**OCR A Level Computing Coursework Writeup**

**Analysis**

**The Problem**

**Stakeholders**

The end user is an owner of a construction firm named Kieran Reed. His business works to place construction workers who are looking for employment in construction related jobs in to said positions by having those who are looking for workers come to his company to hire said construction workers. His business makes a profit by charging a fee to individuals who hire the construction workers. For example, a homeowner may decide they wish to have an extension done to their property. The homeowner would approach Kieran and ask to hire a builder, perhaps an electrician and a plumber as well. Next, Kieran would charge a commission of 3% of the total cost of the hiring, which is where Kieran makes his profit. The workers are sent out by Kieran to the relevant home, where work will begin.

As seen above, Kieran groups his customer’s needs based on what sort of worker they need to hire. As such, he groups his workers in this way, making them have a job title, which could be a plumber, electrician, builder and more. This needs to be verified by the worker and checked that they are needed by the customer as well.

As Kieran receives requests from construction sites to provide workers, it is vital that work provided to the contractors is:

1. Close to their respective abode
2. Contain jobs which the contractor is qualified to do
3. Fairly compensated in relation to the skills the contractor has, as well as experience

Currently, Kieran stores his records of workers and building sites in a paper-based database, which is prone to damage and takes a long time to sort through. A large amount of time is spent filing papers to ensure they are filed properly to make assigning jobs to contractors as fair and easy as possible. This leads to lost time which could be spent on more important things, such as counting penguins given by Mr Travi to his computer science group, dealing with customers, and otherwise growing his business.

By creating a database which stores various information on Kieran’s contractors, Kieran can much more easily direct contractors to suitable jobs using heuristics to measure distances from a construction site to a worker’s home, matching contractor’s skills to a keyword or category making selection easier, and allows Kieran to securely store records of payments far more easily.

**Benefits**

The clients of this program would be those who are in a situation similar to Kieran; they own a contracting agency of some form where a worker’s location is relevant to the jobs they are assigned to. One example might be that of a teacher’s agency, where substitute teachers may be required and as such teachers who are not currently in work can sign up for the agency.

This program would prove to be extremely helpful to both clients and contractors, however, as it would provide a much more reliable and quicker method of retrieving data on payments made to them, allowing potential legal issues to be carried out much more effectively.

**Why Use a Computational Solution?**

Currently, Kieran has a large number of employees and building sites which he needs to keep track of. This leads to a large amount of paper in his files in his office.

This leads to multiple problems such as:

* Organising files regularly
* Finding a place in the office to put files
* Look through files when looking for a specific job
* Look through files when looking for a specific person
* Maintain the quality of the paper inside the files
* Be careful to avoid damage to files to avoid financial loss
* Store sensitive data in a way adhering to data legislation acts, which is much harder in a paper form

These problems can all be hugely mitigated or removed by using computational methods to solve Kieran’s problem.

**Computational Methods Which Will Be Used to Solve the Problem**

**Recognition**

The problem here is finding a way to allow Kieran to store data and process data to find the most suitable candidate for the job, and store sensitive details about said job. As a result, multiple parameters will likely make an impact on the candidate selected, such as distance from the construction site and days since the last job received by the worker. Once this has been solved, one will likely need to apply a method to estimate or calculate distances from building sites to homes of the workers, apply a GUI to make accessing the database easier, and use a secure way to store sensitive details.

Another aspect of the problem involves reducing the time Kieran spends on mundane tasks in order to keep his data in line, such as filing and managing the data. This will be able to be performed digitally, meaning he can use his computer to enter new data and look for it. Kieran will not need to maintain his data nearly as often, although he will likely need to back the data to a cloud computing solution in order to keep data safe.

**Decomposition**

This program can be decomposed further. The following is an idea of what these steps are:

1. Create a secure database capable of safely storing sensitive data like passwords and payment details.
2. Make a GUI that is both friendly for Kieran to use and has the functionality he requires in order to use the program effectively.
3. Combine the data stored within the database or workers and the data on location of construction sites alongside heuristics to suggest workers who would be suitable for the job, and fit the requirements outlined by the construction site.

Each of these steps can then be further broken down in to submodules or routines, which will form the basis of the procedures written in the program.

**Abstraction**

Abstraction will be used in this program to hide unnecessary details, such as the age or gender of a worker or their name when calculating which workers are most optimal for the jobs outlined. This will save lots of space in the GUI, making it cleaner and far easier to use for the consumer, who doesn’t need to know every single function in the program, nor the details of classes used.

The use of abstraction can also help hugely when considering the Data Protection Act. Abstraction can be used to hide unnecessary detail, such as confidential information, when it is not needed. This means Kieran no longer runs the risk of accidentally viewing confidential information, meaning he can ensure his client’s data is more adequately managed in line with the Data Protection Act.

**Researching Other Solutions**

Currently, there are a few similar solutions to Kieran’s problem on the market, but none which fully encapsulate what he needs.

**BuilderTrend**

BuilderTrend is a database software that is designed for building firms with multiple workers and a home office, which attempts to make it possible for workers to communicate and manage jobs more easily.

The biggest advantage of this software is the fact it can be used on mobile devices, making the software very convenient to use as well as user friendly. As well as this, the software can log photos and date jobs which are under progress, allowing for more information to be provided to a client on where the construction process currently is as well as how long it will take to complete.

However, the biggest disadvantage of this software to Kieran is that it assumes transport from location to construction site is not a problem, as all workers are likely to be close to the home office anyways. This is not the case for Kieran, who must manage workers over a very large geographical area, where it may not be feasible for all workers to reach all construction sites, meaning the software cannot choose the most optimal worker for the job either.

*Talk about what I can take from this*

**Oracle Aconex**

Oracle Aconex is a service provided by Oracle that for the most part is designed for large scale building projects from companies who may have to run multiple projects at a time, suiting Kieran well for his needs. It uses a cloud computing solution to allow workers to be able to access data from construction sites, meaning needed updates to tasks can be pushed out to workers.

However, many of the features which Aconex uses are completely unneeded for Kieran; the construction sites are the ones who would handle how and where the workers are needed, not him, so a cloud computing solution would be excessive and wasteful of resources. In addition, workers may end up seeing more of the project than they needed to, leading to privacy issues which Kieran could save himself the trouble of fixing.

In addition, Aconex also does not provide a heuristics tool to allow Kieran to easily determine which workers should be assigned to each construction site, although it could help to sort workers who had the skills required to work at said site.

**Takeaways from these programs**

Both of the programs talked about above have many desirable traits that would be beneficial for Kieran. However, there are also many traits which are simply excessive, unneeded or are too expensive for Kieran to upkeep to make it sustainable and optimal for his, or any other person in a similar situation’s position.

BuilderTrend does a fantastic job storing relevant details about each construction site easily, with the ability to store addresses, names, payment information and much more. However, some of the features are simply unnecessary. An example of this is the ability to be used on mobile devices. While this is great for anyone who has a large scale business with multiple administrative employees, Kieran is the only administrator in his company. As a mobile app would require the database to be stored on a server, which would require a cloud computing solution as previously mentioned, this option is not viable as it would cost a large amount of money to establish a server.

BuilderTrend’s ability to log progress on a site may be helpful to other firms, but for Kieran who rents his builders out for a fixed amount of time instead of when the job is done, the ability to take photos is unneeded and unhelpful. As such, I will likely choose not to implement it in to the project.

However, BuilderTrend’s use of a database is an approach I will use. Its UI is simple, which I will also attempt to recreate as well.

Aconex, however, has a slightly intimidating UI which takes some skill to navigate. This is not an approach I will take, as the program I need to make for Kieran is much simpler and less consuming.

On the other hand, Aconex is a powerful database that uses referential integrity principles from the ACID model to work effectively, which I will need to implement in to my program.

Both programs feature cloud computing in ways that aren’t helpful to the program. However, we can still utilise cloud computing to back the database up, to ensure any possible damage done is not permanent and will not lead to serious business issues for the client.

**Problem Investigation**

In order to gain a deeper understanding of the problem at hand, I felt necessary to interview the relevant stakeholders; namely Kieran, a building site owner, and a construction worker.

**Questions For Kieran:**

1. How long do you spend organising your files per week?

2. What’s your current system for organising work and workers?

3. How long do your workers work in a day?

4. How experienced are you with a computer?

5. What would a computer based solution mean for your company?

6. How else do you spend your time?

7. How do clients contact you and provide their details?

**Interview with Kieran:**

1. “I wouldn’t be able to say for sure, but I’d say give or take 3 to 5 hours per week? This includes simple tasks like placing the details of my new clients in, and moving files around to make sure everything is in order and easy to find.”

2. “Right now, I have separate folders for storing confidential information and information I can freely view. I have current jobs in a folder on my desk, and multiple archive files where I place details about jobs which have been fully completed. I also have a separate board where I keep jobs where payments are due. Workers are stored in a separate folder, where I can look through to choose a worker who is most suitable for the job. The worker’s name and address is written on the job’s description sheet to ensure the payment goes to whom it is due.”

3. “Our workers work from 9am to 5pm.”

4. “I can use simple programs like Word, Excel and my internet browsers, but past this I am not very experienced. I don’t have experience with any programming languages.”

5. “It’d definitely save me a lot of time, which I could put towards other endeavours like visiting sites of my clients, dealing with complaints and generally providing better customer service. I could also spend more time with my clients, making sure they get the workers they need, increasing customer satisfaction and creating more regular clients for my business to grow.”

6. “I think I mentioned most of this previously. Apart from filing work, I have to deal with various clients who may e-mail me with concerns about their jobs, where I will have to schedule an appointment. For that, I already use Microsoft calendar, which provides a fantastic solution for my appointment problem.”

7. “Clients contact me by e-mail predominantly, while others may phone or text me. Normally, I will book an appointment for them to come to our office and speak to me about what they need from the job being done, so I can adequately provide them with the best service.”

**Questions For Building Site Owner:**

1. What’s the most inconvenient part of the process of booking a worker?

2. How experienced are you with a computer?

3. Do you generally know what you need when you come to book a worker?

4. How experienced are you with construction by the time you come to book a worker?

**Interview With Building Site Owner:**

1. “The most inconvenient part is finding a time where both I and Kieran have time to come in and discuss. Both of us work full time, so often our availability clashes, leading to both of us having to take time out to book a time to talk about the project.”

