NEA PROJECT WRITE UP

Task Assignment System

By Issa Aboobaker

Table of Contents

Analysis

Current Problem

The problem which has driven the idea of my project is derived from the difficult and time-consuming process of assigning tasks to staff within a company or organisation. It can take very long for a manager or anyone in a position of responsibility to choose the most suited person for a task and in fact sometimes it can be hard to choose somebody at all. As someone who has experienced this problem first-hand during working hours of my job it has inspired me to come up with a solution.

The three problematic outcomes of this situation in a working environment are as follows:

- The right person is chosen but after a very valuable amount of time is wasted
- Nobody is identified as suitable enough and therefore nobody is chosen
- The wrong person is chosen, which can have consequences

These are ongoing problems within many working environments and if they can be solved, work processes will become catastrophically more efficient. As someone who has experienced these problems first-hand during working hours of my job it has inspired me to come up with a solution. I aim to create a software to solve all of these issues.

My vision is to create a software which will allow managers to easily assign tasks to staff under their control who will be able to view their current tasks in order of their priority and can mark them as complete when necessary. This will then be fed back to the manager who can feel confident that tasks are being completed by the right people in the correct order. The software will be designed to be used by everyone in the organisation but will act differently for people in management. The managers would be able to input details of a task into the software and then choose if they want it to be assigned automatically or manually. If they want to do it manually, they can simply just select the worker to assign it to. They also have the choice to assign it automatically where a built in algorithm will analyse the task and select the most suitable person for it. The software ensures that the worker is informed of their task and its then up to them to complete it and mark it as complete on the software which will then notify the manager. The manager will also have the option to view al tasks that have been assigned to asses the progress of all the workers.

Research

To help me gather information on the problem and to decide a way to solve it I first had to use primary research meaning I obtained all my information first-hand, so it is totally accurate and useable. The best and easiest way to do this in order to gather enough details within the timeframe was to conduct a survey. I did two surveys, firstly to find out general thoughts towards the problem and secondly their opinions on my approach to solve it. I have put all my results into tables to make them easier to interpret and have shown the actual survey results in screenshots below. The survey involved a rating system which is explained above each table. For privacy reasons I haven't shown the name of the people who took part in the survey, so I have just labelled them with letters corresponding to the order they completed the survey in.

Survey 1 – General Thoughts Towards the Problem

The rating system used for this survey:

1. Strongly Disagree
2. Disagree
3. Not Sure
4. Agree
5. Strongly Agree

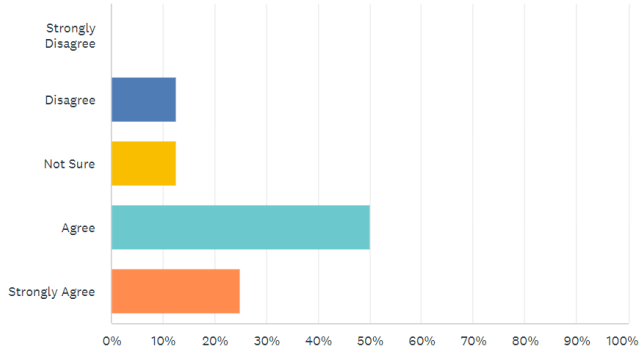
Person	The whole process of assigning tasks to the right people within an organisation can be time consuming	Picking the right person for a task within an organisation can be difficult	Picking the wrong person for a task could end up being costly
A	4	4	3
B	4	4	4
C	5	5	4
D	2	2	2
E	5	4	4
F	4	4	3
G	3	4	2
H	4	4	4

Q1

Customize Save as

The whole process of assigning tasks to the right people within an organisation can be time consuming

Answered: 8 Skipped: 0



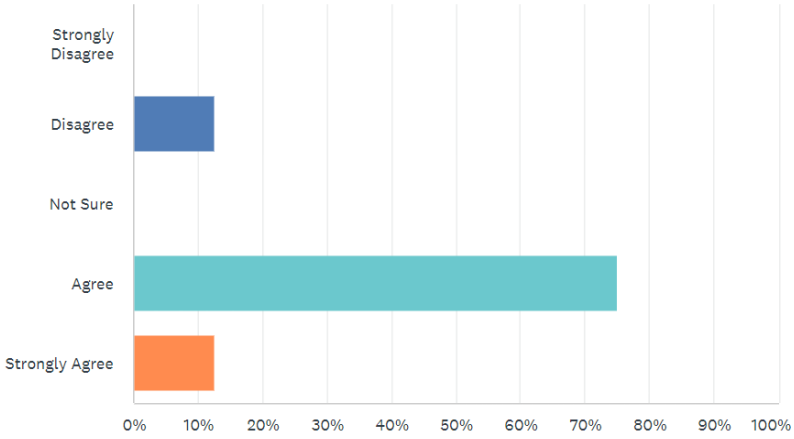
ANSWER CHOICES	RESPONSES	
Strongly Disagree	0.00%	0
Disagree	12.50%	1
Not Sure	12.50%	1
Agree	50.00%	4
Strongly Agree	25.00%	2
TOTAL		8

