**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 login system that:
   * Allows labourers to log in to an account.
   * Allows customers to log in to an account.
   * Contains account details for both labourers and customers.
   * Adequately stores these account details in line with the data protection act.
   * Provides a sign up option for labourers.
   * Provides a sign up option for customers.
   * Requires a customer to sign up before being able to hire a labourer.
   * Requires a labourer to sign up before being eligible for hiring by a customer.
   * Ensures the sign up process for customers collects all details needed to hire a labourer.
   * Ensures the sign up process for labourers collects all details needed to be hired out as a labourer.
2. Provides an information storage system that:
   * Stores information about labourers.
   * Stores information about customers.
   * Stores information about orders placed by customers.
   * Stores information about payments due and paid by customers to labourers.
   * Allows easy access to information relevant to a labourer when needed.
   * Allows easy access to information relevant to a customer when needed.
   * Groups data in tables, storing data relevant to a person in the same table.
   * Contains references to other tables to allow for easy access to information through foreign keys.
   * Is normalised to 3rd normal form for efficient use.
   * Runs smoothly while being scaled up regularly, and storing larger amounts of data.
3. Accounts must be able to:
   * View data immediately relevant to the account they are logged in to.
   * This includes hired workers, order start dates and end dates, and payments that are due.
   * Delete the account alongside all data related to the account holders on request.
   * Change the data on their account in the event of a change.
   * For example, a worker who moves houses needs to change their address, and should be able to do so.
   * Be accessed via a graphical user interface.
   * Be easy to access.
   * Be easy to change the data on.
   * Be easy to view data on.
   * Be easy to delete.
4. Data must be suitably protected:
   * According to the principles of the Data Protection Act, meaning:
     + Data must be processed fairly, lawfully and transparently.
     + Data cannot be used other than the purpose it has been collected for.
     + Must be limited in use as much as possible.
     + Must be accurate.
     + Must be kept for a reasonable amount of time.
     + Must be handled securely.
     + The holders are responsible for the data’s storage state.
   * Hashing algorithms should be used to store passwords securely.
   * Hashing algorithms should be used to store information such as credit card numbers and expiry dates.
   * Hashing algorithms should be impossible to reverse without finding millions of possibilities.
5. The program must contain a user interface that:
   * Is easy to read for clients.
   * Uses a large font to help those who are visually impaired.
   * Uses a simple colour scheme to avoid intimidating clients.
   * Is intuitive for those not well versed in computers, and perhaps haven’t used them as much as others.
   * Directs clients with clear and precise instructions.
   * Gives a sense of professionalism.
   * Uses commonly used constructs, such as push buttons and spin boxes, to direct customers easily.
   * Contains friendly language to help ease the user in to the program.
6. The database must be fully relational to allow for optimised performance among larger data sets. This includes adhering to the rules of database integrity, outlined by ACID:
   * Atomicity – Every entry in the database must contain one piece of data.
   * Consistency – Any changes to the database must leave the database in a valid state.
   * This means leaving the database’s changes either fully made or not made at all.
   * Isolation - Any commands performed on the database performed concurrently must have no difference if they are performed in either order, in series.
   * Durability – Any changes to the database must remain if the database loses power.
7. The database must contain:
   * Labourer:
     + Names
     + Date of birth
     + Address
     + Phone number
     + Username
     + Hashed password
     + Unique identifiers
   * Customer:
     + Names
     + Date of birth
     + Address
     + Phone number
     + Username
     + Hashed password
     + Unique Identifiers
   * Order:
     + Unique Identifiers
     + Start date
     + End date
     + Labourer ID
     + Customer ID
   * Payment:
     + Unique Identifiers
     + Order ID
     + Date paid
     + Amount paid
8. The program must have heuristics that:
   * Use the postcodes from the database of labourers
   * Evaluate the postcodes, ranking them in order of closest and farthest from the construction site
   * Allows the heuristics program to return a labourer ID of the labourer closest to the construction site.

**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

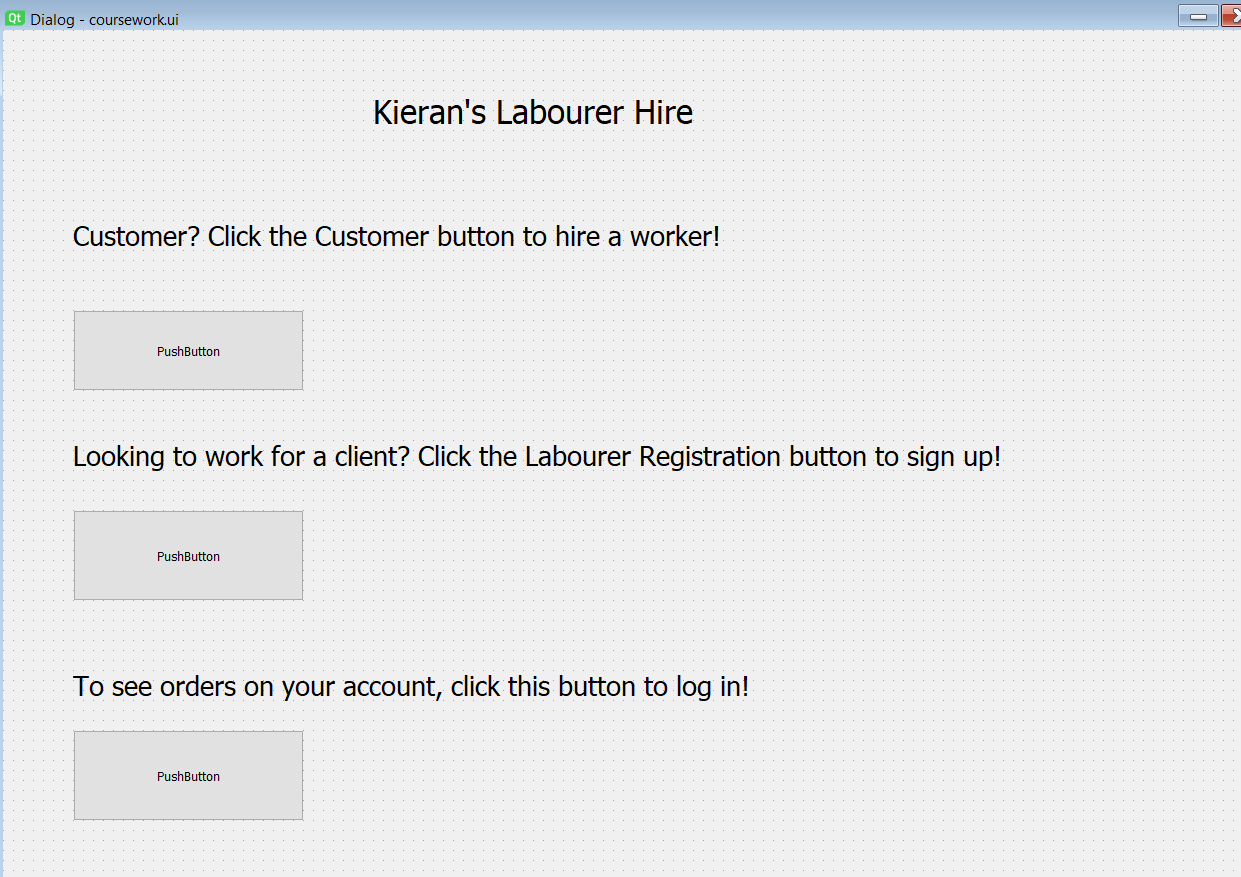


Figure 1

**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

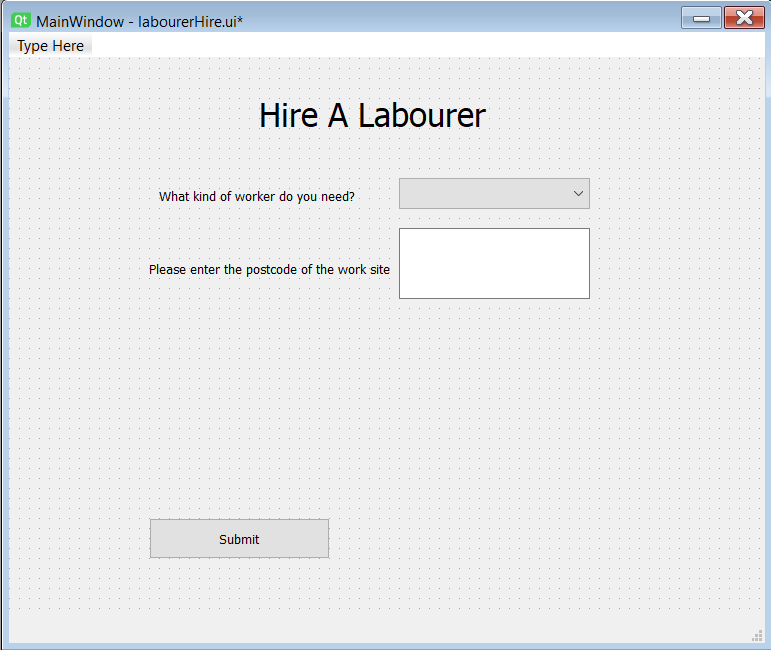


Figure 2

**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

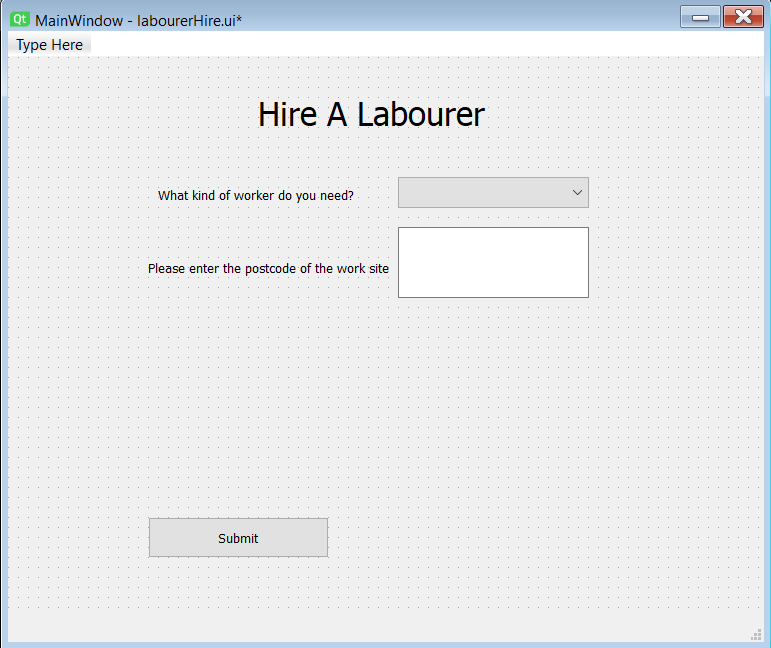


Figure 3

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.