2. “I have a basic knowledge of how to use one, and can navigate my way through simple programs.”

3. “Generally, I come in with a good idea of what I need, as most people coming to Kieran are experienced in the industry or have had some advice from a friend or architect.”

4. “Personally, the first time I came to see Kieran I was fairly sure on what I needed from the advice of my architect, who helped me and many others. Generally, I’d say most people coming to see Kieran for the first time would know what they need”

**Questions For Worker:**

1. How important is location for you when working?

2. How often do you come in to the office?

3. How many hours do you work in a day?

**Interview With The Worker:**

1. “I like to be close to the site, because it’s more convenient that way. It does matter to me, but not an incredible amount.”

2. “I only visit the office when I have issues getting paid or when I came to get the job in the first place. Otherwise, I don’t ever need to come in.”

3. “I work from 9 till 5.”

**Limitations**

There are limitations to the program, such as the budget which can be allocated towards Kieran’s computing solution. Ideally, Kieran would not need to buy any additional hardware, like a server, meaning cloud computing cannot be used as it would simply cost too much. While Kieran could make use of an external cloud computing solution, it too would cost a certain amount of money to service, which is not ideal.

In addition, Kieran’s requests come in many forms, as outlined above, being phone calls, emails and texts. These cannot have the information easily pulled from them, and so there will need to be some standardized way of obtaining the necessary information. One idea for such a solution is to simply have all clients fill a form at the office, which provides these details directly on to the system.

**Final Success Criteria**

The objectives for the program are as follows:

1. The program must provide a client with their own unique account with a secure password, and assign an ID to the account.
2. Payment details must be stored safely and securely for each client’s account.
3. Client accounts must be able to be created, deleted and updated with new information from within the user interface.
4. Passwords must use hashing techniques to securely store passwords, hashing the entered password to obtain a confirmation of whether it is correct.
5. The program must contain a user interface that is intuitive and simple to use, with minimal clutter.
6. The database must be fully relational to allow for optimised performance among larger data sets.
7. The database must contain all information needed on both clients and employees, including date of birth, dates of hire, addresses for client building sites and employees home addresses, as well as many more relevant pieces of information.
8. The program must be able to tell how far a site is from an employee’s location and choose an employee to let out based on these details.
9. The database must contain specifics on what job the employees/freelancers carry out- for example, specify if the employee is a plumber, builder or other worker related to construction.
10. The database must allow for one to track payment dates and times to provide records of transactions and payments to employees.
11. The program must provide accounts to employees and clients which allows them to see their job history, payment history and job location. For employees, they will be able to view client details and for clients they will be able to view employee details.
12. The database must adhere to the ACID model of referential integrity, keeping values atomic, the database durable and consistent while isolating entered data.

**Design**

**Decomposition**

As explained previously, decomposition aims to break a larger problem in to smaller, easier to manage parts. In this case, we will need to break the larger database problem into many solvable sub modules that can be more easily coded. Advantages of this approach include the easier to code submodules that are produced, which can lead to easy code re-use later on in development if one part of code turns out to be particularly useful, as seen very often with many programs. Another advantage is the more robust code that is produce, as each function needs to work as intended, which is much easier than going through a larger program to check everything works as intended. The possibility for code re-use that was previously touched on also allows for better code efficiency, reducing space needed by the program.

Diagram

Description automatically generated

**GUI**

The GUI will be coded using PyQt 5, a wrapper that allows the C QT library to be used inside of python, my programming language of choice for this program. This has a number of advantages, such as the inbuilt GUI designer, meaning I do not need to program the GUI, instead I simply need to include a number of GUI files which I swap between when needed.

The PyQt library requires one to use object oriented programming to initialise objects (your windows) to start the program. As such, it is a more complex approach than procedural programming, but is a much better solution to the problem of establishing a GUI.

**Design Features**

The GUI will make up all of the visible program for the user, so it is important that it is easy to navigate and use. As such, I have taken many factors in to consideration when creating mock up designs for my GUI.

**Colours**

Colour is a large part of any design, and this is no less true for GUIs. The default PyQt package supplies a clear and easy to read set of penguin-like colours, in particular black and grey, although you might be hard pressed to find grey on any penguins I have personally met. These colours are clear, contrast well, and ultimately work nicely to make text legible and clear.

**Layout**

The layout should be clear, with minimal clutter and boxes lined up. This will not only provide an air of professionalism and polish to the program, but also help the user as a simple and effective layout will do wonders for general user usability.

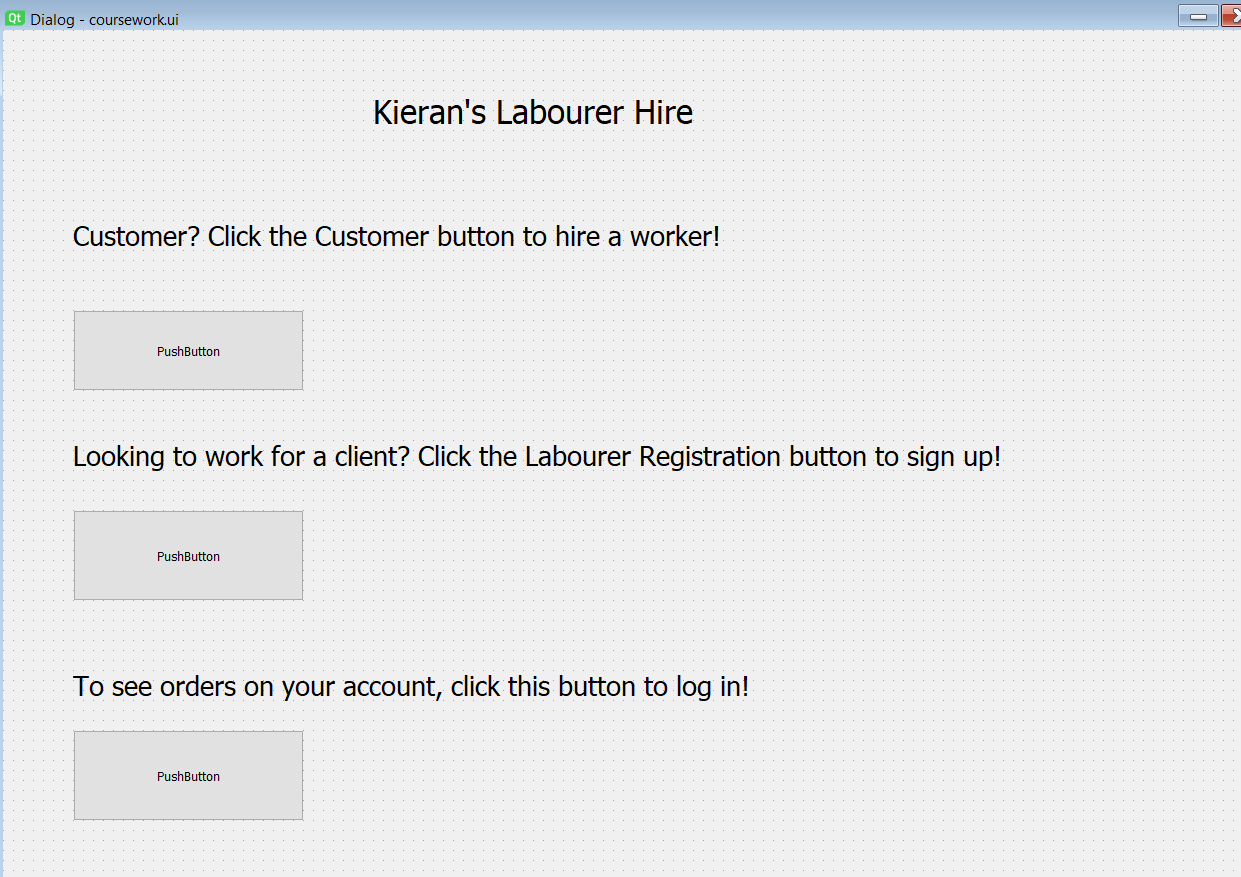
**Font**

The font itself should be clear and easy to read. For example, a font like Bradley Hand would not be suitable for this program, as it cannot be read by most people easily, and hence would not be consumer friendly.

Similarly, font should be of adequate size to allow those with less adept eyesight see the text easily.

**Main Menu**

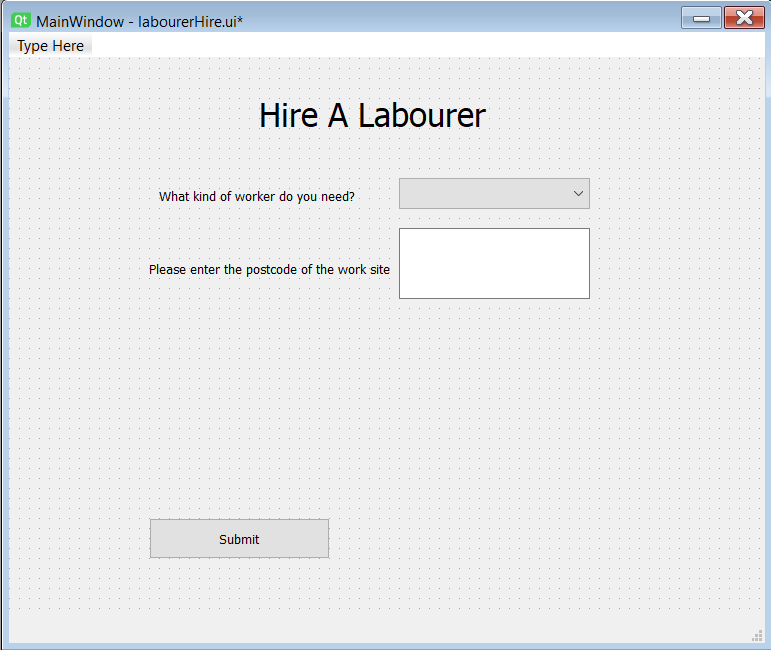
* The main menu needs to be clear so as to not confuse people trying to use the program
* This is important as most people who come to Kieran aren’t incredibly adept with computers, hence it is important to make the process as simple as possible
* The main menu therefore also needs clear instructions to direct the clients
* Utilising intuitive and commonly used items such as push buttons will make it easier for clients to become familiar with the software