Q2

Customize Save as

Picking the right person for a task within an organisation can be difficult

Answered: 8 Skipped: 0



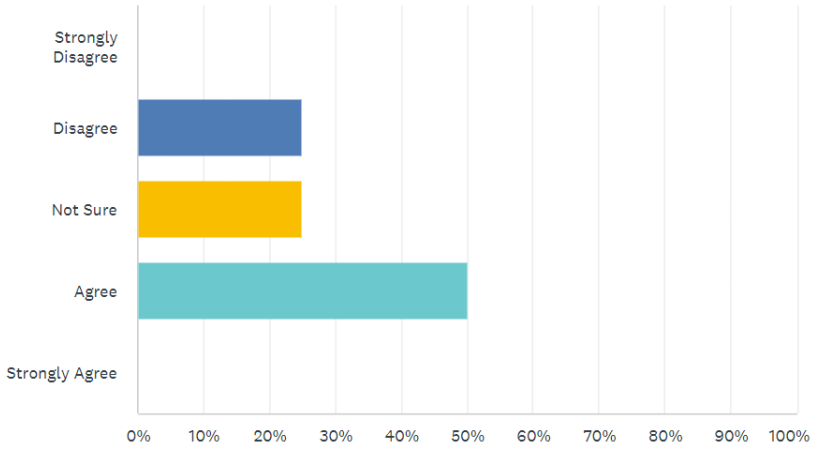
ANSWER CHOICES	RESPONSES	
Strongly Disagree	0.00%	0
Disagree	12.50%	1
Not Sure	0.00%	0
Agree	75.00%	6
Strongly Agree	12.50%	1
TOTAL		8

Q3

Customize Save as

Picking the wrong person for a task could end up being costly

Answered: 8 Skipped: 0



ANSWER CHOICES	RESPONSES	
Strongly Disagree	0.00%	0
Disagree	25.00%	2
Not Sure	25.00%	2
Agree	50.00%	4
Strongly Agree	0.00%	0
TOTAL		8

I found the results of Survey 1 to have gone very well. There were lots of 4s and a few 5s meaning that in general there is agreement that my identified problem is a valid problem. There were no 1s so nobody strongly disagreed about the problem. The general pattern is agreement with a few not sures. The only exception was person D who disagreed with everything but since this was only a minority it doesn't raise any concerns for now.