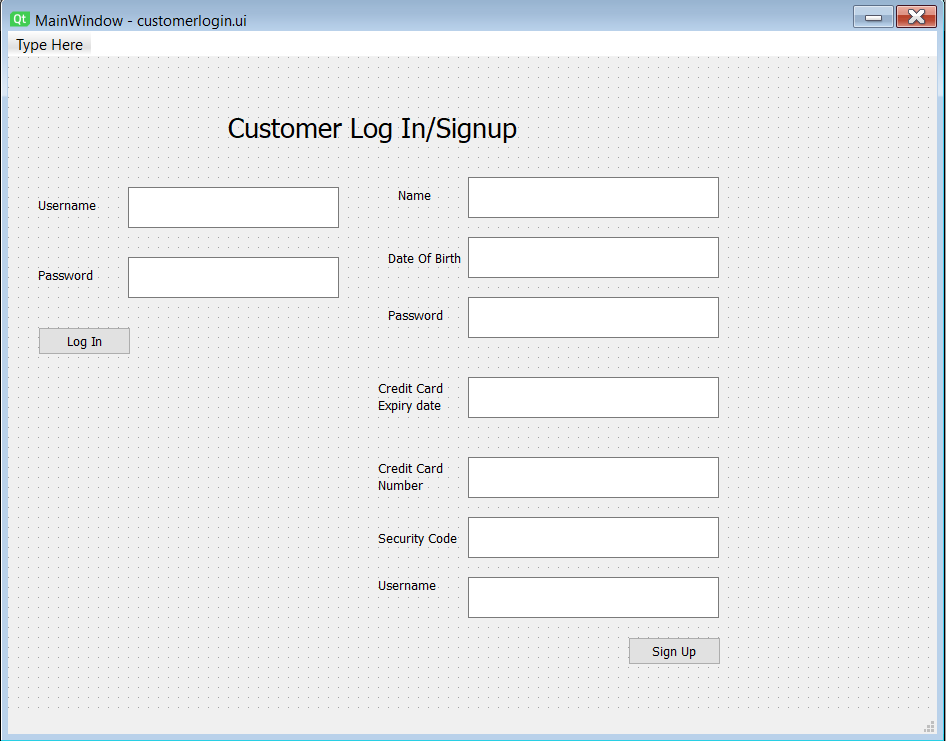


Figure 4

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**

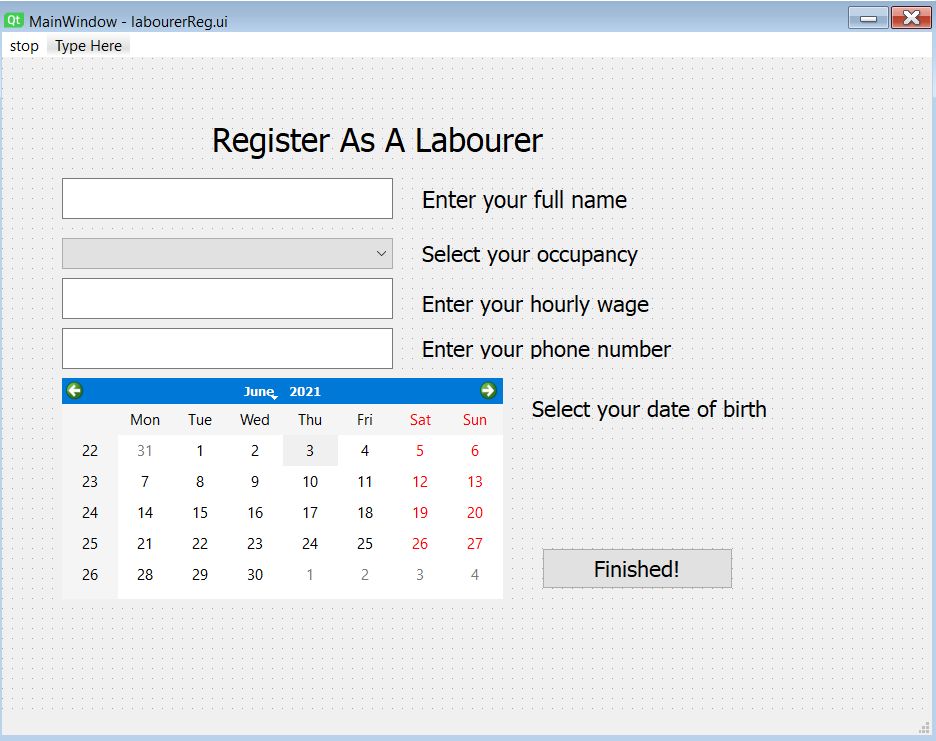


Figure 6

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

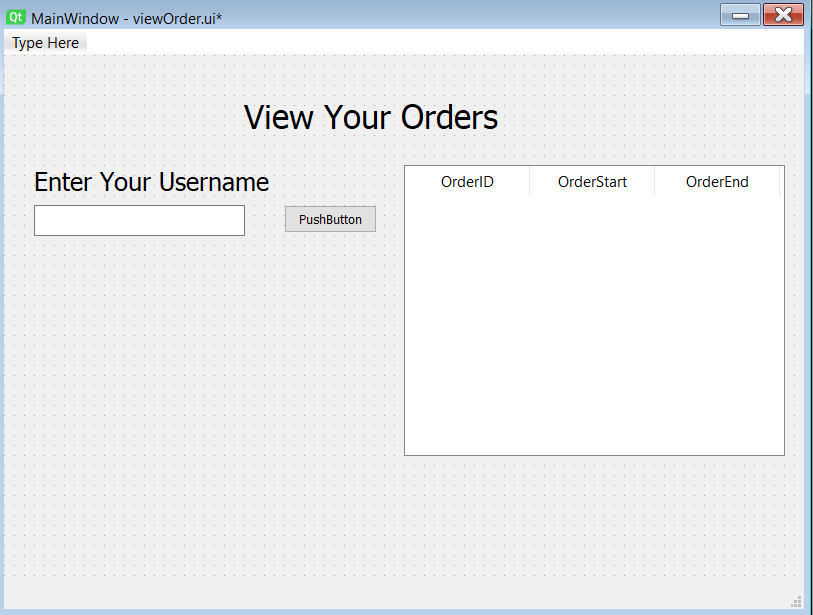


Figure 7

**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.

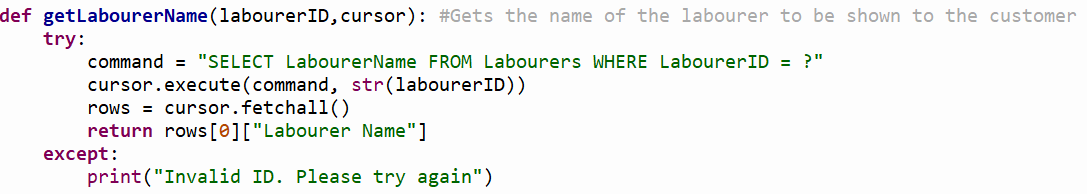


Figure 8

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.

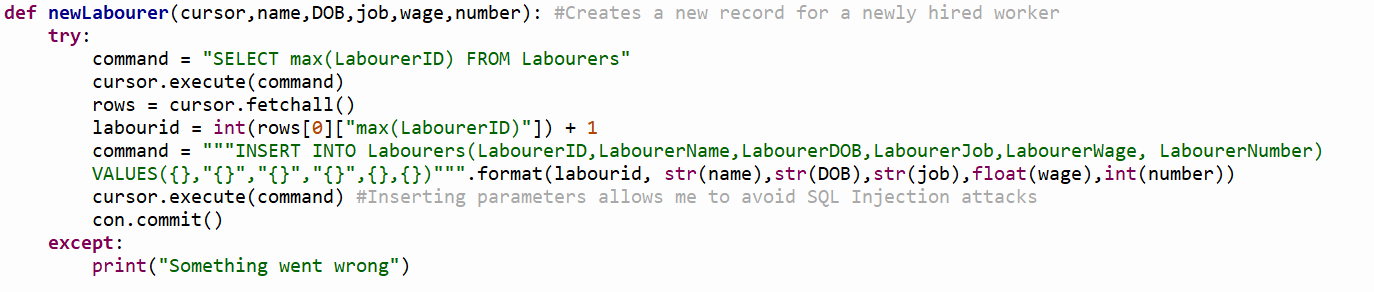


Figure 9

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.

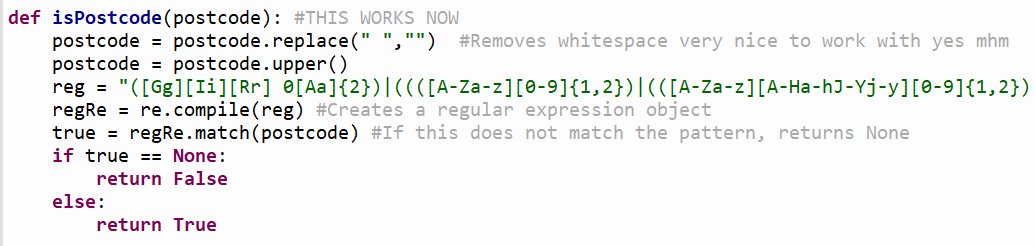
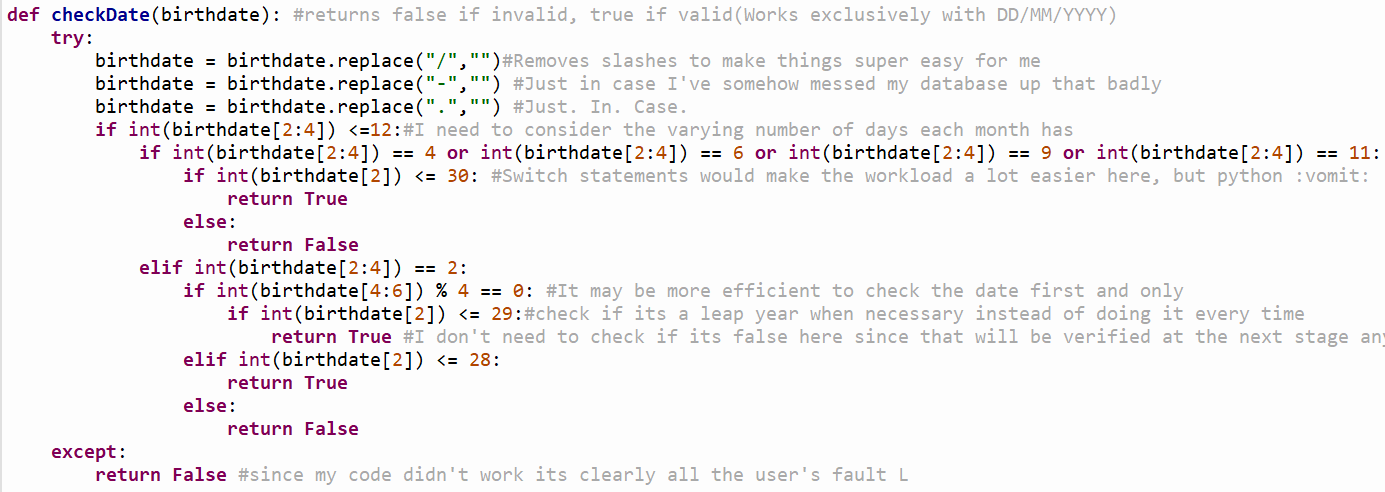
****

Figure 10

****

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**

**Database**

Databases make up a large portion of the program, so it is important to get them right. As such, I started my development with the databases.

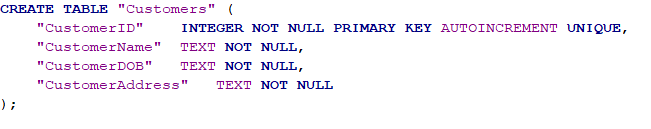


Figure 12

CREATE TABLE creates a table called customers, which has been given 4 columns; CustomerID, CustomerName, CustomerDOB, and CustomerAddress. All of these have been set to Not Null, meaning any empty entries in to the database will return an error back to the python program. Each of these columns have been given other requirements, such as specifying the data type. SQL has 5 data types – 1 more than most other programming languages, the date data type, which is, as deduced from the name, a date.

The other requirements are listed by the CustomerID column, being Primary Key, Auto Increment, and Unique.

Primary key specifies this column as the primary key column – this means that all the attributes in the table are determined by this attribute.

Auto increment automatically makes the field in the database one larger than the previous entry. For example, if the last CustomerID was 1, the database would automatically update the database’s next record to 2.

Unique means that no two values in the column can be the same. This is particularly important in this case, as the primary key needs to be unique for each customer.

