Logo

Description automatically generated

|  |  |
| --- | --- |
| First name | Charles |
| Surname | Weller |
| Student number |  |
| Centre number | 11298 |

Contents

[The problem: 2](#_Toc134624008)

[Current solutions to the problem: 3](#_Toc134624009)

[Interviews: 4](#_Toc134624010)

[Project summary: 5](#_Toc134624011)

[Potential technical solution: 6](#_Toc134624012)

[Limitations: 9](#_Toc134624013)

[Aims and Objectives: 10](#_Toc134624014)

[Window creation overview: 12](#_Toc134624015)

[Creation of the Login page: 13](#_Toc134624016)

[Create of the Create account page: 15](#_Toc134624017)

[Creation of the Main page: 17](#_Toc134624018)

[Creation of the Upload job page: 21](#_Toc134624019)

[Class relationship diagram: 23](#_Toc134624020)

[Database design: 24](#_Toc134624021)

[Database relationship diagram: 25](#_Toc134624022)

[Databases input and outputs: 26](#_Toc134624023)

[Data validation: 29](#_Toc134624024)

[Research: 30](#_Toc134624025)

[Testing outputs and inputs: 32](#_Toc134624026)

[Data base tactical solution: 52](#_Toc134624027)

[Login page technical solution: 53](#_Toc134624028)

[Create account page technical solution: 56](#_Toc134624029)

[Main page technical solution: 59](#_Toc134624030)

[Upload job page technical solution: 69](#_Toc134624031)

[Degree of solution success: 71](#_Toc134624032)

[User evaluation: 74](#_Toc134624033)

[Possible Improvements/ Things I would change: 75](#_Toc134624034)

[Advantix: 76](#_Toc134624035)

Analysis

## The problem:

The problem which I have identified and will be solving is a gap in the job finding market as there is no easy way to look for zero-hour contracts. (A zero-hour contract is a type of employment contract between an employer and an employee whereby the employer is not obliged to provide any minimum number of working hours to the employee). Currently in the UK there are 32.6 million people employed and only 3.1% are seen to be zero-hour contracts. This is surprising to me as for students it is a much better option then permanent employment. It allows the student to take a few weeks off during exams and more difficult periods of schooling. The student will also not lose their contract with their place of employment. Additionally, it allows the student to adapt their work timetable week by week to better suit their current situation.

I will be solving this problem with university students as the intended user, as, while talking to other students, I have become aware of the difficulty of holding jobs during stressful times of the year. This is due to the minimum expected hours (part of a work contract that states the minimum house an employee must work a week). The application will be made with students in mind; the application will be easy and fast to use so that less of the day is spent finding employment. This can be achieved by a large range of functions and all required functions being available in on place.

This will make it easy for both sides of the job sector to use as it will allow an employer to have many applicants and job listings up at once. The easy application process and searching for the job would be beneficial for employers as well, as roles could be filled within a short time frame.

## Current solutions to the problem:

Recruitment Agencies:

Recruitment agencies are organizations which match employers to employees. This can be seen as private and public sectors allowing anyone to have access to these services.

|  |  |
| --- | --- |
| **Pro** | **Con** |
| Large level of connection and partnered companies. | Can take a while. |
| Specialist helps with the finding of a job. | Not prioritized. |
| Job places may be open to agencies before the positions are opened to public. | Pressure to take jobs you don’t want. |
| Access to more job openings. | Competition with other candidates. |
| Employer vetting is done for you. | Not all recruiters may understand your industry. |
| Career guidance. | Commission driven relationship. |
|  | 40,000 in the UK, with 1782 in London alone; there is a great level of competition. |

Online job finder websites:

Online job finders are websites which can be used by employers to help display open positions in a company. There are also websites where people can include their experience and employers can look to find people to fill their roles (which is less common as large company will use head-hunters more often than not).

|  |  |
| --- | --- |
| **Pro** | **Con** |
| Large range of areas. | Can be complicated to use. |
| Ability to search for a small area. | User may not be sure what “what” means on the opening page. |
| CV builder (not a function of all websites). | User will not be sure what keyword is related to in search. |
| Will take less time than other choices to use. | Would be catered for everyone trying to find a job. |
| Filter options. | Large amount of competition. |
|  | Everyone on the website will be able to see the same thing you do; it may be hard to be chosen. |

Education support job finders:

Education support can be seen throughout education but is most commonly seen in university. These are schemes/plans which help students to find jobs, better their financial situation, and help build wider skills.

|  |  |
| --- | --- |
| **Pro** | **Con** |
| Will be for the exact level of the student. | May take a while. |
| School shares a large range of connection with companies. | May require an extension in your number of university years. |
| Companies prefer students from school as they know they will be well educated and fond of the subject. | Will not be able to find small jobs, such as those which run off zero-hour contracts, as schools are seen to find internships and apprenticeships (usually 3, 6, 12 months). |
| Low competition. |  |
|  |  |

When I create the application, I will be taking all the pros and cons of previous efforts at solving this problem into consideration. For example, a filter option so that certain jobs could be searched for. I will produce the user interface in a way which is easy and efficient to use. Unfortunately, due to a restriction in time, a range of the pros in existing solution will not be able to be matched, such as the benefit of a CV maker which would be found on a website and having relations with companies.

## Interviews:

I have chosen to interview three students, asking them the questions stated below and condensing their answers. The three students are all in different stages of their university education, with one first year, third year and a postgraduate.

The students were asked the following questions:

1. Why do you not have a job?

The students stated that jobs are hard to find in a close proximity to their location which would work around school hours. Additionally, any job which a student would manage to find would put pressure on them at the end of the year during exam season. This would lead to students having to use all their holidays or quit jobs entirely, as they would not be able to focus on both or complete the required hours.

1. How are you trying to find your new employment?

The students have been finding jobs online using websites, social media posts, and asking shops about any possible positions. There was a complaint from a student that the time it takes to find a job is too long and leads to a drop in living standards, as savings are needed to pay for vital resources.

1. Why do you think you are having no success?

The students believe that, due to an extensive range of competition with online applicants, businesses would neglect students. This is because of their inability to work constantly, as it would be better to have a full-time employee than a part-time employee who would need a month off for exams, creating stress for other staff members.

1. What would make you use this application?

The students would prefer this application as it is easier to use and can find jobs in a smaller amount of time than counter parts. Also, there is the ability to apply to a large range of job at once.

1. What makes a good application for you?

The students believe a good application is based on results, ease of use, and if it can cover a large range of functions so they don’t need to use several websites at once.

1. Do you belive it is important to be able to save your data externally?

All studance expressed great interest in the ability to save data externally, as it would allow the data to be accessed then the devise is offline and allows easy sharing of data.

1. What is the key function you would like to see?

The students voiced interest in a large main display to see multiply job at once, pages made in an easy-to-use fashion (easy to understand thew the uses of labels and easy to find all function), and a filter option to help specify jobs which are available in their location (in local postcode such as in SW11).

## Project summary:

This application will be made with students in mind and is aimed to make it easy for them to find a job. This would be done by making the code easy to use and using pros from different existing solution, then bringing them together to allow a single application for all the benefits of other ways to find a job.

After looking into current solutions to the problem and talking to the targeted user of the application, I have decided to use a range of functions and information the user will have access to.

Firstly, I have decided to use two tree view to display information to the user. There will be one large main tree view which will display the current open job listings with information on the jobs, ranging from the company name to the location where the employer would expect the user to work. A secondary tree view will also be used; unlike the main tree view, the data in the secondary tree view will be changed based on user interaction. This will allow the user to be able to display their favourited jobs and, if a user has a job listing up, will offer employees the option to view the details of the applicant.

Next, the creation of the button to allow the user to access different pages and use inbuilt functions. There will be a button created to allow the user to open the Upload job page, create or delete a job listing which they have made. A user will be able to favourite and apply to a job listing, and display the user favourited position (on the secondary tree). They can also save the application of the user’s job listing to an external file, and a filter function will allow the user to filter through the job listing which they have on the main page.

Then, due to the large level if interest in the ability to save data externally by the studance. I have decided to create a methord in the application which allow a user to save the applicants of their job listings on an external file. The idea of the ability for a user’s to save their favourites (users favourited job listings) was looked at. But as job listings can be taken down at every time and would not affect any external files meaning a studant way be using an outdated job list causing then to be missed leaded about the avalible job listings.

Next, the ability of admin access was seen to be a necessary addition to the application as the ability for a user to be able to create any job listings they want could lead to impersonation or inappropriate jobs listings being created. The admin account would be able to delete any job listings no mater if they made it or not allowing the ability for moderation of the server.

Finally, the creation of the database to allow the user to save their favourites, account, job listing, and applications will have to be made to allow the user to have all the needed information without making the application too complicated.

## Potential technical solution:

Website-based application:

A possible solution would be through the creation of a client-based website. This would allow users to access this service from any part of the world. In addition to this, the website would use a server that contains all user and job details. The page would use a user key and a UserID, containing all user data. This would be accessed once logged in.

**Advantages**

Creating a website-based application would have personal benefits related to gaining knowledge and understanding of coding a website in HTML or JavaScript. It also has the potential to become a functional business. The website would run though a server, so the processing time would be much quicker than if all the data was being processed by local devices. It would also allow a much larger amount of data to be held in the database, increasing the possible number of users and jobs available. Furthermore, the process of distributing this service would be much simpler and faster than having to send user code and the database though a service, such as email.

**Disadvantages**

A website-based application would have a large range of expenses related to it, as a domain and server would need to be purchased. In addition to this, the maintenance crew needed to maintain the servers daily would lead to extra cost in wages. Moreover, the complexity needed to write the level of code in HTML would be extremely demanding and unrealistic without a large amount of code being found on the internet. On top of this, the continuous debugging would be frustrating.

**Procedural Programming:**

Another possible solution to this problem would be creating the application in Python, using functions. These procedures would gather user input, such as the username and password. It would output data to the user, such as eligible jobs through print statement, on to a Python IDEL shell. The program would be built using subroutines to carry out procedures. Examples of this would be procedures allowing the user to log in and create an account. They would then gather data, which the user may use, but would not have any form of GUI (Graphical user interface) or interactive option past entering values.

**Advantages**

Creating a solution using Procedural Programming would have a range of benefits related to the time and complexity of the programming. This can be seen because the code will be simple to write as it is linear. It would therefore not take any learning or further research to create the necessary code. The complexity of the code means it would take up less memory than other methods.

**Disadvantages**

Procedural Programming can be seen as being better suited for the processing of data than looking after data. This makes Procedural Programming an unsuitable solution to the problem, as there is a large amount of data which needs to be processed, so it may not be updated in real time. Another disadvantage is that a procedural language does not use abstraction to modularise the code. This would mean that it is difficult to add new data types or function without going back to previous sections of the code and modifying them.

**Object orientated programming:**

The final possible solution which is taken into consideration for the creation of the application is Object Orientated Programming (OOP). This is a programming model that organises software design around data and objects, rather than functions and logic. An object can be defined using classes that have unique attributes and behaviour. As the code is written in this way, it allows data to be affected without impacting other functions of the program and to run sections of the code using instantiation of the classes. Additionally, the code which relates to sections of the running codes can be split up to increase the efficiency. This is due to the fact that the code being run is only related to the correct function of the code. There are 4 main functions of OOP: abstraction, encapsulation, inheritance, and polymorphism.

**Advantage**

As code would be created in the form of objects, it can be reused and recalled at any time. The code running constantly increases efficiency and decreases the size of the code helping with the distribution of the application. Methods and attributes can be hidden, which would stop a user or programmer from changing essential sections of the code, as this would lead to unwanted outputs in the code. Code is also easier to add to and change due to all section of the code being modularised. This also helps with error checking, as they can be limited to small sections of code.

**Disadvantage**

Object Orientated programs are much larger files in comparison to their counterparts, such as procedural competitors. This is due to the increase in complexity and detail on GUI and code. This leads to the complete solution being much larger than other solutions requiring more time and energy to code and run.