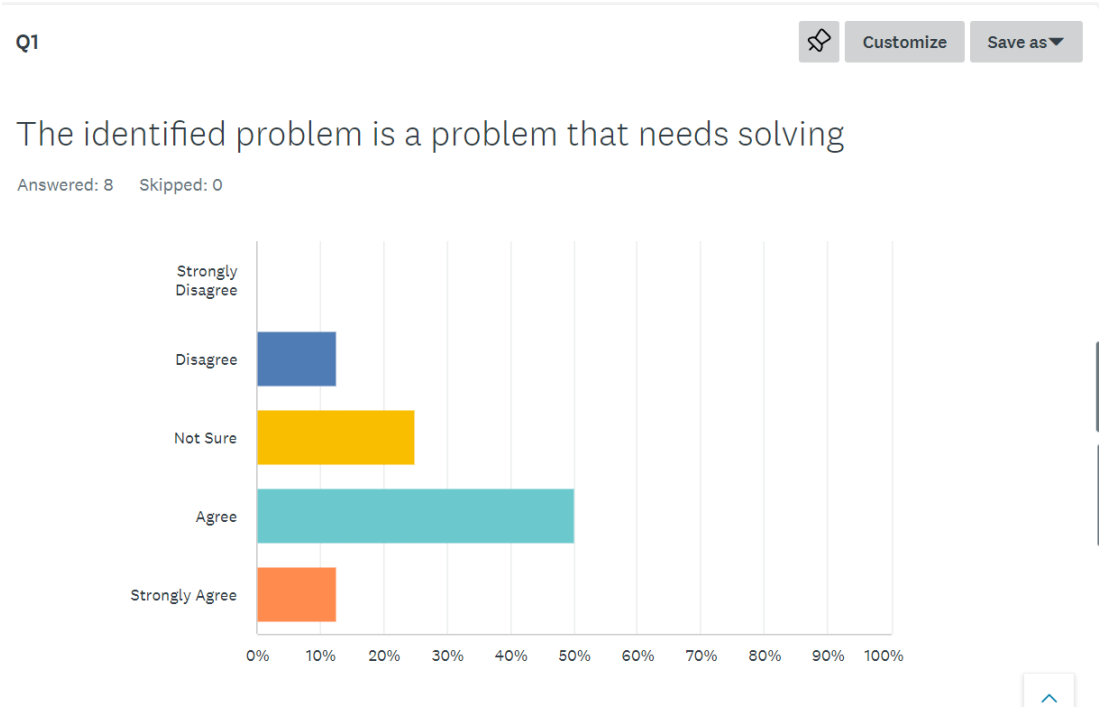
Survey 2 – My Solution to the Problem

The rating system used for this survey:

1. Strongly Disagree
2. Disagree
3. Not Sure
4. Agree
5. Strongly Agree

Note: Persons A,B,C,D,E,F,G,H correspond to the same person of that same letter in Survey 1 so relations have been analysed between their opinions from both surveys.

Person	The identified problem is a problem that needs solving	The best solution to the problem should be through software on a computer	The software I have proposed to create is effective enough to solve the problem to a suitable standard	Would you be happy to use the proposed software in a working environment?
A	4	4	4	5
B	4	4	5	5
C	5	5	5	5
D	2	3	3	3
E	4	5	5	5
F	3	4	4	4
G	3	4	3	4
H	4	4	4	5



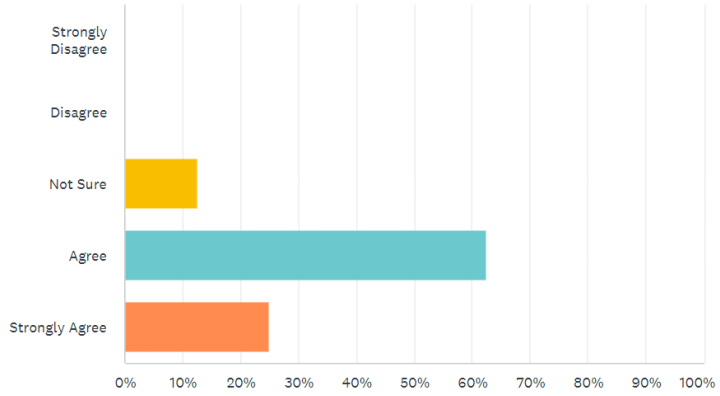
ANSWER CHOICES	RESPONSES	
Strongly Disagree	0.00%	0
Disagree	12.50%	1
Not Sure	25.00%	2
Agree	50.00%	4
Strongly Agree	12.50%	1
TOTAL		8

Q2

Customize Save as

The best solution to the problem should be through software on a computer

Answered: 8 Skipped: 0



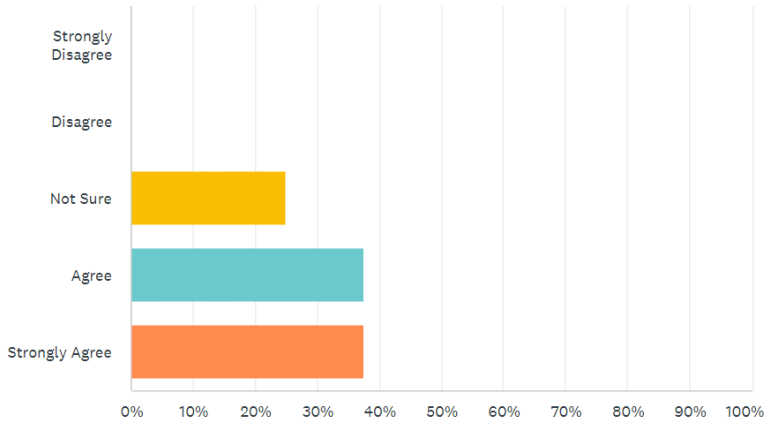
ANSWER CHOICES	RESPONSES	
Strongly Disagree	0.00%	0
Disagree	0.00%	0
Not Sure	12.50%	1
Agree	62.50%	5
Strongly Agree	25.00%	2
TOTAL		8