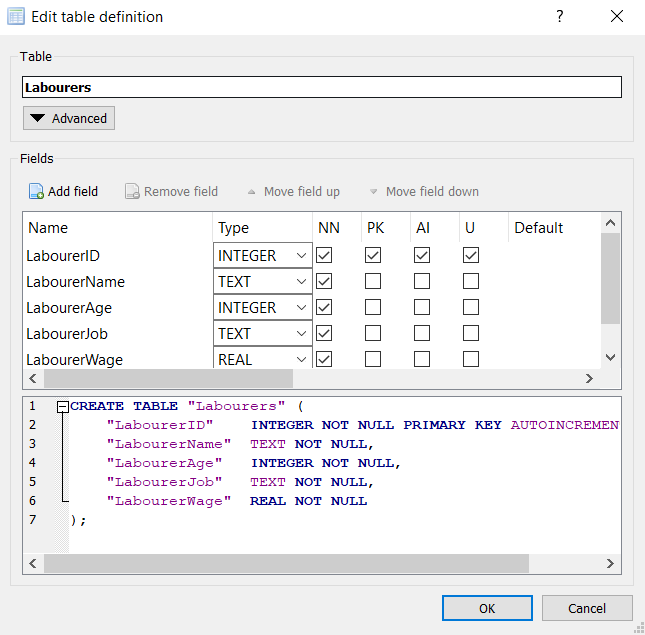


Figure 13

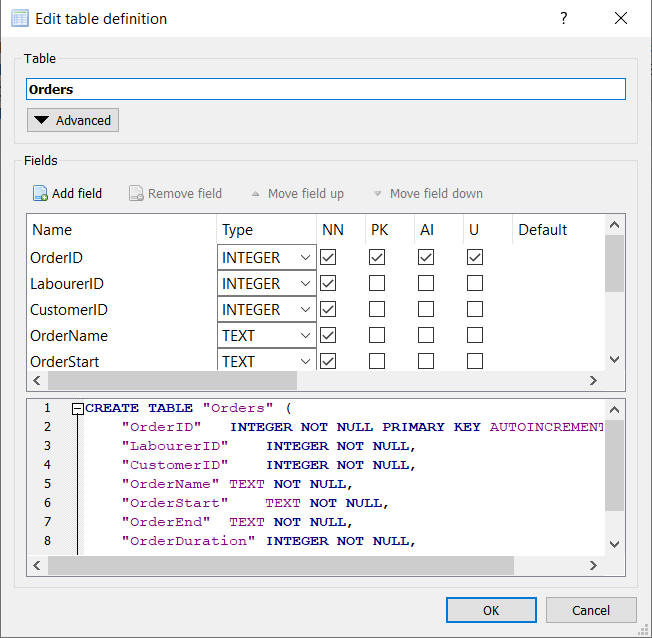


Figure 14

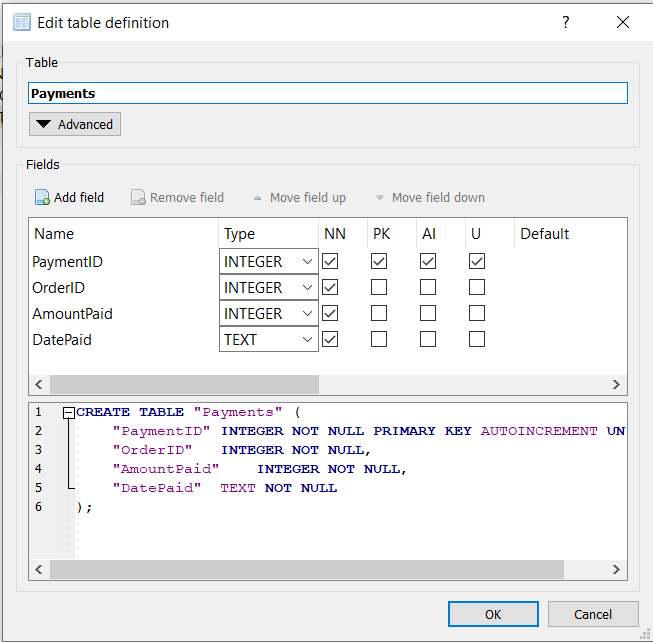


Figure 15

The database browser I used for this project had a very easy way of setting these tables up, where I simply had to specify the rows I wanted to create and the conditions I want to impose on them. This also meant I didn’t need to hard code the creation of the table in to my python, saving vast amounts of storage space for Kieran.

However, this doesn’t work well as there is no specification of foreign keys inside of this. Attempting to reference the keys as I wished to led to a lack of errors which I wanted from the database to ensure its referential integrity. Hence, we need to add foreign keys.

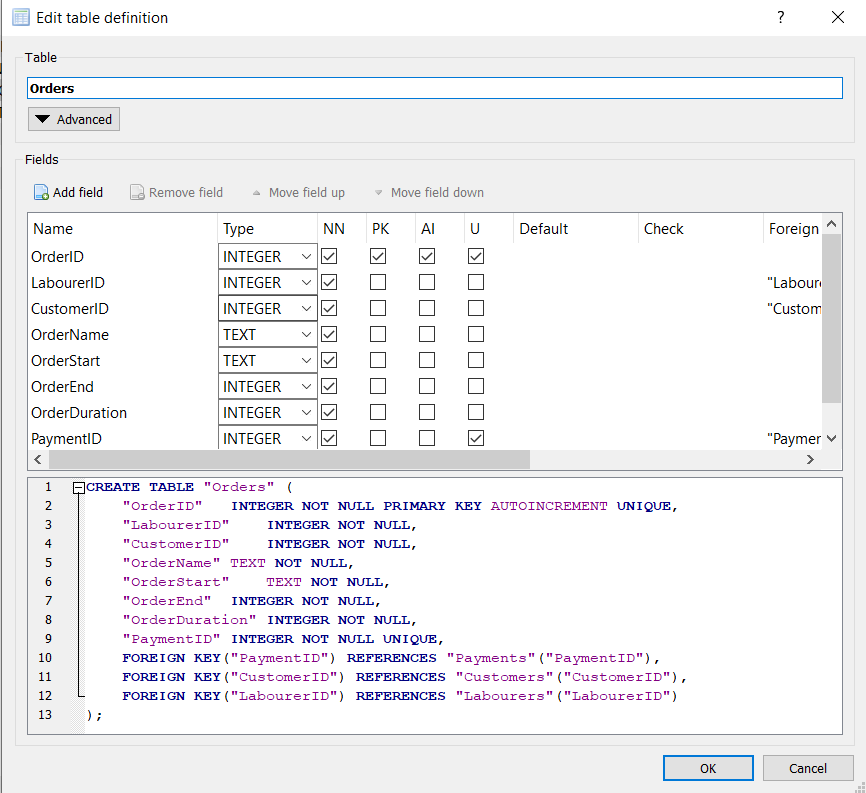


Figure 16

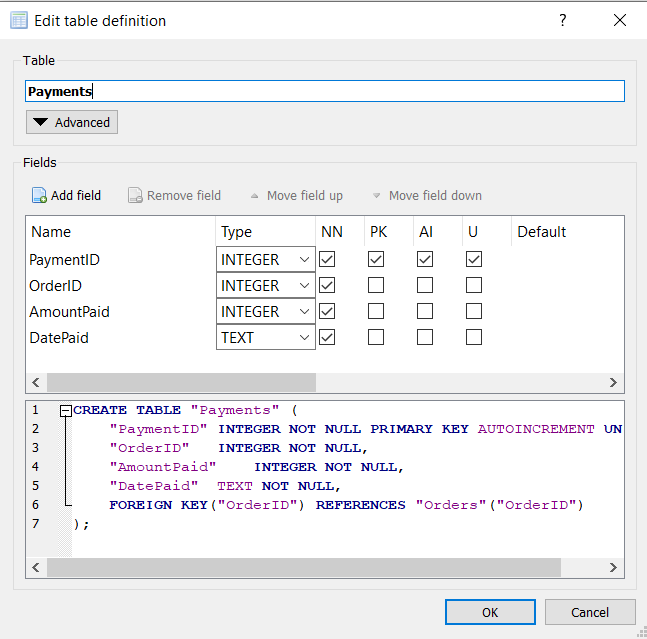


Figure 17

These now have declarations at the bottom, specifying the column that is a foreign key, and what the foreign key refers to. For example, OrderID here refers to OrderID from the Orders table. This means any data from here will be related to the order with the same key as specified here, allowing for easy grouping of information.

Now that foreign keys have been added, there will be an error thrown when there is an attempt to enter data to the database without an existing reference point. The error message thrown from the browser will look like this:

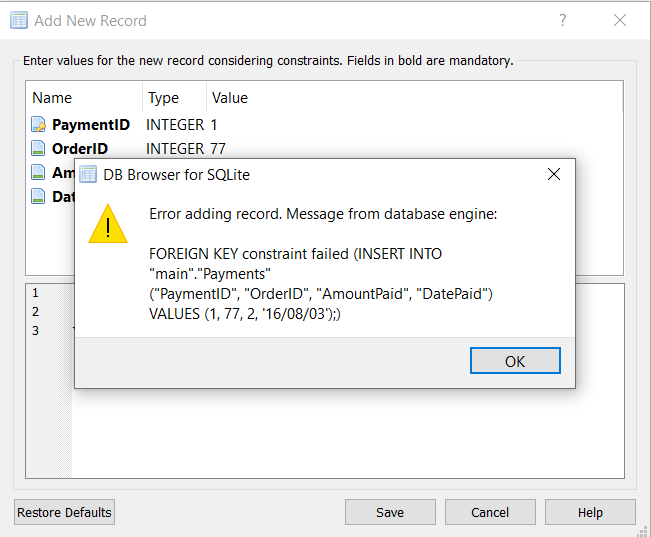


Figure 18

The same problem will be thrown in to python when this occurs. However, this will not happen with proper function structuring before entry of data in to the database.

The structure of the database has now been decided, however, there needs to be more added to maintain and ensure the database’s referential integrity. This ensures the database is robust and stays robust throughout use, no matter the circumstances it is put under.

The model used to maintain this has an acronym of ACID; atomicity, consistency, isolation and durability. Atomicity ensures that there is only one item in each entry. Consistency means the database must stay valid at all times; any power outages or similar occurrences must cause the database to stay valid. Isolation means no matter the order of operations, the end result of the database must be the same. Durability means all data must stay stored in the event of a power outage.

The database is already atomic based on the structure we have created. However, consistency has to be implemented via the SQL functions we write, and durability must be caused by using non volatile memory, such as a hard disk drive. Kieran already owns a hard disk drive, so there is no need to worry about the durability of the database. Consistency must be coded in to the procedures used, so each function occurs all at the same time before making any times, meaning the database has all data written to it simultaneously. Isolation is not relevant as there is only one operation occurring at the same time.

**SQL Functions**

The SQL functions that make up the backend of the program are important, as without them the program could not access the database. As such, it is imperative the code works efficiently.

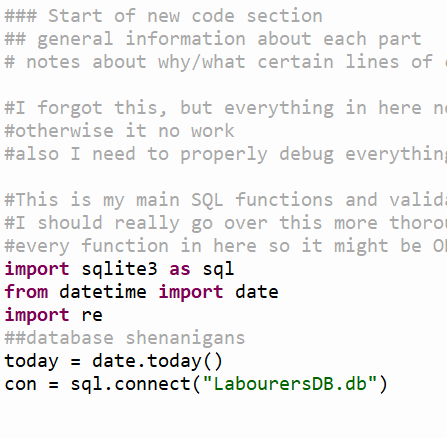


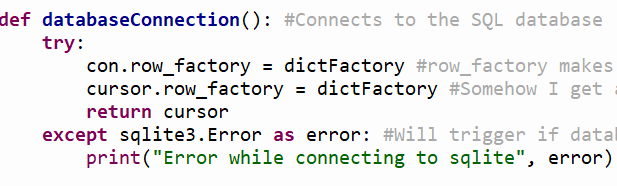
Figure 19

Here, I have made use of libraries to import various functions to help access my database, as well as validate my entries in to the database. SQLite3 is simply the library to interact with the API of SQLite, the format of SQL that I am using in my project. It contains a large amount of instructions that can be used with the database, and can be used to perform CRUD actions (create, read, update, delete.)