**Chosen solution:**

Taking the advantages and disadvantages of a website-based application into consideration, I believe it would be in my best interest to use a different solution to the problem. This is due to the time needed to learn a web-based coding language, and because the running of a server, even if run from my laptop, would not be suitable for coursework of this scale.

The possibility of the final solution being made with a mobile app has been taken into consideration but is seen to have the same advantage and disadvantages as a website. On one hand, it would have the possibility to send notifications to the user, which is seen to be a great advantage over all other possible solutions and would be the only one with the ability to do this. However, even taking this into consideration, the disadvantage outweighs the pros of making the final solution with a mobile app.

Considering the advantages and disadvantages of creating the final solution using Procedural Programming, I believe it would be a suitable solution, but it would not allow the depth or detail desired to put into the code. Therefore, it would not be functional if I decided to take this project further. Consequently, I will not be using it to create this application.

Taking the advantages and disadvantages into account, I have chosen to use Object Orientated Programming for my final solution. The reason for this is the advantages related to code, which can be made using objects and methods. These will allow me to divide my programme into parts, which will indicate the different Tkinter windows in my code. This will lead to debugging being a lot easier and the source code being more logically structured, allowing for easy identification of sections of code. The next best option for my final solution would be Procedural Programming. However, it would not allow me to create a GUI, while Object Oriented Programming does. As classes in the code would represent each window, I used composition and instantiation to link them together. This is because it will allow the code to affect variables, and classes would not be running. Also, through the use of composition, windows can be run so that when the last window is closed, the new window remains running. Therefore, if instantiation is used to call the class it may also be destroyed when the container class gets destroyed.

A GUI will make the working environment for the user. This is cleaner, and it removes some of the difficulty around the use of the application. This will be done through Tkinter in the Python interface, a GUI library. I decided to use Tkinter because it will allow me to easily programme the interfaces and use different widgets that the library offers, such as entry boxes, labels, frames, etc.

I will also be using SQL light, to store details and data in a database. SQL allows the subsets of information within a database to be selected, deleted, added to, or altered. It can also be used to add data to the database (which would be used when creating an account). For example, when a new account is made, details would be added to the database and functions like autoincrementing would be used to add UserID. I would need to create several tables which, through SQL, can be easily linked and can help sections of the code run smoothly.

I will be using encryption in my application to increase security and allow user data to be safer. After researching the security and difficulty factors of different method of encryption, I decided on a form of Caesar Cipher.

## Limitations:

Limitations can be seen in to be present in two major areas knowledge, and hardware.

Hardware causes a limitation in the creation and functionality of the application. This is down to the current location of the database being local to the user, even though that application is in a complete state. When a job is listed it will only affect the local database. This would lead to it not being displayed to any user outside of the device which the listing has been created.

This problem could be easily solved with a central server which holds the database, allowing every user to have access to all listings created by other users, no matter the geological distance.

Knowledge will lead to limitation in the quality of the application but, unlike hardware limitations, I will be able to overcome this limitation with a range of research into methods and inbuilt functions to help with the creation and increase the efficiency of the application. (Research I completed can be found in Research under the design section.)

## Aims and Objectives:

Main objectives are objectives which must be completed, unlike secondary objective which could be completed to help improve the application but are not mandatory.

Creating a database:

* Databases should consist of separate tables, consisting of all atomic data.
* Tables must be normalised (normalization is the process of organizing data in a database).

Create a login page:

* This will be the first page which is seen by users when the programme is started.
* A button will be needed to be able to call the Create account page, quit the application, and sign in.
* Entry boxes will be created to allow the user to enter their username and password.
* Label will be displayed to show the user what each entry box is linked to.
* An image (application logo) will need to be placed at the top of the screen.
* Validation on user inputs will need to be carried out (all fields filled and correct length).
* Validation will need to be carried out to see if the username and password are correct.
* If validation is passing the Main page should be opened.
* Error messages are to be displayed if any invalid entries are detected.
* The GUI should be made user-friendly by keeping all paged on a single page and easy to understand labels, button, tree views and layout.

Create a Creating account page:

* This will be displayed to the user if the “Create new account” button is pressed (on the Login page).
* A button will be needed to be able to create a new account and quit the application.
* Entry boxes will be created to allow the user to enter their account details (username, password, name, email, age).
* A tick box will be needed to give the user the option to create an admin account (if box is selected as entry box, it should replace the tick box to give the user the opportunity to enter an amin code).
* Label will be displayed to show the user what each entry box is linked to.
* An image (application logo) will need to be placed at the top of the screen.
* Validation on user inputs will need to be carried out (all fields filled, entries consist of the correct data type, email is valid (if @ is present), and if the admin code correct if selected).
* Validation will need to be carried out to see if the username is not taken.
* If validation passes, the account should be created (details saved to database).
* Error messages should be displayed if any invalid entries are detected.
* The GUI should be made user-friendly by keeping all paged on a single page and easy to understand labels, button, tree views and layout.

Create a Main page:

* This will be displayed to the user if the “sign in” button is pressed (on the Login page and validation is passed).
* A button will be needed to be able to add a job listing, view favourite, add a favourite, view applicants, apply to a job, filter, delete job, view extra information, refresh trees, and close the page.
* Label will be displayed to show the user what each entry box is linked to.
* Tree view needs to be created. A main tree view which displays the job information and a secondary tree which is used to display favours or applicant data.
* A way to select data on the main tree is needed.
* Validation will need to be carried out to see if the search returns no data.
* Validation is needed to see if a job is selected when needed.
* Error messages should be displayed if any invalid entries or selected are detected.
* The ability to save the details (Job title, applicant name, and applicant contact details) of applicants to user jobs in an external file.
* The GUI should be made user-friendly by keeping all paged on a single page and easy to understand labels, button, tree views and layout.

Create Upload job page:

* This will be displayed to the user if the “Add Job” button is pressed (on the Main page).
* A button will be needed to be able to create a new job listing and close the page.
* Entry boxes will be created to allow the user to enter their job details (Company name, job title, age requirement, location, hours, pay, and job description).
* A tick box will be needed to give the user the option to add extra information (if box is selected as entry box should replace the tick box to give the user the opportunity to enter the information).
* Label will be displaced to show the user what each entry box is linked to.
* Validation on user inputs will need to be carried out (all fields filled; entries consist of the correct data type).
* Validation will need to be carried out to see if the username is not taken.
* If validation passes, the job should be created (details saved to database).
* Error messages should be displayed if any invalid entries are detected.
* The GUI should be made user-friendly by keeping all paged on a single page and easy to understand labels, button, tree views and layout.

Secondary objectives:

* A filter option which can be applied to the main tree.
* A tick box will be needed to give the user the option to filter the order that data is represented on the tree views.
* A drop down to help with the filter option.
* Entry boxes will be created to allow the user to enter a search for a job.
* Create a way for the user to add the people which have applied to their jobs too an external file.

design

## Window creation overview:

All the windows should be easy to understand and uses, to improve the quality of the time that the user spends on the application. This can be seen though the uses of lables which informs the user in what infermation needs to be enterned into each entry box, buttons that are approparly labled to the funtion they are related to, and a drop diown allowing the user to filter around pre-determined options (such as pay and hours).

Each widnow has had a flowchart alicated to it which displays the basic process which is conducted through their respective sections of the code. In the flowcharts, a box is seen to be a process, diamonds are test interactions or decisions, and the cylinders is the code accessing the data base. If a decision (diamond) does not have an output coming out of it, it means that there is no more process conducted and the application is wating for the next stop or interaction by the user.

Diagram

Description automatically generated

This flowchart represents, in the simplified form, how each window is opened and the order that must be followed to open the page (E.g to open the Upload Job page the user must gain access to the Main page first.)

## Creation of the Login page:

All avalible user inputs into the login page:

|  |  |  |
| --- | --- | --- |
| Page | Input | Uses |
| Login page | Username | Username and Password will be used in input checks to make sure the user entered the correct information. Then they will be used to validate the user’s account though the data base, by seeing if the username exists and lines up with the password. If all validation is passed it will lead to the opening of the main page. |
| Password |
| Sign in button | The sign in button will trigger the validation on user inputs (as seen above) then, if all validation is passed, it will lead to the opening of the main page. |
| Create account button | The Create account button is used to open the Create account window. |
| Exit button | The Exit button is used to exit the entire application. |

All avalible outputs onto the login page:

|  |  |  |
| --- | --- | --- |
| Page | Output | Reason for output |
| Login page | “Fill all entry boxes” | To inform the user if not all the input boxes have been filled. |
| “Minimum 6 characters for username” | To inform the user if the entered username is an incorrect length. |
| “Minimum 6 characters for password” | To inform the user if the entered password is an incorrect length. |
| “Username or Password is incorrect” | Will inform the user if the username entered doesn’t exist in the database. |
| “Your Account has been created” | To inform the user that their account has been created. |
| “Do you want to close this application?”.  (Options yes or no) | Gives the user the ability to go back on their decision to close the application. |

Diagram

Description automatically generated

This is an expample of the wanted outcome for the “Login” page. It provides a clear and simple way for the user to login and gain full access to the application and it’s function.

This window has been created using 3 button to allow the user to close the applicaion, gain access to the “main” page (through the login button), and create an account which they would then be able to login with. The error message will not show unless the user has failed a validation check to login.

Diagram

Description automatically generated

This flow chart represents the basic function of the “Login” page and what will be carried out when a user interacts with the page. For example, when the “Login” button is pressed, it will lead to validation being run (seen on the left side of the flow chart) which, if successful, then the main page will be opened. Each set of tiles represents a set of instructions which the code will follow, and it shows when the data base needs to be accessed to revive data to carry out the current process.

## Create of the Create account page:

All avalible user inputs into the Create account page:

|  |  |  |
| --- | --- | --- |
| Create account page | Username | All inputs will be used in input checks to make sure the user entered the correct information. Then the username will be used to see if an account already exits. If all validation is passed it will lead to the account being created. The account will be added to the data base. Then the create account page closes and returns to the login page. |
| Password |
| First name |
| Surname |
| Age |
| Admin (If chosen) |
| Create account button | The Create account button will run validation on user inputs (as seen above) then, if all validation is passed, it will lead to the account being created by adding the account details to the data base, closing the create account page, and returning to the login page. |
| Return to login button | The Return to login button will close the create account page. |

All avalible outputs onto the Create account page:

|  |  |  |
| --- | --- | --- |
| Create account | “Fill all entry boxes” | To inform the user if not all the input boxes are filled. |
| “Minimum 6 characters for username” | To inform the user if the entered username is an incorrect length. |
| " Minimum 6 characters for password" | To inform the user if the entered password is an incorrect length. |
| “Email not valid” | To inform the user the email entered is not valid. |
| “Age should only consist of numbers” | To inform the user that age should only consist of numbers. |
| “Admin code is incorrect” | To inform the user if the entered admin code is incorrect. |
| “Username is taken” | To inform the user if the entered username has already been taken. |
| “Do you want to return to the login page”.  (Options yes or no) | Gives the user the ability to go back on their decision to close the page. |

Graphical user interface, diagram

Description automatically generated

These are examples of the wanted outcome for the “Create account” page would look like. It provides a clear and simple way for the user to create a new account or admin account.

This window has been created using 2 button to allow the user to close this page, and create an account which they would then be able to login with. The error message will not show unless the user has failed a validation check to create the account. The use of the tick box (seen on the left screen) is used to allow the user to enter the admin code to create an account with admin access, as once it is selected, the tick box is removed and an entry box is displayed (seen by the right screen).

Diagram

Description automatically generated

This flow chart represents the basic function of the “Create account” page and what will be carried out when a user interacts with the page. Such as when the “Create Account” button is pressed, it will lead to validation being run (seen on the right side of the flow chart) which, if successful, then leads to the closing of the “Create Account page” and the account being saved to the database. Each set of tiles represents a set of instruction which the code will follow and shows when the data base needs to be accessed to revive data to carry out the current process.