Figure

**Customer Hire**

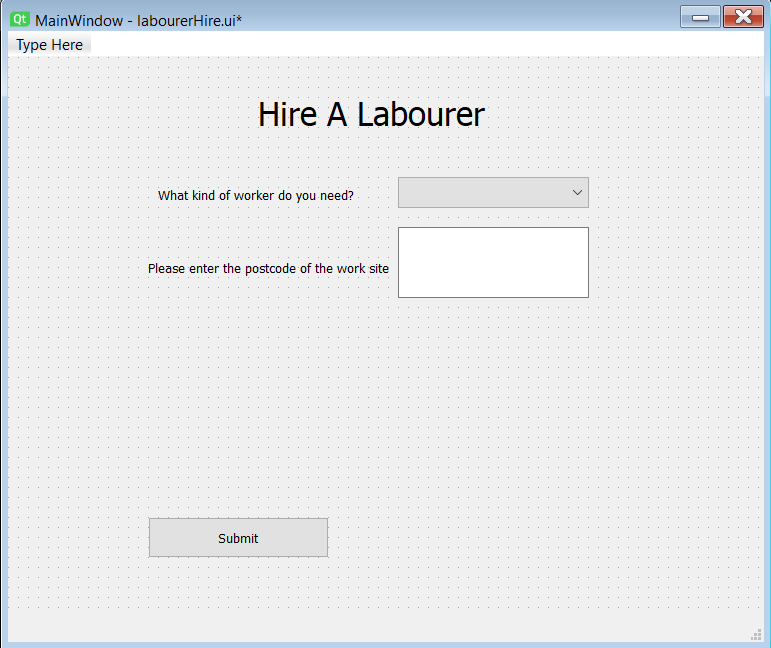
* Here, the customers need to enter the worker they require and the address of their work site
* This will be standardized using REGEX to ensure valid inputs are entered
* The kind of workers are limited, so a drop down menu will be used
* This ensures there is a lower chance of an SQL injection attack being successful
* This will then have a submit button to enter the details and check them against a database
* If valid workers are found, then the next stage will appear, where customers have to confirm a worker



Figure

**Customer Hire Continued.**

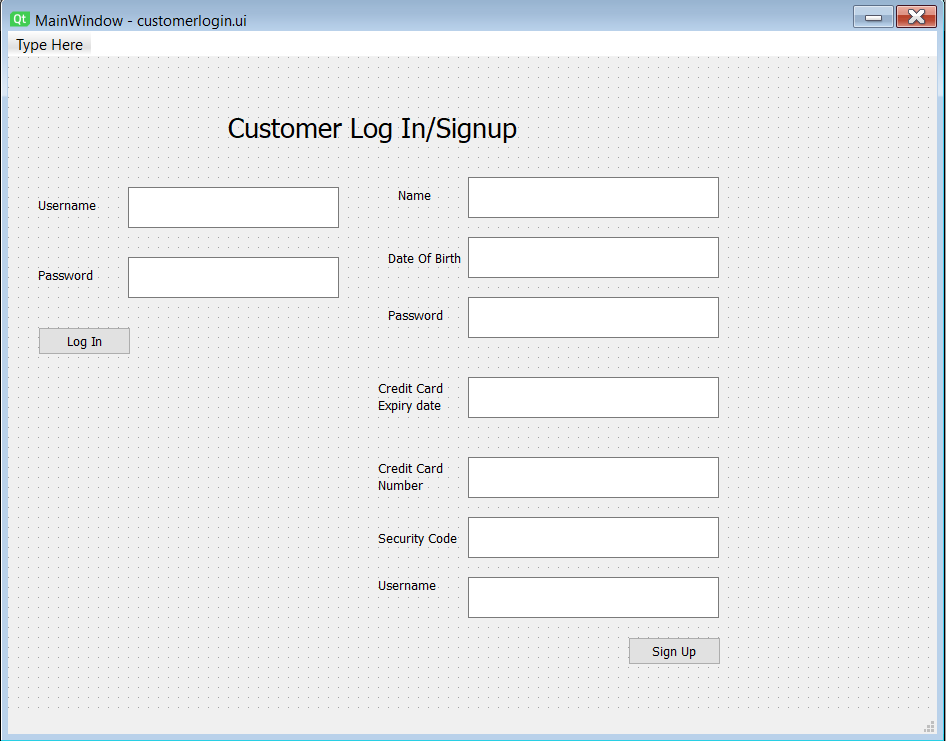
* Customers will have the option to hire a worker here after being presented with a worker’s name and occupation
* Next, a customer will need to either log in or create an account after confirming they wish to hire this worker
* The worker’s ID will be saved, and the customer will enter the relevant information depending on whether they are creating an account or logging in
* Switching between screens will be done using the .show() and .hide() functions built in to PyQt 5
* The password entered here needs to be hashed to make sure it cannot be read by anyone storing the information



Figure

By instead opting to enter a postcode, the REGEX I have to do is much more limited, as I now do not need to handle validating and extracting information from the various addressing formats there are. This also helps massively to avoid SQL injection attacks as I can much more easily validate the query.

The submit button will simply cause a label to appear detailing the worker on the screen, upon which clicking the submit button again will cause the user to shift to the next customer login or signup page.



Figure

Here, I have opted to include sensitive information, such as passwords and credit card details. It is hence vital that these data pieces are held with extreme care, appropriate to what is required under the Data Protection Act. In this case, it means the data needs to be properly protected from Kieran’s eyes, as all he needs to do is verify it is correct. This can be done with a simple hashing algorithm, which will place the hashed values in the database, making harvesting passwords impossible.

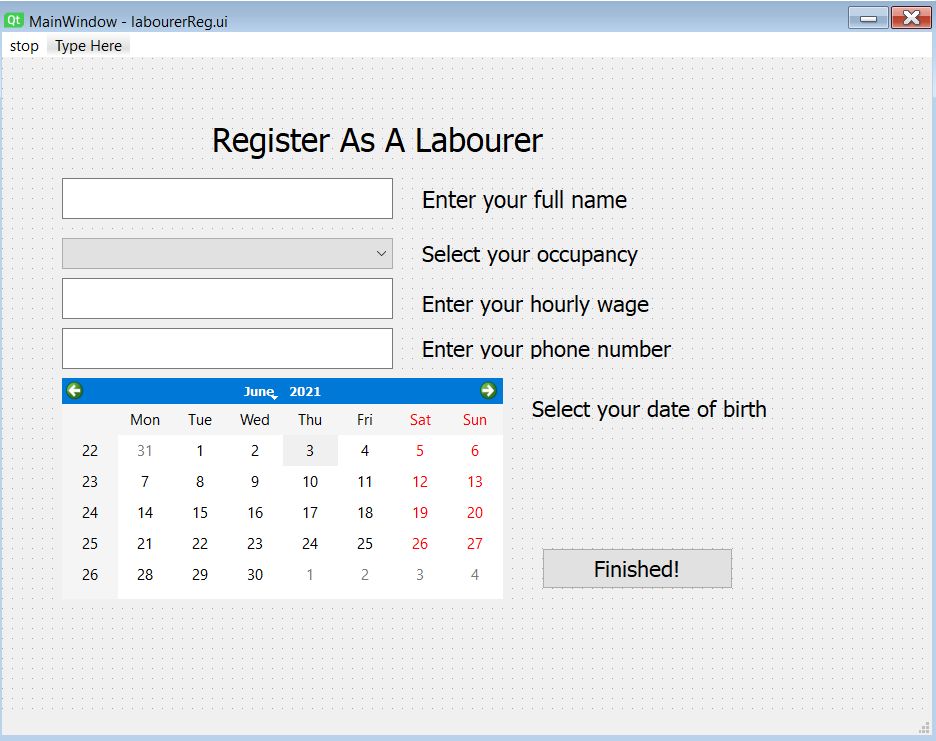
Notably, I haven’t used an age section but instead a date of birth. This is important as age can change, which can cause large problems in the program if this isn’t accounted for. Kieran will be using this for some time most likely, so it is important this doesn’t happen. Age can be calculated easily by calculating the difference in days and using the DIV function in python.

Diagram

Description automatically generated

Figure 5

**Labourer Registration**



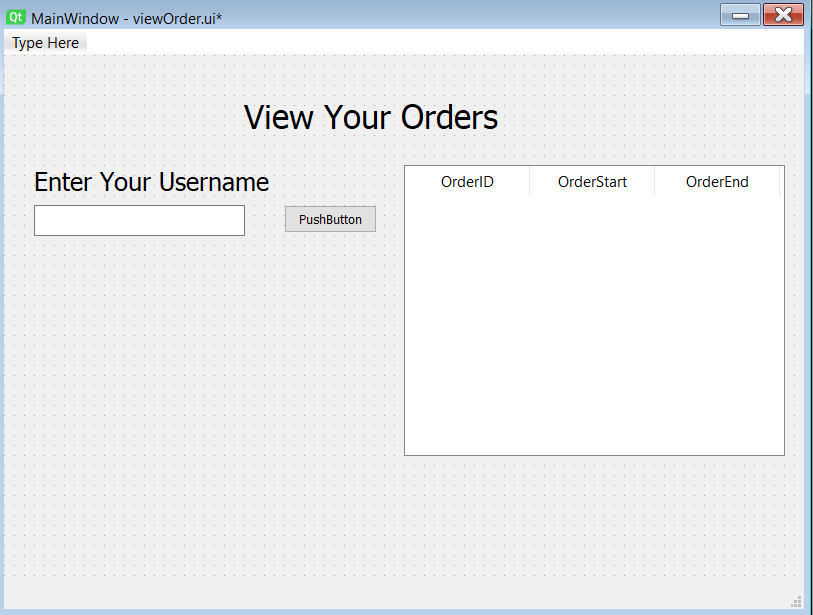
Figure

Here, I have decided to use the calendar widget, as there was a sufficient amount of space. However, that was not available on the last menu, meaning I likely would not have been able to implement a good utilisation of the calendar widget. I could bypass this by creating separate login and signup pages, but this would require larger amounts of code to be written to determine which page needs to be switched to, as well as classes for each respective screen.