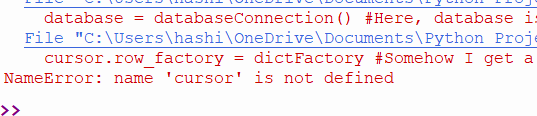
Datetime is another library that allows me to work with time in my database, which is very important as time is an integral part of managing Kieran’s workflow, as well as validating input from the users, and getting the current date.

By obtaining the date at the start of the program, I do not need to repeat the instruction when needed, and can instead just use today and treat it as a date object. I can compare date objects by converting them to strings, allowing for comparison of size and further validation.

The variable labelled con is simply a connector between the database and the python file. This allows for the transmission of data from and to the python file. The .connect command used is one of many provided by the SQLite 3 library.



The following command is intended to return a cursor attribute from the connector created earlier, which can parse through the database to return values. The cursor attribute can be created by utilising a getter method from the connection object, giving the cursor. However, here I forgot to define the cursor attribute, returning an error when one attempts to run the program.



The corrected version is shown below.

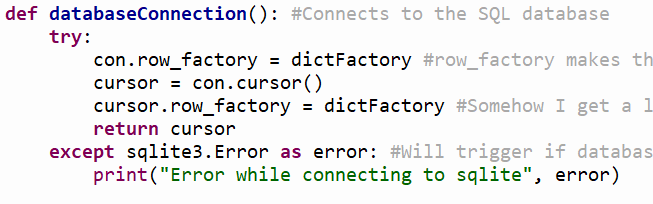


Figure 20

Def database connection defines a function called database connection. This function is intended to run every time I run the program, to allow for the program to return the cursor every time the program runs. This cursor object is an object with attributes given to it by the SQLite 3 library. I have encapsulated the entire function in a try, except clause to allow for exception handling to occur. This allows for a higher degree of professionalism in the final product, and also stops the product from crashing out randomly in the absence of, say, the correct function from the SQLite 3 library.

The first part of this function causes the program to return the cursor object. This object can travel through the whole database as needed, and can parse information from it. This object takes the information back to the python program, where it is returned as a list of dictionaries, allowing for indexed retrieval of the relevant information.

The advantages of this are that it becomes very easy to handle large amounts of data, as a list of all relevant records are returned. This means the records can be further thinned down in some cases when needed.

An instance where it may be needed is later on in the development process, when the heuristic algorithm is produced. By thinning down the results based on proximity to the site, certain workers who are close can be prioritised over others.

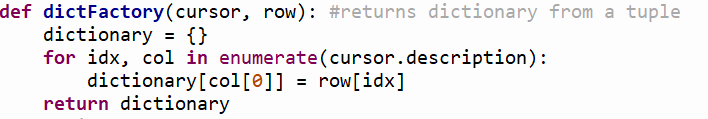
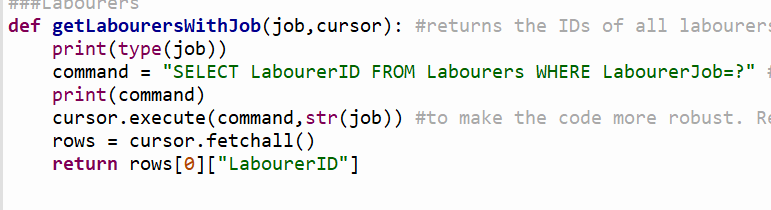


Figure 21

This piece of code defines the function used previously, causing a dictionary to be returned by the function. It does this by collating all read data in the dictionary, and then returning all the read values from the tuple and creating a new variable which is returned at the end, which is now the same as the tuple but now in the form of a dictionary, which is much more helpful as it can be read far more easily.

The list of dictionaries shown in the previous function is a result of the cursor object having many tuples (records) inside of it, meaning many dictionaries therefore have to be returned.



Figure

The function getLabourersWithJob is a function that gets all labourers from the database with a certain job. This function is mainly geared for use in the labourer hire part of the program, where a customer has to choose a labourer to hire in the first place depending on their postcode and job description.

However, the function as it is now doesn’t work as intended. This is because the bindings entered are treated as separate characters, and so need to be separated out, else it will not work.

The fixed version of the code has been shown below:

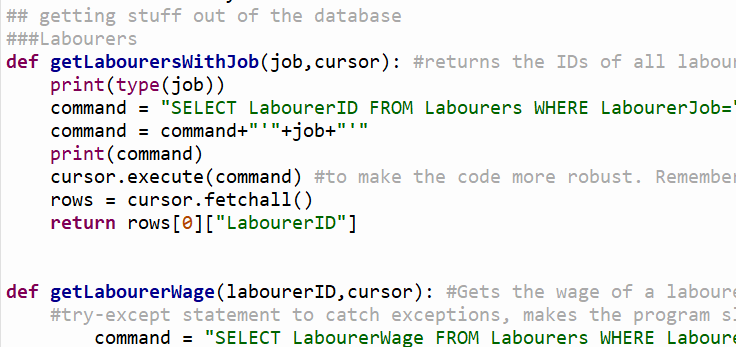
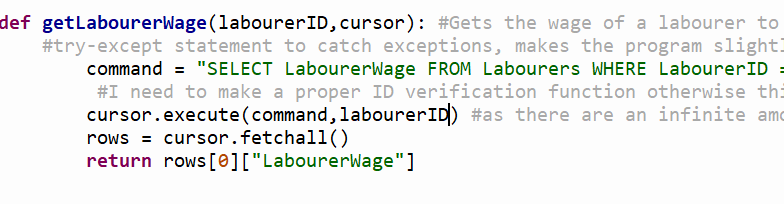


Figure 23

The idea for the GUI of that process is one where a drop box is present to select a job from. Hence, it is very unlikely that any attempt to stop SQL injection is needed for this function, as the input would already be sanitized based on the fact that there are only a few inputs possible, of which none would cause a crash or unwanted data to be shown to the user.

An alternative to this method could have been the use of a switch statement, which would be appropriate as there is a fixed number of outcomes. However, my programming language of choice, python, does not have the functionality of a switch statement, and hence this technique was foregone. 

The function getLabourerWage finds the wage of the labourer selected. This is also aimed for use within the customer hire section, where the payment of the labourer needs to be calculated by the program by taking the time the labourer worked and their wage and multiplying it together.

Initially, I made the same mistake here as I did in the previous step. This meant that any labourer IDs greater than 9 would return an error message, causing the program to crash. Hence, the function needed to be fixed.

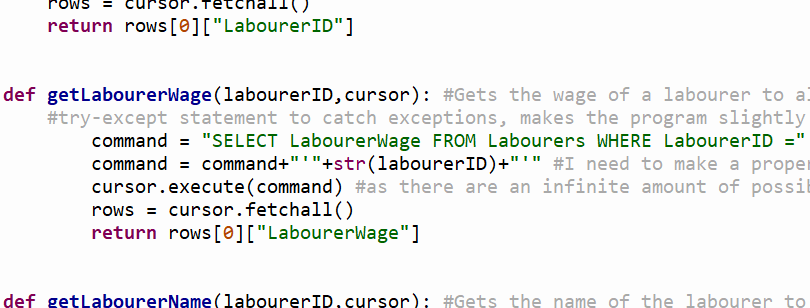


Figure 24

Fixed code shown above. By using the labourer ID and embedding it in to the string, one can run the command much more easily. However, this opens up avenues of malicious attacks via SQL injection. To circumvent this problem, the input has been validated to ensure it is a valid integer.

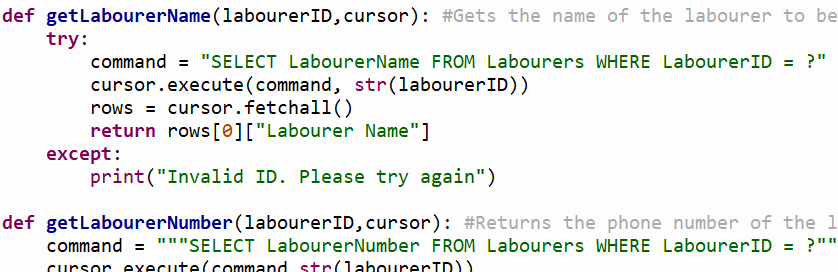


Figure 25

The function getLabourerName is designed for use in the same customer hire section, to display the workers which the customer has been allocated. As such, the labourer’s ID has already been given, as with most of these functions, so can be used to find any other details about the labourer.

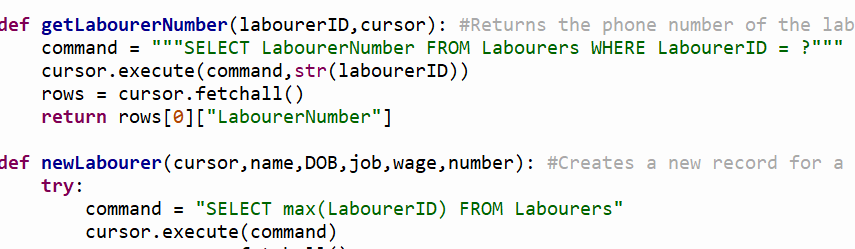


Figure 26

Get Labourer Number is a function to retrieve the labourer’s number, for use at the end of the customer hire program. This allows the customer to contact the worker they have hired, and pass on messages about the job, timings, and any other important details the worker needs to know.