Q3

Customize Save as

The software I have proposed to create is effective enough to solve the problem to a suitable standard

Answered: 8 Skipped: 0



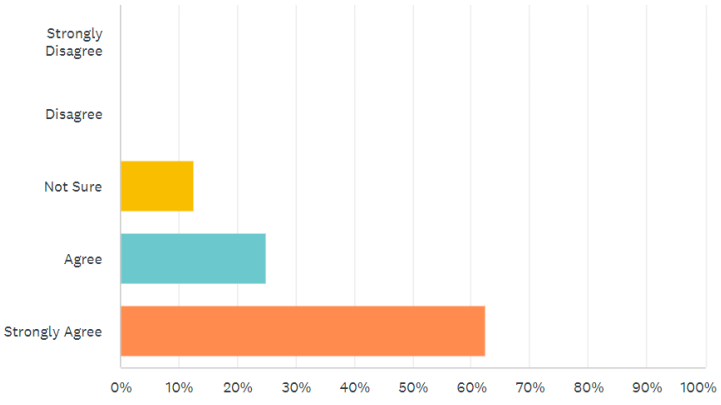
ANSWER CHOICES	RESPONSES	
Strongly Disagree	0.00%	0
Disagree	0.00%	0
Not Sure	25.00%	2
Agree	37.50%	3
Strongly Agree	37.50%	3
TOTAL		8

Q4

Customize Save as

Would you be happy to use the proposed software in a working environment?

Answered: 8 Skipped: 0



ANSWER CHOICES	RESPONSES	
Strongly Disagree	0.00%	0
Disagree	0.00%	0
Not Sure	12.50%	1
Agree	25.00%	2
Strongly Agree	62.50%	5
TOTAL		8

With the results from Survey 2 I have noticed that once again the general pattern is agreement that the problem needs to be solved. From the third column, apart from Person D everybody else agrees that my proposed solution is good enough for the problem.

Interview

From the results of the surveys above I chose the people who could be a potential user for my software and conducted an interview with them to gather solid information which would act as the foundation for my project. Rather than a rating system I wanted to do an interview here to collect much more useful and meaningful qualitative data which would be very influential in how I approached my project design.

The surveys helped me identify the right people that I needed to interview by letting me see those who agreed that the problem I identified was indeed an actual problem and that my solution to the problem is valid. I concluded that the best people to interview were Persons A, B, C, E and H. Now the interview is needed to help me think further about how I'm going to adapt my idea into a working piece of software.

I once again found it useful to present the results in a table. Each response has been recorded exactly how it was given.

Person	Other than the features already specified what else could the software include to make it better?	From the things I have decided to include, is there anything that shouldn't be there, if so what and why?	Considering the identified problem and the given timeframe is my proposed solution the best it possibly could be? If not why?
A	"Give an option for a manager to choose someone themself to assign the task to rather than have it auto done every time. Also allow workers to respond to a task saying whether or not they are able to do it."	"No"	"Not quite, its an almost complete solution, but a few extra features could be added to cover all situations such as the ones I have already suggested for the first question."
B	"Have a shared task feature where the task is assigned to multiple people who are all suited to do it and then allow any one of them to decide to do it based on their availability."	"Everything there is necessary"	"Absolutely"
C	"Nothing, its perfect as it is"	"No"	"It is perfect"
E	"Other things could be added but they wouldn't be necessary"	"Nothing"	"Yes"
H	"Should add a feature to assign tasks to a number of staff because some tasks might require multiple people in a realistic situation"	"No"	"I feel it could be made better but only with more time, so considering the limit you have it is as good as possible."

After reviewing the responses to the first question in the interview I decided to add to my intended design the first suggestion by person A as I feel it is possible for me to include a manual assignment option for managers and it would significantly improve my code. After careful consideration I decided that I wouldn't be able to do any of the other suggestions within the time frame, although they are good enough ideas.

After reviewing the responses to the second question there wasn't much to change in my design, and it was encouraging to hear that everything I have proposed to use as a solution should be there, in the opinion of the interviewees.

The responses to the third question were great to hear and has assured me that given the circumstances my proposed solution is correct and I feel confident with proceeding to code it.

Conclusion of Research

Overall the research into the problem was very successful. It has equipped me with the knowledge I need to perfectly code my solution to the problem which I know now is completely real and needs solving.