## Creation of the Main page:

All avalible user inputs into the Main page:

|  |  |  |
| --- | --- | --- |
| Main page | Search box | Search box will be used to help the user filter the jobs on the tree view and gather specific data from the database.  Highest to lowest and lowest to highest are used with the filter option to allow the user to filter the jobs which they are looking at. Highest to lowest and lowest to highest are also run though a check to see if both are selected.  The tree view selection will be used with applying, favouriting, and viewing extra information. |
| Highest to lowest (Tick box) |
| Lowest to highest (Tick box) |
| Filter option (Dropdown) |
| Tree view selection |
| Apply to job button | The apply to job button is used to apply for a job. It is done by selecting a job in the main tree view, then pressing the button which will save the application to the database. |
| View applicants’ button | The view applicants’ button is used to show data on the secondary tree. When the button is pressed, it will lead to the heading of the secondary tree changing to job title, applicant name, and contact details. Then it will populate the table with the relevant data. |
| Favourite job button | The Favourite job button is used to favourite a job. It is done by selecting a job in the main tree view, then pressing the button which will save the favourite to the database. |
| View favourites button | The View favourites button is used at show data on the secondary tree. When the button is presses, it will lead to the heading of the secondary tree changing to Job title, company name, and job description. Then would populate the table with the relevant data. |
| Upload job button | The upload job button is used to open the Upload job page. |
| Delete job button | The delete job button is used to delete jobs from the database. When the button is pressed, the code will go through all the tables (not including Login and User) and will delete all entries related to the job selected if the user is the one which placed the job, or the user holds an admin position. |
| Extra information button | The Extra information button will display the extra information from the job which was selected on the main tree. |
| Refresh tree button | The refresh tree button will be used to update the main tree. |
| Filter button | The filter button will be used to show a limited number of jobs on the main tree or allow the user to filter the job around selected parameters. |
| Save applicants | The save applicants button will give the user the ability to save the applicants of all their jobs which they have in the data base to an external file. |

All avalible outputs onto the Main page:

|  |  |  |
| --- | --- | --- |
| Main page | “Your job listing has been created” | To inform the user that their job listing has been created. |
| “Job has not been selected” | To inform the user that they have not selected a job from the main tree. |
| “Only a single filter option can be selected” | To inform the user that only one filter option can be selected. |
| “This is not your listing” | To inform the user that the job that they are trying to delate has not been created by them. |
| Main tree View | Allows the user to see all the available jobs currently on the system. |
| Secondary tree view | Allows the user to see their favourites and, if the user has a job, ups the ability to see who has applied for the position. |
| “Do you want to return to the login page”.  (Options yes or no) | Give the user the ability to go back on their decision to close the page. |
| Applicant file | To let the use save the details of their applicant to an external file. |

Table

Description automatically generated

This is an example of the desired outcome for the “Main” page. It provides a clear and simple way for the user to look at existing job listings and complete many other functions, such applying to a job which they have selected. This page has been made in mind of both an employer and a member of the public looking for a job. It allows both to have easy access to their respective functions (using functions related to the employer will not affect any function related to finding a job as it allows the user to swich between other user types without problems or several accounts).

The window has been created using several buttons to allow the user to acsess a range of functions:

1. The ability to favrouite job.
2. The ability to apply to a job.
3. The ability to view a users favrouited job on the secondery tree.
4. The ability to view the applicants of all job lisings created but to curent logged in user on the secondery tree.
5. The ability to make a new job lsiting.
6. The ability to refresh the main tree.
7. The ablility to view the extra infrmation saved for a job listing.
8. the ablility to close the page.
9. The ability to delete any job lisitng made by the currently logged in user unless the user us a admin.
10. The ablilty to filter the data in the main tree
11. The ablilty to save the the applicants of all job lisings created but to curent logged in user into a external file.

The use of the two tree view is so several section of data can be looked at at one and relevent data can be accsessed when it is wanted.

Also, error messages have been created to help inform the user on any failes in validation and the reson for the fail so the user can correct there mistake.

Diagram

Description automatically generated

This flow chart represents the basic function of the “Main” page and what will be carried out when a user interacts with the page. For example, when the “View Applicants” button is pressed, all applications of the jobs, which the user is the creator of, will be gathered and printed on the secondary tree. Each set of tiles represents a set of instruction which the code will follow, showing when the database needs to be accessed to retrieve data to carry out the current process.

## Creation of the Upload job page:

All avalible user inputs into the Upload job page:

|  |  |  |
| --- | --- | --- |
| Upload job page | Company name | All inputs will be used in input checks to make sure the user entered the correct type of information. If all validation is passed, a new job will be created using the user inputs. It will save to the data base, where adding a job page will be closed and user will return to the main page.  The account page will return to the login page. |
| Job title |
| Age requirement |
| Location |
| Hours |
| Pay |
| Job description |
| Extra information  (If chosen) |
| Upload job button | The upload job button will run validation on user inputs (as seen above) then, if all validation is passed, it will lead to the account being created by adding the job details to the data base, closing the upload job page, and returning to the main page. |
| Close page button | The close page button is used to close the upload job page and return to the main page. |

All avalible outputs onto the Upload job page:

|  |  |  |
| --- | --- | --- |
| Upload job page | “Fill all entry boxes” | To inform the user if not all input boxes are filled. |
| “Use numbers only for age” | To inform the user age should only consist of numbers. |
| “Postcode shouldn’t have a space” | To inform the user postcode should not have a space when entered. |
| “Use numbers only for hours” | To inform the user hours should only consist of numbers. |
| “£ is not needed for Pay” | To inform the user postcode should not have a “£” when entered. |
| “Use numbers only for pay” | To inform the user pay should only consist of numbers. |

Table

Description automatically generated

These are examples of the desired outcome for the “Upload Job” page. It provides a clear and simple way for the user to place a job listing.

This window has been created using 2 buttons to allow the user to close this page, and create their job listing. The error message will not show unless the user has failed a validation check when making the job. The use of the tick box (seen on the left screen) is used to allow the user to enter extra information about their job listing which would create an entry box once the tick box has been selected.

Diagram

Description automatically generated

This flow chart represents the basic function of the “Upload job” page and what will be carried out when a user interacts with the page.

This can be seen when the “Upload job” button is pressed. It will lead to validation to being run (seen on the right side of the flow chart) which, if successful, then leads to the closing of the “Upload job” and the account being saved to the database. Each set of tiles represents a set of instruction which the code will follow and shows when the data base needs to be accessed to retrieve data to carry out the current process.

## Class relationship diagram:

Diagram

Description automatically generated

The black diamonds represent composition between the classes. Composition is a design technique in Object Oriented Programming which implements a relationship between objects. Composition in Python is achieved by using instantiation to call the new call. Due to the composition relationship, if the main window is closed, it will not close the login window, but if the login window is closed, the main window will close too.

## Database design:

User Table

|  |  |  |  |
| --- | --- | --- | --- |
| Attributes | | Data Type | The User Table will be used to hold user/account details with have a level of encryption surrounding the Password attribute. |
| UserID | PK | Integer |
| Name | | Text |
| Email | | Text |
| Age | | Integer |
| Admin | | Integer |
| Username | | Text |
| Password | | Text |

Job Table

|  |  |  |  |
| --- | --- | --- | --- |
| Attributes | | Data Type | The Job Table will be used to hold details related to Job. This will be linked to the User table as the UserID of the account that uploads the job will be saved with each Job. Also, extra information will have the ability to not hold any information if the User doesn’t select the option upon creation of the job. |
| JobID | PK | Integer |
| CompanyName | | Text |
| JobTitle | | Text |
| Location | | Text |
| Hours | | Integer |
| Pay | | Integer |
| Age | | Integer |
| JobDescription | | Text |
| Extra Information | | Text |
| UserID | | Integer |

Favourites Table

|  |  |  |  |
| --- | --- | --- | --- |
| Attributes | | Data Type | The Favourites Table will be used to save favourites of user which use the function. There will be a link to the Job and User table, as the UserID is saved for the user when they favourite it with the Job. |
| FavouriteID | PK | Integer |
| JobID | | Integer |
| UserID | | Integer |

Applicant Table

|  |  |  |  |
| --- | --- | --- | --- |
| Attributes | | Data Type | The Applicant Table will be used to save applications of user which use the function. There will be a link to the Job and User table as the UserID as saved of the user when the application is made with the Job which was selected by the user. |
| ApplicantID | PK | Integer |
| JobID | | Integer |
| UserID | | Integer |

All PK’s have been set up to autoincrement when data is added to the database to allow for there to be a unique identifier for each section of data.

Pk stands for primary key.

## Database relationship diagram:

Diagram, table

Description automatically generated with medium confidence

This is a diagram which shows the relationship between the different tables in the database. There is a type of relationships represented in the diagram. A one-to-many relationship seen between the User and Job Table.

A one-to-many relationship refers to a relationship between two tables, such as User and Job, in which an element of User may be linked to many elements of Job, but a member of Job is linked to only one element of User.

These can be identified on the diagram as the straight line with 3 lines coming of one side is a one-to-many relationship.

## Databases input and outputs:

Login Page:

The Login page don’t have any inputs into the database.

|  |  |  |  |
| --- | --- | --- | --- |
| Data gathered | Parameters on data recovery | What is done with the data | Table gathered from |
| Password | Password is gathered based on the Username which the user entered. | Used in decryption of Password, then used to see if the Password entered by the user is correct. | User |

Create an account page:

|  |  |  |  |
| --- | --- | --- | --- |
| Data inputted | How data was gathered | Valid example inputs | Table inputted into |
| Username | User input | MrTeacher | User |
| Password | “” | Noonewillknow | “” |
| Name | “” | Teacher | “” |
| Email | “” | Techer@email.com | “” |
| Age | “” | 32 | “” |
| Admin | Entered based on if the admin was chosen and the correct code was entered. | EGTHM06 | “” |

|  |  |  |  |
| --- | --- | --- | --- |
| Data gathered | Parameters on data recovery | What is done with the data | Table gathered from |
| Username | Username is gathered based on the Username which is entered. | Used to see if an account has been created with the entered username. | User |

Main page:

|  |  |  |  |
| --- | --- | --- | --- |
| Data inputted | How data was gathered | Valid example inputs | Table inputted into |
| JobID | Job which has been selected by user |  | Favourites |
| UserID | Saved in program |  | “” |
| JobID | Job which has been selected by user |  | Applicants |
| UserID | Saved in program |  | “” |

|  |  |  |  |
| --- | --- | --- | --- |
| Data gathered | Parameters on data recovery | What is done with the data | Table gathered from |
| FavouriteID | FavouriteID which are linked to the current User logged in | Used to control the number of times the Gather commands loop to make sure all data is gathered | Favourite |
| JobID | Gathers Favourites which are linked to the account which is user logged into | Allows the code to find the job in which the details are needed | “” |
| JobTitle | Gathered based on the job which has been selected | The data gathered is placed on the secondary tree view for the user to see | Job |
| CompanyName | “” | “” | “” |
| JobDescription | “” | “” | “” |
| ApplicantID | ApplicantID which are linked the current User logged in | Used to control the number of times the gather commands loop to make sure all data is gathered | Applicants |
| JobID | Gathers Applicants which are linked to the account which the user is logged in on | Allows the code to find the job in which the details are needed | “” |
| JobTitle | Gathers based on the job which has been selected | The data gathered is placed on the secondary tree view for the user to see | Job |
| Name | Gathers user detail which have applied to the job which has been selected | “” | User |
| Email | “” | “” | “” |
| Name | Gathered based on user which is logged in | Used to display the name of the user who is currently logged in | “” |
| JobID | Gathers the number of JobID in the Job table | Used to control the number of times the Gather commands loop to make sure all data is gathered | Job |
| CompanyName | Gathered based on the JobID which has been selected in the loop | The data gathered is placed on the main tree view for the user to see | “” |
| JobTitle | “” | “” | “” |
| Location | “” | “” | “” |
| Hours | “” | “” | “” |
| Pay | “” | “” | “” |
| Age | “” | “” | “” |
| JobDescription | “” | “” | “” |
| JobID | “” | The data is gathered to store in an invisible column in the main tree view, so when a job is selected it can be identified | “” |
| UserID | Gathered based on the job selected by the user | The data is used to check to see if the current user has permission to delete the selected job. | User |
| ExtraInformation | Gathered based on the job which has been selected by the user | Data is gathered to be able to display the extra information of the selected job to the user | Job |
| Admin | Gathered based on the user which is currently logged in | Used to see if the user which is logged in is an admin | user |

|  |  |  |  |
| --- | --- | --- | --- |
| Data effected | How data is selected | Parameters around deletion | Tables effected |
| All information related to job which is being deleted | Data is chosen if the user which is logged in is the owner of the job listing. | JobID | Job |
| “” | “” | “” | Favourites |
| “” | “” | “” | Applicants |

Upload job page:

|  |  |  |  |
| --- | --- | --- | --- |
| Data inputted | How data was gathered | Valid example inputs | Table inputted into |
| CompanyName | User input | Natwest | Job |
| JobTitle | “” | Project manager | “” |
| Location | “” | SW01 IZT | “” |
| Hours | “” | 40 | “” |
| Pay | “” | 27 | “” |
| Age | “” | 18 | “” |
| JobDescription | “” | You will be overseeing current projects and working with other heads on next step decisions. | “” |
| ExtraInformation | If the option selected is User input, but if not selected the data entered will be “NULL” | User will be expected to provide their own equipment. | “” |
| UserID | Saved in program |  | “” |

Upload job does not gather any data from the database.