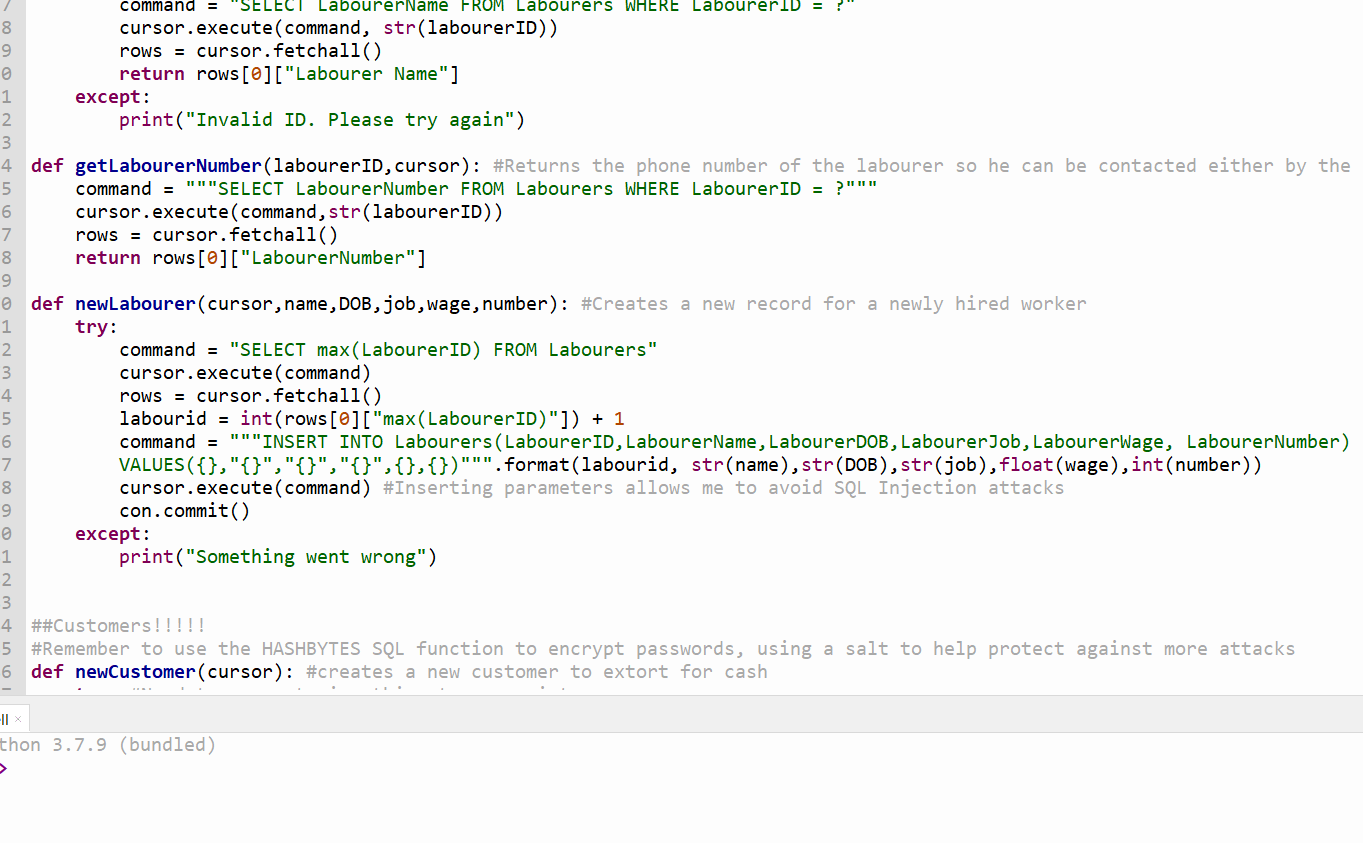


Figure 27

The newLabourer function creates a new record for a newly hired worker, for use in the labourer registration portion of the program. All data added will need to be validated first, however, as there is no validation in the function itself apart from checking all items entered can be converted in to the correct data type.

Here, the VALUES SQL function is used alongside the INSERT statement to create a new record with values specified in the function. .Format allows for the correct values to be entered in its place, as part of the SQLite 3 library.

In this function, I have also had to adjust my input for the labourerID column. While normally when entering the data in to the database via SQL, the column would auto update, here I have had to increment the ID myself to allow for possible error in the system taking place.

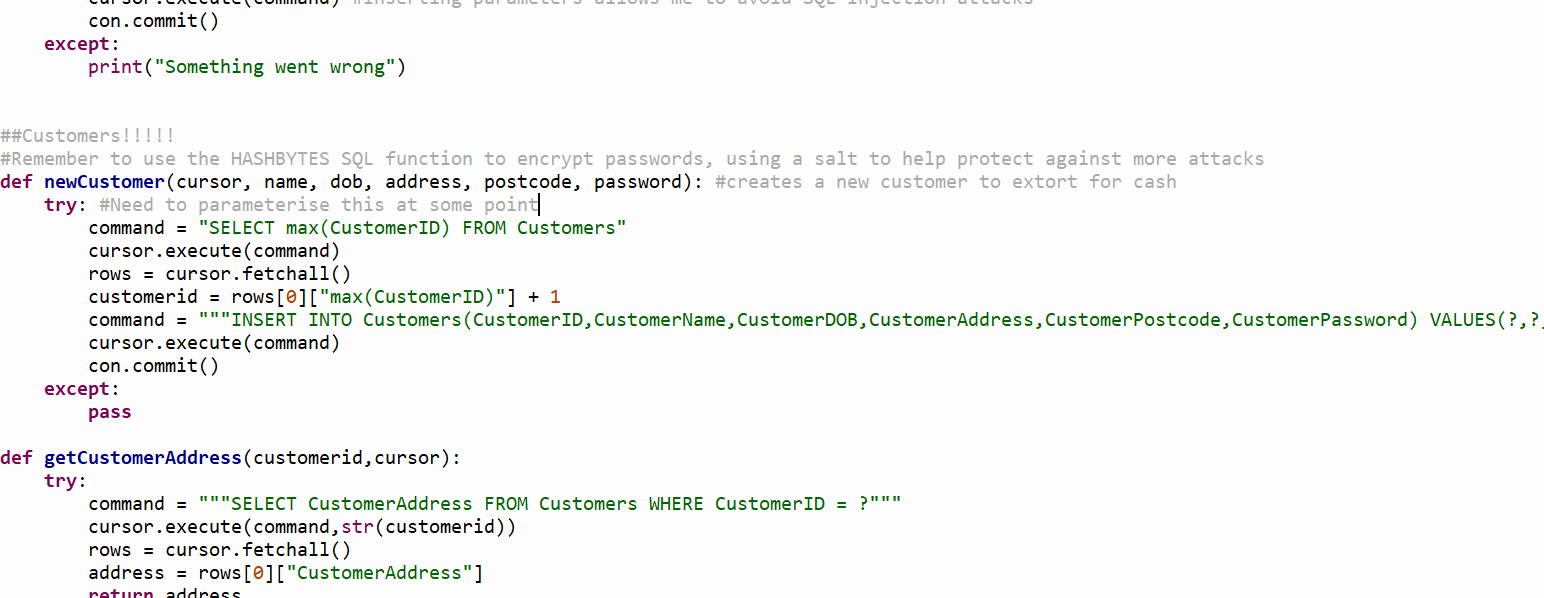


Figure 28

The newCustomer function functions the same as the newLabourer function. However, it has less that needs to be passed in to the database and function, and operates on the customers table. The pass command in the exception handling simply skips the program and halts a crash from occurring.

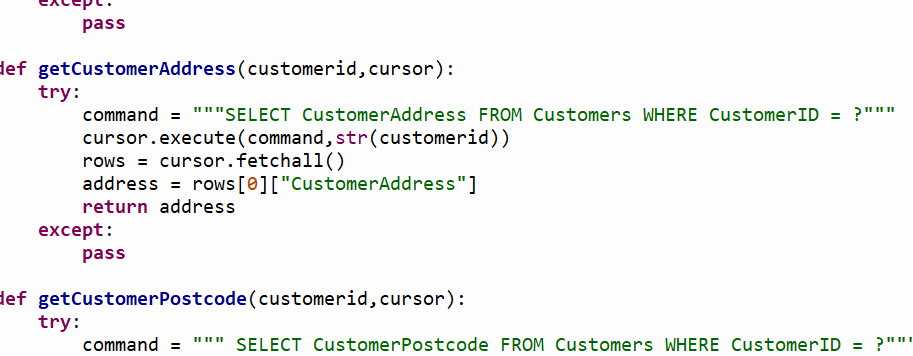
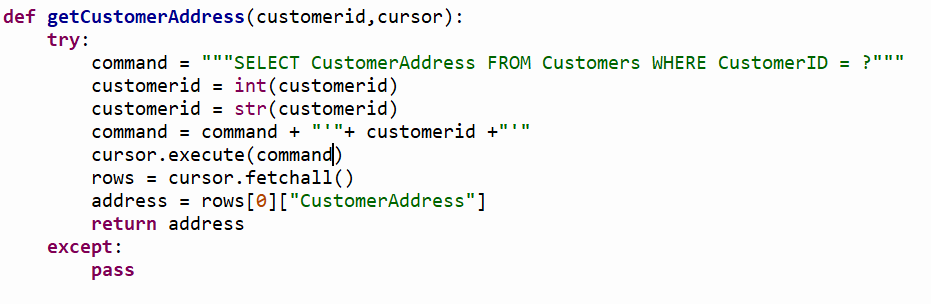


Figure 29

GetCustomerAddress function retrieves the customer address from the database. The command and execute functions in the function define a piece of SQL code which is written, which is then executed. The execute command can only take 2 arguments, so only the command and the customer ID can be passed in at the same time, otherwise a crash would occur.

However, this code is faulty. Customer IDs greater than 9 will cause a crash, as the program will treat each character as an individual binding. Instead, one needs to embed the ID in to the command itself, shown below:



Fixed code shown above. Once again, this fix requires one to note the increased risk of SQL injection attacks. Hence, the conversion to an integer is necessary for validation, and to ensure the database is not attacked and compromised maliciously.

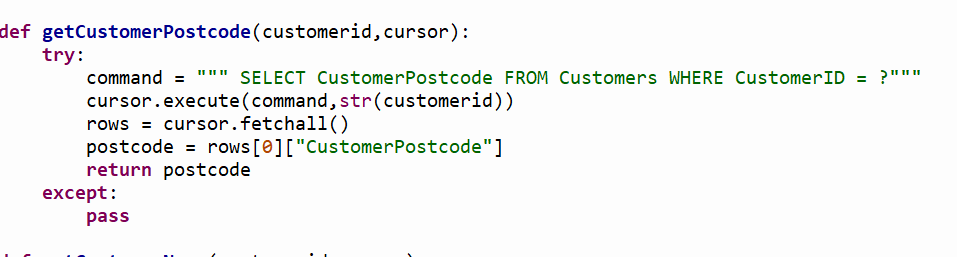
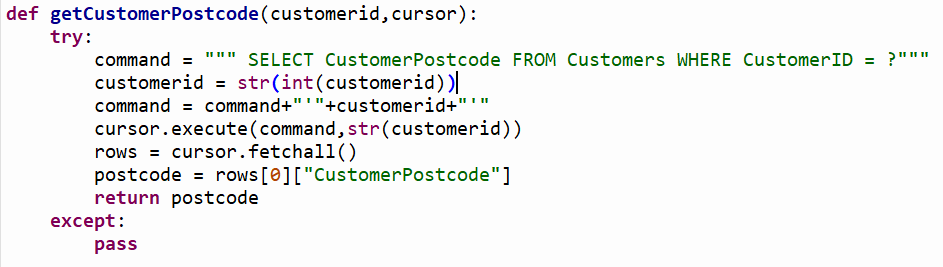


Figure 30

The getCustomerPostcode function retrieves the customer’s postcode from the database. The postcode is used by accessing a list of dictionaries called rows, which is returned by the cursor object when it is passed in to the function. Hence, it needs to be addressed first and then had the postcode removed from it. As there is only one customer with any customer ID, there will only be one dictionary in the list, and hence it is addressed at index 0 of the array, as python indexes arrays from 0.

This is the same as the address of the construction site, and so is viewable by the labourers from the view orders screen.



The code before suffers from the same issue encountered previously, where a customer ID greater than 9 causes the function to crash. Here, I have done the same as before, adding the parameter directly in to the function and executing it after verifying the integrity of the input.

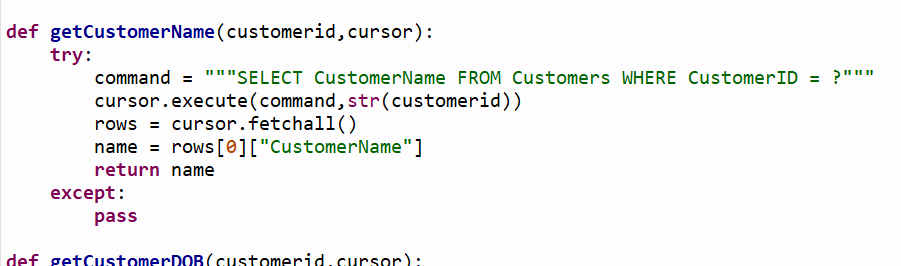


Figure 31

The getCustomerName function retrieves the customer name from the database. The name is used when viewing orders from the view of the labourer, for easy identification of what order is which, and from whom the labourer needs to be paid by.