In addition, this would need much more space as new UI files would be needed.

**View Orders**

* This section will first check whether you are a customer or a labourer by selecting a button to click
* After this, you will be taken to a different screen where your username is input
* Your orders will then appear in a table to the right



Figure

**SQL Functions**

SQL will be used widely in this project, as Kieran will be storing data inside of a database. In order to extract this information in and out of the database easily and effectively, SQL needs to be used. My SQL version of choice is SQLite, because it is a self-contained version of SQL, meaning it only needs one computer to run, and it comes with a database browser, allowing for easy reading of the database when needed by Kieran.

However, the program is being written in Python. As such, a library needs to be used to execute the SQL commands and interact with the SQLite API. Hence, there needs to be functions which get and retrieve important details from the database.

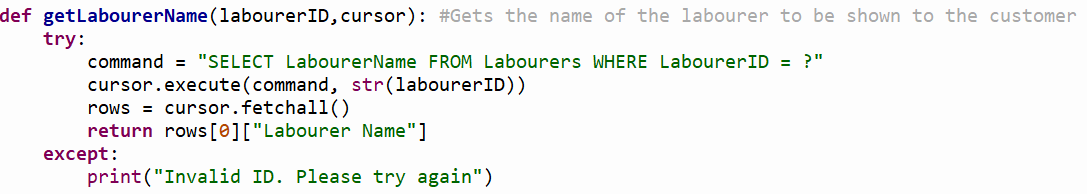


Figure 8

The above is an example of a getter procedure, which has a common structure throughout the program in the form as shown above. These getters are highly versatile, meaning they can be re-used many times, reducing code redundancy.

In addition, I have used a try, except clause here which catches exceptions to stop the program from crashing. Crashes in the program could occur if its used in unexpected ways by the client, so it is important to avoid these crashes as it presents badly in front of clients.

The reason I can call this method a getter is due to the fact that the cursor passed in earlier is an object, which can have code executed on it to fulfil any role needed. In this case, it retrieves a piece of data, making it a getter method.

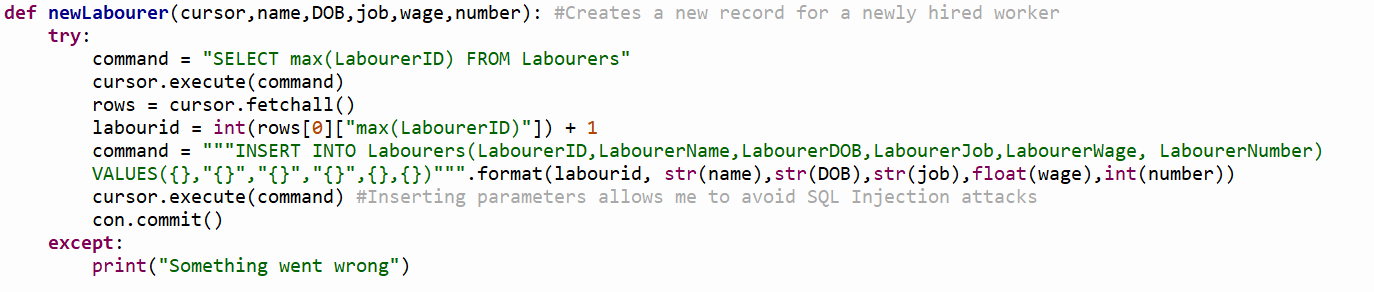


Figure 9

This is an example of a setter method for the reasons listed above. The try and except loop is there for the same reasons.

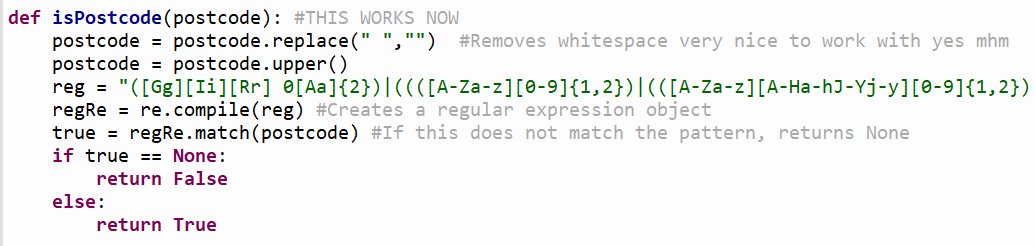
In this case, I have used the SQL command INSERT, which creates a new record in a table. Values specifies what is being inserted, and the part before specifies the table and its fields.

This function, however, has no in built validation checks. While it may be more worthwhile to implement these checks in to the function, it will likely lead to code redundancy as code is re-used many times, where it could have simply been its own function from the start.

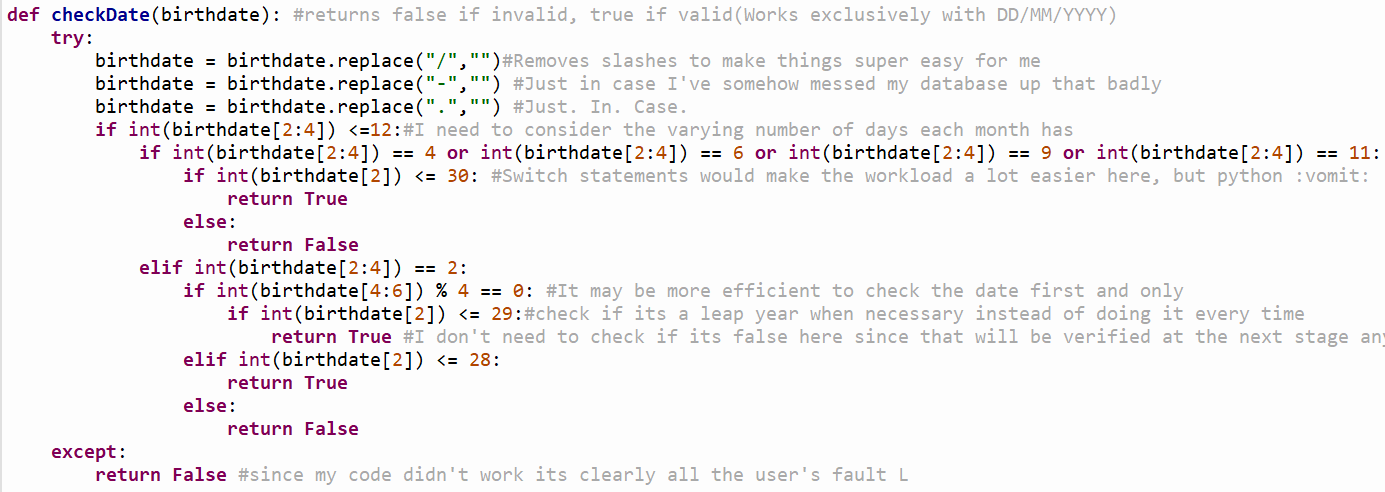
**Validation Functions**

Input validations is very important in this program, as we are dealing with sensitive information and SQL code, which is prone to SQL injection attacks if not guarded against. SQL injection attacks are when malicious SQL code is input instead of normal user input, causing the program to execute the code. This is very dangerous, as it could lead to unknown parties being able to read the data, violating a key principle of the Data Protection Act – data must be stored securely.

One way of avoiding these attacks is by using validation of the user’s input, checking that it is in the form we requested. As such, one way this can be done is by the use of a regular expression. In the case of postcode validation, the UK government provides an official REGEX expression to check the input with to check whether it is a postcode.

****

Figure

****

The other check function I will need to write is the check date function. I will just need to check it is valid, and is easily done with some check statements; there is no need for a regular expression here.

**Program Flow Diagrams**

Diagram

Description automatically generated

Figure 11

**Test Plan**

My test plan is to use an iterative development solution, checking the code functions as intended as I go through the development of the program. As such, there will be instances where code does not work and I need to go back and change it, modifying it as I go through.

At the end, I plan to do a final test of the full project in its entirety, checking to see if all criteria specified at the start of the development process have been met. If this is the case, the program can be called a success.

**Test Schedule**

* The first part to be tested will be the SQL functions, as they will be the first to be written. By checking these work as intended, I can ensure that the building blocks of the backend will work as intended, and can be implemented as I wish in to the program.
* Next, I will need to test the validation functions, to ensure data is properly handled by the function and returns the correct values. The program needs to be able to validate the input so as to avoid the SQL injection attacks, so it is important this is tested after the SQL functions are tested to ensure they work as intended.
* After, I will test the main menu GUI to ensure the GUI module works as intended, and push buttons are detected as intended. This is important as if these simple functions and the GUI don’t work as intended the backend of the program will not be able to run as wished.
* Next, I will test the switching mechanisms in the GUI to check they work as intended. This is important as there will be a considerable amount of switching between windows as the program runs, meaning this needs to be done efficiently and easily.
* After the functions have been implemented, I will test that all the functions come together to work as intended, entering code in the database as planned.
* After this, I will test the heuristics function, which will sort the closest worker to the construction site. This is important to test as it will function fine as a stand alone function but will interact in a significant way with the GUI modules, and so can safely be tested last, allowing me to adjust the complexity as needed in addition to this.

**Varieties of Test Data**

As I test these functions, it will be done with varying sets of test data with certain characteristics which each present their own challenge to the program. The program will need to handle each piece of data in an appropriate manner to ensure the program always runs smoothly.

**Normal Data**

This is data that is entered on normal use by the client. This sort of data assumes the client will never enter anything which isn’t as expected by the user. Hence, the data should be processed as intended by the creator, allowing the client to easily move on to the next step.

**Boundary Data**

Boundary data is data that lies on the boundary between being valid and invalid. An example of this may be entering a birthday on the 29th of February, to see if the program handles it as expected.

**Error Data**

This is data that is invalid and should not be processed by the program. Examples of this may be SQL code that the user is trying to inject. The program should detect this is invalid and return the fact this is invalid data.

**Null Data**

This is the lack of data in a response. The program should return the data back to the client and ask for them to enter once again. If there is no check, the program may unknowingly accept this data, so checks for this must be implemented.