## Data validation:

Login page

|  |  |  |
| --- | --- | --- |
| Validation | If Invalid | If validated |
| Entry boxes are filled. | Error message | Next check |
| Username and Password are at least 6 characters long. | “” | “” |
| Check if the username and password are correct (using database). | “” | Open Main window. |

Create account page:

|  |  |  |
| --- | --- | --- |
| Validation | If Invalid | If validated |
| Entry boxes are filled | Error message. | Next check. |
| Username and Password are at least 6 characters long. | “” | “” |
| Check if email is valid. | “” | “” |
| Check if name only consists of letters. | “” | “” |
| If chosen check if admin code is correct. | “” | “” |
| Check if username is free (using database). | “” | Save account details to database. |

Main page:

|  |  |  |
| --- | --- | --- |
| Validation | If Invalid | If validated |
| Entry boxes are filled | Error message. | Will carry on with the filter function. |
| Job selected | “” | Collect the Job ID of selected job. |
| Single filter selected | “” | Will carry on with the filter function. |

Upload job page:

|  |  |  |
| --- | --- | --- |
| Validation | If Invalid | If validated |
| Entry boxes are filled | Error message. | Next check. |
| Username and Password are at least 6 characters long. | “” | “” |
| Check if the Age requirement consist only of Integers. | “” | “” |
| Check if location is valid. | “” | “” |
| Check if Hour only consists of Integers. | “” | “” |
| Check if pay is valid. | “” | Saves job to database. |

## Research:

Research was completed for the main page to be able to implement a drop-down feature to assist with the filter process. After looking into possible solutions using buttons or an entry box (both seem to be inefficient for the intended purpose), a decision was made. The Tkinter widget, OptionMenu, would give a dropdown feel to the user. It would also allow a recovery of the selected option when needed and would display the selected option to the user once chosen.

During the creation of the input validation, it was brought to my attention that the method of checking if inputs consist of only numbers and/or letters would need to be reduced. This is because the current solutions to the problem consisted of each character of an input being compared to a list of letters or numbers, to make sure that the inputs were valid. After researching ways to increase the efficiency and the file size, methods, such as seeing if values would successfully be used in an equation to validate a number, were used. The discovery of three inbuilt methods in Python is Numeric (checks if an input only consists of numbers), Isalpha (checks if an input only consists of letters, and Isalnum (checks if an input consists of only letter and numbers) allows the input validation to be shortened to one line. This decreases the file size and increases the efficiency.

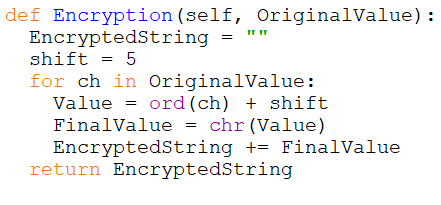
During the creation of the Create account page and Upload job page, the user wanted the function to select to add extra information. The use of a button or another page of the extra information was chosen, but after investigation it was deduced to be inefficient and not the best possible solution to the problem. After further research, the discovery of the Tkinter widget, Checkbutton, would be used, as it would be closed once selected and replaced by an entry box. This would help with the functionality of the page and would make the function more comprehensible to the user.

During the creation of the application, the problem arose around classes being instantiated in the previous class. This led to previous pages having to be kept open (this can be seen as the Login page must be kept open when Main page is in uses). Thusly, research and planning were performed to find solutions for this problem. The research led to the idea of making all classes static, meaning that they can be called without instantiation. However, it would also mean that virtual methods or implementing an interface would not be supported. This would significantly decrease the quality of the application below exemptible standards. Planning into a viable solution opened the idea as a main class, in which all pages would be called on so that it would never lead to pages not being instantiated on each other. This was seen to be an inefficient option as the file size would be greatly increased, leading to a larger power requirement to run the application, and was consequently too complicated for the time scale of the project.

During the planning phase of the project, research was done into different types of encryptions. Chaser Cipher was chosen to help encrypt the Password of the user before being saved into the database. After development, a solution was created (see below) but after testing and evaluation it was seen to be reducing the possible level of encryption. It would be the optimal solution if the saved answer had to stay as only letters and number, however symbols can be used when saving the encrypted data to the database. This would make it harder for a person to use brute force to break the encryption as 9 encrypts to >. Thusly, the code was simplified to the second shown solution.

Text

Description automatically generated



Testing

## Testing outputs and inputs:

All validation will be done with the Test account and the test job listing, so the details of the account and job can be expected.

|  |  |
| --- | --- |
| Test account details | Test job details |
| UserID = 2 | JobID = 3 |
| Name = TestName | Company name = Test.co |
| Email = [TestEmail@Test.com](mailto:TestEmail@Test.com) | JobTitle = TestCoder |
| Age = 37 | Location = SW11GBF |
| Admin = 0 (not an admin account) | Hours = 10 |
| Username = TestUsername | Pay = 26 |
| Password = TestPassword | Age = 22 |
|  | Job description = Test description |
|  | Extra information = Test extra information |
|  | UserID = 2 |

Login page:

Graphical user interface, application

Description automatically generatedDiagram

Description automatically generated

The login window has been created with every desired feature and is made in the same layout of the original wireframe but is seen to be more visually pleasing to the user and laid out in a more professional format.

Testing Inputs:

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Reason for Entry | Expected outcome | Screenshot of outcome |
| The Sign in button.  Username: TestUsername  Password:  TestPassword | To login into the Main page | The main page will open when then valid username and password have been entered. | [Successful]  Proved as the user’s name has been displayed on the top left of the main page |
| The Create Account button. | The open the Create account page. | The Create account page will open. | Before the button is presse:    After the button is pressed:    [Successful] |
| The Exit button | To open a widget that askes the close the application | A widget being displayed to the user and when the yes button is presses on the widget it leads to all pages being closed and the code stops running. | [Successful] |

Testing outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Reason for Entry | Expected outcome | Screenshot of outcome |
| Username: TestUsername  Password: “” | Missing data | An error message will be displayed saying “Fill all entry boxes”. | [Successful] |
| Username: “”  Password: TestPassword | “” | “” | [Successful] |
| Username: Test  Password: TestPassword | Username entered data is under 6 characters long | An error message will be displayed saying “Minimum 6 characters for username” | [Successful] |
| Username: TestUsername  Password: Test | Password entered data is under 6 characters long | An error message will be displayed saying “Minimum 6 characters for password” | [Successful] |
| Username: TestUser  Password: TestPassword | Valid username and password have been entered | An error message will be displayed saying” Username or Password is incorrect” | [Successful] |
| Username: TestUsername  Password: TestPass | “” | “” | [Successful] |
| Username: TestUsername  Password: TestPassword  Name: TestName  Email: [TestEmail@Test.com](mailto:TestEmail@Test.com)  Age: 37  (Into the create account page) | If an account is made successfully in the Create account page a conformation message will be printed | A message will be displayed to the user saying, “Your account has been created”. | [Successful] |
| If a widget pops up to ask the user if they want to quite the application. | The quit button has been pressed | A widget will pop up conferring with the user if they want to close the application | [Successful] |

“Create account” page:

Graphical user interface

Description automatically generated Graphical user interface

Description automatically generatedGraphical user interface, diagram

Description automatically generated

The Create account window has been created with all wanted features and is made in the same layout of the original wireframe but is seen to be more visually pleasing to the user and laid out in a more professional formatted.

Testing Inputs:

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Reason for Entry | Expected outcome | Screenshot of outcome |
| The admin tick box that will show an entry box when selected. | To choose the option of an admin account | An account should be created with an admin value of 1. | Before tick box has been pressed:    After tick box has been pressed:    [Successful] |
| Create account button presses.  Username: TestUsername  Password: Testpassword  Name: TestName  Email: [TestEmail@Test.com](mailto:TestEmail@Test.com)  Age: 37  Admin: (not selected) | The create account button being pressed. | The create account page will be closed and a message reading “your account has been created”, should be displayed on the login page | Account details filled in to make new account:    New database entry for the account details:    The account creation confirmation on the login page:    [Successful] |
| Create account button presses.  Username: Test99  Password: Test99  Name: test  Email: test@  Age: 66  Admin: EGTHM06 | “” | “” | Account details filled in to make new account:    New database entry for the account details:    The account creation confirmation on the login page:    [Successful] |
| The return to login button pressed | The return to login button being pressed. | A widget being displayed to the user and when the yes button is presses on the widget it leads to all pages being closed and the code stops running. | [Successful] |

Testing outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Reason for Entry | Expected outcome | Screenshot of outcome |
| Username: “”  Password: TestPassword  Name: TestName  Email: [TestEmail@Test.com](mailto:TestEmail@Test.com)  Age: 37  Admin: (not selected) | Missing data | An error message will be displayed saying “Fill all entry boxes”. | [Successful] |
| Username: TestUsername  Password: “”  Name: TestName  Email: [TestEmail@Test.com](mailto:TestEmail@Test.com)  Age: 37  Admin: (not selected) | “” | “” | [Successful] |
| Username: TestUsername  Password: TestPassword  Name: “”  Email: [TestEmail@Test.com](mailto:TestEmail@Test.com)  Age: 37  Admin: (not selected) | “” | “” | [Successful] |
| Username: TestUsername  Password: TestPassword  Name: TestName  Email: “”  Age: 37  Admin: (not selected) | “” | “” | [Successful] |
| Username: TestUsername  Password: TestPassword  Name: TestName  Email: [TestEmail@Test.com](mailto:TestEmail@Test.com)  Age: “”  Admin: (not selected) | “” | “” | [Successful] |
| Username: TestUsername  Password: TestPassword  Name: TestName  Email: [TestEmail@Test.com](mailto:TestEmail@Test.com)  Age: 37  Admin: “” (selected) | “” | “” | [Successful] |
| Username: Test  Password: TestPassword  Name: TestName  Email: [TestEmail@Test.com](mailto:TestEmail@Test.com)  Age: 37  Admin: (not selected) | Username entered is under 6 characters long | An error message will be displayed saying “Minimum 6 characters for username”. | [Successful] |
| Username: TestUsername  Password: Test  Name: TestName  Email: [TestEmail@Test.com](mailto:TestEmail@Test.com)  Age: 37  Admin: (not selected) | Password entered is under 6 characters long | An error message will be displayed saying “Minimum 6 characters for password”. | [Successful] |
| Username: TestUsername  Password: Test  Name: TestName  Email: TestEmail  Age: 37  Admin: (not selected) | Email entered doesn’t contain a @ symble. | An error message will be displayed saying “Email not valid”. | [Successful] |
| Username: TestUsername  Password: Test  Name: TestName  Email: [TestEmail@Test.com](mailto:TestEmail@Test.com)  Age: eh  Admin: (not selected) | Age has been entered with incorrect data type. | An error message will be displayed saying “Age should only consist of numbers”. | [Successful] |
| Username: TestUsername  Password: TestPassword  Name: TestName  Email: [TestEmail@Test.com](mailto:TestEmail@Test.com)  Age: 37  Admin: SDEL (Selected) | The entered admin code has been entered incorrectly. | An error message will be displayed saying “Admin code is incorrect”. | [Successful] |
| Username: Test99  Password: TestPassword  Name: TestName  Email: [TestEmail@Test.com](mailto:TestEmail@Test.com)  Age: 37  Admin: (not selected) | The username the has been entered is already taken by another user. | An error message will be displayed saying “Username is taken”. | [Successful] |
| If a widget pops up to ask the user if they want to return to the login page. | The return to login button has been pressed | A widget will pop up conferring with the user if they want to return to the logon page. | [Successful] |

“Main” page:

Graphical user interface, text

Description automatically generated with medium confidenceTable

Description automatically generated

The Main window has been created with all desired features and is similar to the layout of the original wireframe, but it is seen to be more visually pleasing to the user and laid out in a more professional formatted.