Current System

I have decomposed the process of assigning tasks within a workplace into three steps:

1. Pick a suitable person for the task
2. Inform the chosen person that they need to do the task
3. Check that they complete it

1. Picking a suitable person for the task

This is the most difficult step of the process. Choosing out of a huge number of possible people who specifically will carry out the task in the quickest time while maintaining the best standard. This is vital for all companies and organisations. Currently this stage is having to be carried out by managers themselves. For small companies this may be effective as the manager may know lots about the workers and can easily identify a suitable person for a task. However, this is dependant on the fact that the number of staff is small and that the manager knows them well enough to decide who is most suitable. My system doesn't depend on any of these factors and will carry out this step no matter what, so for working areas with a large amount of staff or places where the manager doesn't know everybody well enough to choose someone, or both, my system will be needed. In a perfect scenario where the manager is easily able to identify the right people for a task, they could use the manual assignment option within my system. This means that they can still make the choices and also have my system there to implement it and to help overcome any problems with the other two steps.

2. Informing the chosen person of the task

After deciding a person for a task the manager would need to inform them of the task. This could be difficult in a large working environment making it hard to find them. Also the person may be busy with another thing meaning the manager will have to wait to deliver the task to them. My system overcomes this by sending the task to the workers page and whenever they are free they can check their page and will be informed of the task.

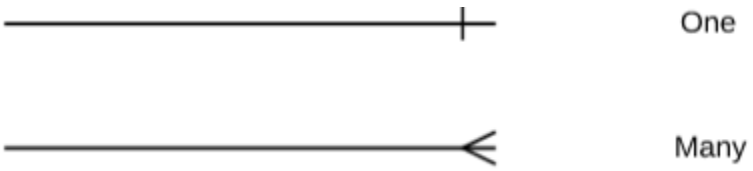
3. Checking that the task has been completed

In some cases the manager will want to know as soon as the task has been completed. This can be made difficult by the fact that the worker would have to find the manager and tell them manually. The manager could be hard to find within the large area or may be busy. Once again my software solves this problem by including a feature where the worker can mark an assigned task as completed. This will show on the managers page, so if they assign an important task which they need to know when it has been completed they only need to check their page and will be notified when it is done.

Entity Relationship Diagram

In the current system the managers communicate directly with the workers to assign them tasks. As explained before this is not very efficient and lots of problems can arise from it. The following entity relationship diagram clearly show the relationship between the manager and the workers. In this ER Diagram the process has been simplified to model a situation where there is many managers communicating with many workers, hence a many to many relationship.

Key



IPSO Chart

IPSO	Section of Program	Item
Input	Inputting Task	Task Details
	Manual selection	Manager manually selects a worker for the task
Process	Check if worker is available	Is the worker free?
	Assign task to worker	If worker is now free assign the task to them
Storage	Storing Tasks	Task Details stored on paper or on computer
Output	Assigning tasks to workers	Communicate task to chosen worker

Potential Solutions to Problem

Languages

To solve the problem the solution must exist as some form of code or software on a computer. The possible different programming languages that could be used are:

- Python
- JavaScript
- Java
- C#
- C
- C++
- Pascal

For my proposed solution I have chosen to code using Python.

Developer Environments

With my chosen programming language (Python) to solve the current problem there are a few possible developer environments to use:

- PyCharm
- Spyder
- IDLE

I have chosen to use IDLE because it is one that I am very familiar with it so I know how it works and will have no problems in coding with it.

Constraints and Limitations

For all organisations using any system to solve this current problem there will be some limitations:

- To use a computer based system, it requires that the organisation will need to ensure that all potential users have access to a computer and are able to use it
- All user computers within the organisation will need to have the system downloaded and ready to use
- All computers will need the appropriate hardware and software to run the system
- All users will need to regularly use the system and check/manage tasks
- All users will need to have some understanding of how to use the system
- The users may not be comfortable with sharing their own personal details which may be required for any system

Clients

The software is designed for professional use by companies or organisations who use workplaces with lots of staff, and someone in control of the staff like a manager. The organisation will download their software to every computer for it to be used by all who need it. User accounts will be created for all by the company and the company will also have to provide the necessary information about the employees for the software to operate.