**Development Log**

19th Oct 2019 – Started work on a library program. The program consists of a 2D array with the name of the book and whether it is available to lend inside of each individual array. Functions written so far allow for one to take out and return a book to the library, changing a boolean inside of the array. I plan to add a function to add a new book to the library.

21st Oct 2019 – New function added which allows one to add a new book to the library. This is accomplished by appending a newly created array in to the 2D array. My next plan is to add date/time validation to the program, changing each array to have a “next in by” date, i.e. the date at which the book will be returned. This date is compared to the current date to see whether the book is available, and will ask how long the user needs it for and therefore when the book will be expected to be back in the library.

4th November 2019 – I have managed to retrieve the current date/time from python and been able to compare it to another using the substring function to compare dates. Next, I plan to alter the 2D array to accommodate dates instead of bools.

17th November 2019 – The 2D array now accommodates dates instead of booleans. My next plan is to use an SQL flat file database instead of a 2D array. To do this I will use SQLite and directly transfer my layout for a 2D array on to the flat file. In order to adapt my program, I will need to write functions that can read and write from an SQL database which will likely take much longer to do than previous iterations of the program.

13th February 2020 – I have not worked on coursework for a long time since last year, as I have been busy with exams and preparation for them. I have a basic structure for my program’s database mocked out in the SQL database editor I have made, as follows as shown in figure 1.

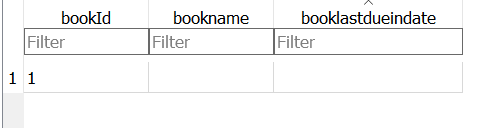


Figure 1

1st March 2020 – Lockdown seems imminent due to coronavirus, which presents a large risk to the progression of my coursework. I have chosen to reform my database to one which suits a database of labourers or handymen – a handyman can sign up to the service, listing their profession for clients to easily find and access and hire them. As such, each handyman will require a separate listing in the database, as will each customer. I have created a new database which is now fully relational and in 3rd normal form to make it as efficient as possible, presented below.

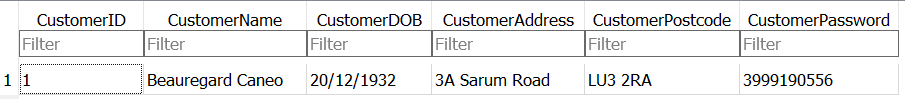


Figure 2

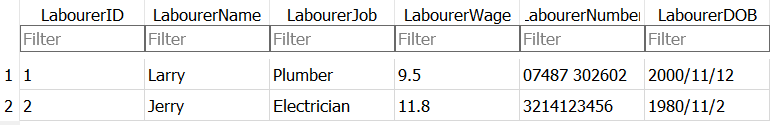


Figure 3

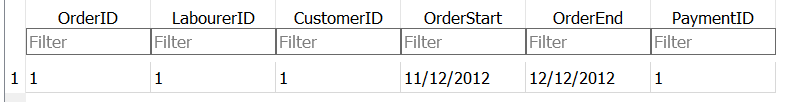


Figure 4

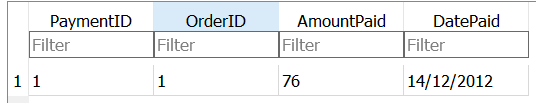


Figure 5

Customer Passwords are not stored as plaintext inside of the database; the password shown is a hashed version of the true password. I have yet to figure out what I will use for the hashing algorithm, so for now I’ve randomly generated a number to put in as a placeholder. I will likely need to use some ascii character code conversions to make it work.

Customer and Labourer tables have a many to many relationship, which I have bridged using the orders table as shown. In order to track payments I have implemented a payment table. While at first it may appear to not meet requirements for 3rd normal form, it is important to realise that not all of an order’s required payment will necessarily be paid all at once, and hence I have had to make it possible for multiple payments to assign to one order placed.

I have also denoted any fields where a primary or foreign key has been used with a 1.

2nd August 2020 – I have not done much work on my coursework in a long time. As such, I have realised that it is important I continue promptly to allow myself ample time to prepare for A Level exams when they come around next year.

I have made functions which allow me to read from my SQL database, which all follow the same template.

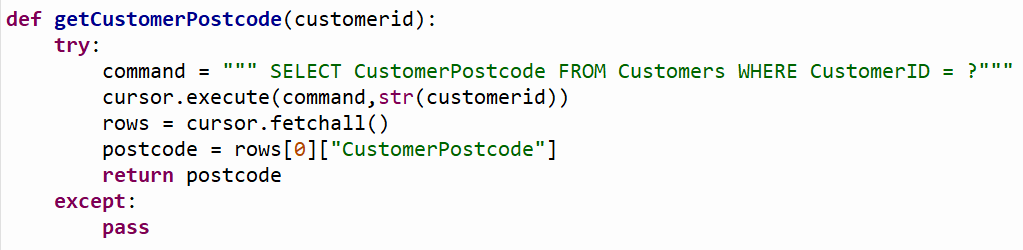


Figure 6

Initially, I had an issue as I replaced the question mark with an exclamation mark, which treated it as a wildcard and attempted to find anything like what I entered, which was blank. As a result, it threw an exception. I replaced it with a question mark, which allowed me to pass the customer id in to the statement easily.

All of the functions written follow this structure. However, in order for it to work, it is notable that the “cursor.fetchall()” function returns an array filled with dictionaries. This is done with the following function:

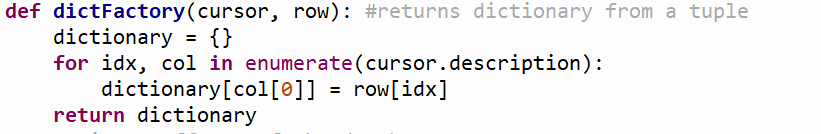


Figure 7

This returns a single dictionary. However, when the following function is run:

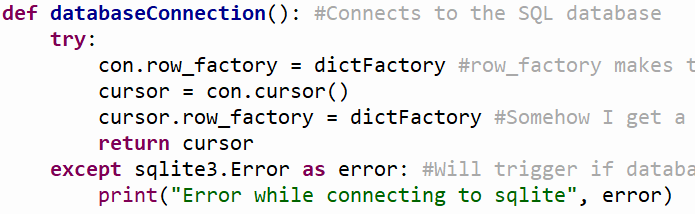


Figure 8

This is repeated for all instances inside of the database, giving the array of dictionaries as shown before.

The reason I have done this is to make it far easier to edit the contents of the dictionary if needed, as a tuple cannot be edited, and sometimes cannot be read either. This method makes it far easier to create functions that work with the contents of the database.

18th September 2020 – I have made an error with my program, as I neglected to realise I have called on the cursor variable which has only been defined in the local scope of the databaseConnection function, but has not been passed in to the new functions. As a result, the program will not work as the cursor will throw an object not defined error. I have amended the function as shown below:

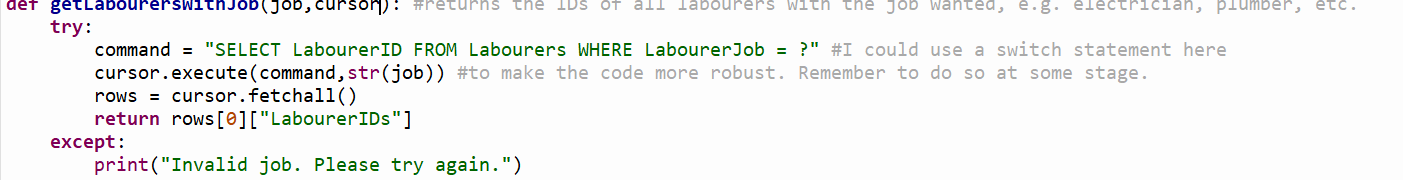


Figure 9

Try and except occasionally makes it rather difficult to debug code without removing them as well. During testing, I would simply run the program without calling the function and assume it was working. I realised this would not work as the program would assume the try and except loop worked with another piece of code to tell whether it was supposed to run something when that was not the case.

I found using the debugging tools built in to my IDE also provided a lot of help, especially with testing and finding issues with my code. Stepping through the code made it very easy without needing to go through the troubles of drawing out a trace table for every function, as well as see what my variables actually look like as I go through.

1st October – Began work on functions that will add to the database instead of just read from it. This provides a unique challenge, as the code is more complex due to needing to find the next needed ID for the table, as well as validate each input to prevent SQL injection.

Currently, my plan to prevent injection is to insert the values using the VALUES SQL function, which will forego any processing of the input itself, in a manner shown below:

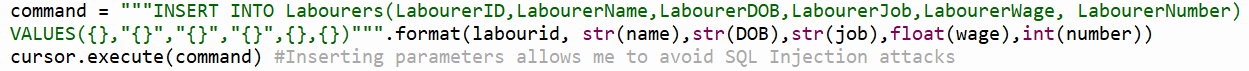
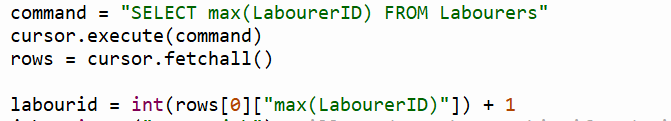


Figure 10

It is important to note that not having quotation marks around curly brackets which will store a string will cause the code to break.

20th October – More work done on a general function which adds to the database. I have done the following to find the max ID and find which ID needs to be found next:



The dictionary actually returns the key by the command I ran to get it, so for the retrieval function I have to call like so in order to get the ID.

4th November – Finished general writing function. Shown below.