Testing Inputs:

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Reason for Entry | Expected outcome | Screenshot of outcome |
| The filter option (dropdown) | To help the user pick if they want to filter around a certain detail about a job. | A dropdown with several options that the user can pick from. | Before interaction:    After interaion:      Also works for all other option in the drop down (shortest and longest options have been displayed above).  The option which has been chosen can be retrived using the code:  self.MainPageDropdown.get()  (as seen in the filter methord in the Main\_Window class)  [Successful] |
| Selection of a job of the main tree. | To allow the user to select a job from the main tree to complete a function with. | A job listing can be selected on the main tree turning to selected. | When a job is not slected:    When a job is slected:    Then the SelectedJob methord in the Main\_Window class can be used to retrive the JobID of the seclected job.  [Successful] |
| The applying to a job button. | To allow the apply a selected job listing. | After a job is selected and the button is pressed, the application should be saved to the database with the correct UserID and JobID. | When the TestUsername user appliaes to the Natwest job listing this is saved to the database:    [Successful] |
| The View applicants’ button. | To allow the user to view the applicants to the job listings in the secondary tree. | The applications to all the job listings which the user has made will be displayed on the secondary tree (the job title, applicant name, and contact details). | The test will be done using the test99 user with the the TestUsername user having applied to the two job listings which have been created.  The data in the database:    The data displayed on the secondery tree when the View applicants’ button is pressed:    [Successful] |
| The favourite job button. | To all the user to save a job listing as a favourite. | After a job is selected and the button is pressed, the favourite should be saved to the database with the correct UserID and JobID. | When the TestUsername user favriouts the Natwest (JobID of one) job listing it is saved to the database:    (seen by the second line)  [Successful] |
| The View favourites’ button. | To allow the user to view there favourited job listings in the secondary tree. | The user’s favourited job listings will be displayed on the secondary tree (the job title, company name, and the job description. | The data in the dataabse:    The data displayed in the tree view:    [Successful] |
| The upload job button. | Open the Upload job page. | The Upload job page should be opened. | When the button is presses the upload job page open:    [Successful] |
| The delete job button.  (Job listing owned by user) | To allow the user to delete a job listing from the data base. | All aspects related to the selected job will be deleted from the database. | Main tree view before deleting a job:    If the TestUsername user deletes the TestCoder job listing:    [Successful] |
| The delete job button.  (Job listing is not owned by the user) | “” | An error message will be displayed saying “This is not your job listing”. | If a user tries to delete a job which is not these:    [Successful] |
| The extra information button. | To allow the user to view the extra information of a job. | A widget will be displayed showing the user the extra information of the selected job. | If a user slected the extra information for the Test.co job lisitng:    [Successful] |
| The refresh tree button. | To allow the user to refresh the main tree | The main tree should be refreshed with any changed to available job listings. | Before the refresh button is pressed:    After a job is added to the daatabse and the refresh button is pressed:    [Successful] |
| The filter button. | To allow the user to filter the job listings on the main tree. | The job listings should be filtered by the option which the user selects. | [partially successful]  Due to this being related to a secondery funtion only parshul sections of the code works. |
| The save applicants’ button. | To allow the user to save the applicants to their job listing to an external file. | The applicants to all the job listings which the user has made will be saved to an external file with the name “Applicants.txt” (the information saved will be job title, applicant name, and contact details). | Then the button is presses the file will be created and saved in the location of the code:    The data in the file will match the data which is displayed in the secondery tree when the view applicants’ button is pressed. (As seen below):    [Successful] |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Reason for Entry | Expected outcome | Screenshot of outcome |
| The creation of a job listing. | If a job listing is made successfully in the Upload job page a conformation message will be printed | A message will be displayed to the user saying, “Your job listing has been created”. | When the job is added it will be added to the main tree one the refresh tree button is pressed.    [Successful] |
| Job that has been selected on the main tree | A function which uses a selected job will cause an error to run if no job is selected. | An error message will be displayed saying “Job has not been selected”. | [Successful] |
| Highest to lowest: Selected  Lowest to highest: Selected | If both tick boxes are selected. | An error message will be displayed saying “Only a single filter option can be selected”. | [Successful] |
| Job the user has chosen to delete. (Not a listing of the user) | When the user tried to delete job listing which they do not own. | An error message will be displayed saying “This is not your listing”. | [Successful] |
| The job listings on the main tree view |  | All job listings to be displayed on the main tree view. | The job listings being displayed to the user.    Job listings in the database:    [Successful] |
| The correct job listings/ applicant details on the secondary tree view. | When the user presses the View favourites’ button. | All user favourites job listings to be displayed on the secondary tree view. | The data in the dataabse:    The data displayed in the tree view:    [Successful] |
| “” | When the use presses the View applicants’ button. | All user applications to their job listings to be displayed on the secondary tree view. | The test will be done using the test99 user with the the TestUsername user having applied to the two job listings which have been created.  The data in the database:    The data displayed on the secondery tree when the View applicants’ button is pressed:    [Successful] |
| If a widget pops up to ask the user if they want to return to the login page. | The return to login button has been pressed | A widget will pop up conferring with the user if they want to return to the logon page. | [Successful] |
| If the code can put a user’s applicants on an external file. | Then the Save application button is pressed. | All a user’s applications saved to an external file. | Then the button is presses the file will be created and saved in the location of the code:    The data in the file will match the data which is displayed in the secondery tree when the view applicants’ button is pressed. (As seen below):    [Successful] |

“Upload” Job page:

Graphical user interface, application

Description automatically generated Graphical user interface, application

Description automatically generatedTable

Description automatically generated

The Upload job window has been created with all the desired features and is made in the same layout of the original wireframe but is more visually pleasing to the user and is laid out in a more professional formatted.

Testing Inputs:

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Reason for Entry | Expected outcome | Screenshot of outcome |
| The extra information tick box that will show an entry box when selected. | To choose the option of extra information for a job listing. | A job should be created with data in the extra information attribute on the table (not Null means it was not selected). | Before the tick box has been selected:    After the tick box has been selected:    [Successful] |
| The upload job button. | To allow the new job listing to be saved to the database. | The new job should be added to the database with the message displayed onto the main screen telling the user the job has been saved. | Job details being entered:    Job saved in databse:      Message on the main page:    [Successful] |
| The close page button. | To allow the user to return to the main page. | The Upload job page will close, and it will return to the main page. | [Successful] |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Reason for Entry | Expected outcome | Screenshot of outcome |
| Company name: “”  Job title: TestCoder  Age requirement: 22  Location (passcode): SW11GBF  Hours: 10  Pay (per hour): 26  Job description: Test description  Extra information: (not selected) | Missing data | An error message will be displayed saying “Fill all entry boxes”. | [Successful] |
| Company name: Test.co  Job title: “”  Age requirement: 22  Location (passcode): SW11GBF  Hours: 10  Pay (per hour): 26  Job description: Test description  Extra information (not selected) | “” | “” | [Successful] |
| Company name: Test.co  Job title: TestCoder  Age requirement: “”  Location (passcode): SW11GBF  Hours: 10  Pay (per hour): 26  Job description: Test description  Extra information: (not selected) | “” | “” | [Successful] |
| Company name: Test.co  Job title: TestCoder  Age requirement: 22  Location (passcode): “”  Hours: 10  Pay (per hour): 26  Job description: Test description  Extra information: (not selected) | “” | “” | [Successful] |
| Company name: Test.co  Job title: TestCoder  Age requirement: 22  Location (passcode): SW11GBF  Hours: “”  Pay (per hour): 26  Job description: Test description  Extra information: (not selected) | “” | “” | [Successful] |
| Company name: Test.co  Job title: TestCoder  Age requirement: 22  Location (passcode): SW11GBF  Hours: 10  Pay (per hour): “”  Job description: Test description  Extra information: (not selected) | “” | “” | [Successful] |
| Company name: Test.co  Job title: TestCoder  Age requirement: 22  Location (passcode): SW11GBF  Hours: 10  Pay (per hour): 26  Job description: “”  Extra information: (not selected) | “” | “” | [Successful] |
| Company name: Test.co  Job title: TestCoder  Age requirement: 22  Location (passcode): SW11GBF  Hours: 10  Pay (per hour): 26  Job description: Test description  Extra information: “”  (selected) | “” | “” | [Successful] |
| Company name: Test.co  Job title: TestCoder  Age requirement: 22  Location (passcode): SW11GBF  Hours: 10  Pay (per hour): hu  Job description: Test description  Extra information: (not selected) | Pay has been entered with incorrect data type. | An error message will be displayed saying “Use only numbers for pay”. | [Successful] |
| Company name: Test.co  Job title: TestCoder  Age requirement: 22  Location (passcode): SW11 GBF  Hours: 10  Pay (per hour): 26  Job description: Test description  Extra information: (not selected) | Location have been entered with a “ “ present. | An error message will be displayed saying “Postcode shouldn’t have a space”. | [Successful] |
| Company name: Test.co  Job title: TestCoder  Age requirement: sb  Location (passcode): SW11GBF  Hours: 10  Pay (per hour): 26  Job description: Test description  Extra information: (not selected) | Age has been entered with incorrect data type. | An error message will be displayed saying “Use only numbers for age”. | [Successful] |
| Company name: Test.co  Job title: TestCoder  Age requirement: 22  Location (passcode): SW11GBF  Hours: tz  Pay (per hour): 26  Job description: Test description  Extra information: (not selected) | Hour has been entered with incorrect data type. | An error message will be displayed saying “Use numbers only for hours”. | [Successful] |
| Company name: Test.co  Job title: TestCoder  Age requirement: 22  Location (passcode): SW11GBF  Hours: 10  Pay (per hour): £26  Job description: Test description  Extra information: (not selected) | Pay have been entered with a “£” present. | An error message will be displayed saying “£ is not needed for pay”. | [Successful] |

## Data base tactical solution:

Text

Description automatically generated

The first two lines are used to connected to the database unless it is not present in the users’ files where one will be created.

The rest of the code displays the creation of the tables and their attributes in the database (will only create is the tables or attributes are not present in the users database file).

## Login page technical solution:

Text

Description automatically generated

This represents the creation of the Login page, starting with the creation of the window (scale and name). Next, the creation of the labels (used to tell the user what each entry bock is linked to), entry boxes (used to gather user input), company logo (gathered from the user files and displayed), and the buttons (used to allow the user to proceed with processing the application and call new windows if needed).