Users

The software is intended to be used by both managers and staff in a company. Each user has their own personal login to their own account. Within a staff account, they can view their current tasks, in order of assignment and mark them as complete when necessary. They can also view their profile which displays all the information about them that is stored in the database. Within the manager accounts, they have an option to assign a new task which they enter in and can have the algorithm automatically assign it for them or can choose to manually assign it by selecting someone from the search/drop down menu in Tkinter. The managers have access to a feature which allows them to view information from the database about the staff to help them decide who they want to assign a task to if they prefer to do it manually. If the manual option is used, the main task queue will not be used to store a task if the staff are busy, it will store it in a personal queue for that person and as soon as they become available they will be assigned it. Manually assigned tasks have priority over tasks assigned by the algorithm. The staff also have a list of every task which has been completed and can clear the list if they choose to.

User Needs and Requirements

After interviewing potential users of the software, I have come up with criteria which needs to be entirely met for the users to be properly satisfied with it. The criteria consists of the most common requests between the users:

- To be able to use the software entirely through the Graphical User Interface without having to know about how the software works
- The Graphical User Interface is easy to use for all users
- To be able to login with their own username and password to their own personal page which they use
- Any data about the users is stored securely, is only to be used by the company for work purposes and is not to be shared to anyone outside of the company

Audience

The audience for this code is the workers who have been assigned tasks. They receive an output from the system telling them what they need to do. The managers also receive an output from the code telling them when a worker has completed a task.

Objectives

The most important part of the analysis stage. I have identified the fundamental targets for my code and tweaked them to be SMART, to ensure they are all relevant and realistic for the given problem. With each one I have provided necessary details about the objective itself, why the objective is important and how the success of it could be measured when the code has been completed.

1.The code utilises a proficient algorithm which analyses any relevant task and correctly assigns it to the most suitable person:

- This is the focal point of my entire code. It solves the problem and everything else in the code is to apply it to be able to used by the users. Regarding the importance of my algorithm I will need to spend a lot of time considering different factors needed to create the algorithm and identify a step by step procedure to process an input and give a suitable output.
- The output will need to go to the right place and this will be achieved by use of primary and foreign keys within the database. Each primary key corresponds to a different user so they can easily be assigned the given output
- To measure the success of this objective it is not as simple as whether the code contains something or not, it will need to be measured how well the algorithm works to solve the specific problem. To do this the code will need to be tested with some given tasks and see who they are assigned to. Then a manager should manually assign these same tasks to the person they think is best suited for that task. This should be compared with who the algorithm assigned the tasks to in order to measure how well the algorithm works at deciding the best person for a task.

2.The Graphical User Interface is clear and easy to use whilst also being complex enough to be used professionally:

- The Graphical User Interface will be coded using the built in Python library TKinter which will help me meet my objective because I have experience using TKinter so I can use it properly and make a useable and also likeable interface.
- This objective is very important because the interface is the only part which will be shown and used by the users and clients within practice. It is very necessary for the interface to be simple to use and doesn't require much coding skill hence I chose to create a Graphical User Interface rather than a Command Line Interface.
- While being easy to use, it must still maintain a certain level of professionalism to how it looks for it to be used within an organisation.
- The GUI is primarily designed to allow the intended users to utilise the code so in order to decide how well I have made the GUI I will need to gather information from potential

users. To measure the success of this objective the GUI will need to be thoroughly tested by a range of users and a questionnaire should be carried out by each user which will gather their opinions on:

- How it looks
- How it functions
- How simple it is to use
- How enjoyable it is to use
- Would they be happy to use it in their working environment?
- Any extra general comments on it.

3.Contains a user login system where each user has their own personal login details to access their own page which looks and acts uniquely for each user

The group of "users" is split into two separate categories:

- The managers who assign the tasks
- The workers who are notified of the tasks and need to carry them out

This means that the GUI needs to be different for people of different categories. Theoretically the page could be completely unique for each user but in the given timeframe I am unable to code this so instead I intend to create two different templates for workers and managers. The code is designed to be used within a real life workplace so there will need to be at least 11 different users each with their own login details to their own page which will be the same as other users in the way it looks and operates based on the template, but each user will be notified of any task they specifically have been assigned only on their page. For the managers they only need to assign tasks and be notified when tasks have been completed so the pages can be exactly the same for each manager.

This objective is very important because in order for my software to be practically used to solve the problem, it needs to be accessed by many different users and the interface needs to be adapted specifically for people of different roles.

