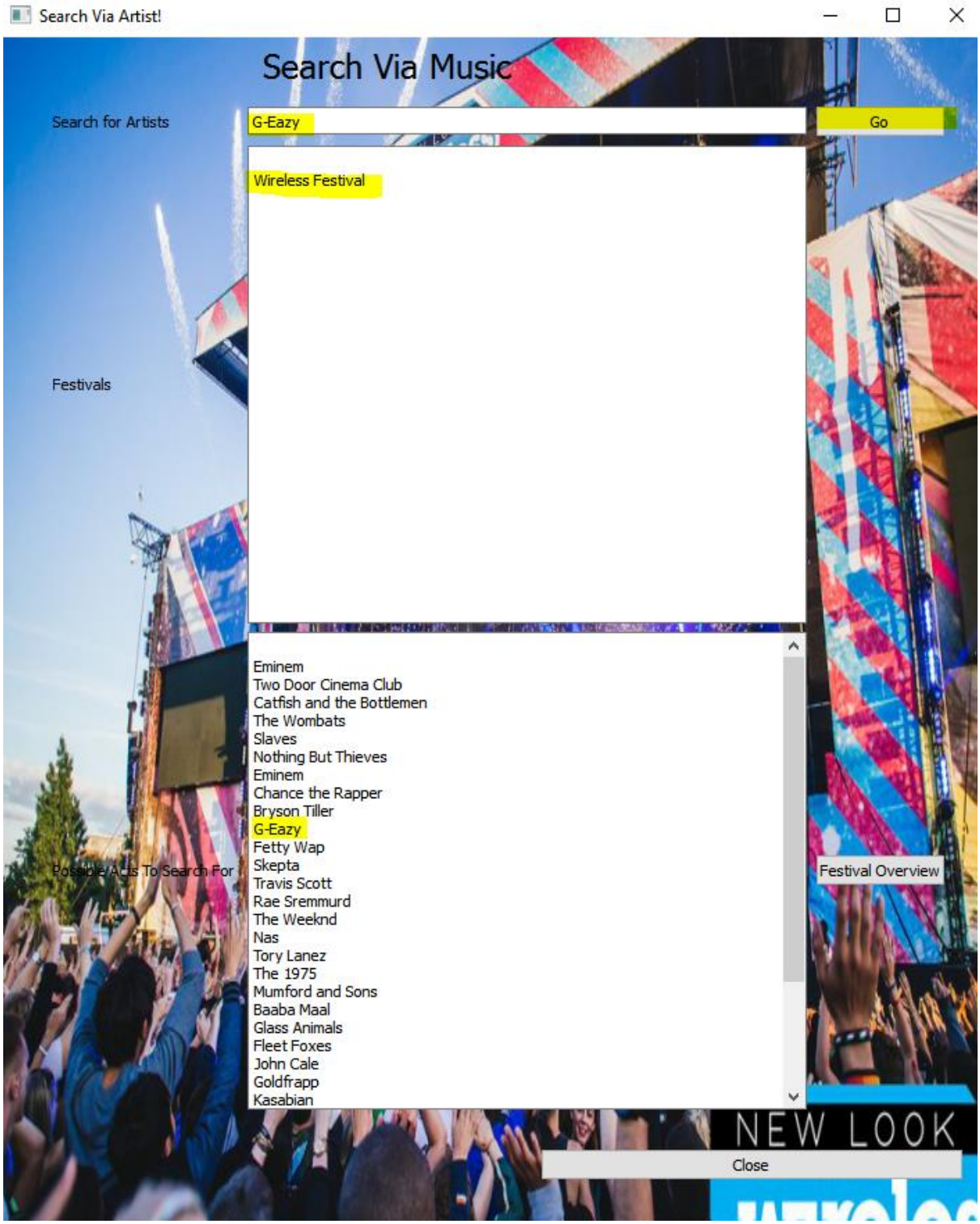


Criteria 34

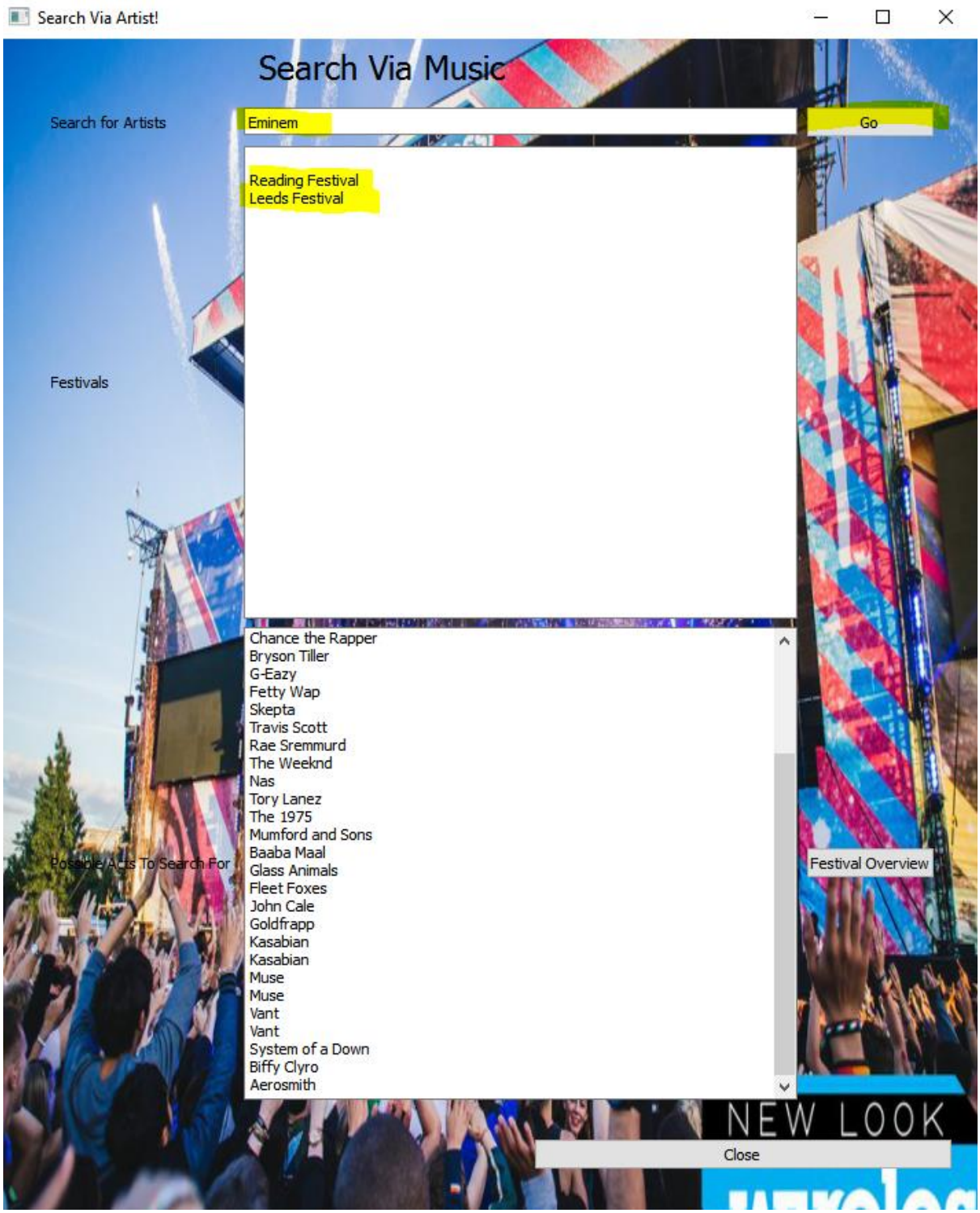
These buttons do hardly any processing, so do not distress the program in any way whilst being pressed repeatedly.

Criteria 34,35,36 (Data Entry, Correct Data and Non-Listed Data)

The program yields the correct result for each artist:

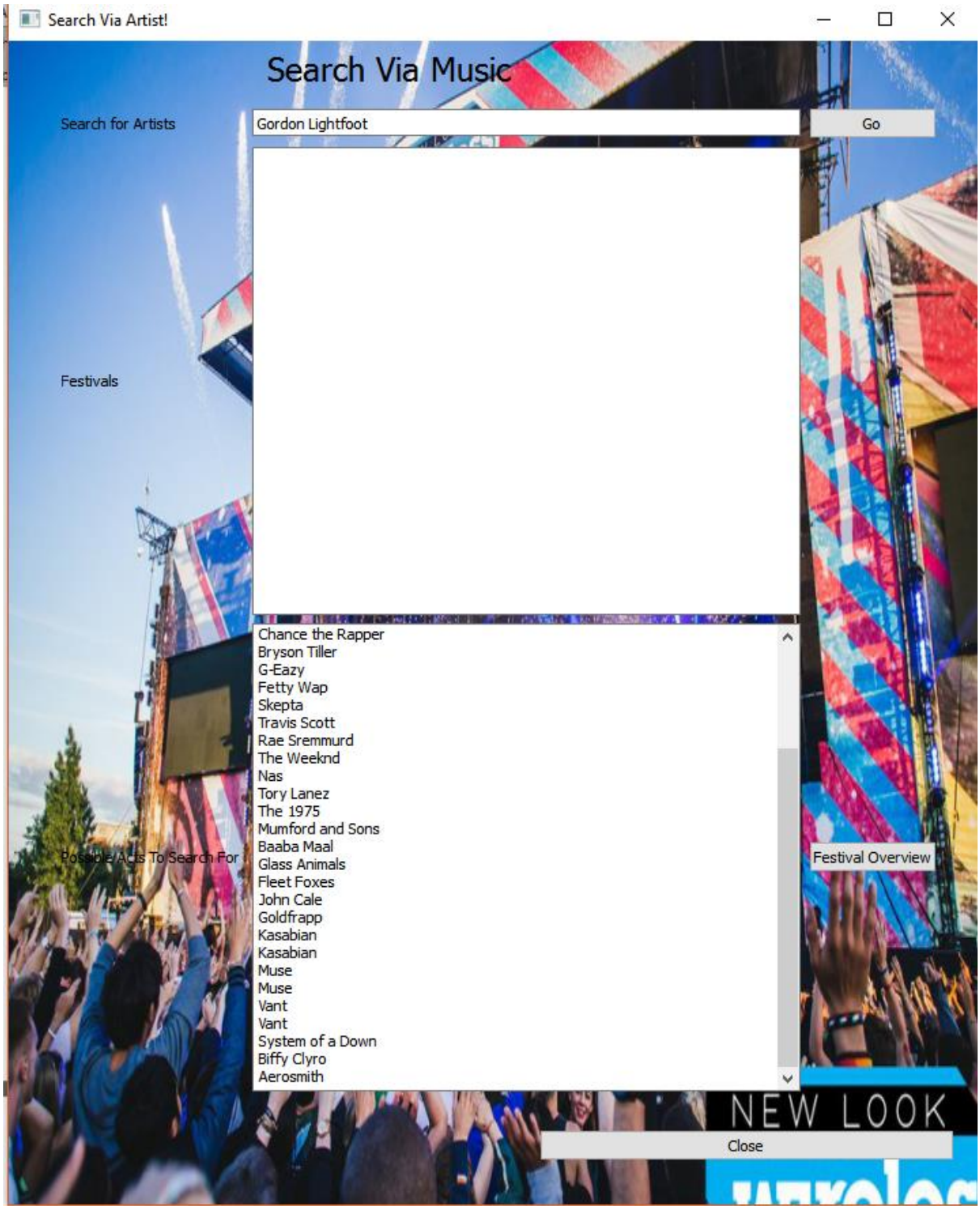


Criteria 34


Search Via Artist!
Search Via Music
Search for Artists
Eminem
Go
Reading Festival
Leeds Festival
Festivals
Chance the Rapper
Bryson Tiller
G-Eazy
Fetty Wap
Skepta
Travis Scott
Rae Sremmurd
The Weeknd
Nas
Tory Lanez
The 1975
Mumford and Sons
Baaba Maal
Glass Animals
Fleet Foxes
John Cale
Goldfrapp
Kasabian
Kasabian
Muse
Muse
Vant
Vant
System of a Down
Biffy Clyro
Aerosmith
Possible Acts To Search For
Festival Overview
NEW LOOK
Close

Criteria 35

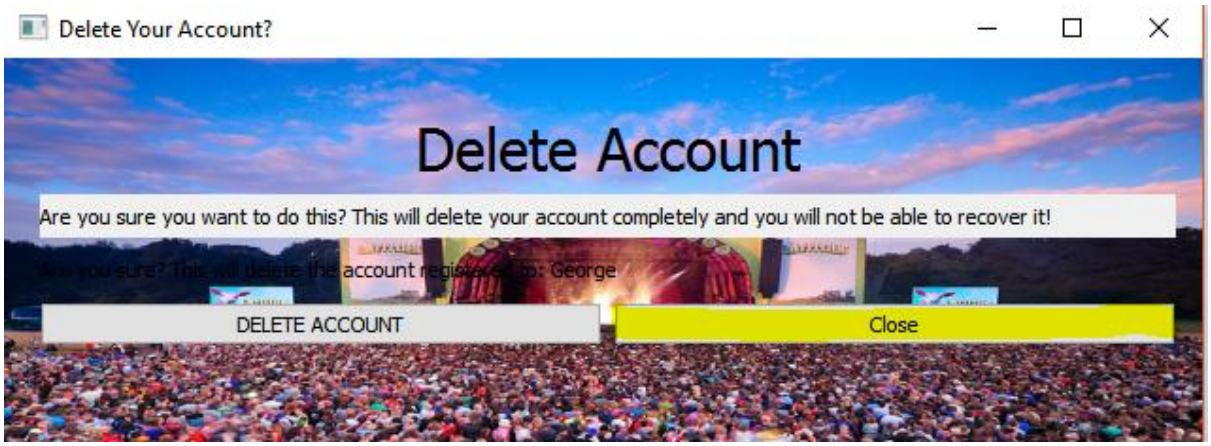
And does not display output for non-existent entries:



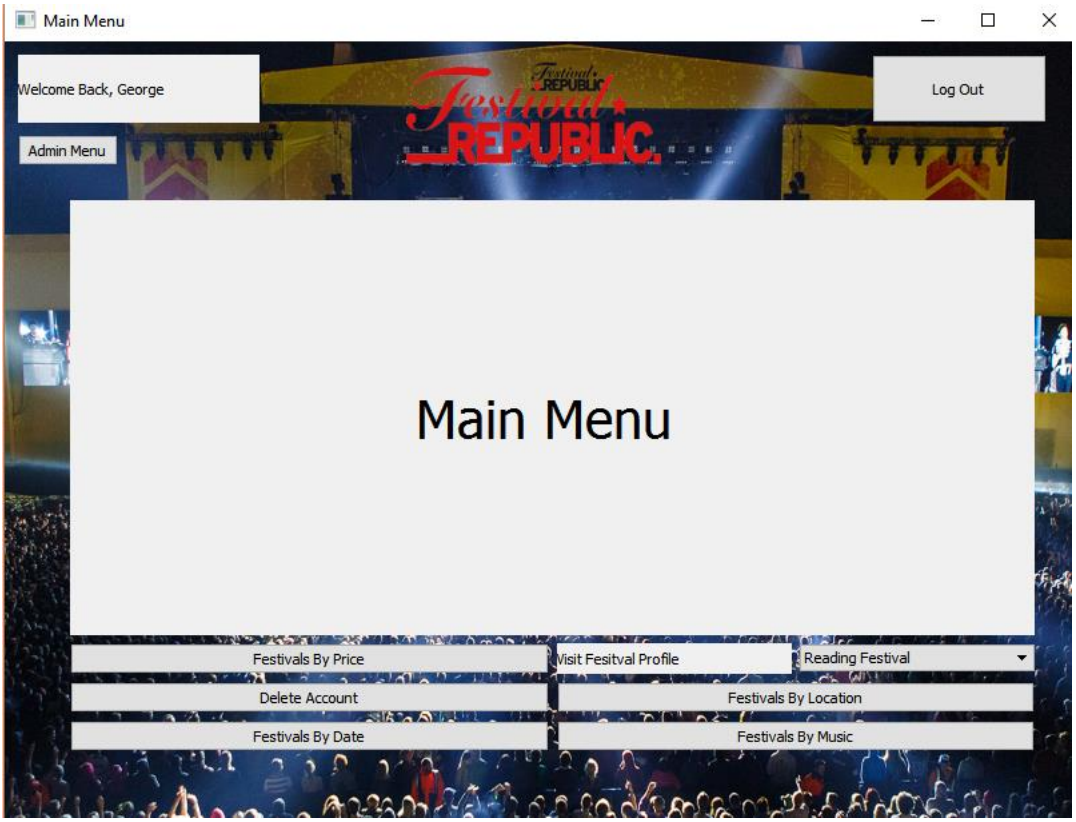
Criteria 36

Delete Class (Criteria 37-38)

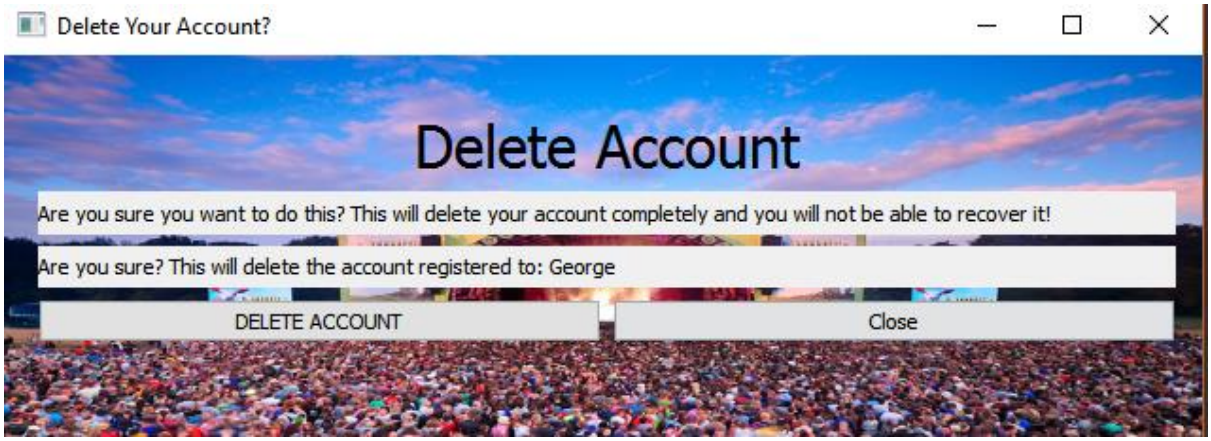
The Delete Class' buttons work perfectly:



Criteria 37



Criteria 37



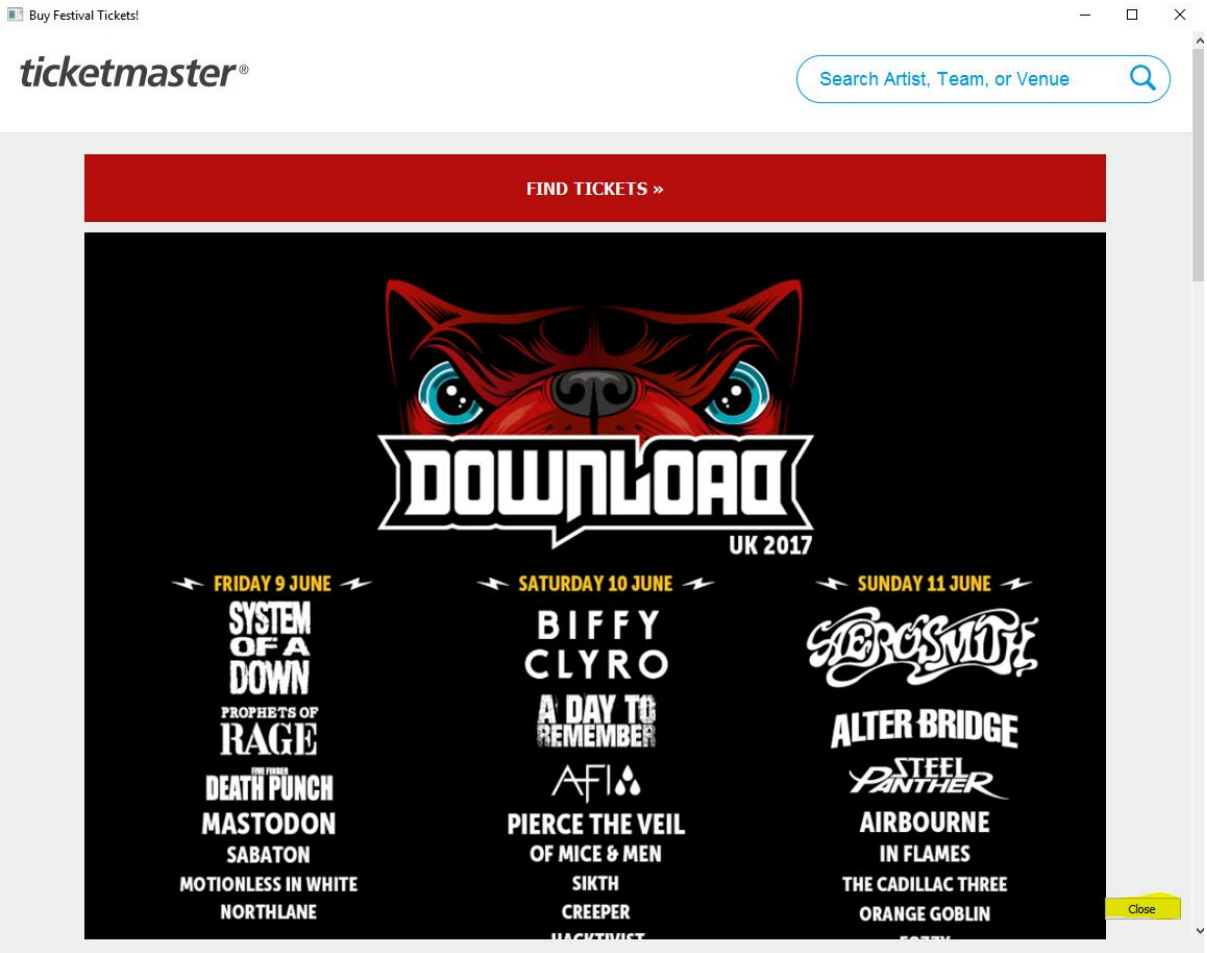
Criteria 38

With the delete button itself successfully removing the user -

	userID	user	pass	email	perm_lvl
1	1	a	a4f647be6ad4ac0ae959985ccb362e71	majesticseabass@gmail.com	2
2	2	admin	6f2457b608622214f753c8522bf63a38	cyberhylian@gmail.com	1
3	5	complex	0d91a55ac531df50ede54f22b23ed9ec	philnsjspam@talktalk.net	2
4	8	jdsalinger	e7d8993bf6de68a2d4e3a8a24e51f4fd	ltmattwelly@gmail.com	1
5	10	jessica	db1a0dee243af63d8db3703ab7d80e53	jess@4818.com	1
6	12	richard	d0bc30e069ddd34121be0c4a8b00231b	richardj99@sky.com	2
7	13	Evaluate	e8a63b225a5c2730367036816fd5c44d	evaluation@coursework.com	1

Buy Class (Criteria 39-40)

The Buy class' buttons work perfectly:



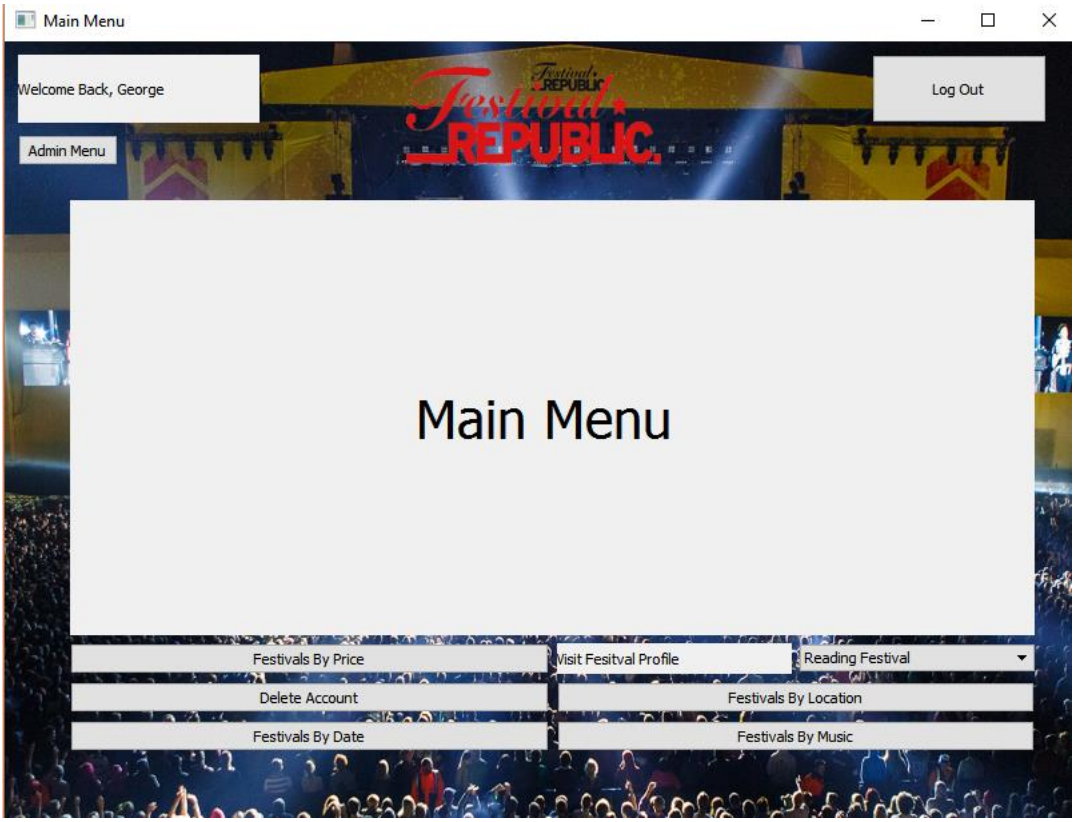
Criteria 39

Criteria 40

And each profile outputs the correct Buy Screen, as shown in Criteria 20-22.

Menu Class (Criteria 41-43)

The Menu Buttons work perfectly, each linking to a separate window:

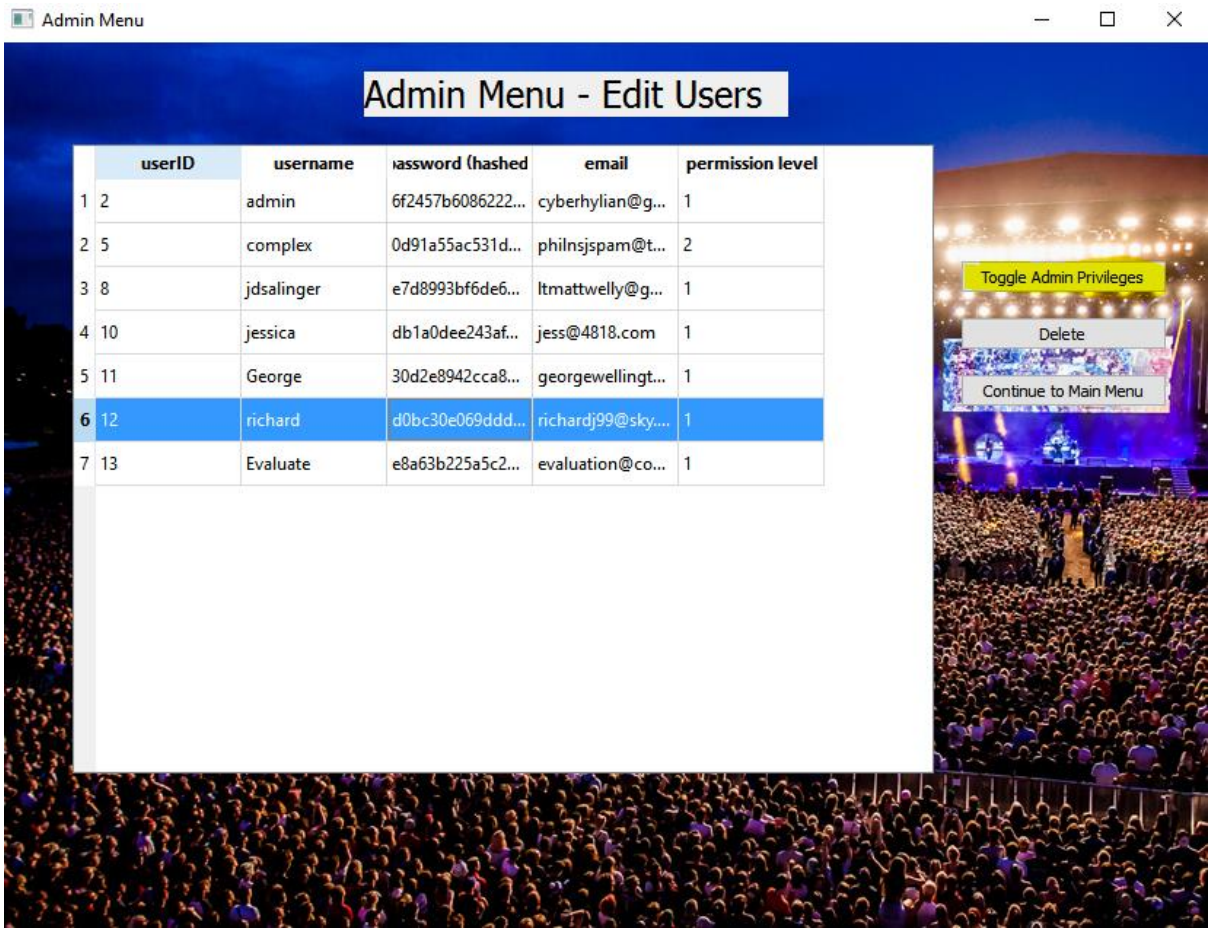


Criteria 41

Logout Button:

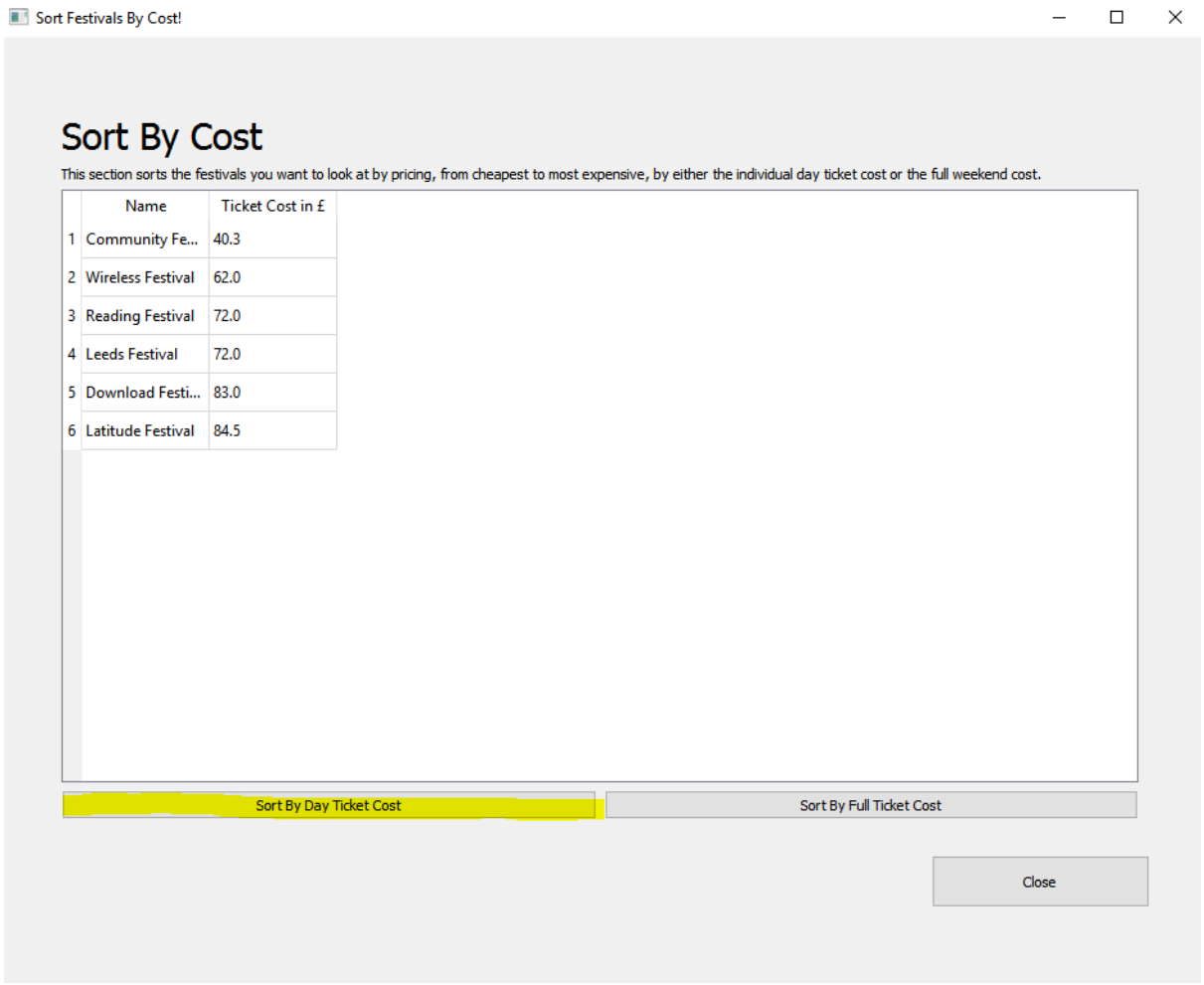
Criteria 41

Admin Menu Button (For admins only):



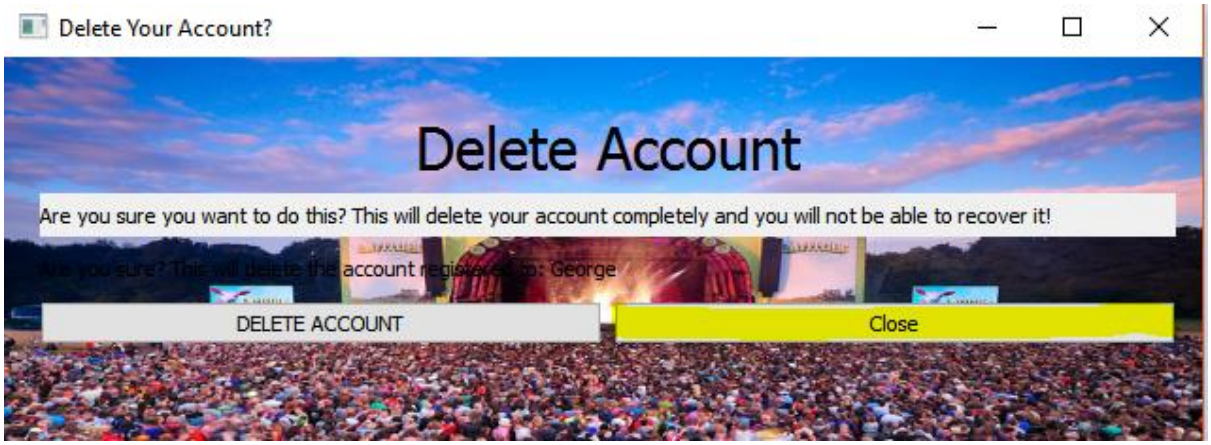
Criteria 41

Price Button:



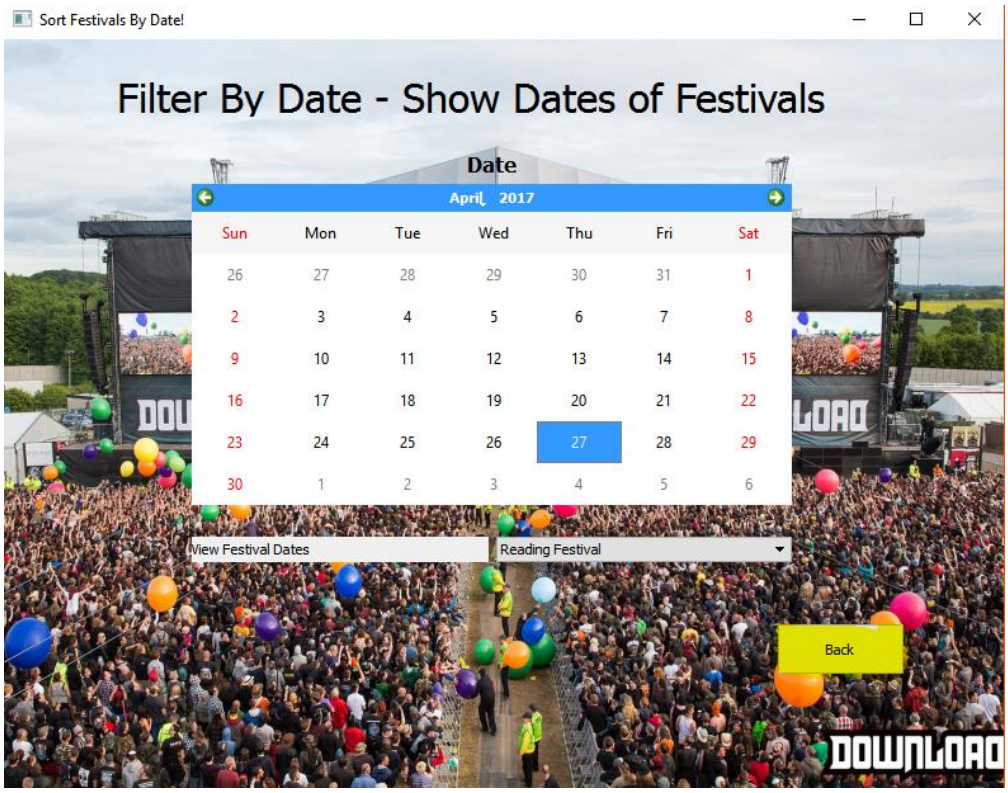
Criteria 41

Delete Button:



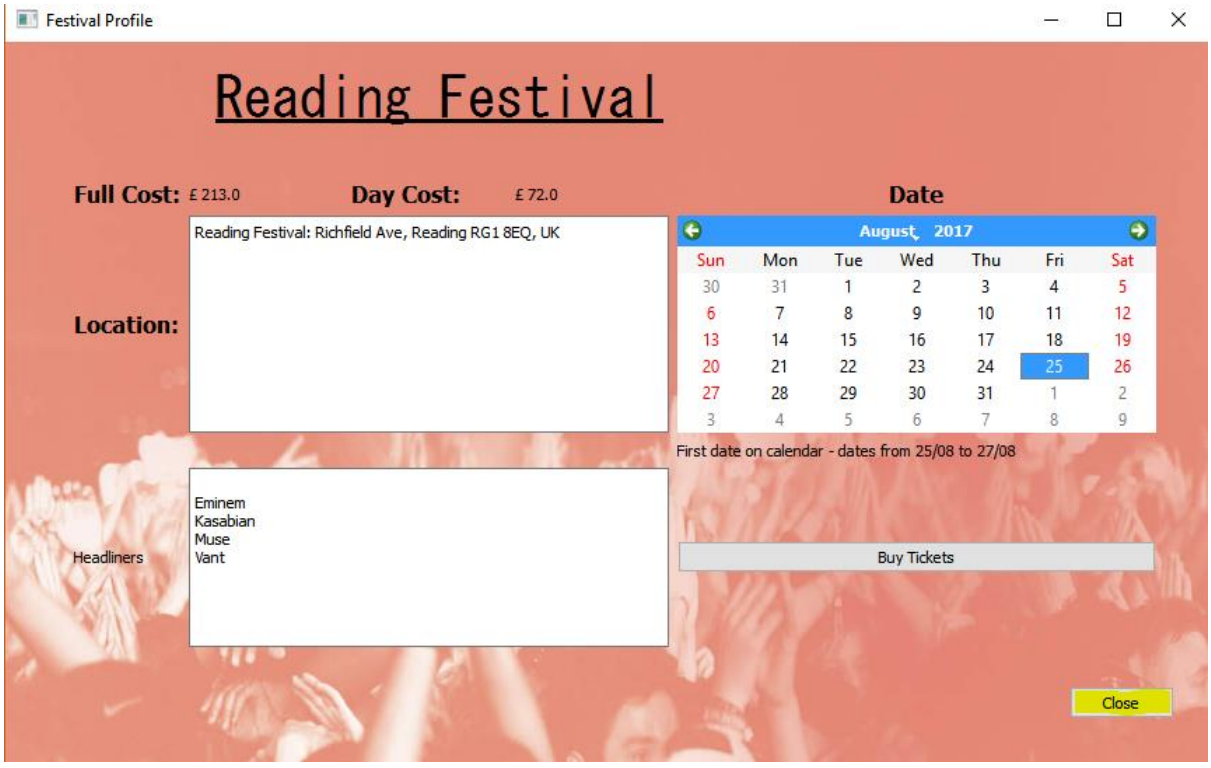
Criteria 41

Date Button:



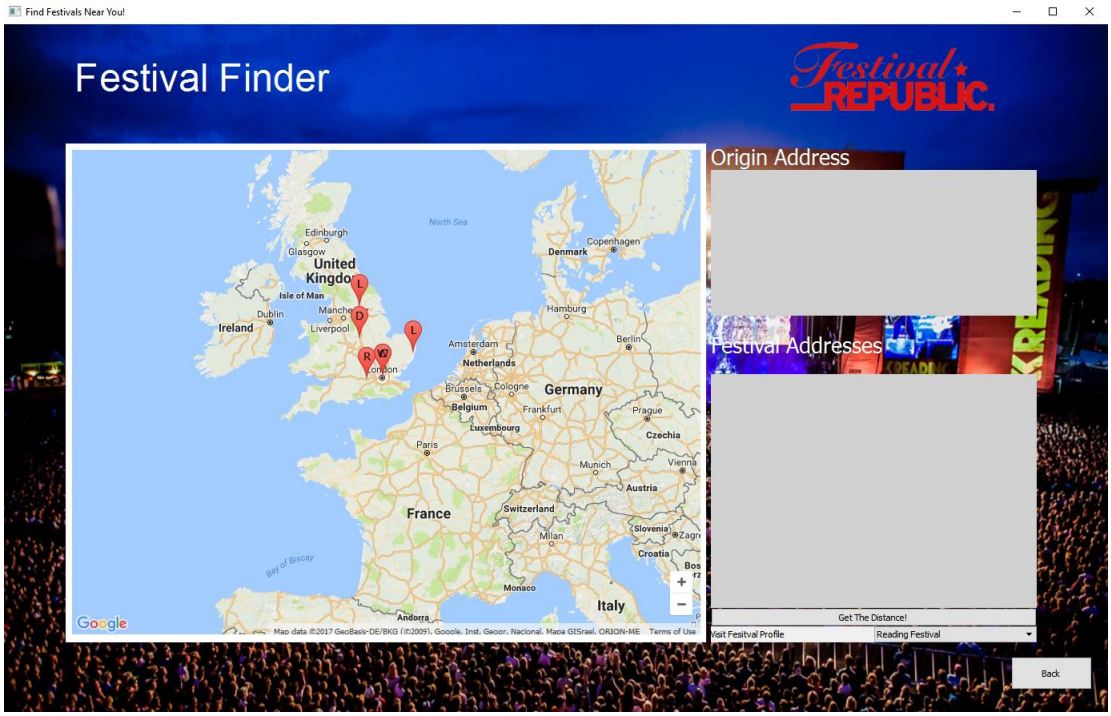
Criteria 41

Profile Button –



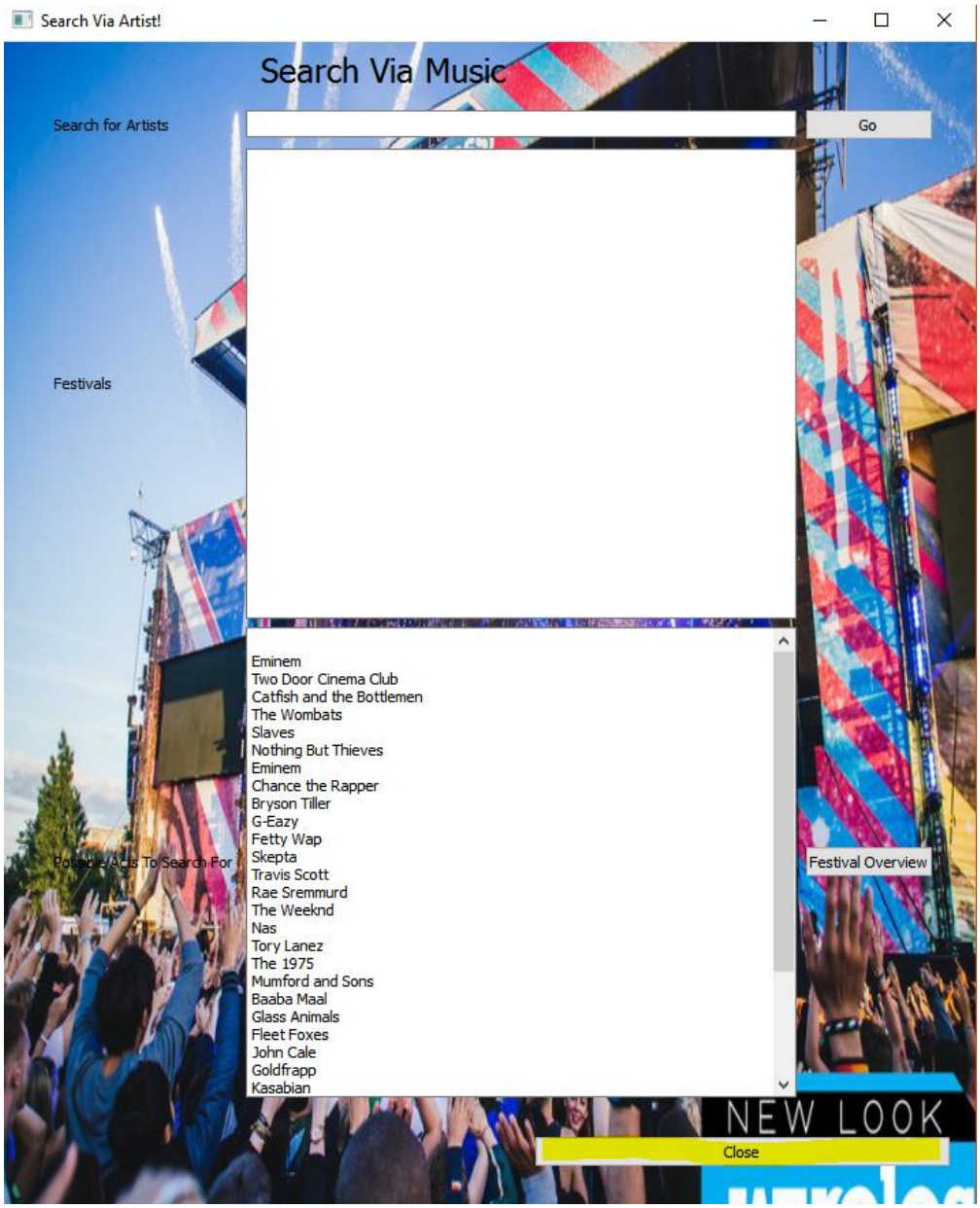
Criteria 41

Location Button -



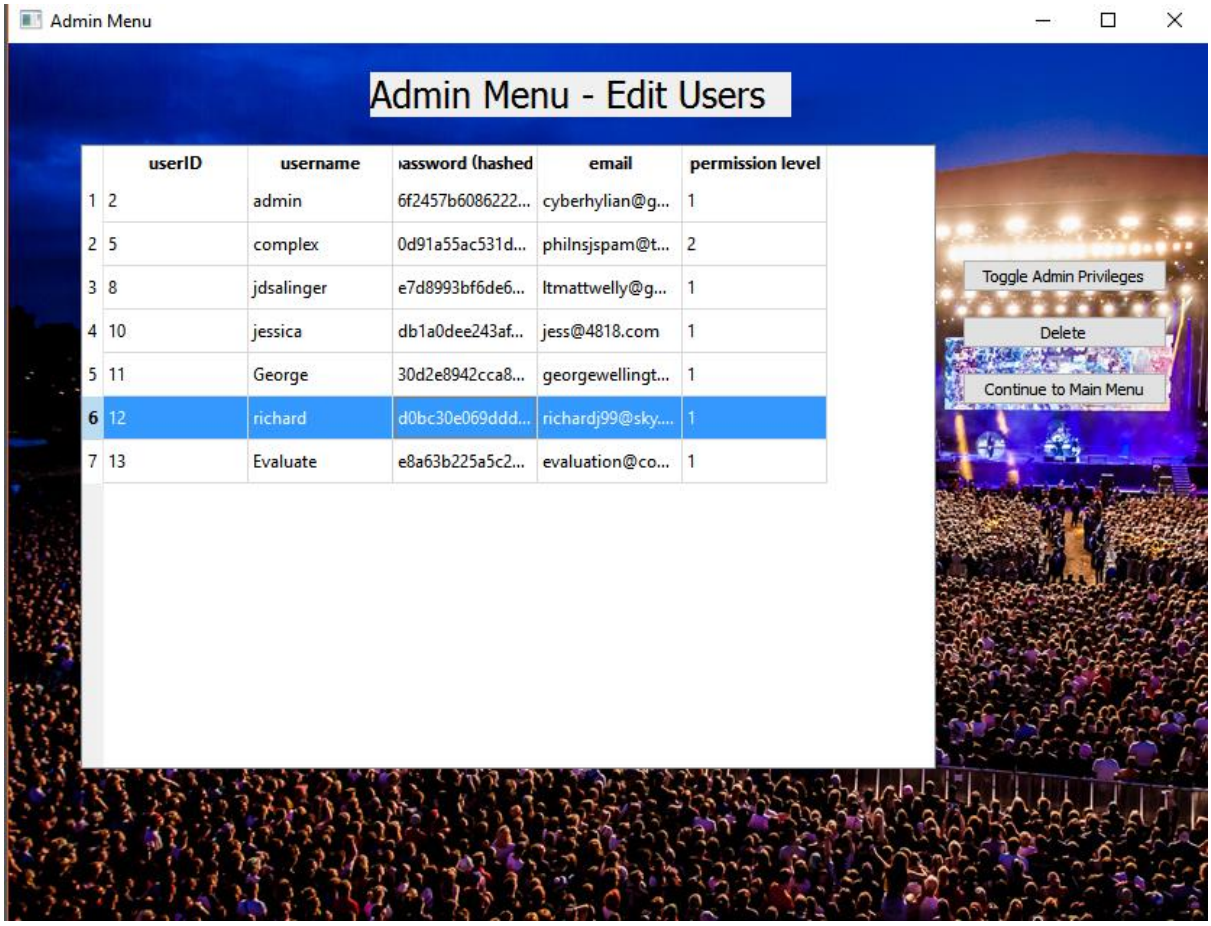
Criteria 41

Music Button –

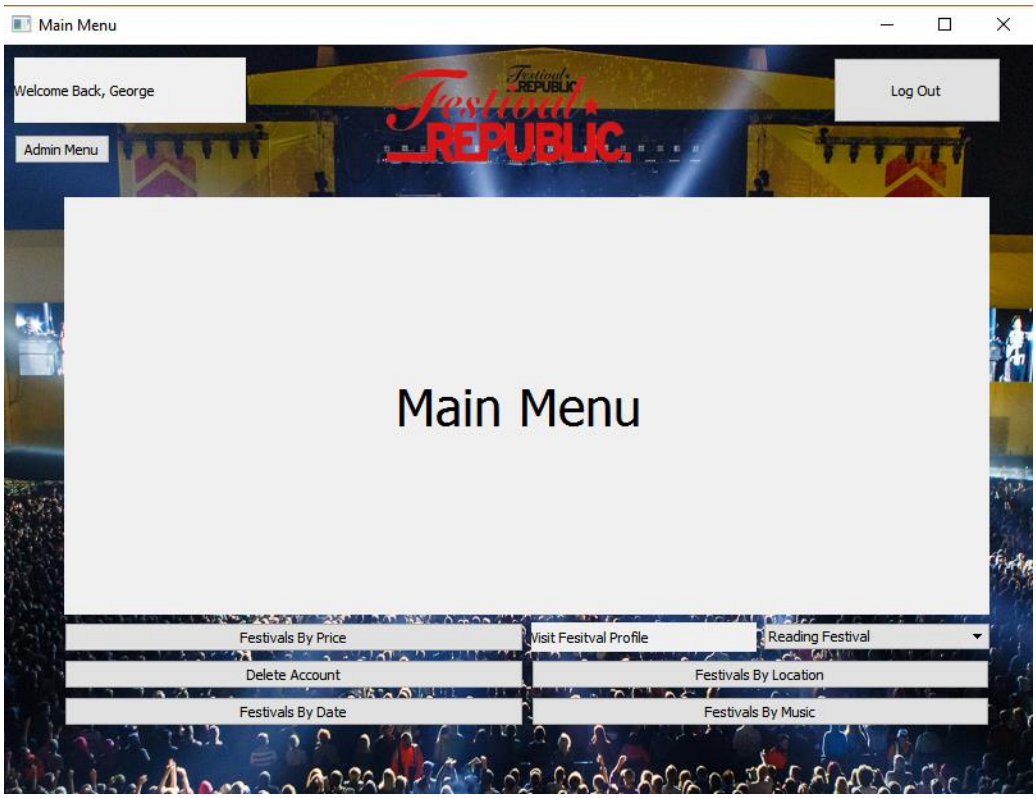


Criteria 41

And only administrator level users can use the admins menu:



Criteria 43

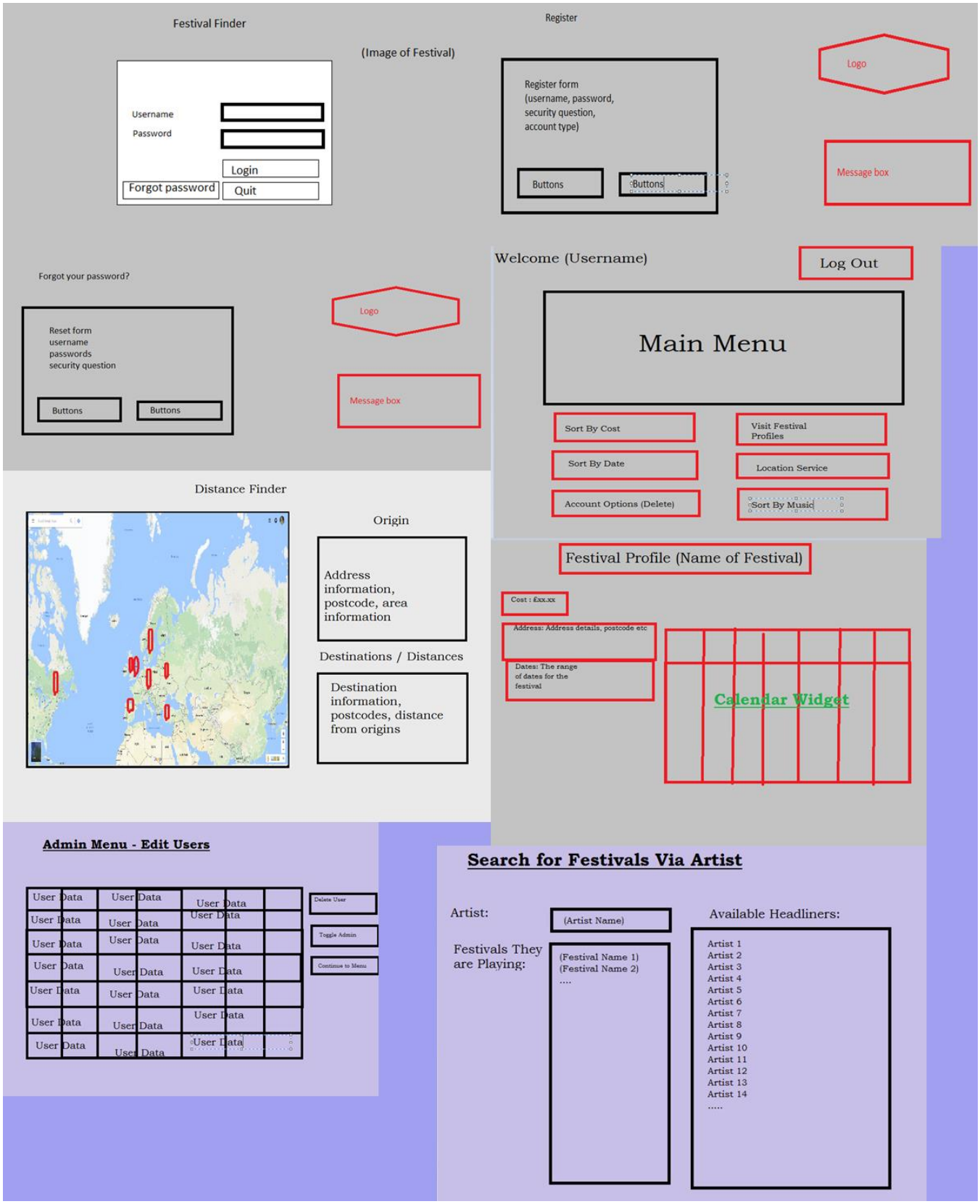


Criteria 43

User Feedback

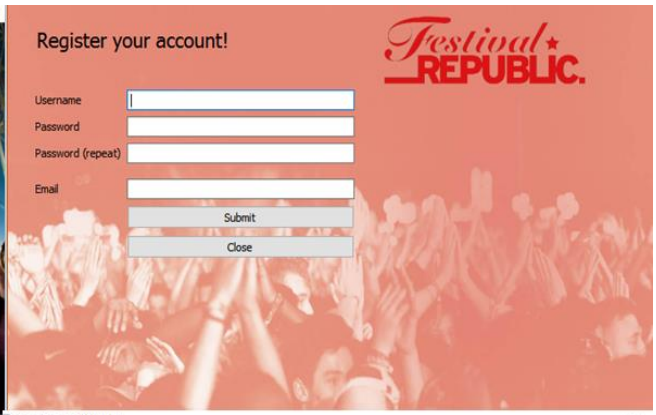
The GUI, Then and Now

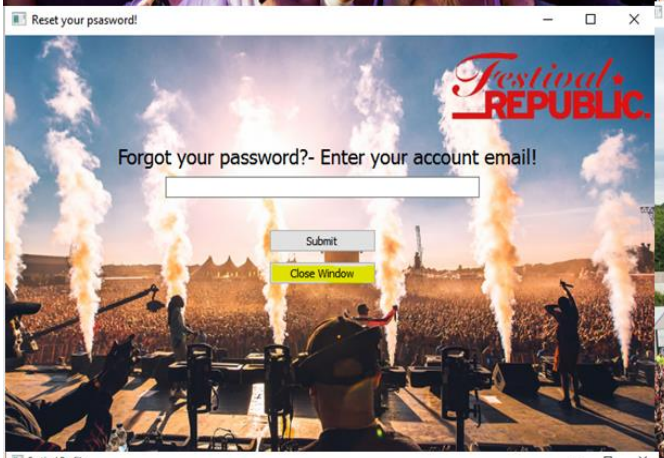
Before

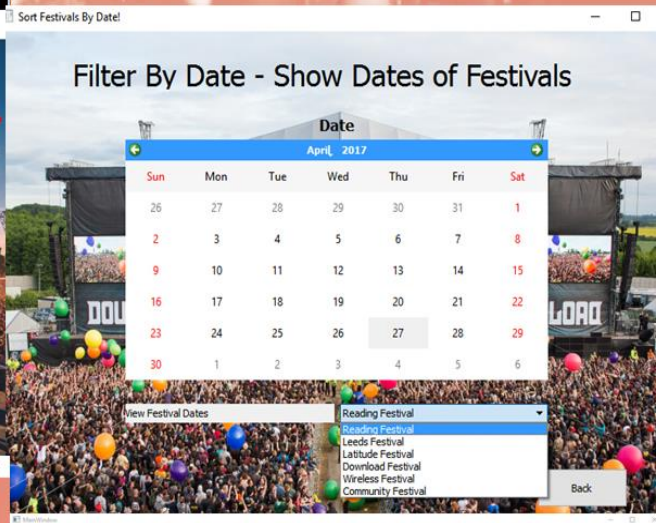


After

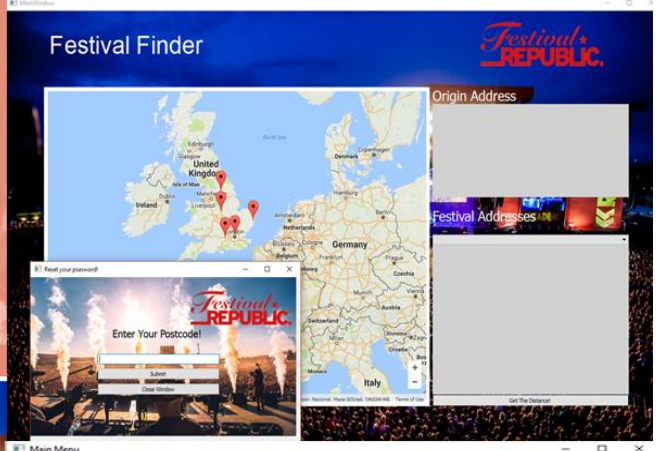

Festival Finder
Username
Password
Close
Log In
Register
Forgot Password?