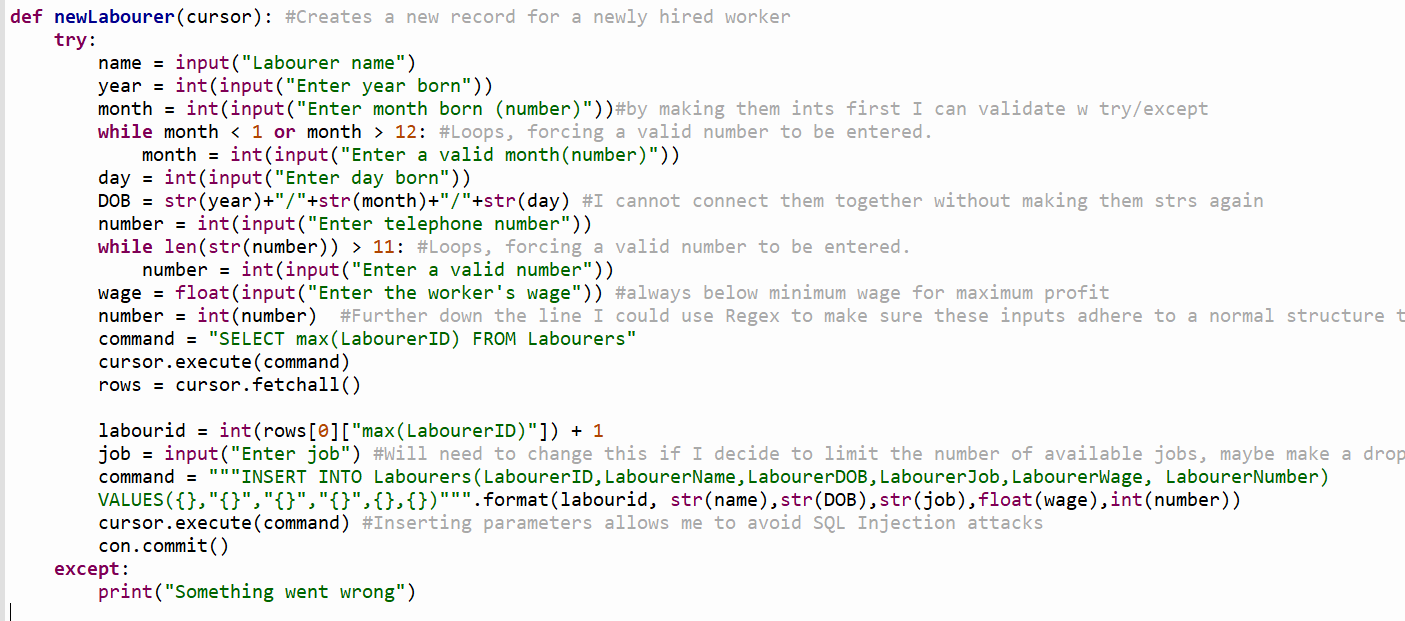


Figure 11

I repeated this structure for each of my tables to give a total of 4 writing functions. When I create my GUI I will likely need to make more functions which are made up of these functions, as well as parameterise these functions to suit the GUI format. This will make my time spent coding the GUI much shorter since I have much less which actually needs to be written up.

17th November – Started working on validation functions for dates as well as postcodes using REGEX. Understanding the syntax of regex is difficult but luckily the UK government provides a regular expression for programmers to use expressively for this purpose. However, it is important to know that the regular expression provided accommodates for any possible postcode, and the postcode does not necessarily need to exist for it to be used. As a result, I have considered using google waypoints API to counter this issue. However, I will look more in detail into this solution later as my current priority is finishing a decent GUI in good time.

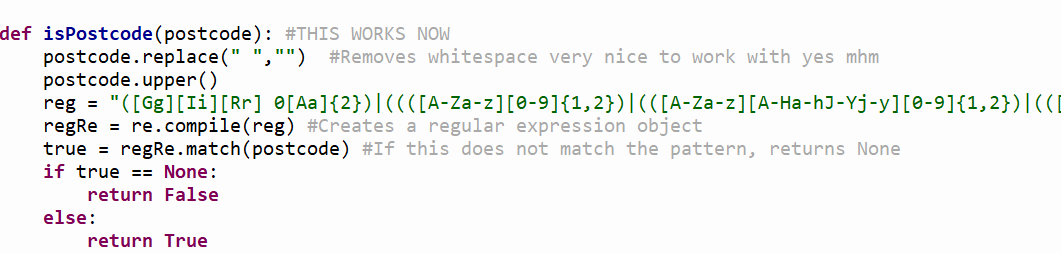


Figure 12

The full expression is as follows:

([Gg][Ii][Rr] 0[Aa]{2})|((([A-Za-z][0-9]{1,2})|(([A-Za-z][A-Ha-hJ-Yj-y][0-9]{1,2})|(([A-Za-z][0-9][A-Za-z])|([A-Za-z][A-Ha-hJ-Yj-y][0-9][A-Za-z]?))))\s?[0-9][A-Za-z]{2})

20th December – Finished a checkDate function which checks if the date entered is valid. Shown below.

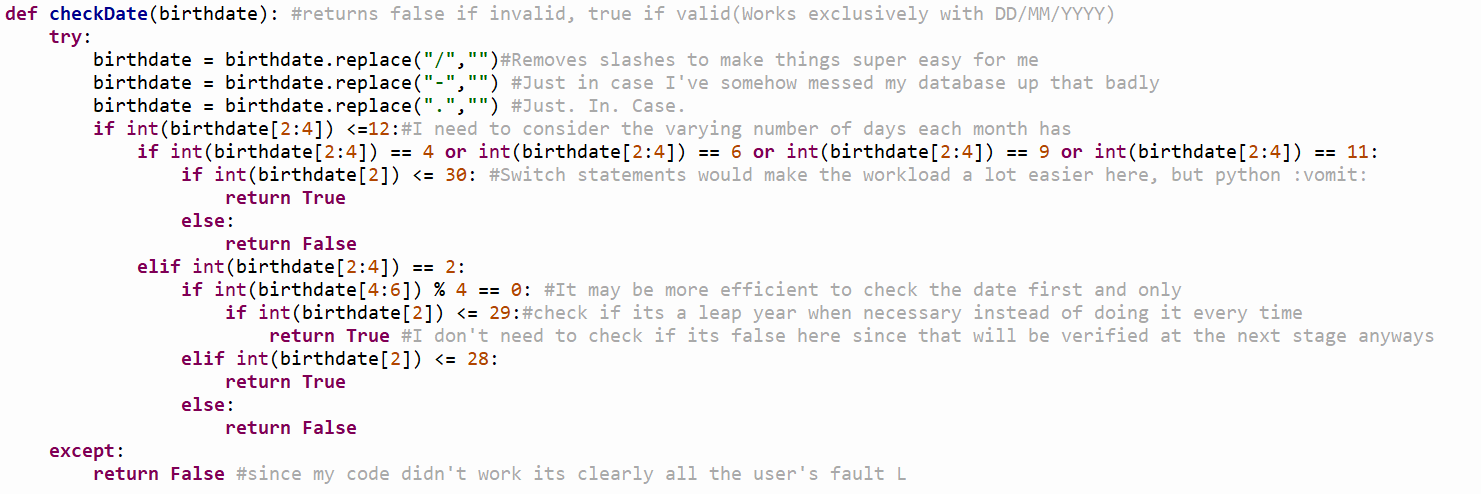


Figure 13

My database stores dates in the form of “DD/MM/YYYY” so ideally the date would not have hyphens or dots between the date, month and year. However, if I mess up along the way or if a user forcibly enters data directly in to the database, this will mean the date will still be considered valid.

I have made considerable use of the substring function provided by python, as items in the string can be called by index and length. This allows me to check specifically the date, month and year.

When considering the issue, it becomes apparent that different months have different numbers of days, and will hence need the month to be checked first. I have a set number of possible outcomes which are conveniently labelled from 1 to 12, which would be perfect for a switch statement. However, this isn’t possible in python, so I have been forced to simply use an if statement to catch all the possibilities.

27th December – I have created a password hashing function which is currently very primitive.

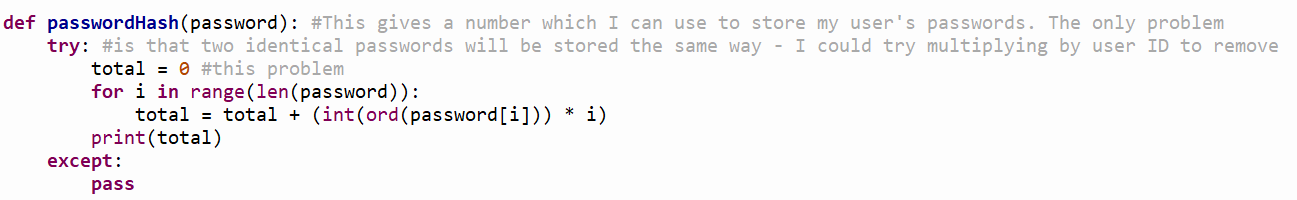


Figure 14

Print(total) would normally be a return statement but I have kept it that way for testing purposes.

The biggest problem with this is that 2 identical passwords will be stored in exactly the same way, meaning if someone gained access to the database, it would be possible for them to find out a user’s password with enough time which is much less than ideal. The solution I proposed in my code comments, multiplying by user id, would mean that it would be harder for the hacker to access passwords as he would need to figure out that he needs to divide each password by the user’s user ID first as well.

7th January – Began working on my GUI. Now my SQL is mostly finished, I can start drawing out what I want to make for my GUI and begin working on it. For now, I have decided to start out on working on showing and hiding a basic GUI, and being able to output a string when a button is clicked.

I have decided to use PyQt5 for this, as the Qt framework is well established, meaning it will be easier for me to get help with any large issues I encounter while using it.

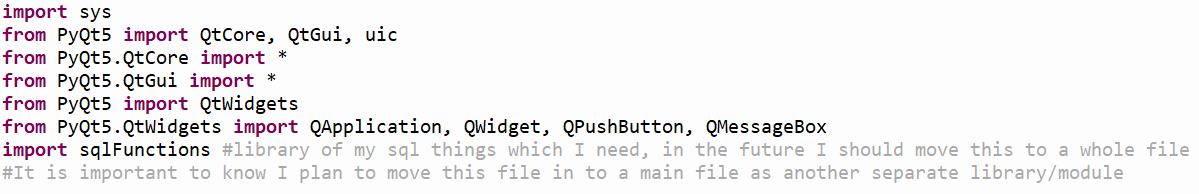


Figure 15

This code is used to import all the parts of PyQt5 which I will be using in my program.

Next, I need to initialise my window objects.