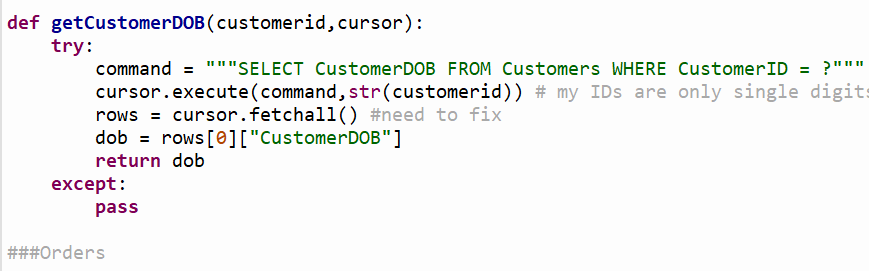


Figure 32

The getCustomerDOB function retrieves the customer’s date of birth from the database.

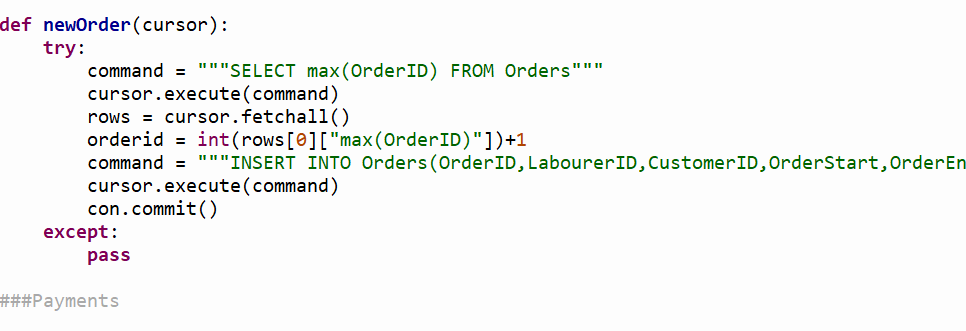


Figure 33

This function creates a new order, and is used when hiring a new labourer. It also fetches the current max ID and increments to create the new ID used to create the new record.

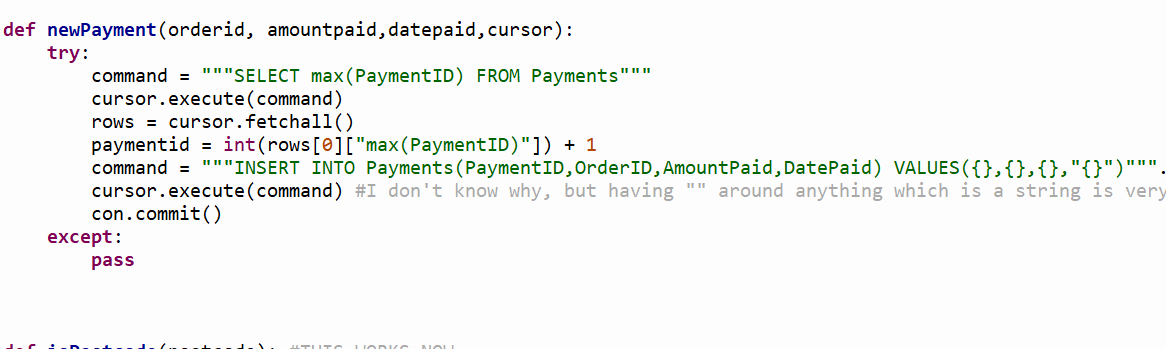


Figure 34

This function creates a new payment, which is used in the hiring labourer part of the program. A new entry is created with an unpaid date paid, meaning all outstanding payments can be viewed quickly and easily.

**Validation and Hashing Functions**

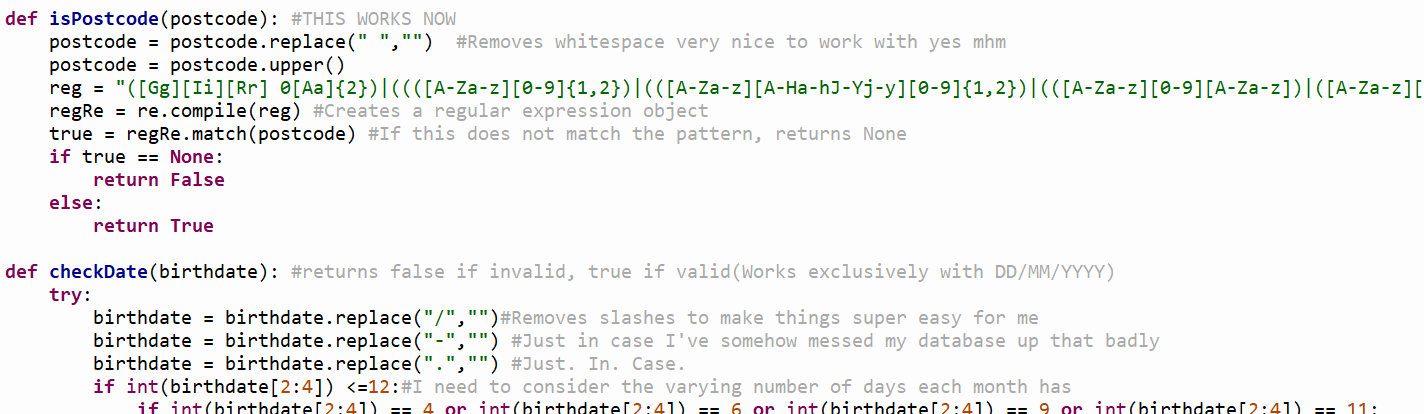


Figure 35

The isPostcode function checks whether the postcode entered fits the postcode regulations set out by the UK government, checking if it works with the regular expression the government themselves use to create new postal codes. This is very helpful, as newly created postcodes will therefore be deemed as possible, and therefore will be considered valid without needing to update the program, making maintenance costs much lower for Kieran.

Here, I have used a new library called regex, which stands for regular expressions. It allows for compilation of the regular expression as shown above in to code which can be used to check against strings to check if they meet the criteria.

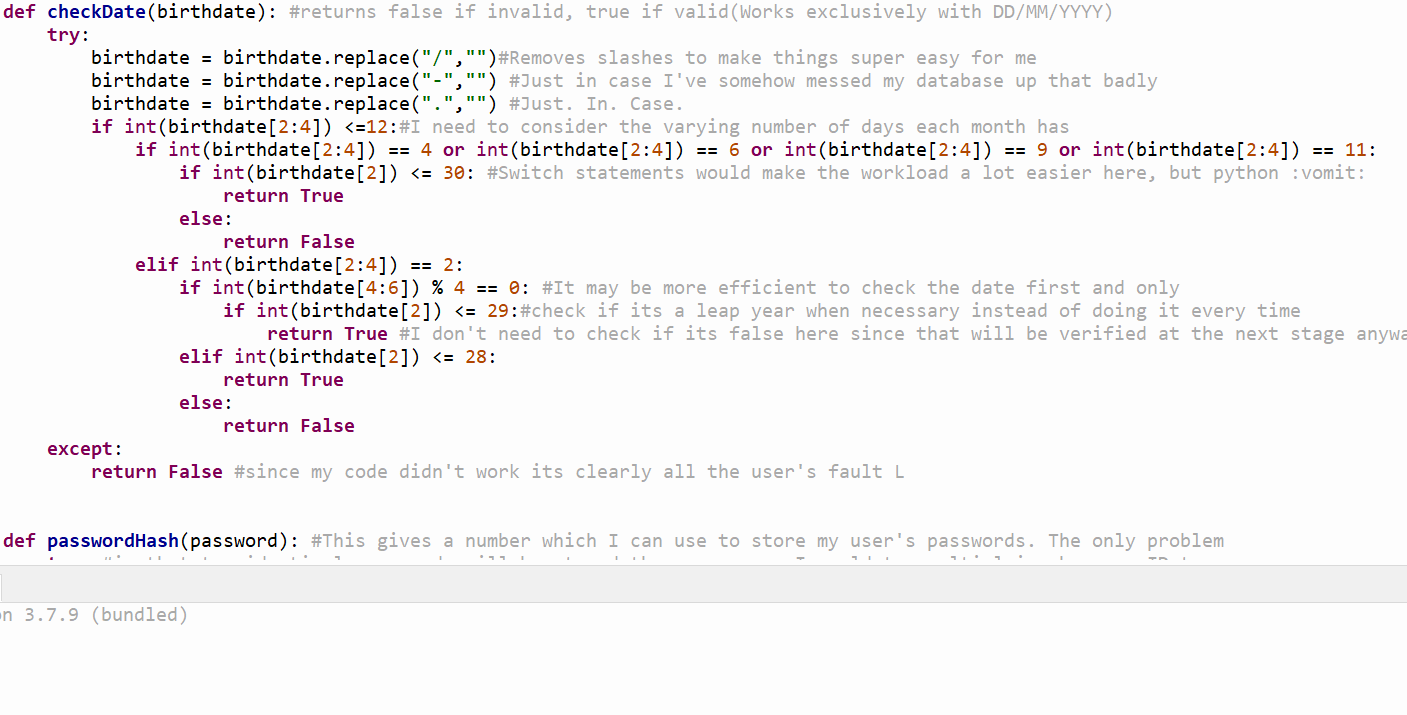


Figure 36

The CheckDate function checks whether a date is valid. It does this by taking the string passed to it and standardising the date in to a certain format, then creating substrings of the newly parsed string to compare the days, months and years.

Here, it is also important to consider the possibility of leap years in the equation. As such, is the year divided by 4 gives a remainder of 0, the year is considered a leap year and so a date of February the 29th is considered valid. Else, it will return invalid.

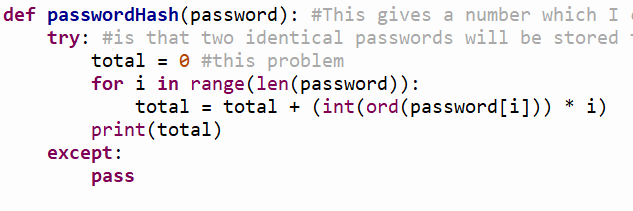


Figure 37

I have used this hashing program for use in Kieran’s program. Most importantly, the hashes themselves cannot be reversed. However, there are multiple hashes that lead to the same result. As such, entering these passwords could lead to an incorrect password being deemed valid and being given entry.

**GUI Code**

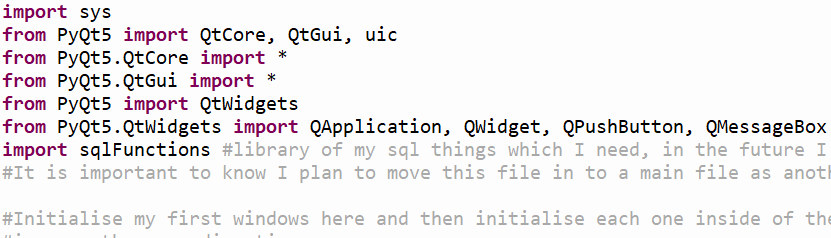


Figure 38

This code imports the necessary libraries to interact with the PyQt 5 .UI files, and initialise the windows themselves as well as work with the widgets which will be used.

In addition to the pre-programmed libraries I have imported the SQL functions, validation and hashing algorithms previously coded in to the new GUI file. This makes writing the necessary methods to interact with the GUI much easier, as I can simply use the SQL functions when they are needed.

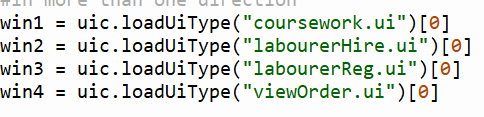


Figure 39

These windows are defined by the ui files assigned to them. This creates them as objects, which have methods specified in the PyQt 5 libraries, allowing you to show, hide, and many other functions with the windows, like change their size. The [0] at the end is a part of the syntax of the function.