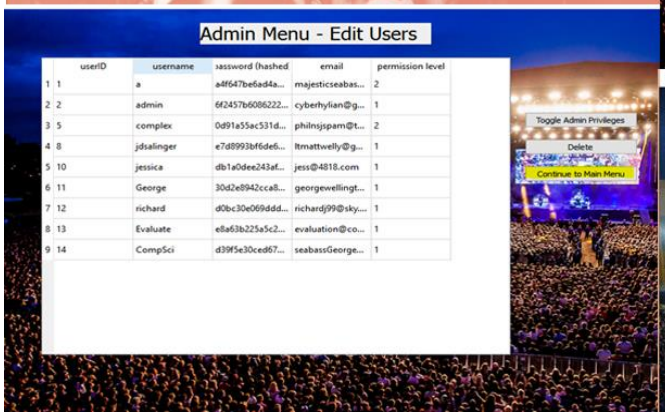

Register your account!
Festival REPUBLIC.
Username
Password
Password (repeat)
Email
Submit
Close


Reset your psasword!
Festival REPUBLIC.
Forgot your password?- Enter your account email!
Submit
Close Window

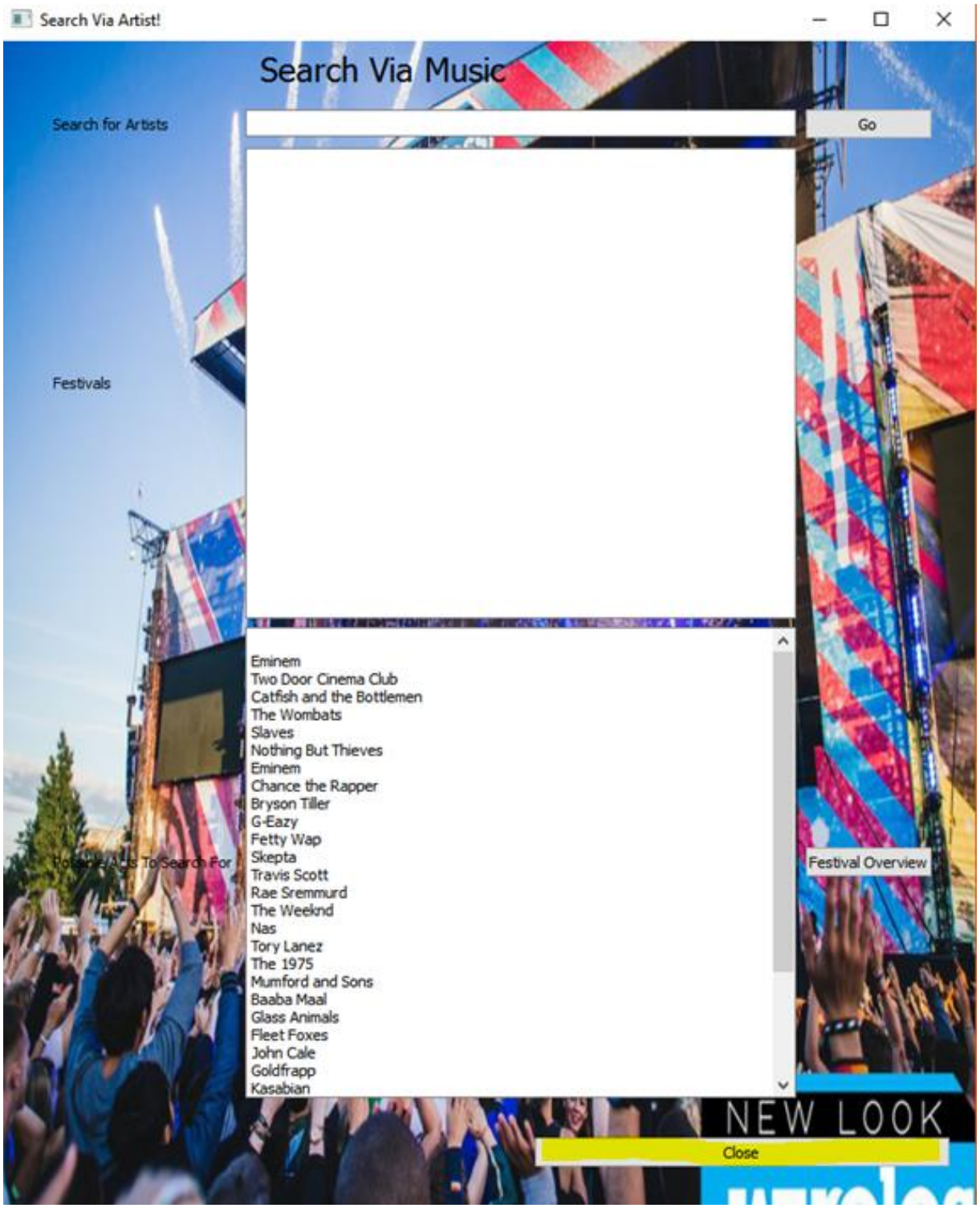

Sort Festivals By Date!
Filter By Date - Show Dates of Festivals
Date
April 2017
Sun Mon Tue Wed Thu Fri Sat
View Festival Dates
Reading Festival
Reading Festival
Leeds Festival
Latitude Festival
Download Festival
Wireless Festival
Community Festival
Back


Festival Profile
Reading Festival
Full Cost: £213.0
Day Cost: £72.0
Date
Location:
Reading Festival: Richfield Ave, Reading RG1 8EQ, UK
August 2017
First date on calendar - dates from 25/08 to 27/08
Headliners
Eminem
Kasabian
Muse
Vant
Buy Tickets
Close


Festival Finder
Festival REPUBLIC.
Origin Address
Festival Addresses
Enter Your Postcode!
Submit
Close Window

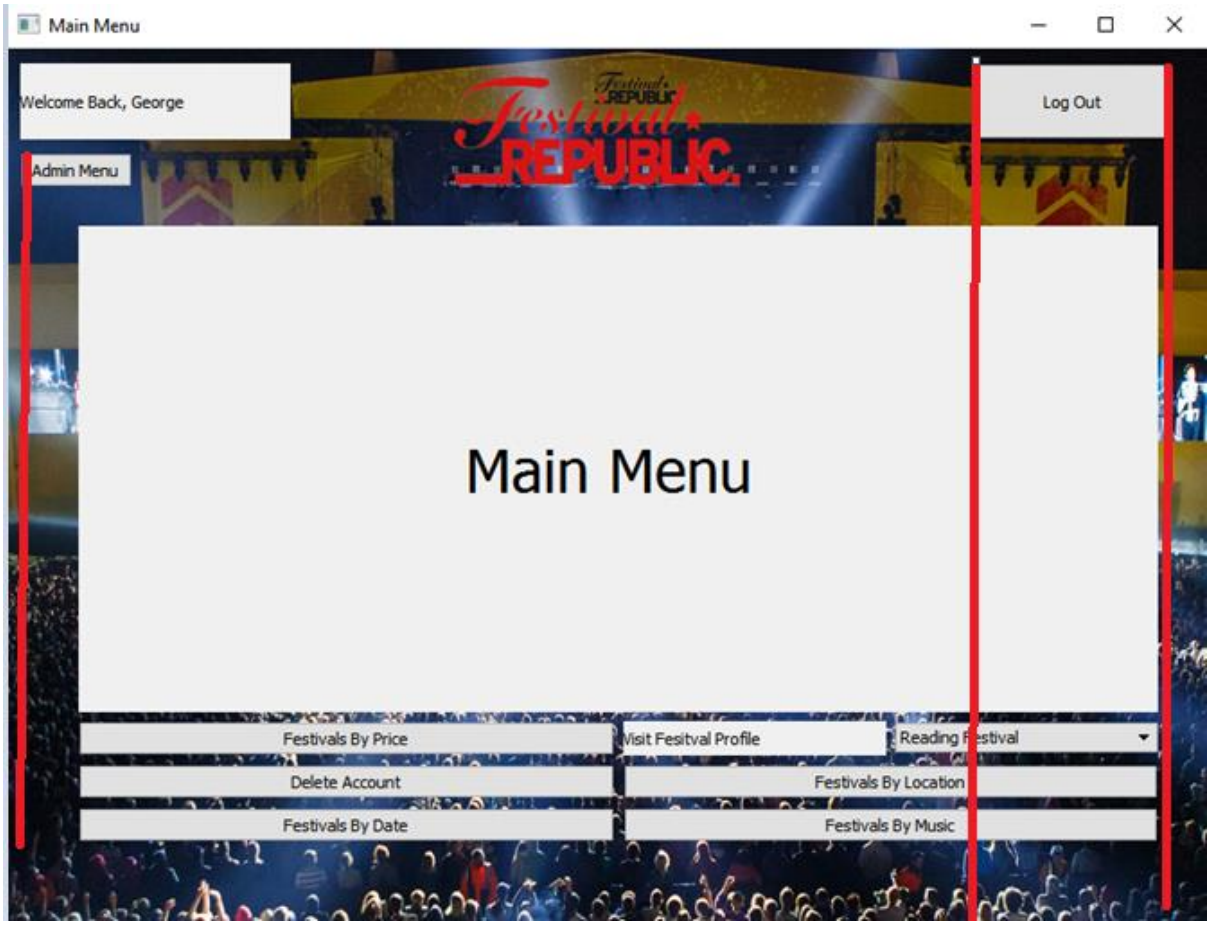

Admin Menu - Edit Users
userID
username
password (hashed
email
permission level
Toggle Admin Privileges
Delete
Continue to Main Menu


Main Menu
Welcome Back, George
Log Out
Admin Menu
Festival REPUBLIC.
Main Menu
Festivals By Price
Visit Festival Profile
Reading Festival
Delete Account
Festivals By Location


Search Via Artist!
Search Via Music
Search for Artists
Go
Festivals
Eminem
Two Door Cinema Club
Catfish and the Bottlemen
The Wombats
Slaves
Nothing But Thieves
Eminem
Chance the Rapper
Bryson Tiller
G-Eazy
Fetty Wap
Skepta
Travis Scott
Rae Sremmurd
The Weeknd
Nas
Tory Lanez
The 1975
Mumford and Sons
Baaba Maal
Glass Animals
Fleet Foxes
John Cale
Goldfrapp
Kasabian
Festival Overview
NEW LOOK
Close

Looking back on our initial designs, I am quite pleased with the progress I made, however given more time and resources, I would like to implement several changes –

- Fix the buttons so they all line up in a grid nicely – for example in the Main Menu:



- Also, I would like to replace the grey-background text boxes with graphic titles and more aesthetically pleasing forms. (Such as the main menu box above)
- Create dynamic image backgrounds, for example with each Festival Profile background creating a background for the festival it represents.
- Make it so that there is a custom pop-up box instead of different ones for both postcode entry and maximum distance entry.
- Make the table widgets size dynamic to the data it represents to avoid whitespace
- Create a theme changer to change background color-schemes (night-mode, chrome, white etc)

Outside Testing by Third Party

Jessica Skelton, a student from the Abbey School in Reading, kindly agreed to test out the program for usability feedback. Jess is a regular festival goer, having attended 5 in the last year. She was given account details to log in to the program. Here is what she had to say in an interview after. (Jessica is in Bold)

Did you have any feedback for the Festival Finder?

Well overall, the program seems to be a very good start into a currently non-existent service, but I do have some suggestions for the screens and services.

What would you suggest?

The current interface looks fairly okay, but perhaps creating a more polished interface without plaintext would be a step in the right direction – also the output for the program could be cleaned up, maybe using alert windows or mouse over text instead of embedded messages in the program?

Anything in terms of actual functionality?

Well this program certainly shortens the time it takes to find the right festival, the only problem is, currently Festival Republic only offers six different festivals – perhaps open the service up to different agencies for more functionality?

How do you feel about the integration of social media and including individual concerts in this product?

I think social media would be a good idea- perhaps pair the program with the users Facebook feed to see where other people are planning on going? In terms of individual concerts over festivals – perhaps if you integrate that you would want to separate the two different choices – festivals and concerts, as to avoid annoying clutter when you're looking for a festival or random festivals when you're trying to find a concert.

Okay, your feedback has been very helpful – thank you for your time!

Jessica demoed most of the features in the program itself in under 5 minutes, spending:

Register: 63 seconds

Login: 10 seconds

Main Menu: 16 seconds

Forgot Password : 0 seconds

Admin Menu : 0 seconds

Date Tracker: 12 seconds

Profile View: 37 seconds

Location and Alert Window: 57 seconds

Cost Overview: 20 seconds

Music Overview: 40 seconds

Delete Menu: 0 seconds

Buy Menu: 35 seconds

The input time seemed to be quite quick, with the register menu taking the longest with the strength values for username and password making quick work difficult.

A Reflection on Success Criteria

Initial Objectives –

Usability

1. The program will provide an interface between the user, a festival ticket buyer, to purchase tickets and find out information about festivals. ✓
2. The program should link to the appropriate ticket vendor for the festival selected. ✓
3. It will provide search/ selection tools that allow the user to narrow down their search by different protocols, for example, location and cost. ✓
4. It should involve full database integration. ✓
5. It provides a graphical user interface for log-on, then provides access to the festival program:
    a. The log-on should have full database compatibility. Usernames and Passwords need to be linked. ✓
    b. The system should store essential information on current users, such as links to orders, and information on the festivals themselves. ✓
    c. The program should provide a registering service for new users, where they can set up a username and password and create an account to use the service. ✓
    d. The log-on screen must be passed by logging in in order to gain access to the product. ✓
    e. The program should have a "Forgot Password" service, either through email or security question, for password recovery. ✓
    f. The programs forgot password service should provide a random number to reset the password to, update the user's table accordingly and email them their forgotten password in a correct format – eg. An Official "Festival Finder – Forgot Password" style email. ✓
    g. To use Regular Expressions functionality in order to regulate username and password form. For example, maintain a form for passwords to be between 8 and 20 characters, including numbers and different cases. ✓
    h. To provide a service that allows a user to delete their account and all records of it if necessary. ✓
    i. The program should encrypt all passwords accordingly, for example using a MD5 encryption when checking, submitting and changing passwords. ✓
6. The Program should provide a suitable menu interface to link all the features together. ✓
7. The programs should show evidence of fetching the current users details and greeting them, e.g. "Welcome back, (Username)" ✓
8. The program should provide a means to navigate the program itself. ✓
    a. The program should provide an easy to use Graphical User Interface which can navigate from one function to another. ✓

   b. The program should provide a menu after the process of logging on to navigate between different search functions – cost, location finder etc. ✓
   c. The program should provide easy navigation buttons to traverse the service – Close, Back, Logout etc. ✓
   d. The program should provide buttons for all the possible needs the user would want to use. ✓
9. The program should provide an accurate depiction of the end-user, to ensure maximum product effectiveness, tailored to our target audience. (In the form of logos, perhaps tailored information for tables) ✓

Performance

10. Provide a permission based system: ✓
    a. Admin users should have access to the users database from inside the program. ✓
    b. Admin users should be able to edit the permissions levels of other uses. ✓
    c. Admin users should be able to delete users from the database. ✓
11. The program should display data from a Database in a table view format: ✓
    a. This format should be able to be sorted via certain parameters – for example with the festivals via cheapest cost to most expensive – providing two options for both day ticket type and the full ticket type. ✓
    b. This data should be output to be viewed by the user in a table model or otherwise for best viewing of information. ✓
12. Provide service to view the "profile" of different festivals – provide the option of all festivals to be chosen via a menu – perhaps a dropdown list design – and then display the overview of that festival. ✓
    a. Profile would fetch the name of the festival and display it as the title of the window✓
    b. Profile would also fetch the cost in GBP. ✓
    c. Profile would fetch the start and end dates. ✓
    d. Festival would display the date range on a calendar view. ✓
    e. Profile would display all this data to the user in a suitable selection of widgets for the GUI . ✓
13. The program should also provide an interface for location services. ✓
    a. Looking up a postcode, an origin, to return a formatted address. ✓
    b. Using Festival Postcode stored in database to return formatted addresses✓

c. Finding the distance between the origin and the festival addresses using a Google Maps API. ✓

d. Dropping all of these locations in markers onto a map of the nearest locations. ✓

e. Displaying all of this information found for the user in a clean format on the GUI. ✓

14. The Program would supply a means to directly view the dates of festivals. ✓
    a. Provide a selection of festivals. ✓
    b. Out the name of the festival. ✓
    c. Output the start and end dates of this festival ✓
    d. Display these dates on a calendar widget. ✓
15. The program should accurately fetch data from the database with SQL queries – for example fetching all postcodes of festivals etc. ✓
16. The Program should allow the user to narrow down their search via music acts performing at certain festivals. ✓
    a. This should fetch and display all the performing acts. ✓
    b. Allow the user to search these acts specifically with a search bar for their selection or favourite. ✓
    c. The Program should then display the festivals for which this act is performing. ✓
    d. It should provide the names of all of the multiple festivals, if applicable. ✓
17. The Program should have the functionality to perform Create, Read, Update, Delete (CRUD) functionality with the databases in real time, this should include: ✓
    a. **C**reate – creating a new entry into our database. This could be employed via the Register form, administrator-level edits for the festival databases (if we expand this program to festivals beyond the Festival Republic brand) or for any ticket purchases ✓
    b. **R**ead – read the data from the database. This is incorporated with the viewing of any data from the databases, such as when the Festivals are researched based on information in their databases. ✓
    c. **U**pdate – Updating tables in the databases and committing the changes would be necessary to amend account information, change the location of festivals and update festival acts each year. ✓
    d. **D**elete – Deleting data from the database would be necessary to delete unnecessary date, amend address information and delete user accounts. (For example when a user would like to cancel or delete an account from our service, we should construct a window and option for that, which executes the suitable query to drop the user account from all the tables. ✓

Reliability

18. To Validate all available inputs – whether they suit the form the program needs, or there is any date at all. ✓
19. Using Regular Expressions to validate the username and password strength upon sign up ✓
    a. Username should be checked for matching regular expressions for basic checks – ie a length between 6-15, only alphanumeric characters or underscores. ✓
    b. Password should be checked for matching regular expressions for basic password strength (7-15 characters long, at least one uppercase letter, one lowercase letter and a number.) ✓
20. Validate inputs for matching passwords in the register form. ✓
21. Validate inputs for valid email addresses, usernames already in use etc. ✓
22. Validate for empty entries without any inputs at all. ✓
23. Make sure that an admin can't edit their own user data. ✓
24. Make sure that a postcode is valid using regular expressions to validate it. ✓

Maintainability

25. At all points in the program where the code isn't easy to understand or there is a loop, selection or procedure, use comprehensive comments to demonstrate for the next maintainer of the program what is involved and what happens at different points, so they can maintain the program accordingly. ✓
26. Meaningful variable names have been used, along with Hungarian notation to specify the usual/initial filetype of variables or data structures. ✓
27. Consistency in code writing, keeping methods as short as possible and making separate modules for aspects of the code will keep it easy to maintain for the next user. ✓

All of the individual objectives were met.