To measure the success of this objective the code simply needs to be run and used like it would be in the workplace. I have created a simple step by step method to test this, but it can also be done in any way the tester finds best:

1. A manager logs in using their own account and checks that their page is aesthetically suitable
2. The manager inputs any random task and the code will assign it to any random user
3. That user should log in using their own account
4. They should check the page is aesthetically suitable as well
5. They should check that the assigned task has been displayed on their page
6. Another user should login using their account to check that they haven't received the task
7. Repeat the previous step with as many different users as necessary
8. Once it's confirmed that the initial user has received the task and that nobody else has, they should mark it as completed
9. The manager checks that their page tells them that the task has been completed

4.Stores user login details within a table in SQL

The previous objective is dependent on this one. In order for users to login to their own personal page, they need unique account details which must be stored in a way that it can be accessed by the code. This will be done using Structured Query Language (SQL). I have chosen SQL because there is built in function in Python for it so I can easily integrate it into my code and also because I have worked with it before so I am confident that I can use it to meet this objective and also implement it elsewhere in my entire code.

To measure the success of this objective it first needs to be made sure that the accounts work and are associated to different user pages (this will be checked during testing for the previous objective). Secondly the table containing the user accounts needs to be thoroughly checked to ensure it stores all the accounts and stores them in the most efficient way possible, using normalised tables (information about the normalised tables I have used can be found in the Current System section).

5.Stores information about the employees within a table in SQL which is used by the algorithm

This objective is the second most important because without it the central algorithm cannot function as intended. For the algorithm to determine who to assign a task to, it needs access to information about the users so it can identify the most suitable person. Therefore, information will need to be collected or supplied by the organisation itself after the users confirm they are happy with sharing such details and give consent for them to be used for the purpose of the code. All of these details will be stored securely within a table using SQL, similar to the previous objective, and will only be used for the purpose of the code. Any information stored will NOT be shared with any external sources. The algorithm should access these details, process them and provide an output of who the given task is to be assigned to.

To measure the success of this objective it will be tricky because there is not a specific criteria to meet. In order to determine whether this objective has been carried out to an acceptable extent, the tables should be fully checked (by someone who has been given permission to view these details) that the details within them are accurate and relevant and that the method of storing them is efficient, by use of normalised tables.

Documented Design

Summary

The software would be coded in python and contain various algorithms. The vital algorithm of the entire code is the one that analyses tasks and decides which staff to assign it to and this is created using procedural abstraction where the code is divided up into subroutines each with a specific purpose. I will achieve this by looking at the entire process as a whole and then decomposing it into smaller subtasks that could each be executed by single subroutines. The completion of each subroutine composed together will ultimately complete the algorithm of analysing tasks and assigning them. For example the whole algorithm would be split into two parts: analysing and assigning. Then each of these would be further broken down.