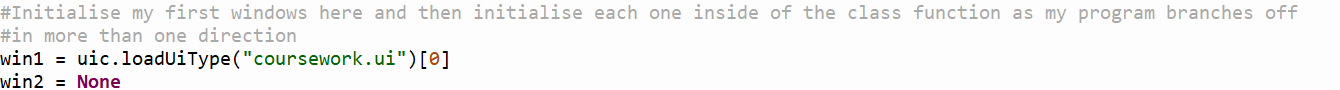


Figure 16

By initialising all the windows at the start of my program, I can simply show and hide each window as needed.

14th January – Made a function to initialise my first screens. Code is shown below.

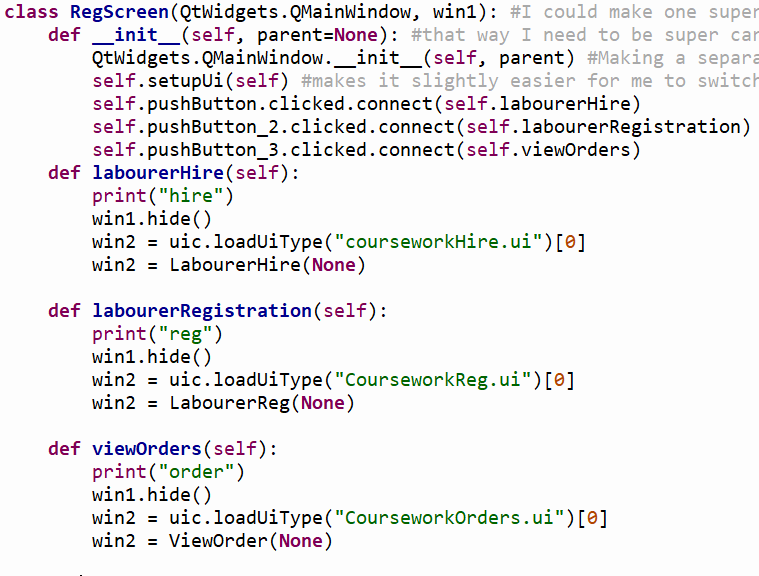


Figure 17

RegScreen is a class I will initialise at the start of the program. I will create classes for each respective window object I create, which will be able to perform functions accordingly. In order to go back and forth between different forms, I will implement the use of a stack which will store the windows being used in order. Each time the back button is pressed an item is popped off the stack, and each time a form is changed to the old form is added to the stack.

Each method I have specified inside of this class I plan to initialise in to another class.

As such, I will create a class for viewOrder, LabourerReg and LabourerHire.

2nd February – Created my first GUIs using PyQt’s designer. The designer allows me to create UI files which allow me to specify object names from within the file, and I can perform in built functions with the object’s attributes using the imported libraries provided by the PyQt5 library.

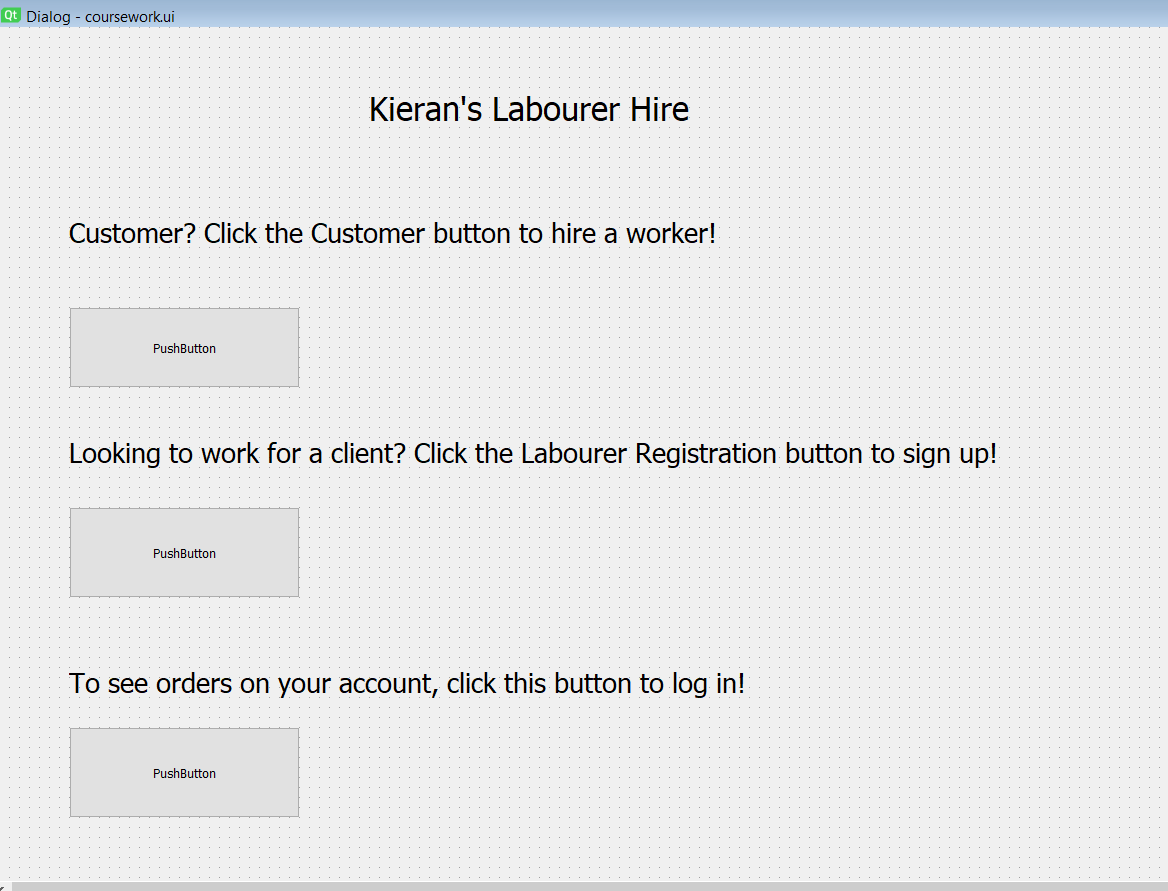


Figure 18

Clicking each separate button runs each method, by running an event whenever the buttons are clicked. When this event occurs, I cause a method to run which launches the next window I require.

10th February – I have had some issues running my SQL functions. I believe I will need to test and reconsider the template I used previously, as it has failed in the case of the following function:

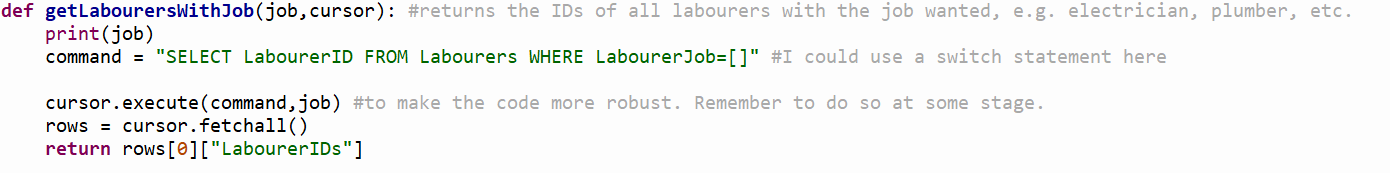


Figure 19

Currently, the job parameter being passed in is a string. However, the program throws an error, saying there is no column with no name. I do not know why, as no empty column names have been specified in the program.

I have as such tested the SQL code from inside the database browser. The code works perfectly fine without square brackets, but throws the same issue with them, so I believe the issue lies with the usage of the square brackets as a place holder value in side of the command itself.

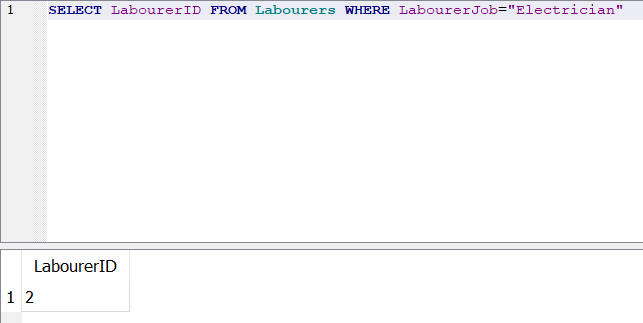


Figure 20



Figure 21

From this, I have realised the only reasons why this seemed to work perfectly fine previously was because my IDs are only single digits.

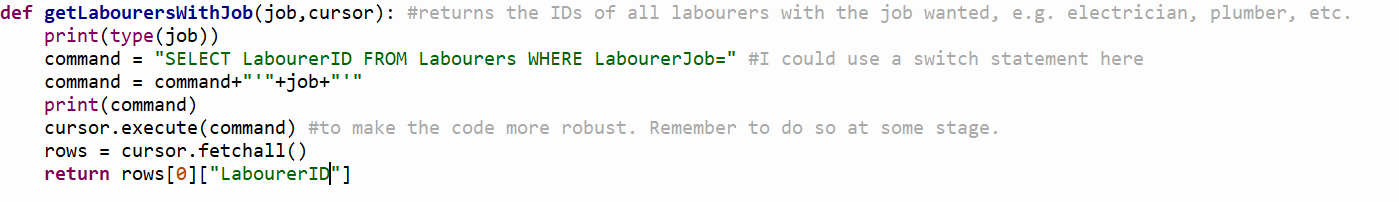


Figure 22

This solution I have used takes advantage of the fact that I know users cannot enter their own code in to the program, meaning I no longer need to consider the possibility of an SQL attack. Instead, this allows me to simply append the results from the combo box to the end of my statement, and avoids me having to make a separate SQL function for every possible combo box option.