Have I met my Success Criteria?

Success Criteria Number	Required Result from Test	Has it Been Met?	Evidence	Comments from End-User
1	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 1, pg. 106	Good work, very robust.
2	The program must reject/ alert user to blank data entries and accept data entry.	**Yes**	Criteria Screenshot 2, pg. 109	Good work, very robust.
3	The program must login for the correct information and not allow entry for incorrect information.	**Yes**	Criteria Screenshot 3, pg. 109	Fully Functioning
4	The program must pass to the administrator table for administrators and pass to the main menu for non-administrator users.	**Yes**	Criteria Screenshot 4, pg. 112	Fully Functioning
5	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 5, pg. 114	Good work, very robust.
6	The program must reject/ alert user to blank data entries and accept data entry.	**Yes**	Criteria Screenshot 6, pg. 115	Fully Functioning
7	The program must reject inconformity and accept conforming inputs.	**Yes**	Criteria Screenshot 7, pg. 116	Fully Functioning
8	The program must reject inconformity and accept conforming inputs.	**Yes**	Criteria Screenshot 8, pg. 116	Fully Functioning
9	The program must reject a new user with an already used email and accept one with an unused email.	**Yes**	Criteria Screenshot 9, pg. 118	Fully Functioning
10	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 10, pg. 120	Good work, very robust.
11	The program must reject/ alert user to blank data entries and accept data entry.	**Yes**	Criteria Screenshot 11, pg. 122	Good work, very robust.
12	The program must send emails out to valid emails and reject invalid user emails.	**Yes**	Criteria Screenshot 12, pg. 122	Fully Functioning
13	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 13, pg. 123	Fully Functioning

14	Test that the delete function works	**Yes**	Criteria Screenshot 14, pg. 124	Fully Functioning
15	The program must display all the data and not allow users to delete their own account.	**Yes**	Criteria Screenshot 15, pg. 125	Fully Functioning
16	Test that the toggle admin privilege feature works	**Yes**	Criteria Screenshot 16, pg. 126	Fully Functioning
17	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 17, pg. 127	Good work, very robust.
18	Test for correct output (List of Festivals)	**Yes**	Criteria Screenshot 18, pg. 128	Fully Functioning
19	Tests must return correct date for each festival	**Yes**	Criteria Screenshot 19, pg. 128	Fully Functioning
20	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 20, pg. 131	Good work, very robust.
21	**Links to the correct Ticket Vendor URL**	**Yes**	Criteria Screenshot 21, pg. 132	Fully Functioning
22	Shows the correct data for the selected festival	**Yes**	Criteria Screenshot 22, pg. 134	Fully Functioning
23	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 23, pg. 138	Good work, very robust.
24	Successfully Plots the user's point of origin	**Yes**	Criteria Screenshot 24, pg. 139	Fully Functioning
25	Successfully Calculates Distances	**Yes**	Criteria Screenshot 25, pg. 140	Fully Functioning
26	Displays the correct locations of all the festivals	**Yes**	Criteria Screenshot 26, pg. 140	Fully Functioning
27	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 27, pg. 141	Good work, very robust.
28	Passing in the correct location to festival Location	**Yes**	Criteria Screenshot 28, pg. 141	Fully Functioning
29	The program must reject inconformity and accept conforming inputs.	**Yes**	Criteria Screenshot 29, pg. 141	Fully Functioning
30	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 30, pg. 141	Good work, very robust.
31	Displays data from a database	**Yes**	Criteria Screenshot 31, pg. 142	Fully Functioning
32	Displays the correct data for each button.	**Yes**	Criteria Screenshot 32, pg. 144	Fully Functioning
33	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 33, pg. 146	Good work, very robust.
34	The program must reject/ alert user to blank data	**Yes**	Criteria Screenshot 34, pg. 148	Fully Functioning

	entries and accept data entry.			
35	Yielding results for correct input	**Yes**	Criteria Screenshot 35, pg. 149	Fully Functioning
36	Displaying the correct data	**Yes**	Criteria Screenshot 36, pg. 150	Fully Functioning
37	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 37, pg. 151	Good work, very robust.
38	Deleting the account that's currently logged in.	**Yes**	Criteria Screenshot 38, pg. 151	Fully Functioning
39	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 39, pg. 152	Good work, very robust.
40	Displaying the correct data	**Yes**	Criteria Screenshot 40	Fully Functioning
41	The program must stand up to stress test and link to the correct functions/classes.	**Yes**	Criteria Screenshot 41, pg. 153	Fully Functioning
42	All the links link to the correct windows.	**Yes**	Criteria Screenshot 42, pg. 159	Fully Functioning
43	Only admin level users can access the admin menu via the main menu	**Yes**	Criteria Screenshot 43, pg. 159	Working fully
44	The program must be suitable for all computers that run on the Windows OS	**No**	All Screenshots	The program has a variety of different window sizes, with the location window being optimised for a 1080p display – this is not suitable for all users.
45	The program must provide full information on all artists performing at the festivals.	**No**	All Screenshots	The program only has act information for headliners performing each festival, not all individual acts.
46	The program must provide an adequate user interface for finding information about festivals	**Yes**	All Screenshots	The program does provide adequate information for the user.
47	The program must provide information relevant to all users	**No**	All Screenshots	The program only contains festivals within the UK – not accessible for all users.
48	All of the GUI is acceptable and is in keeping with the end-user's general look	**No**	All Screens	This criteria was failed because the labelling / titles and general look of the GUI is a bit juvenile –

				this could be rectified with a design team or collaboration with our department.

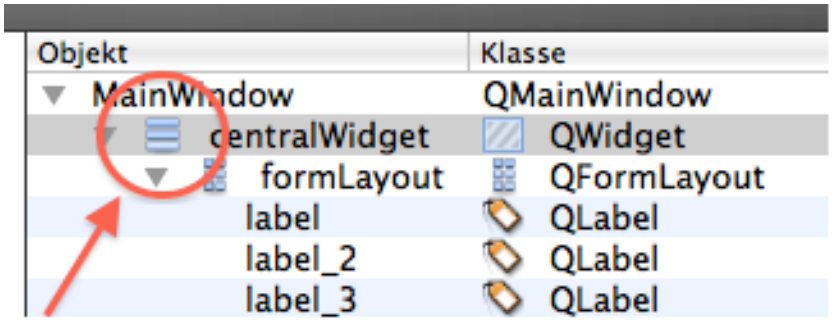
Solving Failed Success Criteria

Although we had very little success criteria that we failed, the success criteria that did actually fail are actually quite large, to solve these, I came up with these solutions –

Success Criteria 44 - The program must be suitable for all computers that run on the Windows OS

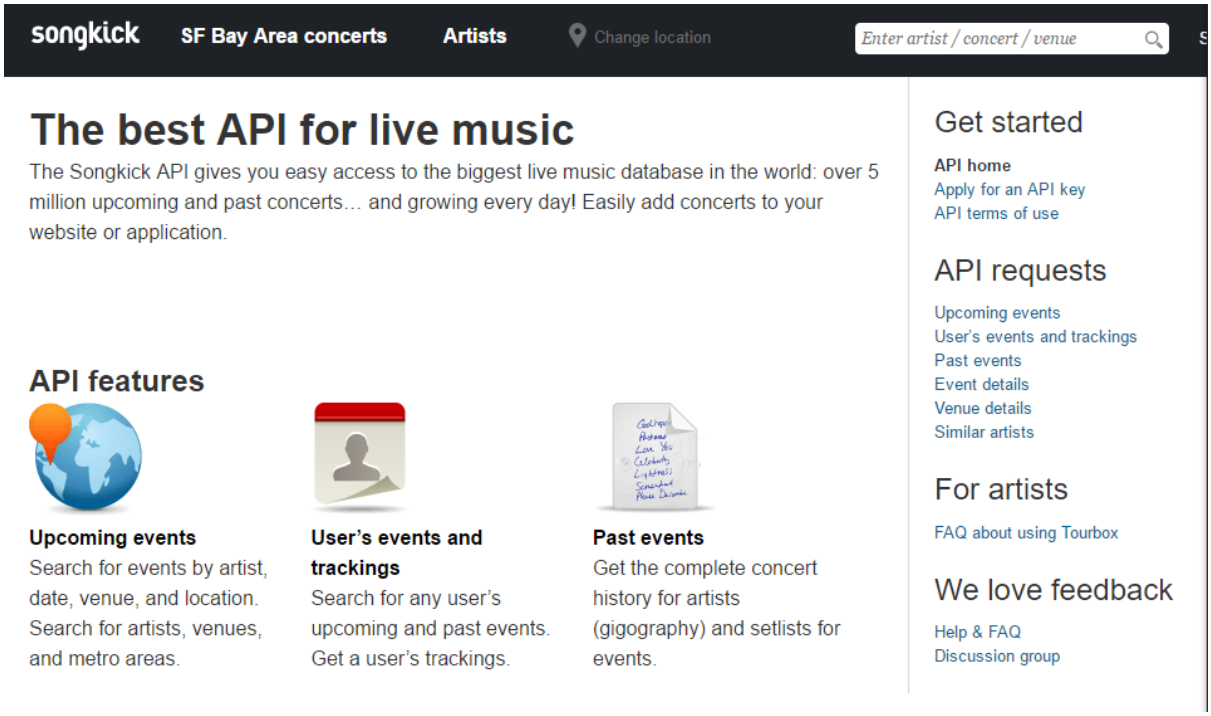
I failed this test because my windows are not suitable for every display – I could rectify this by making both the widgets and the window size dynamic for each display – stretching or constricting the size or the window and widget dependent on the resolution, possibly porting all windows to full screen.

Using the Qt Designer, I could activate the centralWidget and activate a form layout, this would make all the buttons stretch to the size of the window and behave depending on the window's size:



Success Criteria 45, 47 - The program must provide full information on all artists performing at the festivals and The program must provide information relevant to all users.

I failed this test as I did not include the full range of artists performing at each festival – I only included the headliners that were performing. To rectify this, I would either have to add the acts manually to the acts database, making sure to specify their genre with the genreID reference to the table genres, however I would prefer to change the way that the program fetches acts – perhaps using an intetgrating API for fetching this data – one such solution I found online is the developers API "SongKick" :



This API would give me access to a large amount of festival data – including festival names, act names, genres and past concerts. I could expand my Festival Finder to a simple Concert/Festival Finder, with access to information on thousands of different festivals, concerts and other music and give the user the ability to find an access music around the world, whilst keeping the functionality of cost, location, music sorting and a profile interface. The program could plot many more festivals on its map with this information, and I could expand my program to overseas events and more users (Different Festival Providers to Festival Republic). With this information I would also solve Criteria 47, giving festival information to festivals close to any user – in America, Germany, France etc.

Criteria 48 – The GUI's general look, making it more sleek and refined

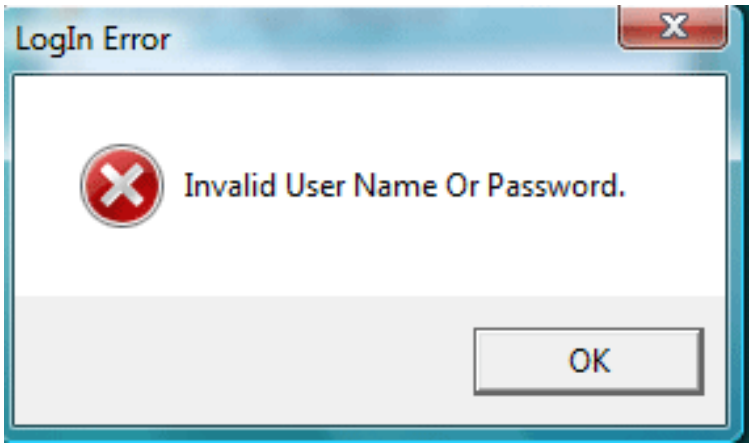
Solving this would be genuinely quite easy to implement, as it would not be to hard to edit the different interfaces to suit the design of the end-user. Perhaps this could be solved with a collaboration with the end-user's design team, or just by some more work with widgets, such as creating a semi transparent overlay for buttons and text, to ensure it looks better, Dynamically coloured buttons to suit the window and using Alert boxes instead of labels that display text for output (Such as "Wrong Password! Does not conform to form!")

One such change:

Before:

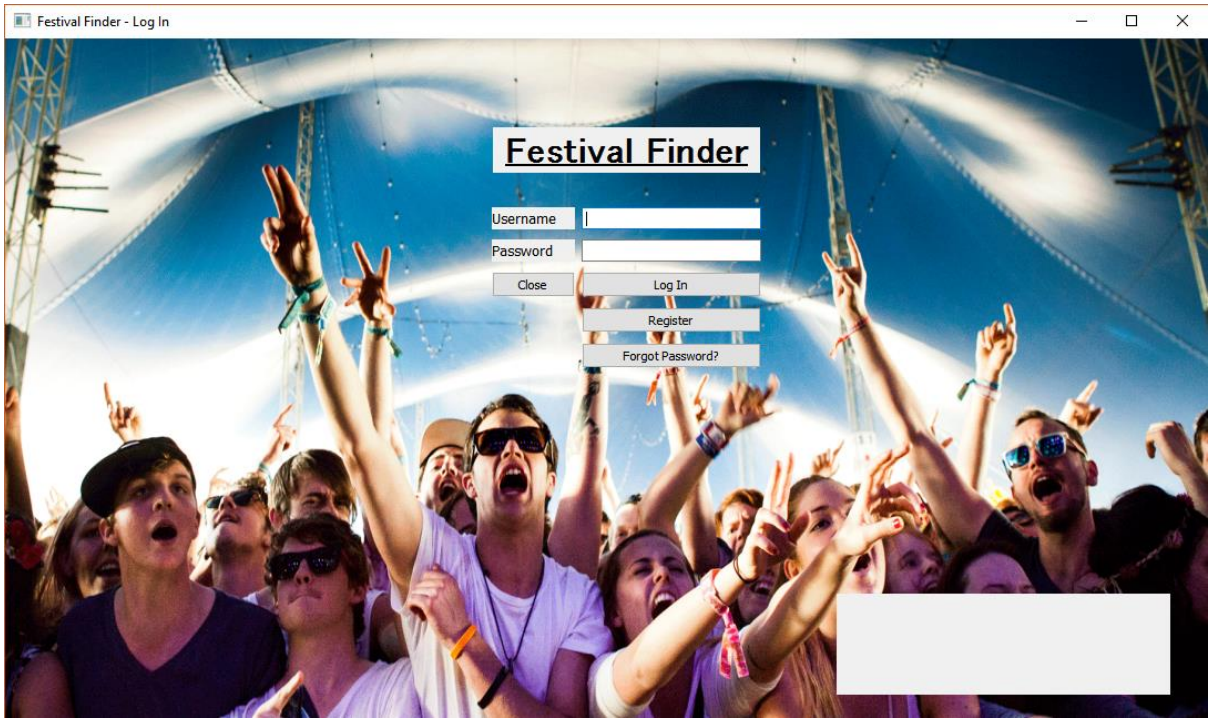

Wrong username or password

After:


LogIn Error
Invalid User Name Or Password.
OK

Describing the Final Product

The Login Window



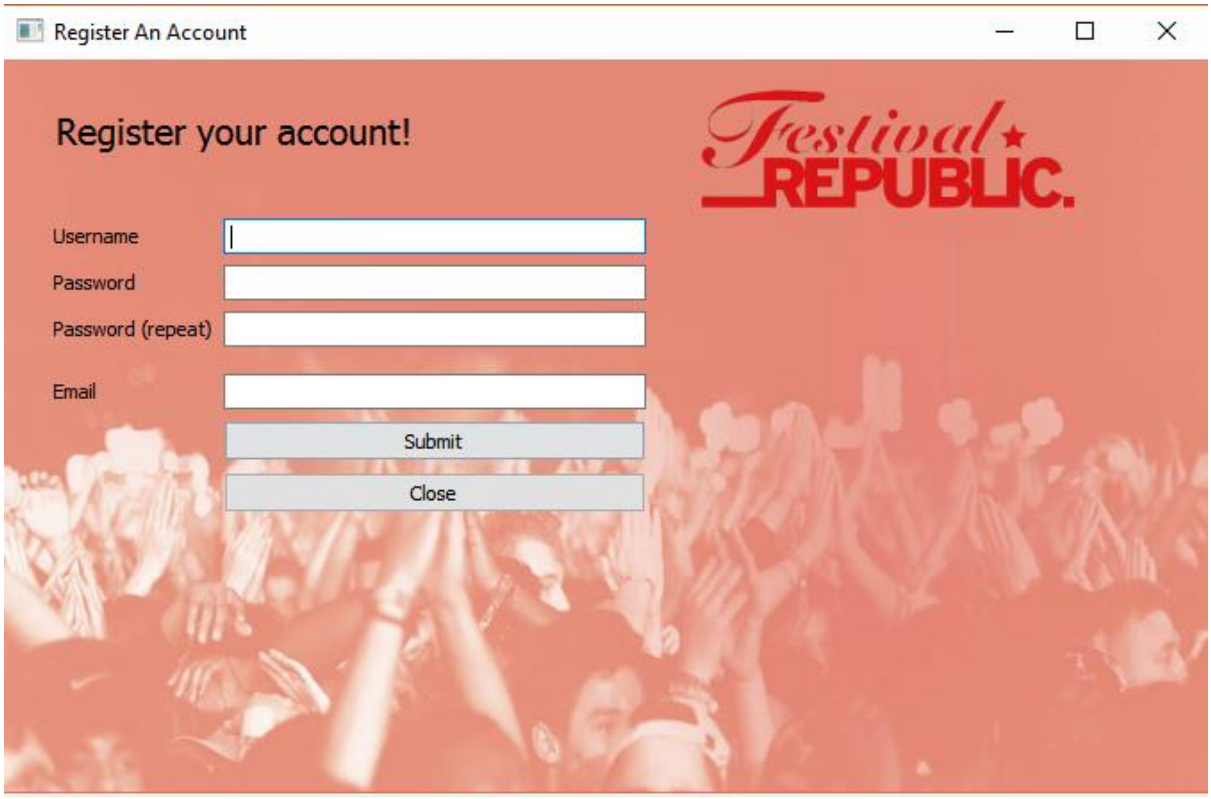
The logon window performs 4 main functions-

- Logging in to the program with a username or password
  - Inputting a wrong username or password results in an error message in the text message box.
  - Inputting no information results in a prompt for input (“Please fill all input boxes”)
  - Inputting a correct set of user information will result in a log in to either the admin menu or the main menu depending on the permission level
- Linking to the Register Window and the Forgot Password Window
- Outputting a Message based on user input (Wrong Password, No input submitted etc)
- Closing the program with a close button

Failed Success Criteria 48 – All of the GUI is acceptable and is in keeping with the end-user’s general look

I could improve this screen by creating a more clean and organised array of pushbuttons, perhaps a command link buttons for forward, back or close buttons and use Graphics in place of Text labels with auto-filled grey backgrounds. I could implement a general colour scheme for the program in general. Output could be displayed by either pop-up alert windows or by mouse over text (for specific input based messages ie. “Don’t Leave this blank” or the strength regulation for regular expressions enabled input fields. The other criteria failed do not related to this section.