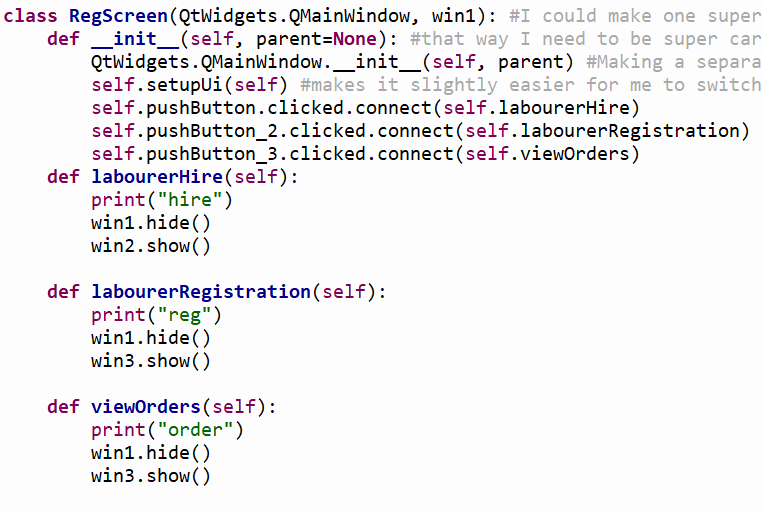


Figure 40

The class reg screen is a class which defines itself as inheriting win1, and hence win1 can use all the methods in this class. Here, each interaction labelled in the constructor method leads to a method being used.

For example, when the first push button is clicked, the labourerHire method is run, which causes the current window to hide, and the relevant window to show. This means that the client can now only interact with that window in particular, and none of the others.

The class also inherits all the methods of a main window, which is shown through its ability to use the .hide and .show functions. This also allows for the use of many other functions, although they haven’t been used here. To reduce space taken by the library, one could only import those specific commands.

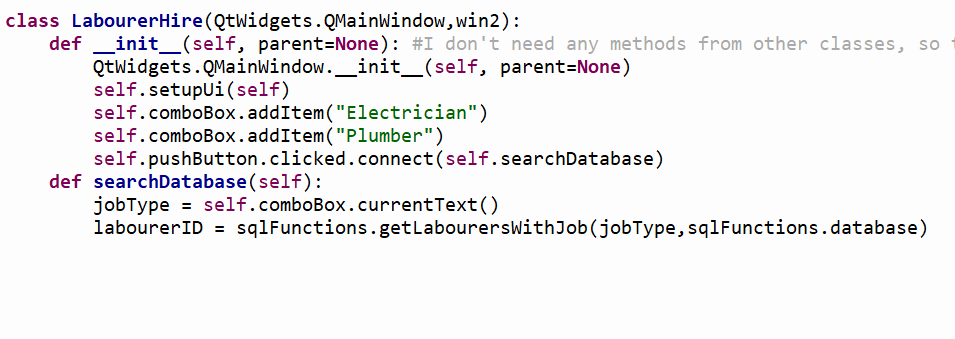


Figure 41

The class labourer hire inherits the second window, and has the method to search the database for labourers with a certain job, using the SQL functions that we imported and coded previously, re-using code efficiently.

**Evaluation**

**Final Testing**

Testing was carried out on the code during development. However, it must be tested once again to check if the program fully meets the success criteria agreed upon by the client and the developer at the start of the development process.

In addition, the code needs to be tested to see if the criteria that have been met are robust and able to hold up to the abuse the client would likely be putting the programs through, such as entering invalid data, null data, and other forms of test data.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No. | Test | Test Method | Expected Outcome | Outcome | Changes made |
| 1 | Speed of page transition | Press buttons to move to next pages. | Little latency between page swaps | Little latency between page swaps | None |
| 2 | Closing down application | Shut down the application | Application shuts down | The application shut down | None |
| 3 | Intuitive user interface | Show the program to a non tech user, and observe their movement through the program | The user will be able to navigate without much trouble | The user encountered some problems between page transitions – it was not immediately obvious one could not go backwards easily. However, the user easily used buttons and the other key inputs without issue | None |
| 4 | SQL functions – new labourer entry | Run the function from inside of the SQL library. Ask the user for input through the IDLE interface. Check the database for changes made to the data, and if the function was successful. | The changes will be made and visible in the database from the database browser. | The changes were visible. | None |
| 5 | SQL Functions – new labourer entry boundary test | Run the function from the SQL library.  Ask the user for input through the IDLE interface. Check the database for changes made to the data, and if the function was successful.  Date entered as the 29th of February,1988. | Accepted in the same manner | The changes were visible and accepted | None |
| 4 | SQL Functions – New labourer entry null data test | Run the function from the SQL library.  Ask the user for input through the IDLE interface. Check the database for changes made to the data, and if the function was successful.  There was no date of birth entered. | Rejected, asked for a valid date. | The blank string was entered in to the database. | Added a validation check to check for blank spaces in the function |
| 5 | SQL Functions – Retrieving information about the labourer from the database | Run the function through the SQL Functions library. Print the information in IDLE and verify | SQL returns correct information | SQL returned the correct information, and was displayed on screen. | None |
| 6 | SQL Functions – Retrieving information about the labourer with an incorrect ID | Run the function through the SQL functions library. | Error message is returned | Error message was returned | None |
| 7 | SQL Functions – Retrieving information about the labourer with an invalid ID | Run the function through the SQL functions library. Enter an array instead of the expected string, containing random/filler data. | Error message is returned. | Error message was returned, without the program crashing out. | None |
| 8 | SQL Functions – Creating a new customer | Run the function from the SQL functions library. Enter inputs via the code, and observe the database before and after for any changes made to it. | Details should be added to the database correctly, while keeping referential integrity. | Details were correctly added to the database. | None |
| 9 | SQL Functions – Creating a new customer with boundary data | Run the function from the SQL functions library. Enter inputs via the code, and observe the database before and after for any changes made to it. Inputs entered such as birthday will be the 29th of February on a leap year, postcodes which don’t currently exist but may in the future. | Details should be added to the database without issue. | Birthday was rejected by the SQL code. | The function did not have the validation from the library correctly implemented. The implementation was fixed. |
| 10 | SQL Functions – Creating a customer with invalid data | Run the function from the SQL functions library. Enter inputs through the code, and observe the database before and after execution for any changes made to it. Inputs will not conform to the expected values. | An error should be encountered. | Error was encountered. Program rejected the invalid inputs. | None |
| 11 | SQL Functions – Creating a customer while inputting no data. | Run the function through the SQL functions library. Enter inputs through the code and observe the database before and after for any changes made to it. There will be blank strings inputted in place of data. | An error should be encountered. | An error was encountered. | None. |
| 12 | SQL Functions – Retrieving customer data | Run the function through the SQL functions library. Enter inputs through the code, observe the database and compare to the output. | Customer data should be correctly displayed | Customer data was not correctly displayed – customer ID was being incremented before retrieving data. | Customer ID is no longer incremented before retrieving data. |
| 13 | SQL Functions – Retrieve customer data with invalid data | Run the function through the SQL functions library. Enter the inputs through the code, and observe for outputs. | The program should return an error message. | Program returned an error message. | None |
| 14 | SQL functions – Retrieve customer data with boundary data | Run the function while using the lowest or highest customer ID in the database. Observe for outputs. | The program should return the correct value. | Program returned the correct value. | None |
| 15 | SQL Functions – Retrieve customer data with no data | Run the function from the SQL functions library. Enter a blank string in place of customer ID. Observe for outputs. | An error message should be returned. | An error message was returned. The program did not crash. | None. |
| 16 | SQL Functions – Create a new order | Run the function from the SQL functions library. Enter the information directly in to the code. Observe the database for changes. | A new order should be created with the details you have entered. | A new order was created with the correct details. | None. |
| 17 | SQL Functions – Create a new order with a customer who doesn’t exist | Run the function from the SQL functions library. Enter the information directly in to the code. Observe the database for changes. Ensure the ID for the customer entered does not exist. | The program should return an error. | The program does not return an error. | Column was not correctly labelled as a foreign key in the database structure. It has now been labelled correctly and fixed. |
| 18 | SQL Functions – Create a new order with a labourer who doesn’t exist | Run the function from the SQL functions library. Enter the information directly in to the code. Observe the database for changes. Ensure the ID for the labourer entered does not exist. | The program should return an error. | The program does not return an error. | Column was not correctly labelled as a foreign key in the database structure. It has now been labelled correctly and fixed. |
| 19 | SQL Functions – Create a new order with invalid data | Run the function from the SQL functions library. Enter the information directly in to the code. Observe the database for changes. Invalid data entered takes the form of an ending date that is before the current date. | The program should return an error. | The program did not return an error. | Further investigation showed there was no code to check whether the date was in front of the current date. The code has been implemented, by converting the total time since 1974 in to minutes and comparing to check for validity. |
| 20 | SQL Functions – Create a new order with boundary data | Run the function from the SQL functions library. Enter the information directly in to the code. Observe the database for changes. Boundary data takes the form of the extremities of valid data for each field. IDs would be the highest available ID, while dates would be the closest possible. | The database should append the new data to it. | The database appended the new data to it. | None. |
| 21 | SQL Functions – Create a new order with no data | Run the function from the SQL functions library. Enter the information directly in to the code. Observe the database for changes. | The program should return an error. | An error was returned. | None. |
| 22 | Create a new payment | Run the function from the SQL functions library. Enter the information directly in to the code. Observe the database for changes. | Changes should be made to the database in accordance to the entered data. | Changes were made in accordance to the entered data. | None. |
| 23 | Create a new payment with boundary data | Run the function from the SQL functions library. Enter the information directly in to the code. Observe the database for changes. Boundary data in this case would be the 29th of February on a leap year. | Changes should be made to the database in accordance to the entered data. | Changes were made in accordance to the entered data. | None. |
| 24 | Create a new payment with invalid data. | Run the function from the SQL functions library. Enter the information directly in to the code. | The program should return an error message. | An error message was returned. | None. |
| 25 | Create a new payment with no data. | Run the function from the SQL functions library. Enter the information directly in to the code. In this case, blank strings would be entered in to the code. | The program should return an error message. | An error message was returned. | None. |
| 26 | Validate a valid postcode | Enter a valid postcode in to the function from inside the SQL functions library. Print and observe the results. | The program should return true. | The program returned true. | None. |
| 27 | Validate an invalid postcode | Enter an invalid postcode that cannot exist now nor in the future. Do this from inside the SQL functions library. Print and observe the results. | The program should return false. | The program returned false. | None. |
| 28 | Validate a postcode containing no data | Enter a postcode of an empty string from inside the SQL functions library. Print and observe the results. | The program should return false. | The program returned false. | None. |
| 29 | Hash a string of text | Enter a random string from within the SQL functions library. Print and observe the results. | The program should return an integer that cannot be reversed. | The program returns an integer that cannot be reversed. | None |
| 30 | Validate a valid date | Enter a date in to the function from within the SQL functions library. Print the results and observe. | The program should return true. | The program returned true. | None. |
| 31 | Validate a boundary date. | Enter a date in to the function from within the SQL functions library. Print the results and observe. Boundary data refers to the 29th of February on a leap year in this case. | The program should return true. | The program returned true. | None. |
| 32 | Validate an invalid date. | Enter a date in to the function from within the SQL functions library. Print the results and observe. Invalid data is any dates which do not exist on the calendar. | The program should return false. | The program returned false. | None. |
| 33 | Validate an empty date. | Enter a date in to the function from within the SQL functions library. Print the results and observe. An empty date is an empty string passed in to the validate function. | The program should return false. | The program crashes. | There was no except loop to catch exceptions, so I have added one which defaults to returning false. |

**Stakeholder Response to the finished program**

With the program finished, I wanted to evaluate the stakeholder’s response to the finished product.

**Kieran (Company Owner)**

The program seems quite polished, despite the fact that it is lacking in many features that were discussed at first, with hopes of making it in to the final iteration of the program. Notably, the GUI is crisp, clean and easy to navigate. It (the program) uses commonly seen button and spin box designs, so even someone who wasn’t as adept with the program could fairly easily grasp the basics on how to use the program to its full potential. In addition, loading between pages was quite smooth and felt professional. Unfortunately, the lack of a back button hampers usability quite a bit, as a wrong touch can send you down a path you cannot leave without restarting the program entirely. This makes for a very hostile design, where a mistake can cost a large amount of time.