I have made the combo box inside of the designer as such:

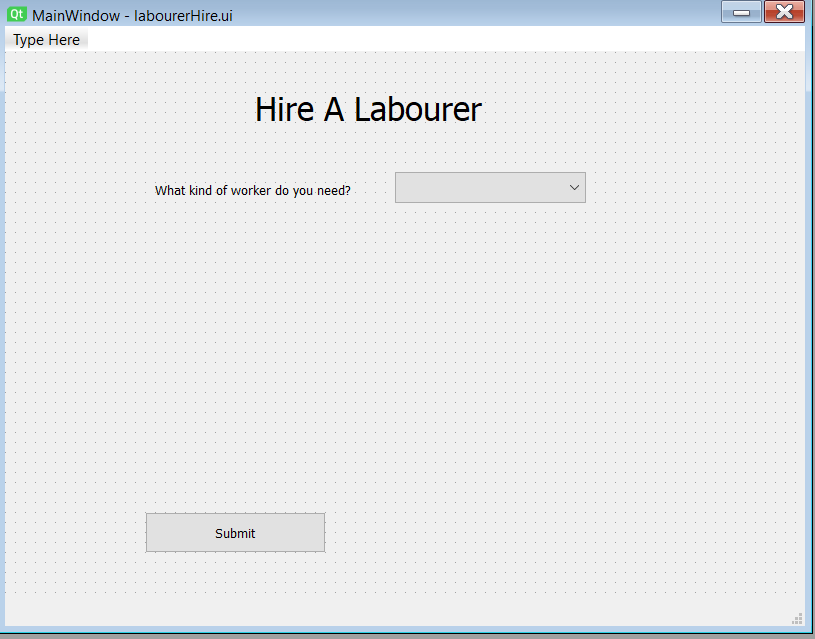


Figure 23

Right now, I do not have much I need to consider as far as choosing my workers goes. Workers can potentially just be suggested at random to the customer, but there is a lot I could do with this in the future, such as a graph colouring algorithm, usage of Google’s waypoints API to choose workers who are close by, and perhaps even looking at a worker’s previous hiring history to evenly distribute jobs.

The bigger problem at hand currently, however, is modifying my SQL functions to allow them to accommodate for cases where more than one character is passed in to the function. I will not tackle the problem for now – I think it’d be more efficient to handle the bug when I write the code for the GUI options which utilise those functions.

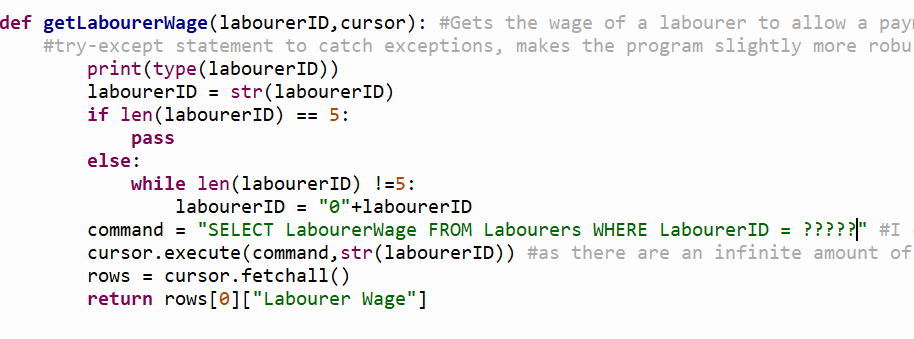


Figure 24

I attempted to implement a solution I suggested earlier, where a certain number of question marks would define how large my database could be. However, it seems that this isn’t possible, due to the way SQL handles the placeholder ? value. It treats each character as a separate entity when there are multiple in a row, rather than placing each character together to form a string.

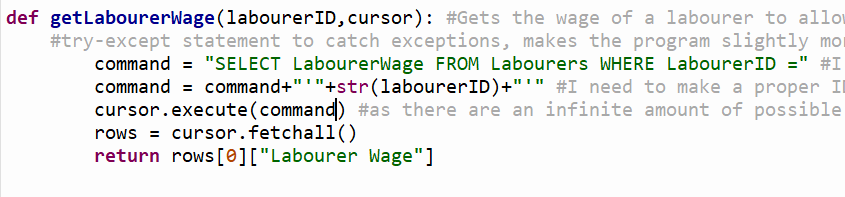


Figure 25

I have instead opted to use the same method as with the previous function. However, in order to keep using this, I will need to write a separate verification of ID function, so I know the user isn’t being naughty and trying to inject some code. I can probably do this with a simple attempted convert to INT attempt, and if it fails return false. I could also find the max ID and check if the INT falls within the range required from it.

16th February – Started working on a basis for my ViewOrders class.

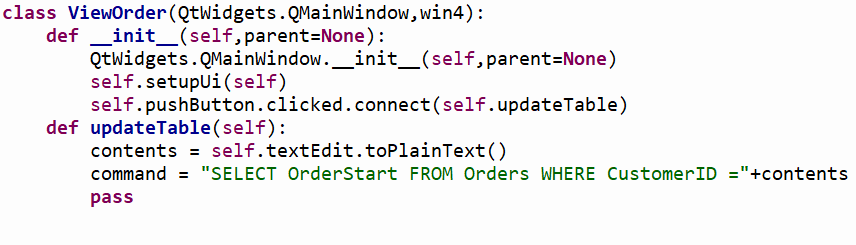


Figure 26

Initially, the command did not function as I needed it to. However, after applying a quick fix, this was remedied.



Figure 27

This meant everything was passed in to the SQL as one string, allowing it to run, whereas before the string contents was not placed as a string but instead as plaintext.

I implemented my table widget here. I plan to make this table widget much more developed in the future, however this class is certainly less essential than the ability to add and take orders out of the database. As such, I think I should focus on those aspects more in the future.

21st February – Did further work on my LabourerReg class. The first thing I need to do is to parameterize my newLabourer function inside of my sqlFunctions library. It now looks like this:

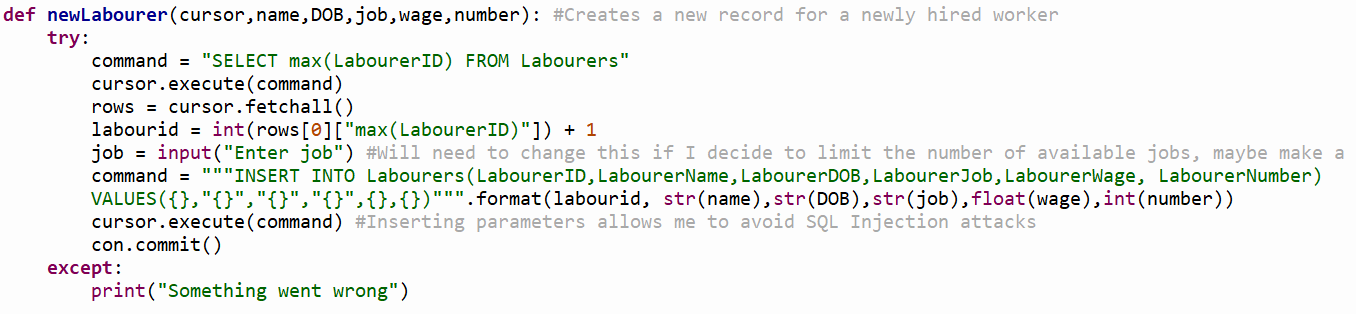


Figure 28

There was a lot stripped out from the function, but this can be easily fixed by simply using the checkDate function within my sqlFunctions library. Inside of my class’s method I will need to make sure this function returns a True Boolean value, as well as validate the rest of the input. In a sense, all I’m doing is moving the code from my library of SQL functions to my GUI library.

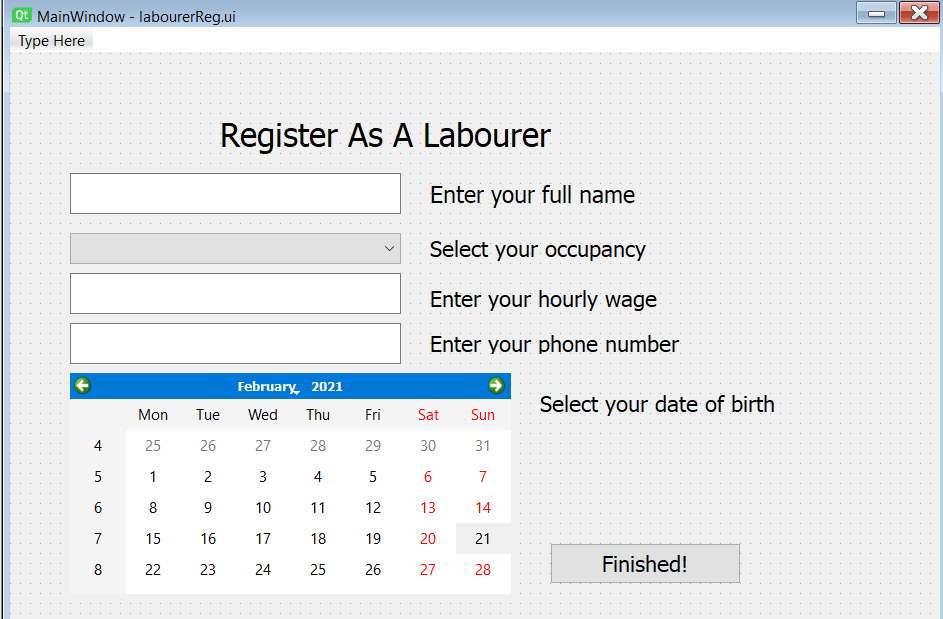


Figure 29

This is how my GUI now looks for the LabourerReg class.

While attempting to validate the inputted number from the GUI, I realised that my checkPostcode function didn’t work as intended.

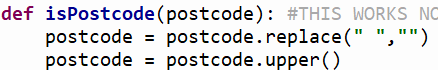
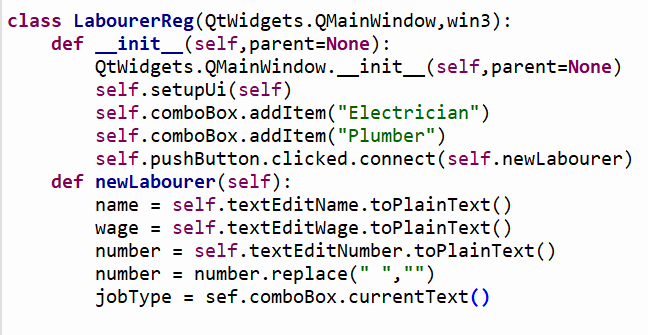


Figure 30

Previously, this function would run the replace and upper functions. However, I didn’t save these changes, meaning the changes would mean nothing and a valid postcode wouldn’t be returned as valid as the postcode would be in varying forms to the form I intended to standardize it in to.



This is what I’ve written so far in the LabourerReg class. I need to write something to extract the date selected from the calendar widget and I will be good to run it in to the function from my library.