The Register Window



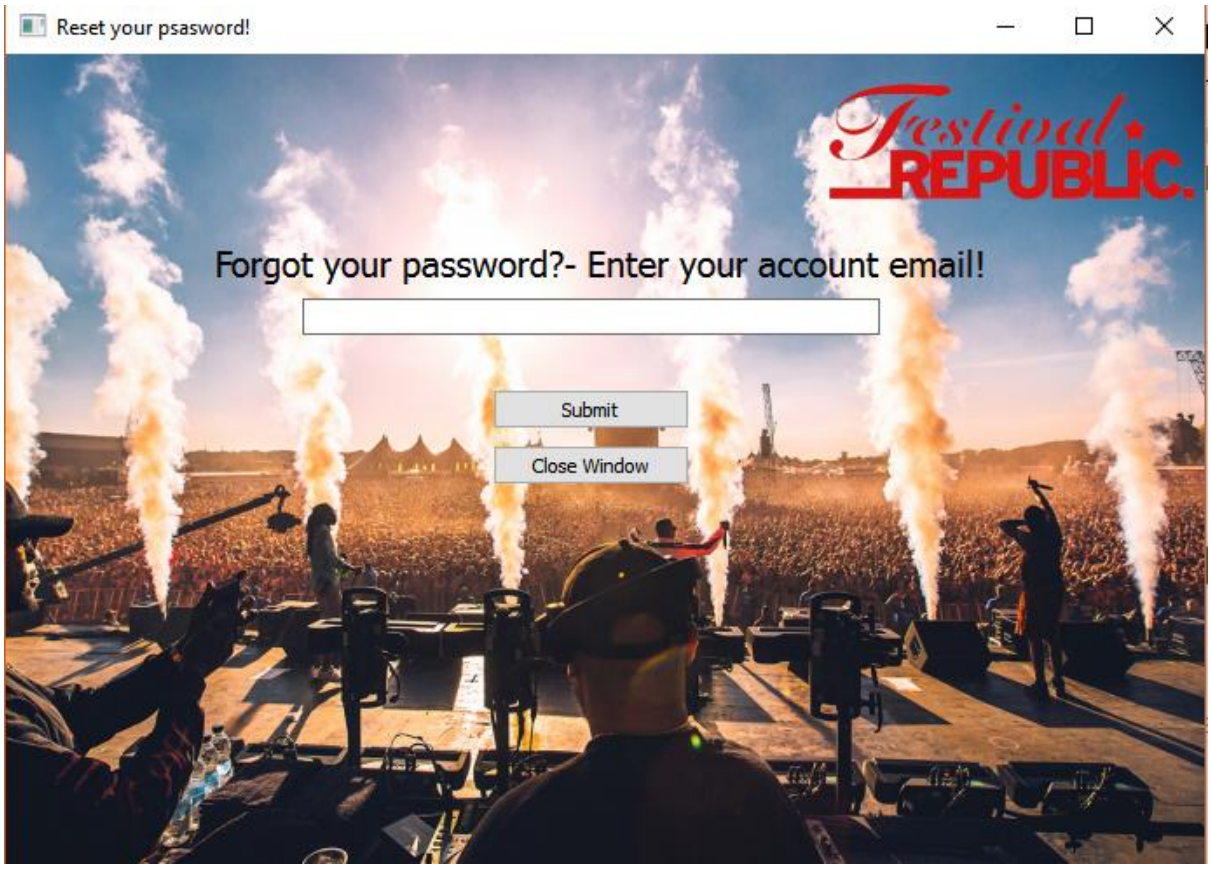
This window has several input boxes with several main paradigms –

- It will accept input from all the boxes
- It will display a prompt if the fields are left blank
- It will check if the username already exists and if so reject the new account
- It will check if the email already exists and if so reject the new account
- It will check the two password fields for a match, accepting only when the passwords are identical
- It will test the username field for Regular Expression conformity (It must be made up with alphanumeric characters, underscores, be in the length 6-12 and start with an alphanumeric character)
- It will test the password field for Regular Expression conformity (It must be made up with alphanumeric characters, underscores, be in the length 6-12 and start with an alphanumeric character)
- It will provide a link to the log-on screen through the close button

Failed Success Criteria 48 – All of the GUI is acceptable and is in keeping with the end-user's general look

I could improve this screen by creating a more clean and organised array of pushbuttons, perhaps a command link buttons for forward, back or close buttons and use Graphics in place of Text labels with auto-filled grey backgrounds. I could implement a general colour scheme for the program in general. Output could be displayed by either pop-up alert windows or by mouse over text (for specific input based messages ie. "Don't Leave this blank" or the strength regulation for regular expressions enabled input fields. The other criteria failed do not related to this section.

Forgot Window



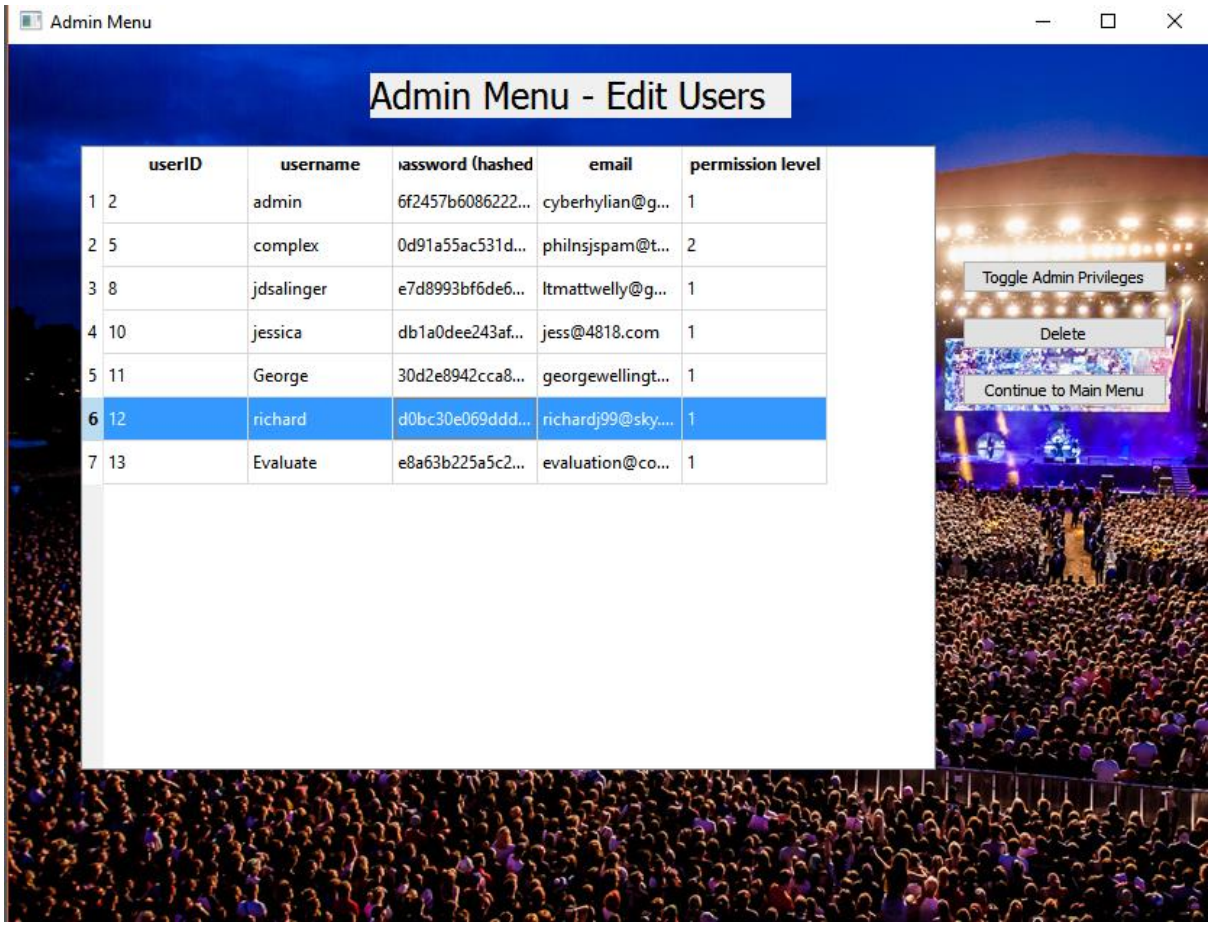
This window does several different things–

- Takes in input and prompts if field is left blank
- Checks if the email is in use – if it is then it resets the password for that account and sends an email to the user with the new information
- Links back to the logon screen with the close button

Failed Success Criteria 48 – All of the GUI is acceptable and is in keeping with the end-user's general look

I could improve this screen by creating a more clean and organised array of pushbuttons, perhaps a command link buttons for forward, back or close buttons and use Graphics in place of Text labels with auto-filled grey backgrounds. I could implement a general colour scheme for the program in general. Output could be displayed by either pop-up alert windows or by mouse over text (for specific input based messages ie. "Don't Leave this blank" or the strength regulation for regular expressions enabled input fields. The other criteria failed do not related to this section.

Admin Window



This window performs several functions –

- Links to the main menu
- Toggles the administrator privileges of the user account selected
- Deletes the account selected

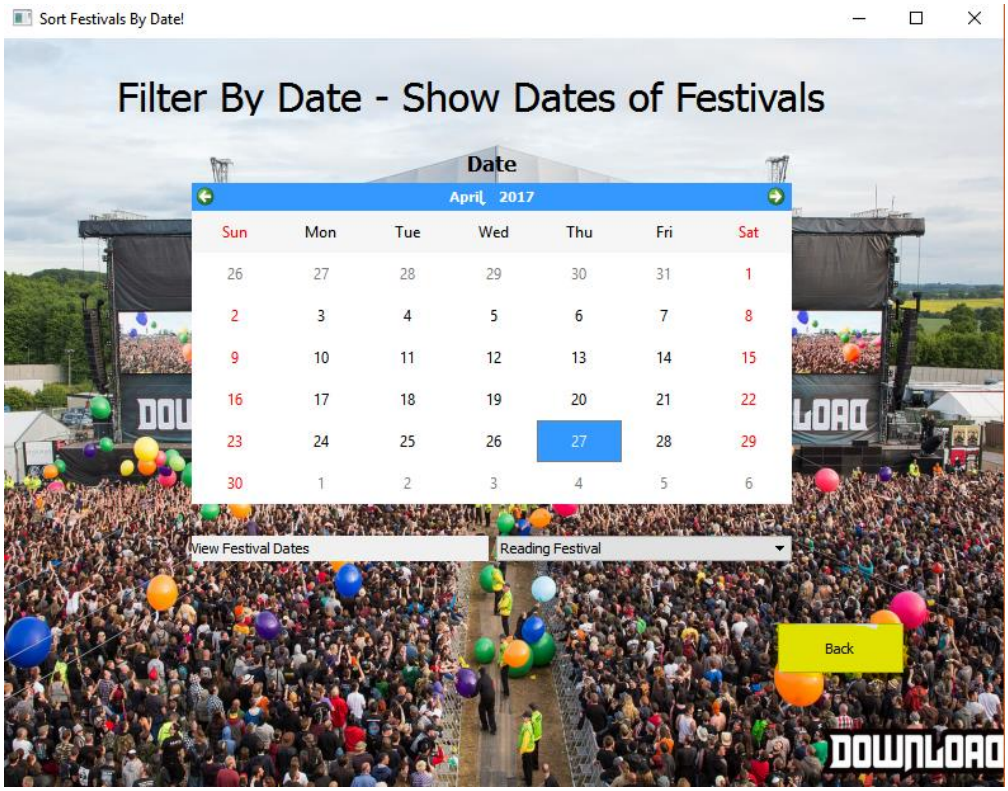
Failed Success Criteria 48 – All of the GUI is acceptable and is in keeping with the end-user's general look

I could improve this screen by creating a more clean and organised array of pushbuttons, perhaps a command link buttons for forward, back or close buttons and use Graphics in place of Text labels with auto-filled grey backgrounds. I could implement a general colour scheme for the program in general. Output could be displayed by either pop-up alert windows or by mouse over text (for specific input based messages ie. "Don't Leave this blank" or the strength regulation for regular expressions enabled input fields. The other criteria failed do not related to this section.

Failed Success Criteria 44 – Making the Display suitable for all screens

This screen could be improved by making the size of the widgets dynamic – ie the table view widget – to suit the screen of the device it's being viewed on and to grow or shrink to fill the available space and the data that is being displayed by the program.

Date Window



This window performs several functions –

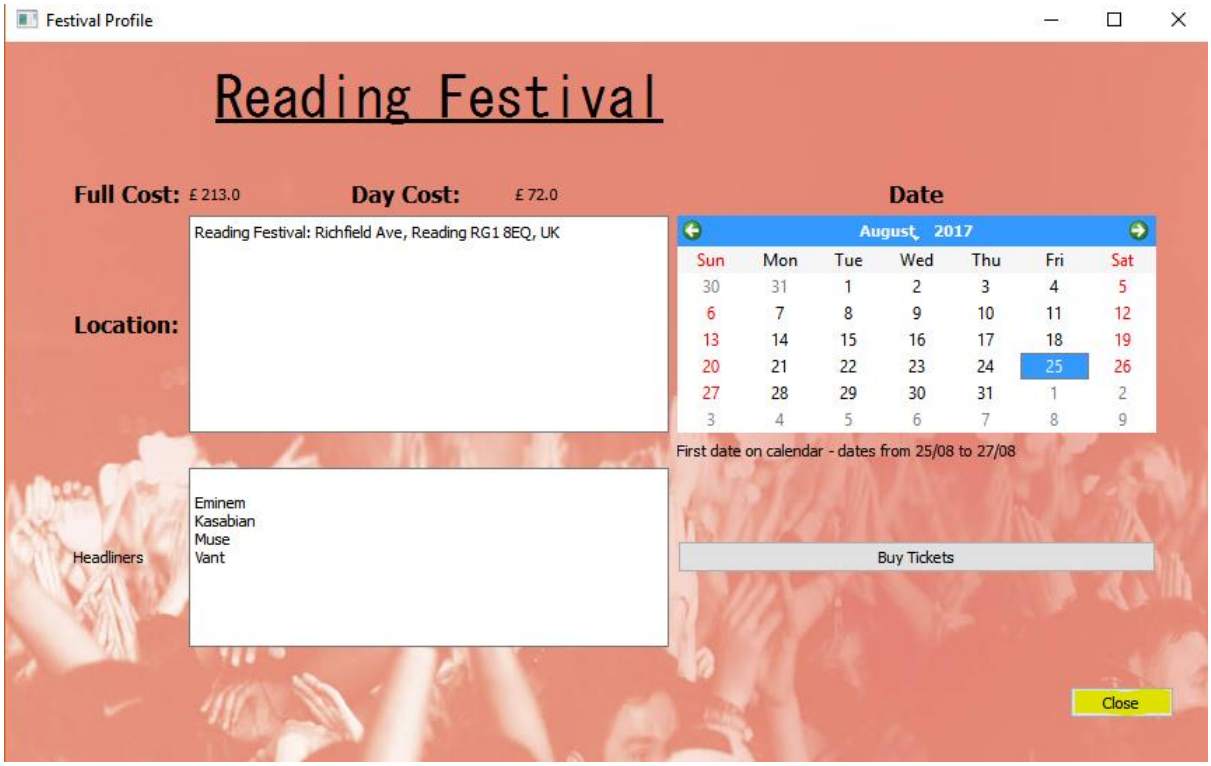
- It provides a link back to the main menu through the back button highlighted for viewing
- It provides a list of festival names through the combo box below the date widget
- Shows the start date and date range when a festival is selected in the combo box

Failed Success Criteria 48 – All of the GUI is acceptable and is in keeping with the end-user's general look

I could improve this screen by creating a more clean and organised array of pushbuttons, perhaps a command link buttons for forward, back or close buttons and use Graphics in place of Text labels with auto-filled grey backgrounds. I could implement a general colour scheme for the program in general. Output could be displayed by either pop-up alert windows or by mouse over text (for specific input based messages ie. "Don't Leave this blank" or the strength regulation for regular expressions enabled input fields. The other criteria failed do not related to this section.

I could improve this window also by incorporating a custom coded version of the calendar widget – making my own polymorphic design by allowing the calendar to select multiple dates and highlight all the dates of the festival in question.

Profile Window



This window performs several functions –

- Links back to the main menu via the highlighted close button
- Links to the buy menu by setting the URL of the buy screen to a concatenated string of a ticket vendor URL and the festival name
- Outputs the headliners in a text browser
- Outputs the location in a text browser
- Outputs the date of the festival
- Outputs the cost of the festival
- Outputs the name of the festival

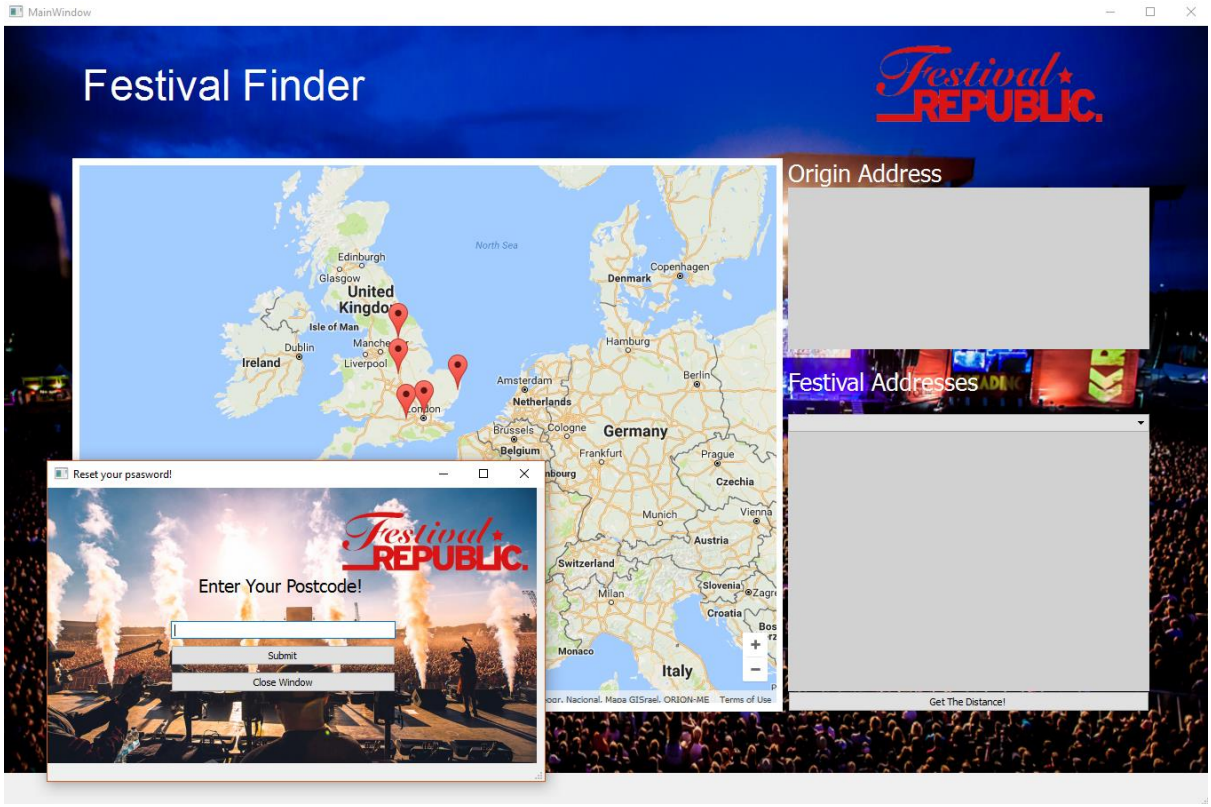
Failed Success Criteria 48 – All of the GUI is acceptable and is in keeping with the end-user's general look

I could improve this screen by creating a more clean and organised array of pushbuttons, perhaps a command link buttons for forward, back or close buttons and use Graphics in place of Text labels with auto-filled grey backgrounds. I could implement a general colour scheme for the program in general. Output could be displayed by either pop-up alert windows or by mouse over text (for specific input based messages ie. "Don't Leave this blank" or the strength regulation for regular expressions enabled input fields. The other criteria failed do not related to this section.

I could improve this window also by incorporating a custom coded version of the calendar widget – making my own polymorphic design by allowing the calendar to select multiple dates and highlight all the dates of the festival in question.

I could also improve this by changing the text browsers to dynamic text browsers that suit the size of the information to be output.