In addition, key features such as the ability to edit a customer and labourers data from within the GUI were notably absent, which does mean I will have to update the information myself. This isn’t too much of an issue though, primarily since people don’t move houses all that often.

Unfortunately, the labourer proximity measurement program is notably absent. This won’t present a huge challenge to myself, since sorting postcodes is still remarkably simple with the database. This just means it’d take slightly longer to obtain the information I want, but its not a huge problem.

Overall, there are large parts of the initial project that are missing or haven’t worked in the way as discussed. However, there is still a large amount of function and usability which I can certainly use to solve part of the problems I faced.

**Worker**

The program seems to do everything I need it to, without much compromise. The only downside I can see to this is my inability to talk to my employer when registering, but that can be remedied over the phone or by email. The GUI seems pretty simple and clear; there were clear signs signalling where to go and what to do for the service you want. My biggest qualm with the program is the lack of a back button – any mistakes I do make require for me to restart the program completely, which I would assume isn’t exactly ideal for someone with very little technological experience, who would come here.

In particular, I find the ability to track the orders I have ongoing and done in the past very helpful. As a self-employed individual, I find filing taxes rather taxing when it comes around, but by monitoring the payments I receive here I can easily do my taxes when the time comes. This also means I save money I would have spent on an accountant to do them for us.

The log in and sign up page for customers, however, does seem semi cluttered, and in my opinion, could do with a separation to simplify the interface. Otherwise, for my purposes, the program is satisfactory and will certainly make working easier.

**Customer**

The program functions nicely, without any large problems apart from the lack of a back button, which is slightly frustrating from a quality of life perspective. Apart from this, the interface is clear and well presented, which helped me get familiar with the software. There’s also a lot of commonly seen items, so that helped me use the software pretty easily, like the buttons and number arrow boxes (spin boxes). There are also plenty of instructions for anyone who uses it, so I don’t think it would overwhelm too many people. Still, the lack of a back button hampers usability by those who are less adept with the software.

The customer related functions aren’t as fleshed out as I’d like them to be. Hiring a labourer requires you to navigate a messy login/signup screen which, while makes sense, it makes the process less user friendly so I think there could be some improvements there.

On the other hand, the ability to see all the orders I’ve placed now and in the past is handy for monitoring workers as well as checking who I still need to pay.

Overall, the program does its job well, but could use some quality of life changes to make it more convenient for use.

**Level Of Success**

The level of success of the program is important to evaluate as to whether the problem can be correctly called solved.

**Success Criteria Check**

Here, I will check whether each success criteria was successfully fulfilled in the problem, and evaluate, if it wasn’t fulfilled, how much of the criteria was developed with details of the process.

|  |  |  |
| --- | --- | --- |
| Criteria | Evidence of success | Criteria met? |
| Allow labourers to log in to an account | The labourer can successfully enter credentials for an account | Yes |
| Allow customers to log in to an account | The customer can successfully enter credentials for an account | Yes |
| Contains account details for both labourers and customers. | Details can be viewed from inside the database | Yes |
| Adequately stores these account details in line with the data protection act. | Meets the specification of the Data Protection Act | Yes |
| Provides a sign up option for labourers. | Labourers can add themselves to the database | Yes |
| Provides a sign up option for customers. | Customers can add themselves to the database | Yes |
| Requires a customer to sign up before being able to hire a labourer. | Customers are blocked from hiring before signing in | Yes |
| Requires a labourer to sign up before being eligible for hiring by a customer. | Labourers are unable to be hired before being registered on the database | Yes |
| Ensures the sign up process for customers collects all details needed to hire a labourer. | Evaluate the details taken from the customers during the sign up process | Yes |
| Ensures the sign up process for labourers collects all details needed to be hired out as a labourer. | Evaluate the details taken from the labourers during the sign up process | Yes |
| Stores information about labourers. | Observe the database for information on labourers | Yes |
| Stores information about customers. | Observe the database for information on customers | Yes |
| Stores information about orders placed by customers. | Observe the database for information on orders | Yes |
| Stores information about payments due and paid by customers to labourers. | Observe the database for information on payments by customers to labourers. | Yes |
| Allows easy access to information relevant to a labourer when needed. | Attempt to access information relevant to a labourer. | Yes |
| Allows easy access to information relevant to a customer when needed. | Attempt to access information relevant to a customer | Yes |
| Groups data in tables, storing data relevant to a person in the same table. | Observe the database and check it is properly sorted in to relational tables. | Yes |
| Contains references to other tables to allow for easy access to information through foreign keys. | Observe the relational tables for foreign keys. | Yes |
| Is normalised to 3rd normal form for efficient use. | Ensure that all items in a table are determined by the whole key, only the key, and nothing but the key. | Yes |
| Runs smoothly while being scaled up regularly, and storing larger amounts of data. | Place a large amount of data in to the database and attempt to access information. | Yes |
| View data immediately relevant to the account they are logged in to. This includes hired workers, order start dates and end dates, and payments that are due. | Log in to the account and attempt to verify any information. | Yes |
| Delete the account alongside all data related to the account holders on request. | Log in to the account and attempt to delete. | No |
| Change the data on their account in the event of a change. For example, a worker who moves houses needs to change their address, and should be able to do so. | Log in to the account and attempt to change information logged on it. | No |
| Be accessed via a graphical user interface. | Attempt to access the program using a graphical user interface. | Yes |
| Be easy to access. | Evaluate the program’s ease of use using stakeholder feedback | Yes |
| Be easy to change the data on. | Evaluate the program’s ease of use using stakeholder feedback | No |
| Be easy to view data on. | Evaluate the program’s ease of use using stakeholder feedback | No |
| Be easy to delete. | Evaluate the program’s ease of use using stakeholder feedback | No |
| Data must be processed fairly, lawfully and transparently. | Ensure data is processed in a clear and fair way. | Yes |
| Data cannot be used other than the purpose it has been collected for. | Ensure data cannot be used for purposes other than Kieran’s use for his company. | Yes |
| Data must be kept accurate | Ensure data kept can be kept accurate, whether this is manual or automatic. | Yes |
| Data must be limited in use as much as possible. | Ensure data is not used excessively. | Yes |
| Data must be kept for a reasonable amount of time | Ensure data can be deleted after a point at which it is deemed unnecessary. | Yes |
| Data must be handled securely | Ensure important data is handled securely and with no prospects of attacks. | Yes |
| Hashing algorithms should be used to store passwords securely. | Ensure hashing algorithms are used to store the passwords. | Yes |
| Hashing algorithms should be used to store information such as credit card numbers and expiry dates. | Ensure hashing algorithms are used to store this information. | No |
| Hashing algorithms should be impossible to reverse without finding millions of possibilities. | Ensure hashing algorithm has multiple possibilities for passwords. | Yes |
| Is easy to read for clients. | Evaluate the program’s ease of use using stakeholder feedback | Yes |
| Uses a large font to help those who are visually impaired | Evaluate the program’s ease of use using stakeholder feedback | Yes |
| Uses a simple colour scheme to avoid intimidating clients | Evaluate the program’s ease of use using stakeholder feedback | Yes |
| Is intuitive for those not well versed in computers, and perhaps haven’t used them as much as others. | Evaluate the program’s ease of use using stakeholder feedback | Yes |
| Directs clients with clear and precise instructions. | Evaluate the program’s ease of use using stakeholder feedback | Yes |
| Gives a sense of professionalism. | Evaluate the program’s ease of use using stakeholder feedback | Yes |
| Uses commonly used constructs, such as push buttons and spin boxes, to direct customers easily. | Evaluate the program’s ease of use using stakeholder feedback | Yes |
| Contains friendly language to help ease the user in to the program. | Evaluate the program’s ease of use using stakeholder feedback | Yes |
| Retains Atomicity – Every entry in the database must contain one piece of data. | Analyse the contents of the database to ensure values are atomic. | Yes |
| Retains Consistency – Any changes to the database must leave the database in a valid state. | Ensure all changes are written simultaneously. | No |
| Retains Isolation - Any commands performed on the database performed concurrently must have no difference if they are performed in either order, in series. | Ensure all changes to the database leave it in the same state no matter the order the operations are performed. | Yes |
| Retains Durability – Any changes to the database must remain if the database loses power. | Ensure the database’s information remains on the database after losing power. | Yes |
| Stores the listed information on labourers, customers, orders and payments. | Ensure information is stored in the database. | Yes |
| Use the postcodes from the database of labourers | Check the postcodes are used. | No |
| Evaluate the postcodes, ranking them in order of closest and farthest from the construction site | Check postcodes are correctly evaluated. | No |
| Allows the heuristics program to return a labourer ID of the labourer closest to the construction site. | Check heuristics program works and returns an appropriate result. | No |

**Evaluation of Success Criteria**

Overall, while most success criteria have been accomplished, many of the more advanced functions have not been properly implemented as I wished them to. Notably, these functions are the heuristic proximity function and the ability to change data relating to an account. Factors contributing to the incompleteness of the program can mostly be placed down to COVID 19 and the lockdown surrounding it, greatly reducing productivity and the ability to interact with clients greatly, hurting the iterative development process massively. However, user and robustness testing signal that the program’s current state is usable and solves a large portion of the issues I set out to solve at first.

**Maintenance and Further Development**

|  |  |
| --- | --- |
| Limitation | Consequence |
| Inability to change data from inside the GUI | Kieran will need to manually update the database when data, such as phone numbers or addresses change, leading to an increase in load. |
| Database stored locally | Kieran will need to use one machine which is available to both workers and labourers at the head office, where they can use the machine to provide service to themselves, still requiring the workers and customers to come in to the office, but at any time. |
| Inability to use heuristics to select a labourer | A labourer will need to be selected at random from the database. However, if the labourer is too far from the site, a new labourer will need to be chosen manually in order to select well by Kieran, increasing workload. |
| Inability to change data from inside the GUI | Kieran will have to be personally contacted for those who wish to delete their accounts for whatever reason. Under GDPR, one is able to ask for all data related to them to be removed, and so Kieran will have to oblige to these requests manually, taking some time from Kieran. |

**Possible Further Developments**

**Back Button**

One of the points user testers were most vocal about was the point regarding the lack of a back button. Claims that it hurt user quality of life were very much founded in reality, as the user of the program would need to restart the program from scratch to make the correct choice, which could be alleviated with a simple implementation of a back button. Implementing the back button could be theoretically done using a stack, and placing the windows opened on to a stack. Each time the program progresses, the window closed is added to the top of the stack, and each time the program has the back button pressed, the stack is popped from, and the window is displayed. This would significantly aid user quality of life.

**Heuristics**

To the dismay of Kieran in particular, the initially described heuristic function to evaluate the closest worker to the construction site was not created due to a lack of resources and time. Further development would likely blossom the creation of this function, and its implementation in to the graphical user interface as part of the customer hire routine.

The method for implementing such a routine would likely revolve around the postal codes, the fetching of these postal codes from the database, the use of Google’s waypoints API and comparisons of the distances.

**Removal of Data From The Database**

Due to GDPR, a company should be ready at any time to provide and delete all data relating to an individual. Currently, there is no easy way to do such a thing, as the data has to be deleted manually by Kieran. However, by implementing a function that does this automatically, Kieran can save vast amounts of time and effort.

**Changing of Data from the database**

Due to the data protection act, data must be kept up to date. Hence, any time a client moves, their address needs to be updated when possible. As such, there is no function that allows this to be done alone by the client. Instead, Kieran must do so himself. As such, one could save vast amounts of time for Kieran by implementing such a function, allowing the clients to update this by themselves.