Text

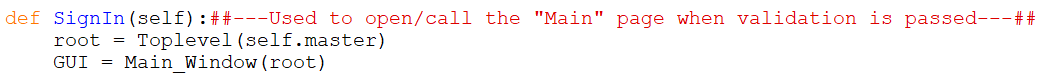
Description automatically generated with medium confidence

Next, in the technical solution, is the input validation which is triggered when the user presses the “Sign in” button. It starts by gathering the inputs which the user has entered then saved in local variables. Then, the code checks if there is data populating the entry boxes and makes sure that they are longer than 6 characters long. If any of the checks are failed, then the relevant error messages will be displayed to the user. If all checks are passed the next step of the validation is called, as seen below.

Text, application

Description automatically generated

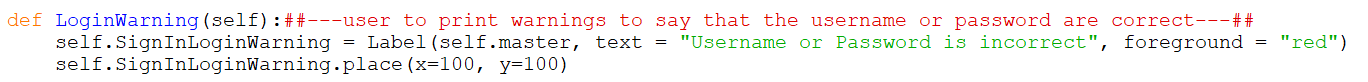
This section of code will be used validate the users’ login details. This is done by entering the try loop and using a statement to call to the data base and request for the password and the user’s unique ID based on the username the user has entered. The reason for the try loop at this point is so that, if the line “DatabaseVaue = cursor.fetchall()” runs and causes an error, it will return to telling the code that there is no saved account linked to the username which the user entered. This causes the exempt section of the loop to run, leading a relevant error message to be displayed using the “LoginWarning” method. The next line in this section of code shows the “Decryption” method (which is seen below) being called to remove the encryption on the password that is pulled back from the data base and saved in a local variable. Then, the decrypted password is checked against the password which the user entered and if they match the “Main” page will be opened (using the “SignIn” method) and the user’s unique ID is saved to a global variable “UserID”. If the password doesn’t match a relevant error message will be displayed using the “LoginWarning” method.



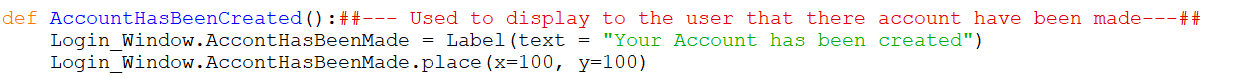
This section of the final solution is called if all the validation during the sign in process is passed and is used to open the “Main” page by calling the “Main\_Window” class using instantiation.



This section of the final solution is called if the Create account button is pressed and is used to open the “Create account” page by calling the “Create\_Account\_Window” class using instantiation.



This section of code called during the validation process (in the “ValidateSignInDetails” method) informs the user that they have entered an incorrect username or password.



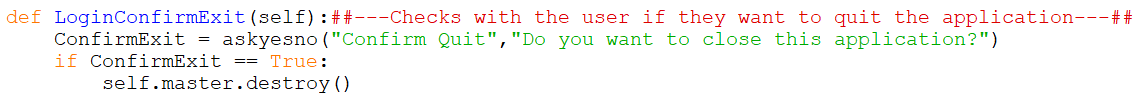
This section of the final section is used to inform the user that their account has been created. It is called after the user’s new account details have been saved to the database (In the “SaveAccountDetails” method of the “Create\_Account\_Window” class).

A picture containing text

Description automatically generated

This is the decryption which is in the input validation (to see if the password is correct) as the saved password will need to be converted back to its original sate to allow the process to take place.

The decryption works by looping through each character of the password and finding the Unicode for each character (using the ord function) then decreasing them by 5 (a hard coded value) to return them to their original value. It then places them back in a string to give the decoded password.

Unicode, formally The Unicode Standard, is an informational technology standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems. 

The “loginConfirmationExit” method is run when the quit button is pressed. When run, it displays a small conformation widget asking the user if they are sure they want to quit the application. If the user presses the enter key or selects the yes button, the application will be closed. This is unlike if the user presses back, as this will cause no change in the code, and it will repeat to carry on like normal.

## Create account page technical solution:

Text

Description automatically generated

This represents the start of the creation of the Create account page which is called to run when the “Create account” button is pressed in the “Login” page. Starting with the creation of the window (scale and name). Next, the creation of the labels (used to tell the user what each entry bock is linked to), and company logo (gathered from the user files and displayed).

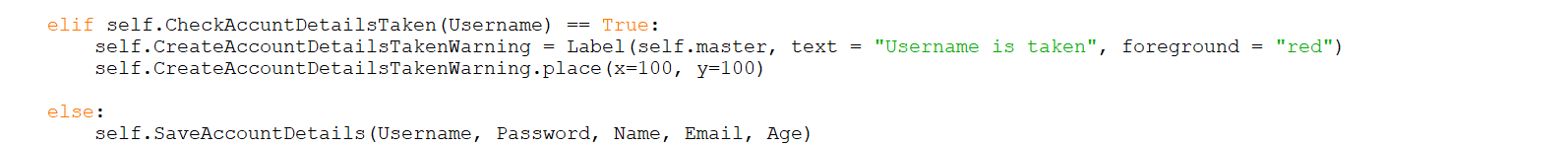
Graphical user interface, text, application

Description automatically generated

This is the second section of creation of the Create account page starting with the creation of the entry boxes (for the user to enter their account details), buttons (used to allow the user to proceed with process of the application and close the window is needed), and variables which are used through the “Create\_Account\_Window” class.

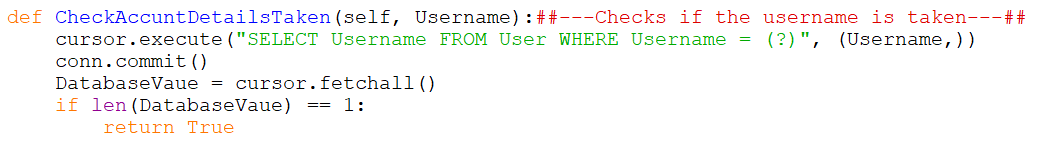
Text

Description automatically generated

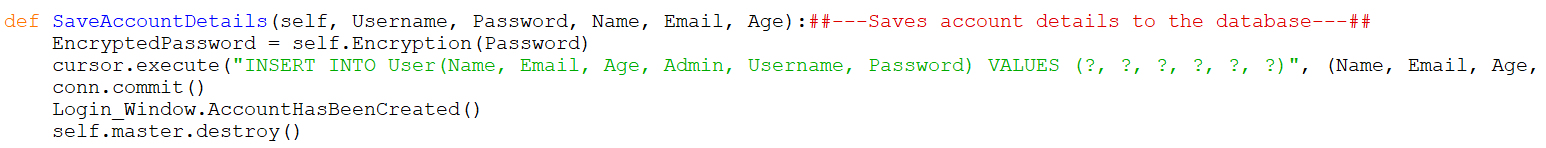


Next, in the technical solution is the input validation which is triggered when the user presses the “Create account” button.

It starts by gathering the inputs which the user has entered then saves it in local variables but, when gathering the user input for the admin code, there is first a check to see if the option selected is of an admin account. If not, and the gather function is run, the code will crash. Then, the code checks if there is data populating the entry boxes and makes sure that the Username and Password are longer than 6 characters long. Checks are then run to see if the @ symbol is present in the email input, check to make sure the age requirement only consists of numbers with a check if correct admin code has been inputted (check will only run if the user selected the admin option). The input validation will end with a check to see if the username which the user has entered has already been taken through the “CheckAccountDetailsTaken” method, with the inputted username entered as an argument. If any of the checks are failed, then a relevant error message will be displayed to the user. If all checks are passed the “SaveAccountDetails” method (as shown below) is called to save the user’s account.



The “CheckAccountDetailsTaken” method is called during the input validation to see if the username the user has entered is taken or not. The method does this by taking the argument (the username with the user inputted) which was entered and using it is a SQL statement to see if the account exists. The test works by seeing if the username is in the database. It attempts to return the entered username if a value is returned or if the username exists. If nothing is returned, the username is not taken. This is used in the if loop it looks at the number of values in the returned list. If the username is returned, then it is returned to the check (which is causes the relevant error message to be displayed to the user).





The “SaveAccountDetails” method is called if all validation is passed during the input validation. The method will receive all the user inputs in the form of arguments from the validation. The method will start with a call to the “Encryption” method with the argument of the user inputted password that will return an encrypted version of the password, which will be saved to the database to add a level of security. The account is then saved to the database with the arguments used for the account username, name, email, and age. The password will be saved using the encrypted password and the admin will be saved using a variable from the constructor (will be 1 of the admin verification passwords and 0 if not). Finally, the “AccountHasBeenCreated” method in the “Login\_Window” will be called and the “Create account” page will be closed.

A picture containing text

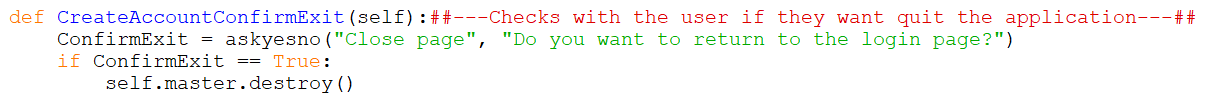
Description automatically generated

The “Encryption” method is called during the “SaveAccountDetails” method to encrypt the user’s password. The method starts with creating a blank string which is saved into the local variable “EncryptedString”, and the shift values are declared. Then a loop is run for each character in the password (brought in through the argument and saved under the variable name “OriginalValue”). On each loop, the value of the character used on that loop is converted to its Unicode number and the shift value is added to it and saved into the local variable “Value”. Then the number saved in “Value” is converted into its character form and added to the end of the string saved in “EncryptionString”. Once all the loops have finished the value in “EncryptionString” is returned.

A picture containing scatter chart

Description automatically generated

The “AdminEntryOption” method is run then the admin tick box is selected. It starts by changing the “AdminAccountChosen” (a variable saved in the constructor) to equal 1 and remove the tick box from the screen. Then an entry box placed in the screen to allow the user to enter the admin code.



The final method in the “Create\_Account\_Window” class is the “CreateAccountConfirmExit” method which is called when the “Return to login” button is pressed. It is created to make sure that the user doesn’t want to exit the “Create Account” page and the click of the button was not a mistake. This is done by displaying a widget to the user (which reads “do you want to return to the login page?”) it gives the user the option to press yes or no. If yes is pressed, the if statement runs and the “Create Account” page is closed. If no is selected, the code goes back into a standby to wait for the next user interaction.

## Main page technical solution:

Graphical user interface, text, application

Description automatically generated

This represents the start of the creation of the Main page, starting with the creation of the window (scale and name). Next the creation of the three variables (if the highest to lowest or lowest to highest tick box been selected- will save as true or false) and this list which is used to help make the external file, which are used through the “Main\_Window” class. Following this is the creation of the labels (used to tell the user what each entry block is linked to and display the name of the user which is currently logged in), as well as entry boxes (used to gather user input to assist with the filter function.)

A picture containing text

Description automatically generated

This represents the next section of the constructor (this is used to make the Main page window) which holds the creation of all the buttons (used to allow the user to proceed with the process of the application and call the “Upload job” windows if needed) and the two tick boxes (used with the filler commands to help the user selected the order which the data on the main tree is displayed).

A picture containing application

Description automatically generated

This shows the creation of the drop down which is made using the Options (hard set in a list needed above the creation of the dropdown itself), which are then filled into the dropdown for the creation. It will start off displaying the work filter on the drop down (set by the 2nd line) which will then change to display the selected option by the user.

Text

Description automatically generated with medium confidence

The final part of the constructor is focused on the creation of the “Main” tree. This begins with declaring the instance of the tree into the variable “self.MainTree”. Then, the number of columns in the tree view with their size and data orientation (if the data is displayed in the centre, left or right of the box) are declared. The header for each column is declared, displaying to the user what the data in each column holds. Finally, the tree view is placed on the screen.

Then, the scrollbar for the main tree is created with its orientation and the tree view is linked to be declared before it is placed on the right side of the tree with its highest set to make it the same size as the main tree. This makes the application more appealing to the user.