Location and Alert Windows



These windows perform several functions –

- Ask the user for a conforming postcode to be entered
- Ask the user for a maximum distance they would like to travel
- Displays geocached addresses of both the user and the festivals in the database
- Displays the distances between the user and these festivals.
- Plots these locations on a map, with the icons for each festival being the first letter of the festival's name
- Plots a green marker for the user's location, along with the maximum distance they would like to travel as a blue radius

Failed Success Criteria 48 – All of the GUI is acceptable and is in keeping with the end-user's general look

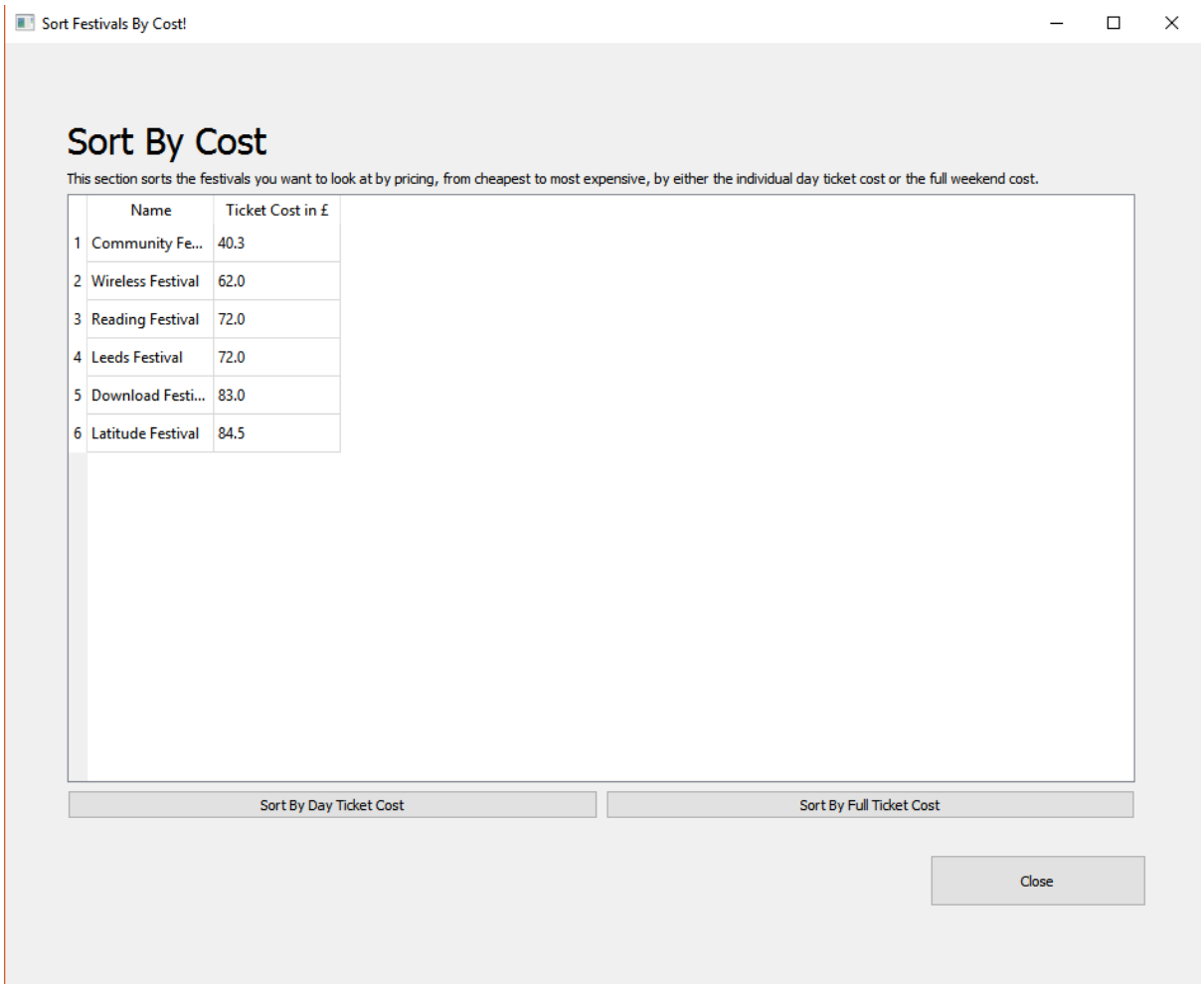
I could improve this screen by creating a more clean and organised array of pushbuttons, perhaps a command link buttons for forward, back or close buttons and use Graphics in place of Text labels with auto-filled grey backgrounds. I could implement a general colour scheme for the program in general. Output could be displayed by either pop-up alert windows or by mouse over text (for specific input based messages ie. "Don't Leave this blank" or the strength regulation for regular expressions enabled input fields. The other criteria failed do not related to this section.

Partial Success Criteria 45-47 – Extending the program for more festivals

This screen would be significantly affected by changing the festivals processed from a database of my own creation to a API sourced database structure with thousands of concerts and festivals. The program could be extended to work as it does but using the data source of the SongKick API,

providing information for festivals around the globe, thus reaching more users, having more information for every act performing and overall providing more user service.

Cost Window



This window performs several functions –

- Displays the Festival Name and Cost of all festivals
- Sorts by the Day Ticket Cost
- Sort by the Full Ticket Cost
- Close links back to the main menu

Failed Success Criteria 48 – All of the GUI is acceptable and is in keeping with the end-user's general look

I could improve this screen by creating a more clean and organised array of pushbuttons, perhaps a command link buttons for forward, back or close buttons and use Graphics in place of Text labels with auto-filled grey backgrounds. I could implement a general colour scheme for the program in general. Output could be displayed by either pop-up alert windows or by mouse over text (for specific input based messages ie. "Don't Leave this blank" or the strength regulation for regular expressions enabled input fields. The other criteria failed do not related to this section.

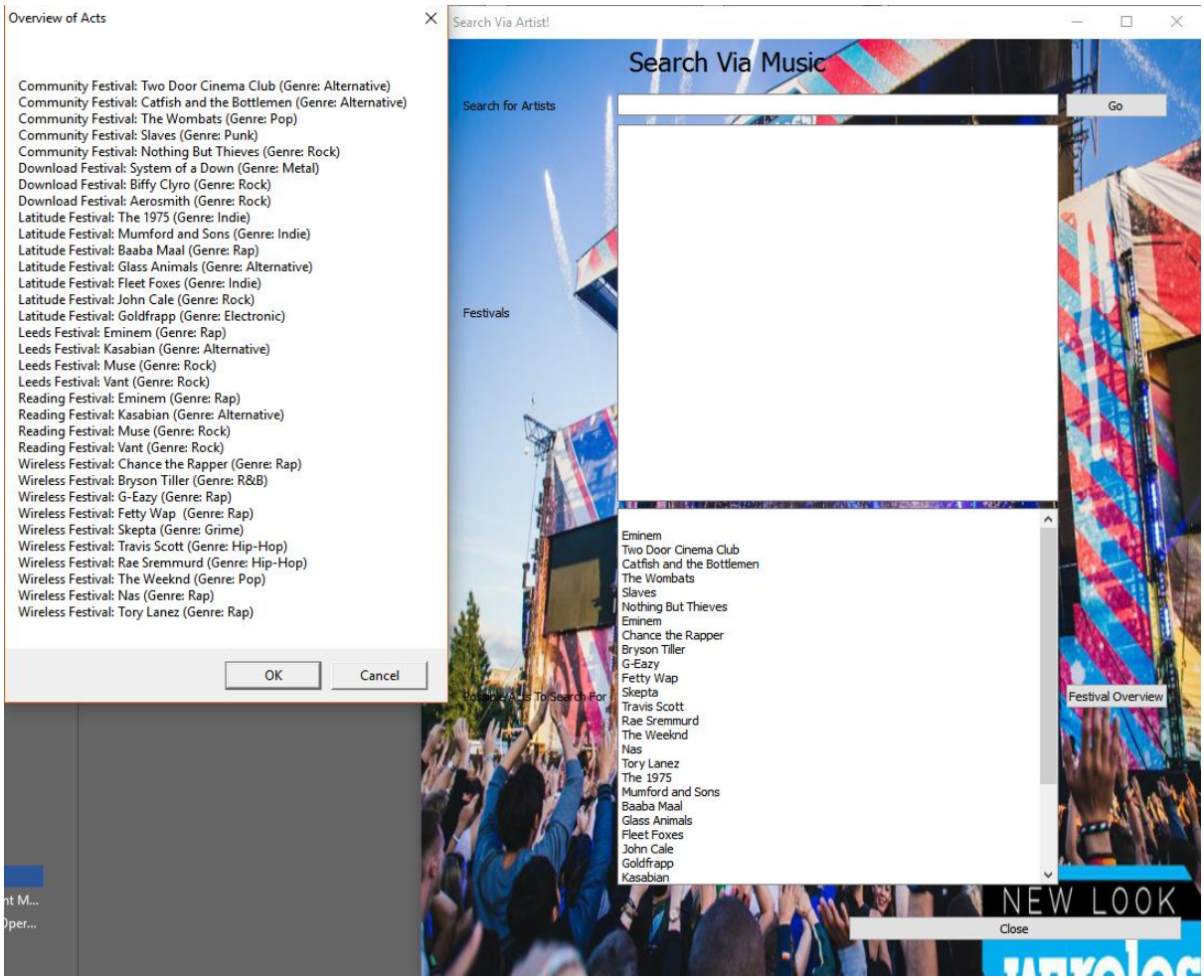
Partial Success Criteria 45-47 – Extending the program for more festivals

This screen would be significantly affected by changing the festivals processed from a database of my own creation to a API sourced database structure with thousands of concerts and festivals. The program could be extended to work as it does but using the data source of the SongKick API, providing information for festivals around the globe, thus reaching more users, having more information for every act performing and overall providing more user service.

Failed Success Criteria 44 – Making the Display suitable for all screens

This screen could be improved by making the size of the widgets dynamic – ie the table view widget – to suit the screen of the device it's being viewed on and to grow or shrink to fill the available space and the data that is being displayed by the program.

Music Window



The Music Window performs several functions –

- Allows the user to search via input of an artist
- Outputs an Overview of the festivals including the Festival Name, Act Name and Genre when the overview button is clicked
- Outputs the available artists
- Outputs the festivals the searched for artist is performing at

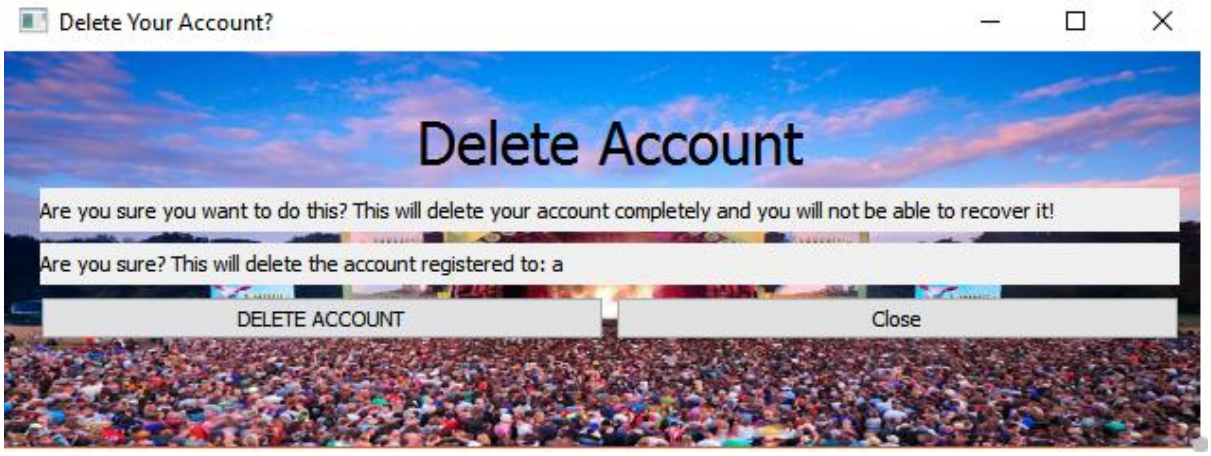
Failed Success Criteria 48 – All of the GUI is acceptable and is in keeping with the end-user's general look

I could improve this screen by creating a more clean and organised array of pushbuttons, perhaps a command link buttons for forward, back or close buttons and use Graphics in place of Text labels with auto-filled grey backgrounds. I could implement a general colour scheme for the program in general. Output could be displayed by either pop-up alert windows or by mouse over text (for specific input based messages ie. "Don't Leave this blank" or the strength regulation for regular expressions enabled input fields. The other criteria failed do not related to this section.

Partial Success Criteria 45-47 – Extending the program for more festivals

This screen would be significantly affected by changing the festivals processed from a database of my own creation to a API sourced database structure with thousands of concerts and festivals. The program could be extended to work as it does but using the data source of the SongKick API, providing information for festivals around the globe, thus reaching more users, having more information for every act performing and overall providing more user service.

Delete Window



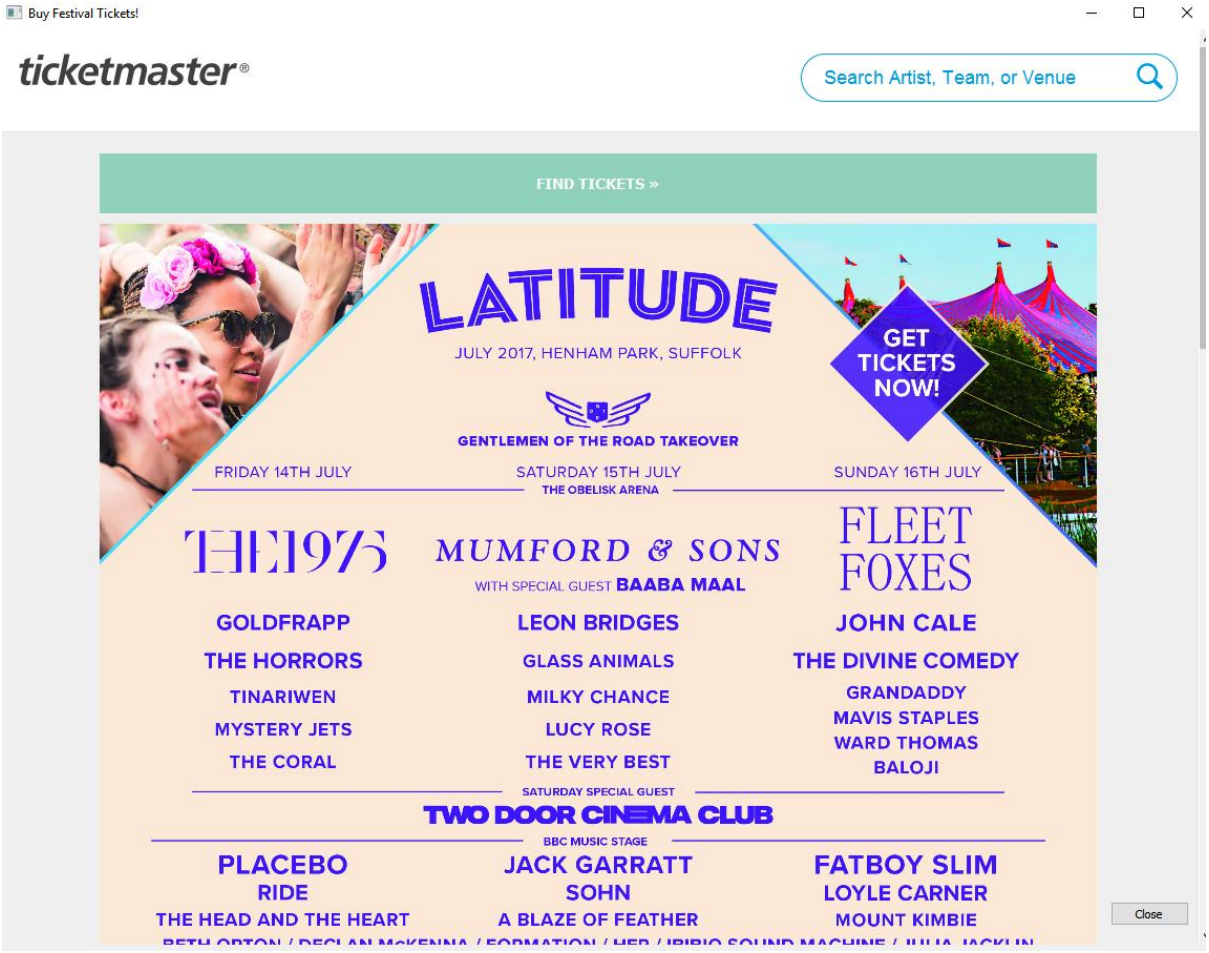
This window only does two things –

- Deletes the currently signed in user
- Links back to the main menu with the close button

Failed Success Criteria 48 – All of the GUI is acceptable and is in keeping with the end-user's general look

I could improve this screen by creating a more clean and organised array of pushbuttons, perhaps a command link buttons for forward, back or close buttons and use Graphics in place of Text labels with auto-filled grey backgrounds. I could implement a general colour scheme for the program in general. Output could be displayed by either pop-up alert windows or by mouse over text (for specific input based messages ie. "Don't Leave this blank" or the strength regulation for regular expressions enabled input fields. The other criteria failed do not related to this section.

Buy Window

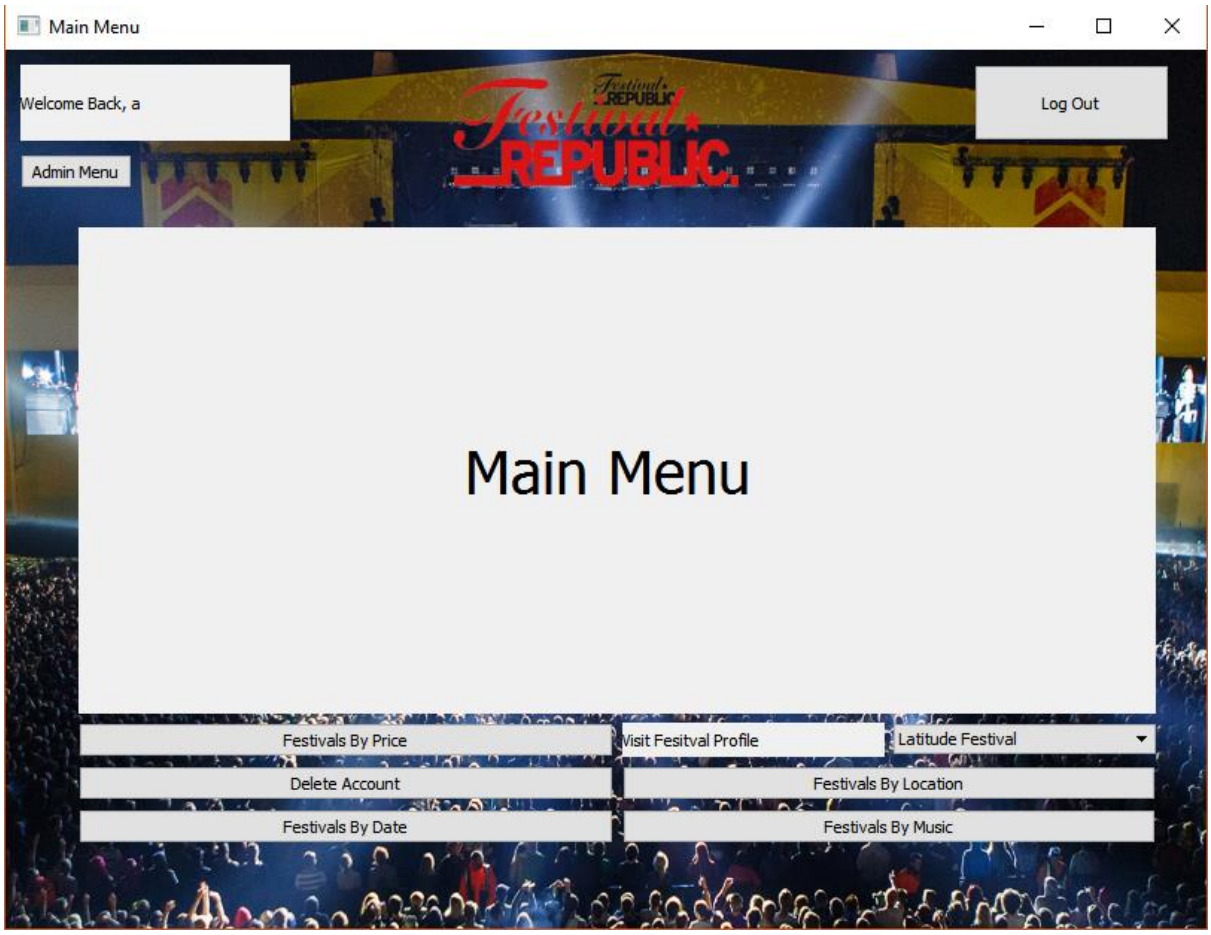


This displays the URL passed in by festival profile and links to the main menu via the close button.