The constructor ends with a call to the “AddingDataToMainTree” method (found below) which is used to populate the “Main” tree with all the job listings saved in the database. The “SecondaryTreeView” method (shown below) is called to insatiate the “Secondary” tree into the “SecTree” variable.

Text

Description automatically generated

The secondary tree is seen to have 3 uses in the code which can be changed based on the argument (data which is pulled into a method) which the “SecondaryTreeView” method is called with. If the method is called with the argument (variable name in the code “SecTOption”) as 0 (seen in the constructor) it will lead to headings being declared with nothing in them, and no data will be outputted. If the argument is 1 (when the “View favourites’” button is pressed) it will lead to new headings being declared with relevant titles to information of jobs which have been favourited by the user and data will be displayed onto the “Secondary” tree view. If the argument is 2 (when the “View applicants’” button is pressed) it will lead to new headings being declared with relevant titles to information of the applicants to the users’ job listings and data will be displayed onto the “Secondary” tree view.

This method at the end of the constructor calls to declare and place the “Secondary” tree on the “Main” page. This begins by declaring the instance of the tree into the variable “SecondaryTree”. Following this, the number of columns with their size and data orientation (if the data is displayed in the centre, left or right of the box) are declared.

Then, the header for each column is declared based on the argument of the tree allowing it to adapt to its current use, seen as the if statements are used to declare the desired headers.

The scrollbar for the Secondary tree is created with its orientation and the tree view is declared before it is placed on the right side of the tree with its highest set to make it the same size as the main tree to make the application more appealing to the user.

Graphical user interface, text, application, email

Description automatically generated

The next section of the “SecondaryTreeView” method starts with the placement of the secondary tree then, if the argument is equal to 1 or 2, data will be placed into the tree.

The code will start (for both argument 1 and 2) by running a for loop to remove all existing data from the tree. This is achieved by using the number of rows of existing data to find the length of time the loop should run for, looping that number multiple times to remove all the data in the tree (a try loop is used as if there is no data in the tree and then only the for loop is use the application would crash due to row being equal to 0).

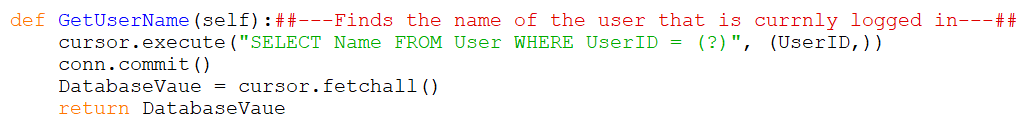
If the argument is equal to 1, the process of adding the user’s favourites to the secondary tree view will be run (as seen above). The adding process will start by finding the number of favourites which the user has, which must be known to tell the code how many times the for loop is run to gather all the user’s favourites from the database. Then, the JobID’s of the job, which have been favourited by the user is added to a list “value” which will take the form of a tuple (a list which contains lists). This is later used in the first of two for loops which are run for the number of favourites the user has. The first loop will take the JobID from the tuple, breaking it down to just its value (removing the values from both lists) then they will be used to gather the desired information about the job which. This will be gathered into the local variable “values”. Then “values” will be used in the final for loop to add the data to the tree view and display the information of the user’s favourites. (This section of the code is in a try loop as, if the user has no favourites, the code will crash when trying to receive data from the database as there is nothing to gather.)

Graphical user interface, text, application

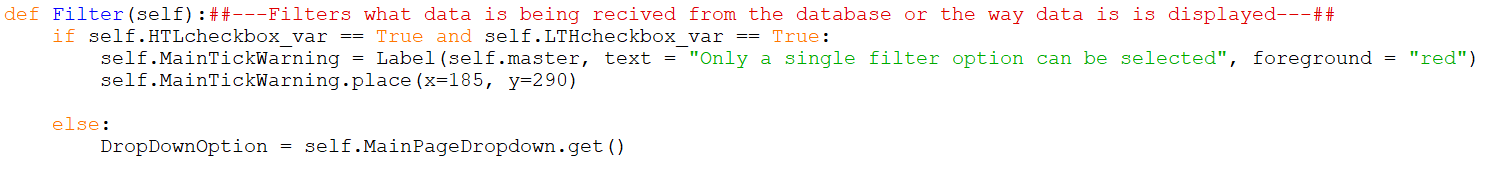
Description automatically generated

If the argument is equal to 2 the process of adding the user’s applications to their job listings to the secondary tree view will be run (as seen below). The start of the adding process will start by finding the number of applicants which the user has to all their existing jobs, which would need to be known to tell the code how many times the for loop is ran to gather all the uses applicant’s data form the database. Then the JobID’s of the jobs which people have applied to are gathered and added to a list “value” which will take the form of a tuple (a list which contains lists) which is later used in the first of two for loops which are run for the number of applications the user has. The first loop will take the JobID from the tuple, breaking it down to just a value (removing the values from both lists). Then they will be used to gather the desired information about the applicants and the jobs they have applied to. This is done using an inner join connecting the job and user table where the UserID match to leave other. This then allows the code to gather the user’s information based on the user ID which has been saved alongside the JobID in the applicant class to gather all the necessary information for the Secondary tree. This is saved into the local variable “values”. Then “values” will be used in the final loop to add the data to the tree view and display the information of the user’s applications. (This section of the code is in a try loop as, if the user has no favourites, the code will crash when trying to receive data from the database as there is nothing to gather.)

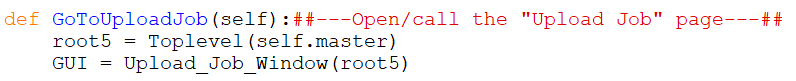
Then the final line of the “SecondaryTreeView” method is run to save all the data of the secondary tree into “self.SecTree” (a variable in the constructor).



The “GetUserName” method is called at the start of the constructor to gather the name of the current logged in user so it can be displayed on the top right of the screen. This is done by using the UserID of the current user (saved in a global variable) to gather their name from the database. This can then be returned and used.



The “Filter” method is the only incomplete method in the code due to it being a secondary objective and being too difficult to execute in this round of the application. The current working of the method makes sure that only one of the two filter options will be chosen by the user and will output a suitable error message if they are both selected. Then, the input from the drop down will be gathered and saved into the local variable “DropDownOption”. (All code created is fully functional and would not need to be changed in a later version.) The next steps for this application would be to gather the input if one has been entered into the search box by the user. The three filter options would need to be brought together and, based on what the user has selected the data in the main tree view, would need to be altered. When this is done it will be changed to the “AddingDataToMainTree” method, which would also need to be completed to make this functional.

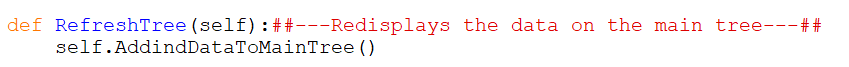


This section of the final solution is called when the user presses the “Upload Job” button to open the “Upload Job” page by calling the “Upload\_Job\_Window” class using instantiation.

A picture containing application

Description automatically generated

The “AddingDataToMainTree” method is called at the end of the constructor to populate the main tree with the data of all the current job listings. The code starts by entering the try except loop, which is there as, if there are no job listings in the database, the application would crash. Then the first section finds the number of current job listings saved in the job table. Once that number is found it is saved to a local variable and run through a loop to save all the JobID’s of the found jobs. That value is then used to tell the code how many times it will need to loop in the next for loop of this section. In this for loop, the code uses the JobID’s and gathers all the job information and displays it onto the main tree view. When the data is added to the main tree view, the data is saved in a phantom row (a row that is not visible to the user) at the end of the tree view (seen by the import into column 7 when the tree view only consist of 6 columns). This is down so when the job listing is selected from the tree, the selection code can grab the unique identifier of the selected job listing.



The “RefreshTree” is one of a few methods which are used to call other methods. This one has been created to update the main tree when the “Refresh tree” button is pressed. This has been made to assist with testing and with the running of the application. It allows the user to refresh the tree if they want to see if any new job listings have been posted. It has also been designed so that if, in an updated version of the code, the user could also be able to update the secondary tree. This would not take any extra methods to be created and only a single line has been added to this method.

Text

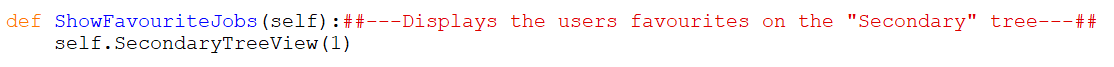
Description automatically generated

The “SelectedJob” method is used many times throughout the code. It is designed to return the JobID of the job which has been selected on main tree. It works though a try except statement. The try is used to attempt to gather the JobID of the selected job listing by looking at the data saved in the phantom row of the tree and returning it to the variable that it has been called to be saved in (as seen below where “Selected = self.SelectedJob()” is called in the Delete function to see what job was selected by the user. The except part of the statement is used to inform the user that they have not selected a job listing.

A picture containing text

Description automatically generated

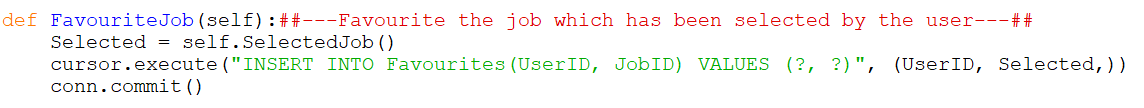
The “DeleteJobCommand” method is called when the “Delete job” button is pressed. It is used to delete the job which was selected from the job listings and all user’s favourites and application (removing all trace of the job listing). This is done by first using the “CheckIfAdmin” command to see if the current logged in user is an Admin (will return 1 if the user is and 0 is the user is not), then the job which user has selected is gathered using the “SelectedJob” method. Then the code will find the UserID of the Job which the user has selected. Then an if loop is used to see if the user has the permission to delete the selected job. (The user has the right to delete the job if they are an admin or are the user who placed the listing). If the user passes the requirements, three delete statements will be run to delete any entries in the database which are related to the selected Job. Once the data is deleted from the database the listings will be deleted from the main tree.



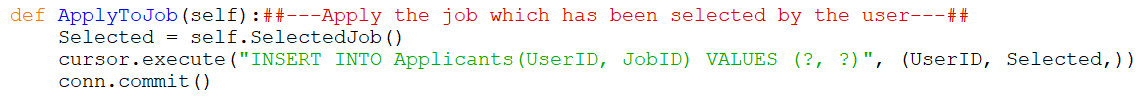
The “ShowFavouriteJobs” method is called when the “View favourites’” button is presses which is used to call the “SecondaryTreeView” method with the argument of 1 so new heading and data get displayed onto the secondary tree which relates the user favourited job listings.



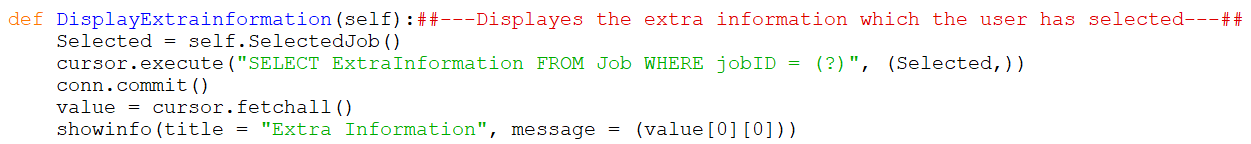
The “ShowApplicants” method is called when the “View applicants” button is pressed. This is used to call the “SecondaryTreeView” method with the argument of 2 so new headings and data get displayed onto the secondary tree which relate the user applicants who have applied to their job listings.



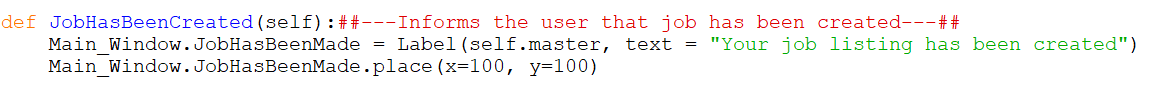
The “FavouriteJob” method is called when the “Favourite job” button is pressed which is used to save the selected job listings. This is done by first calling the “SelectedJob” method to gather the jobID of the job which has been then using the gathered JobID and the UserID of the current user. The favourites are saved to the database for the selected jobs for the user which can later be seen on the secondary tree.



The “ApplyToJob” method is called when the “Apply to job” button is pressed which is used to apply to the selected job listings. This is done by first calling the “SelectedJob” method to gather the JobID of the job which has been selected, then using the gathered JobID and the UserID of the current user. The application is saved to the database to apply to the selected job which the owner can then see on the secondary tree.



The “DisplayExtraInformation” is called when the user presses the “Extra information” button. The method starts by gating the JobID of the job selected by the user using the “SelectedJob” method. Then the returned JobID is used to gather the extra information of the job from the database. Finally, the method uses a widget (which will be displayed on the screen) that is holding the extra information with a close button for when the user no longer needs the information.

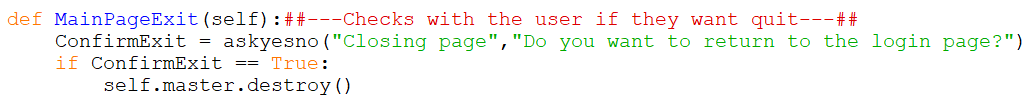


The “JobHasBeenCreated” method is called from the Upload job page (reason why Main\_Window is used and not self) before the page is closed to inform the user that the job has been created.

A picture containing text

Description automatically generated

“CheckIfAdmin” is currently called only in the delete function, but I have decided to make it a full method so, if more admin commands are required later, the code would not need to be remade or repeated. The method starts with a call to the database to see if the current user who is logged in is an admin. If the user is an admin, the method will return 1 (for true) and 0 (for false) to inform the code if the user is admin or not.



The “MainPageExit” method is called when the user presses the “Return to login” button. It is created to make sure that the user does want to exit the main page and if click of the button was not a mistake. This is done by displaying a widget to the user (which reads “do you want to return to the login page?”). It gives the user the option to press yes or no. If yes is pressed, the if statement runs and the main page is closed. If no is selected, the code goes back into a standby to wait for the next user interaction.

Text

Description automatically generated

The final method of the main page is the “SaveApplicantsToFile” which is run when the user presses the “Save applicants” button. The code starts with the name of the file in which the data will be saved into being declared. Then the opening/creation of the file is run with the correct option being selected for the input into the file (declared by the “w” on the second line). Then a for loop is run (this runs for the number of applications which the user has received) and each applicant’s name and contact details, with the job title which they have applied for, is saved and written into the file. This is done with the method ending and the file being closed (meaning it is saved into the user files in the same location where the application is saved).

## Upload job page technical solution:

Text

Description automatically generated

Text

Description automatically generated

This represents the creation of the Upload job page, starting with the creation of the window (scale and name). Next, the creation of the variables, labels (used to tell the user what each entry bock is linked to), entry boxes (used to gather user input), the buttons (used to allow the user to proceed with processing the application and call new windows if necessary) and a tick box (used to give the user the option to input extra information).

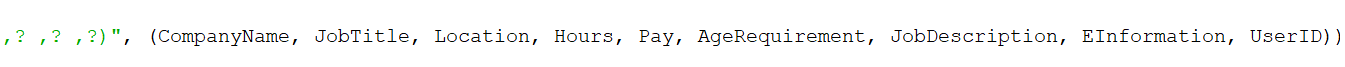
Text

Description automatically generated

Next in the technical solution is the input validation which is triggered when the user presses the “Upload job” button. It starts by gathering the inputs which the user has entered then saved in local variables. Then the code checks if there is any data populating the entry boxes (the check for data in the extra information entry box only runs if it has been selected). After this, the code checks to see if the age only consists of numbers, the postcode doesn’t have a space present, hours only consist of numbers, there is no “£” present in pay, and it only consist of numbers. If any of the checks are failed, relevant error messages will be displayed to the user. If all checks are passed, the job listing is saved to the data base (as seen below).

Text

Description automatically generated with medium confidence



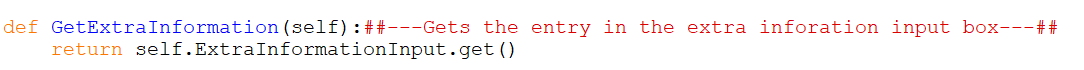
The code displayed above is used to save a new job listing to the data base. It is called when all input validation is passed. First, the code gathers if the extra information section was selected. If it was the data the user entered, it will be gathered and saved into a local variable. If it has not been chosen, the string “NULL” will be saved in its variable instead. Then, the code saves all the job’s information to the job database, including the unique ID of the user, which made the job listing using the global variable (creation show in the “Login” page technical solution). Once the job listing is created a method is called- “JobHasBeenCreated”, which is located on the “Main” page) to display to the user that their job listing has been created, then the “Upload” job page age is closed.

(The second image in the end of the line which exits the screen)

Chart, scatter chart

Description automatically generated

“ExtraInformationEnteryOption” is called when the user choses to add extra information to their job listing (done by selecting the tick box).



This section of code is used to gather the information which the user has entered in the extra information entry box, to help with the efficiency of the code as the data needs to be gathered several times.

Evaluation

## Degree of solution success:

Main objectives are objectives which must be completed, unlike secondary objective which could be completed to help improve the application but are not mandatory.

The data base:

|  |  |
| --- | --- |
| Objective | Objective reached |
| Database should consist of separate tables, consisting of all atomic data. |  |
| Tables must be normalised (Normalization is the process of organizing data in a database). |  |

Create a login page:

|  |  |
| --- | --- |
| Objective | Objective reached |
| This will be the first page which is seen by users when the programme is started. |  |
| A button will be needed to be able to call the Create account page, quit the application, and sign in. |  |
| Entry boxes will be created to allow the user to enter their username and password. |  |
| Label will be displayed to show the user what each entry box is linked to. |  |
| An image (application logo) will need to be placed at the top of the screen. |  |
| Validation on user inputs will need to be carried out (all fields filled and correct length). |  |
| Validation will need to be carried out to see if the username and password are correct. |  |
| If validation is passing the Main page should be opened. |  |
| Error messages are to be displayed if any invalid entries are detected. |  |
| The GUI should be made user-friendly by keeping all pages on a single page and easy to understand labels, button, tree views and layout. |  |

Create a Creating account page:

|  |  |
| --- | --- |
| Objective | Objective reached |
| This will be displayed to the user if the “Create new account” button is pressed (on the Login page). |  |
| A button will be needed to be able to create a new account and quit the application. |  |
| Entry boxes will be created to allow the user to enter their account details (username, password, name, email, age). |  |
| A tick box will be needed to give the user the option to create an admin account (if box is selected as entry box, it should replace the tick box to give the user the opportunity to enter an amin code). |  |
| Label will be displayed to show the user what each entry box is linked to. |  |
| An image (application logo) will need to be placed at the top of the screen. |  |
| Validation on user inputs will need to be carried out (all fields filled, entries consist of the correct data type, email is valid (if @ is present), and if the admin code correct if selected). |  |
| Validation will need to be carried out to see if the username is not taken. |  |
| If validation passes, the account should be created (details saved to database). |  |
| Error messages should be displayed if any invalid entries are detected. |  |
| The GUI should be made user-friendly by keeping all pages on a single page and easy to understand labels, button, tree views and layout. |  |

Create a Main page:

|  |  |
| --- | --- |
| Objective | Objective reached |
| This will be displayed to the user if the “sign in” button is pressed (on the Login page and validation is passed). |  |
| A button will be needed to be able to add a job listing, view favourite, add a favourite, view applicants, apply to a job, filter, delete job, view extra information, refresh trees, and close the page. |  |
| Label will be displayed to show the user what each entry box is linked to. |  |
| Tree view needs to be created. A main tree view which displays the job information and a secondary tree which is used to display favours or applicant data. |  |
| A way to select data on the main tree is needed. |  |
| Validation will need to be carried out to see if the search returns no data. |  |
| Validation is needed to see if a job is selected when needed. |  |
| Error messages should be displayed if any invalid entries or selected are detected. |  |
| The ability to save the details (Job title, applicant name, and applicant contact details) of applicants to user jobs in an external file. |  |
| The GUI should be made user-friendly by keeping all pages on a single page and easy to understand labels, button, tree views and layout. |  |

Create Upload job page:

|  |  |
| --- | --- |
| Objective | Objective reached |
| This will be displayed to the user if the “Add Job” button is pressed (on the Main page). |  |
| A button will be needed to be able to create a new job listing and close the page. |  |
| Entry boxes will be created to allow the user to enter their job details (Company name, job title, age requirement, location, hours, pay, and job description). |  |
| A tick box will be needed to give the user the option to add extra information (if box is selected as entry box should replace the tick box to give the user the opportunity to enter the information). |  |
| Label will be displaced to show the user what each entry box is linked to. |  |
| Validation on user inputs will need to be carried out (all fields filled; entries consist of the correct data type). |  |
| Validation will need to be carried out to see if the username is not taken. |  |
| If validation passes, the job should be created (details saved to database). |  |
| Error messages should be displayed if any invalid entries are detected. |  |
| The GUI should be made user-friendly by keeping all pages on a single page and easy to understand labels, button, tree views and layout. |  |

Secondary objectives:

|  |  |
| --- | --- |
| Objective | Objective reached |
| A filter option which can be applied to the main tree. |  |
| A tick box will be needed to give the user the option to filter the order that data is represented on the tree views. |  |
| A drop down to help with the filter option. |  |
| Entry boxes will be created to allow the user to enter a search for a job. |  |
| Create a way for the user to add the people which have applied to their jobs too an external file. |  |

The application created has been able to solve all outlined problem and more by achieving all main objectives, with a range of functions which were not predicted to be completed in this solution due to the given time scale and complexity (such as the creation of the external file, seen in the secondary objectives).

Green is objective reached.

Orange is objective attempted.

Red is objective incomplete.

## User evaluation:

The three students where shown the final application which was made for this project and were given time to try it out for themselves.

The review of the objectives concludes that the system has met all main objectives. However, to ensure that the final solution of this application meets its indented purpose, I have shown them the application and they were given time to try it out for themselves. This has been done to allow me to have a non-biased and rational idea of the strength and weakness of this application.

When the students were asked about the strengths of the application, they brought up the topic of having the ability to access all the functions of the application form on one page. They stated that the use of the two-tree view was extremely helpful as it allowed them to see all the data they wanted at once.

When the students were asked about the weaknesses, they stated that a way to help to improve the code would be to allow the user to press on a job on the secondary tree to be able to view the full details.

The students' overall impressions were positive, and all agree that if this application was published and created with a centralized sever, they would prefer this application over existing methods of job finding.

Therefore, in terms of the clients’ satisfaction, the optimal job has exceeded their expectations because of the range of functions and useability of this solution to the problem.

## Possible Improvements/ Things I would change:

The application was made and exceeded expectations on objectives’ completion. However, improvements can be made to make the code more efficient or add functions to improve the users’ experience.

Throughout the code, several labels are used, mainly to display error messages to the user. A way to reduce the amount of RAM which is being taken up by the application is to reduce the amount of declared items throughout the code. This could be done by making a single label which can be updated. It would also have the benefit of reducing the file size, thanks to a significant reduction in lines as the location of the label would only need to be declared once.

Another way to improve this application would be to complete the filter function (a secondary object). This would be done to help improve the user experience and the time it would take for the user to find a job which would suit their needs. This would be done by bringing the function of the dropdown, the search bar, and the highest to lowest/lowest to highest tick box to allow a broad range of filter options on the job listings, which are displayed to the user.

A small addition to the code, which would improve the user interaction with the application, is finding a way to allow the user to enter the name of their file when downloading the applications to their job.

## Advantix:

This is an uninterrupted representation of the whole application:

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

Text, application

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

(End of the SQL statement)

Text

Description automatically generated

Text

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

A picture containing background pattern

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated



(End of the SQL statement)