Failed Success Criteria 48 – All of the GUI is acceptable and is in keeping with the end-user's general look

I could improve this screen by creating a more clean and organised array of pushbuttons, perhaps a command link buttons for forward, back or close buttons and use Graphics in place of Text labels with auto-filled grey backgrounds. I could implement a general colour scheme for the program in general. Output could be displayed by either pop-up alert windows or by mouse over text (for specific input based messages ie. "Don't Leave this blank" or the strength regulation for regular expressions enabled input fields. The other criteria failed do not related to this section.

Menu Window



The final window performs many simple functions –

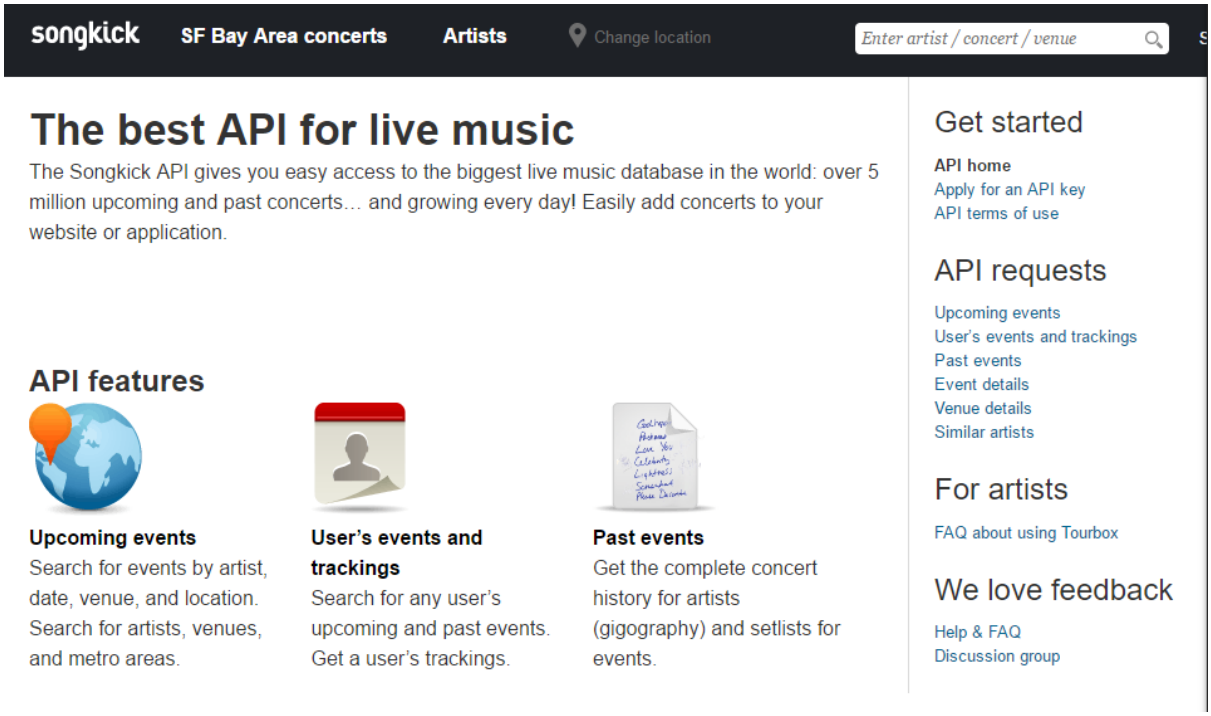
- Outputs a welcome message for the currently selected user
- Allows admins access to the admin menu via the admin menu button
- Allows access to the profile window via the profile combobox
- Allows access to the location window via the location button
- Allows access to the music window via the music button
- Allows access to the price window via the price button
- Allows access to the delete window via the delete button
- Allows access to the date window via the date button
- Signs out the user and returns to the log-in screen with the logout button

Failed Success Criteria 48 – All of the GUI is acceptable and is in keeping with the end-user's general look

I could improve this screen by creating a more clean and organised array of pushbuttons, perhaps a command link buttons for forward, back or close buttons and use Graphics in place of Text labels with auto-filled grey backgrounds. I could implement a general colour scheme for the program in general. Output could be displayed by either pop-up alert windows or by mouse over text (for specific input based messages ie. "Don't Leave this blank" or the strength regulation for regular expressions enabled input fields. The other criteria failed do not related to this section.

Maintainability of Solution – What could be changed? (Potential New Features and Partial Successes I would pursue given more time)

I would prefer to change the way that the program fetches acts – perhaps using an integrating API for fetching this data – one such solution I found online is the developers API "SongKick" :



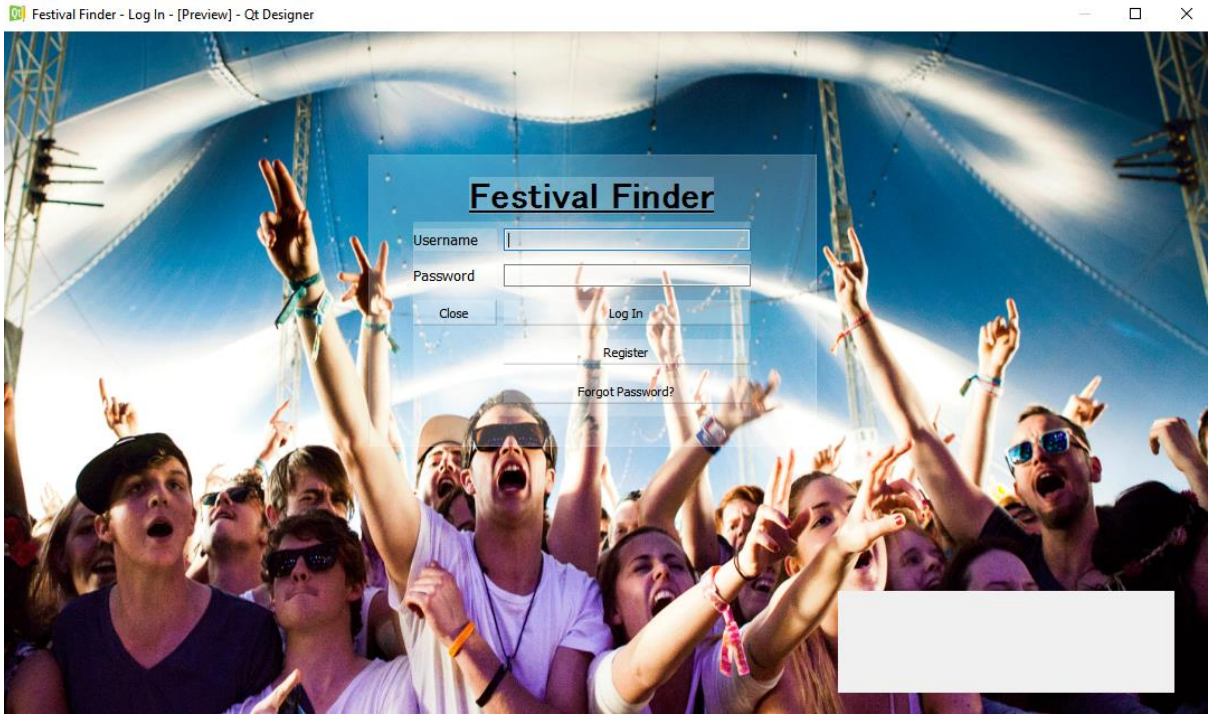
This API would give me access to a large amount of festival data – including festival names, act names, genres and past concerts. I could expand my Festival Finder to a simple Concert/Festival Finder, with access to information on thousands of different festivals, concerts and other music and give the user the ability to find an access music around the world, whilst keeping the functionality of cost, location, music sorting and a profile interface. The program could plot many more festivals on its map with this information, and I could expand my program to overseas events and more users (Different Festival Providers to Festival Republic). With this information I would also solve Criteria 47, giving festival information to festivals close to any user – in America, Germany, France etc.

This could potentially prove a problem if the data is not exactly matching the data I used in the program ie The date screen would have to be changed from recognising "24/07" to the date format in the new API.

Given more time, I would implement this.

I would also like to clean up the GUI in general – turn the titles into graphics and customise buttons and widgets with bespoke designs through Adobe Photoshop.

I could also replace the auto-filled labels with transparent overlays and forms such as a transparent style sheet – here is an example of a concept that I quickly drew up in Festival Login:



This would eliminate auto-filled labels and also aid with the visibility issues with similar looking widgets and backgrounds.

I would like to add social media integration as well – involving linked Facebook accounts to users so they can see who is interested or going to festivals they have found:

I think including a feature like this would close the gap with competitor applications.

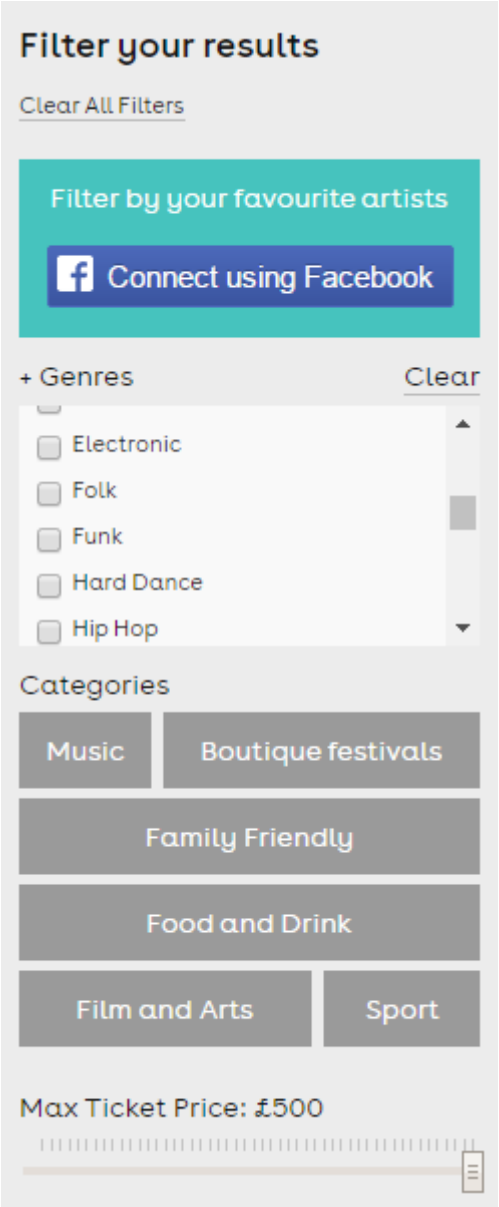
A page like this could be integrated into the festival profile window also -

I could also embed a recommendation function, from festivals the user has viewed previously or an artist they have searched for – artists playing the same festivals or similar genres could be suggested for the user.

Reading Festival
Leeds Festival

This is a potential mockup of what the output would like like

A search engine feature would also be a potential route I could explore – narrowing down festivals by criteria the user enters, ie. Cost boundaries and genres. This is a similar design to the design I would want to implement:



Analysing Time Spent on Different Methods

Name	Call Count	Time (ms)	Own Time (ms)
<built-in method exec_>	1	41455563 100.0%	41455550 100.0%
<method 'read' of '_ssl._SSLSocket' objects>	6	1268 0.0%	1268 0.0%
<built-in method setHtml>	1	1078 0.0%	1078 0.0%
<built-in method __new__ of type object at 0x5B6E6FC0>	124	1042 0.0%	1042 0.0%
<method 'do_handshake' of '_ssl._SSLSocket' objects>	6	241 0.0%	241 0.0%
<built-in method stat>	671	111 0.0%	111 0.0%
<method 'connect' of '_socket.socket' objects>	6	102 0.0%	101 0.0%
<method 'load_verify_locations' of '_ssl._SSLContext' objects>	6	68 0.0%	68 0.0%
<built-in method load_dynamic>	11	75 0.0%	65 0.0%
get_data	120	59 0.0%	50 0.0%
setupUi	1	51 0.0%	45 0.0%
<built-in method show>	2	44 0.0%	44 0.0%
<built-in method listdir>	33	37 0.0%	37 0.0%
<built-in method getaddrinfo>	6	33 0.0%	33 0.0%
<built-in method setUrl>	2	27 0.0%	27 0.0%
<built-in method open>	79	26 0.0%	25 0.0%
<built-in method __build_class__>	847	33 0.0%	21 0.0%
festivalFinder.py	1	41460720 100.0%	21 0.0%
<built-in method OpenKey>	280	20 0.0%	20 0.0%
<built-in method loads>	120	15 0.0%	15 0.0%
__getattribute__	6032	16 0.0%	14 0.0%
__getattribute__	8862	13 0.0%	12 0.0%
<method 'read' of '_io.FileIO' objects>	120	9 0.0%	9 0.0%
__next	5883	9 0.0%	8 0.0%
<connectSlotsByName>	13	8 0.0%	8 0.0%
close	6	8 0.0%	7 0.0%
_get_spec	134	159 0.0%	6 0.0%
_parse	217	21 0.0%	6 0.0%
find_spec	332	144 0.0%	6 0.0%
<method 'close' of '_io.TextIOWrapper' objects>	66	6 0.0%	6 0.0%
<built-in method setColumnCount>	3	5 0.0%	5 0.0%
<built-in method isinstance>	9792	8 0.0%	5 0.0%
_path_stat	625	114 0.0%	4 0.0%
createWidget	155	101 0.0%	4 0.0%
<method '_parse_whole' of 'xml.etree.ElementTree.XMLParser' objects>	13	4 0.0%	4 0.0%
_path_join	1317	8 0.0%	4 0.0%
__call__	755	24 0.0%	4 0.0%
<built-in method setNavigationBarVisible>	2	4 0.0%	4 0.0%

Above is a mix of time slices spent on each part of the program, the all of the time being occupied almost unchallenged by the built in method exec, as that process is running all the time and then other portions of the program, such as online requests and SSL sockets, taking up a significant portion of resources.

The time complexity of all the programming that occurs within the program has a maximum time complexity of O(n) with the longest processes taking that amount of time with n being the information processed. ie for a While loop n would be the amount of circuits. The time complexity of a internet request such as my google maps data however would depend on that server and the internet – which for obvious reasons I am unable to estimate in all situations. The operations themselves should have O(n) complexity with a delay equal to the ping from the server.

I can estimate that the entire program complexity is O(n) + D with n being the amount of data processed and D referencing the delay experienced when initialising the display and fetching information from online sources (ie Geocaching).

Due to the upper and lower bounds of the validation I have for my program, there is no way to enter enough data to significantly alter the time compelxtiy of the program in its current state – the user only enters small amounts of data at a time eg user accounts, postcodes.

Product Methodology

This project, admittedly though better suited to Rapid Application Development, was done in the style of “The Waterfall Lifecycle” or simply the waterfall methodology model – flowing from Requirements to Analysis to Design to Coding to Testing to Maintenance and Evaluation, albeit with iteration between adjacent stages as recommended by its founder, William Royce. It was designed and set out and documented step by step through this process.

Porting The Program For Other Operating Systems

Mobile –

Porting this application to mobile platforms would be difficult, firstly because of the incompatibility of the designs, ie the windows are not designed for use on a small screen or even use with touch screen interfaces. Another thing to consider is that the python/ PyQt package is quite hard to port to mobile, and the amount of devices you could port to would be fairly limited, as the iOS would not accept it unless it was written in a Apple-compatible programming language such as Apple's Swift. Third party software that supports both the programming language (Python) as well as the graphical framework (Qt) would be few and far between, and the code would definitely require tweaking.

Mobile development can be done with QML – The Qt Modelling Language. This is a mobile-compatible interface developer with lots of options

At this moment Qt is working on support for mobile devices. Most mobile operating systems like iOS and Windows Phone are not supported yet for the combination Python/Qt and mostly the support for working projects is not really big in comparison to the preferred languages of the operating systems. Using an app such as PySide would be required for a port.

On last thing to keep in mind is that the Google Maps API allows a extremely easy port to iOS and Android Operating system – this implementation would be made easy by Google.

Web/ Client Based Program

This would prove fairly difficult - the webkit component in PyQt is intended as a content viewer – it cannot be directly ported to a HTML5 app. However using a library such as Twisted, a network programming library, could provide the results by integrating the python code with web based network ports.

Switching Programming Languages –

The QtDesigner framework is supported by many languages, making this port a fair bit easier – the databases themselves and the SQL queries wouldn't need to change as the data stays the same, the most work would be converting syntax from one program to the next. Using the pseudocode, it would be fairly easy to write the program up in another language, as the Qt Framework is cross-platform, having being first written in C++.

Another useful tool for this would be Beaker, a note-book styled itergrated development environment for working with datasets. It uses plugins that allow to switch language with ease -

Maintainability

Finally, the users of Festival Finder and the next developer working on the project will have a fairly easy time - I made sure throughout my code to use descriptive and meaningful variable names in Hungarian Notation, demonstrating what a variable is and its data-type. I also incorporated use of modules that are both available online and ones I made myself, chunking out the code so that a potential maintainer of the code in future could modify the map layout or email system without touching the main code. This in turn creates a simple and easy to extend for future personnel looking to improve or extend the product. Throughout, I've made functions as short as I can and used them multiple times if necessary.

An example of my maintainability can be seen from the register class' submit function:

```
def submit(self):
    self.str_user = self.entry_username.text()  # takes in the items in the entries as variables
    self.str_password = self.entry_password.text()
    self.str_password_2 = self.entry_password2.text()
    str_email = self.entry_email.text()
    debug(self, self.str_password)
    if self.str_password != "" or self.str_password_2 != "" or self.str_user != "" or str_email != "":
        reg_user = r"^[a-zA-Z0-9][a-zA-Z0-9_]{4,12}$"
        # starts with alphanumeric, can only contain alphanumeric or underscore with length 4-12
        reg_pass = r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]{6,12}$"
        # (6-12 chars, at least one lower,upper and number and no specials)
        con = lite.connect("festivalFinder")  # connects to the database
        result = con.execute("SELECT * FROM tbl_users WHERE email = ?", (str_email,)).fetchall()
        existing_user = con.execute("SELECT * FROM tbl_users WHERE user = ?", (self.str_user,)).fetchall()
        debug(self, result)
        if len(result) > 0:  # if there is already an existing user with that email (search yields)
            self.lbl_msg.setText("Email already in use, try another")
        elif len(existing_user) > 0:  # if there is an existing user with that username
            self.lbl_msg.setText("Username already in use, try another")
        else:
            if re.match(reg_pass, self.str_password) and re.match(reg_user, self.str_user):
                # checks if the variables user and pass match the regex criteria above
                self.lbl_msg.setText("success")
                if self.str_password == self.str_password_2:
                    # if the passwords are the same on both attempts
                    self.lbl_msg.setText("passwords successfully match - Account Created")
                    hashed_pass = (hashlib.md5(self.str_password.encode())).hexdigest()
                    # hashes the password
                    con.execute("INSERT INTO tbl_users (user, pass, email) VALUES (?,?,?)",
                                (self.str_user, hashed_pass, str_email))
                    # creates a new user
                    con.commit()
                else:
                    self.lbl_msg.setText("passwords don't match")
            else:
                self.lbl_msg.setText("password or username or pass doesn't match form")
        con.close()
    else:
        self.lbl_msg.setText("fields must not be left blank")
```

Bibliography

Google Maps and SongKick API mentioned:

https://developers.google.com/maps/

http://www.songkick.com/developer

End-User website and details (current system)

http://www.festivalrepublic.com/

https://en.wikipedia.org/wiki/Festival_Republic

www.readingfestival.com/

End User Signatures – Proof of involvement

Seymour Parsons, Senior Member Festival Republic

Gareth Cutbill, Sales Representative, Festival Republic

Johnathon Skelton, Sales Representative, Festival Republic