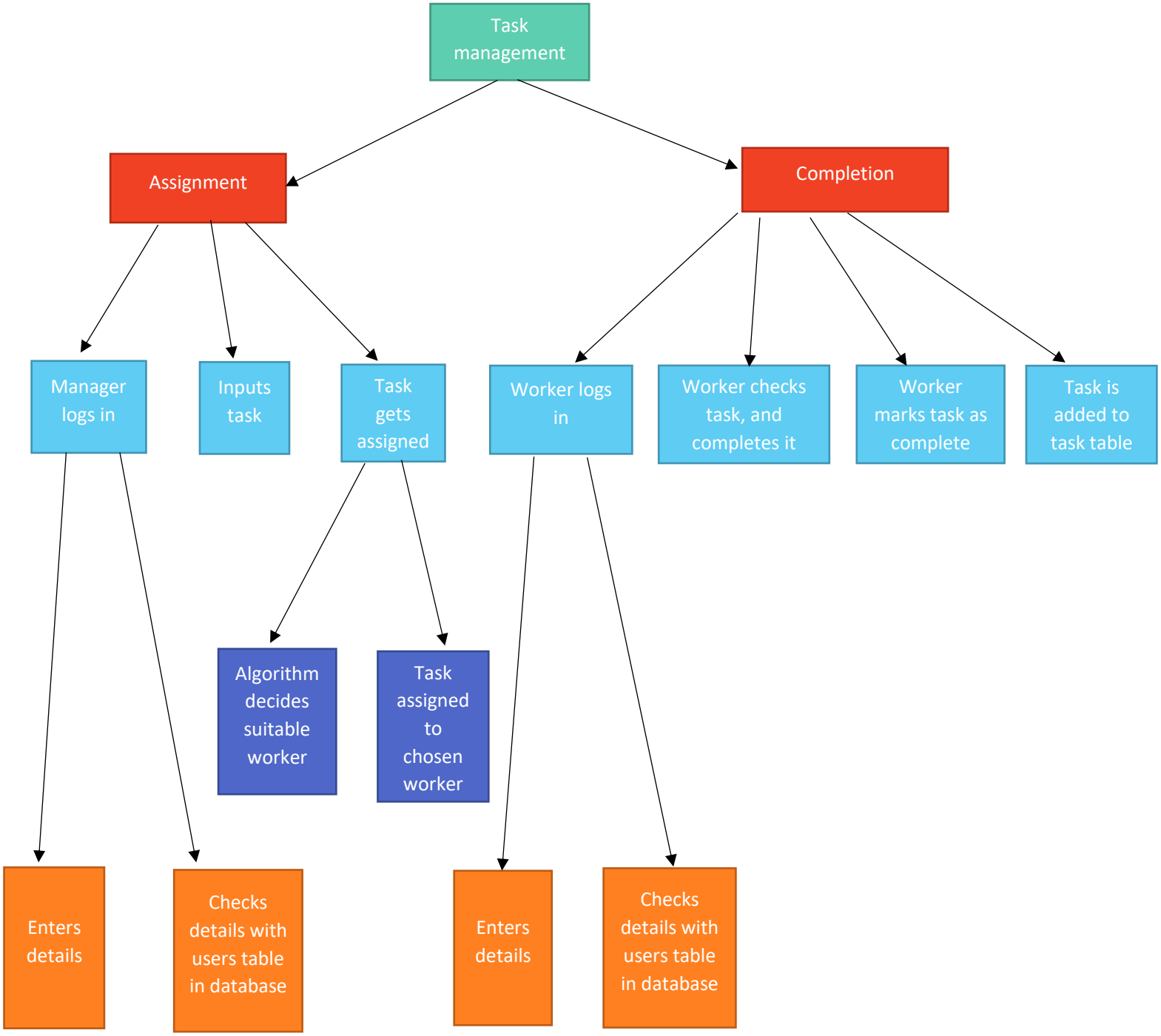
The algorithm will use different factors to determine the best person for a task. For each factor it needs to search the database for people who meet the criteria. This searching will be done using a linear search. Using a binary search would be more efficient but it is not able to be used here because the things which are being searched for cannot be ordered in any way. A factor to be used by my algorithm to decide on who to assign a task to is an attribute of every member of staff which states how long they have been working for. This will allow the algorithm to decide based on how much experience a person has. These attributes would be stored in the database and to compare them they will be sorted via a Merge Sort. I have chosen a Merge Sort because it is more efficient than other sorting methods such as a Bubble Sort and because my software will include lots of data, it is significantly quicker and easier to sort with a Merge Sort.

The code will also have many data structures implemented in it, for example if a task is to be assigned but there is no suitable staff available it will be placed in a queue. I have chosen to use a queue for this over any other data structure to make sure that the first task to be placed in the queue is the first to leave the queue and be assigned, however if there are staff available who aren't suitable to do the first task in the queue it will look at the ones after to check them too, to make this process as efficient as possible.

My code uses information hiding so that the users can make use of it without having to worry about how it actually works. This is done through a Graphical User Interface, using Tkinter within Python.

I have utilised SQL to store the tasks and information about the staff in a database and also the accounts for both managers and staff.

Hierarchy Diagram



Database Design

Workers and Managers Tables – Post Normalisation (3NF)

The database will contain two separate tables each holding account details for both workers and managers. The accounts are given to the users and when they log in the code checks that the Username and Password they input matches the ones stored in the database. For now to be able to code within the timeframe there is 3 manager accounts and 8 worker accounts. In real life practice there would be significantly more of each. The two tables with data dictionaries are shown below along with the SQL statements needed for the tables:

Note: The passwords were completely randomly generated and are the same number of characters as the usernames.

Standard Notation

Managers (ManagerID,Username,Passsword)

Workers (WorkerID,Username,Passsword)

Table Formats

Managers

ManagerID	Username	Password
001	“Manager_1”	“8_x4zAFBg”
002	“Manager_2”	“{Cy:2?MC>”
003	“Manager_3”	“L4#g)tsD&”

Workers

WorkerID	Username	Password
001	“Worker_1”	“md7P@F)7”
002	“Worker_2”	“{mz&4=P&”
003	“Worker_3”	“6^Te{`7D”
004	“Worker_4”	“J>w$5Gjy”
005	“Worker_5”	“Q#e<2Dda”
006	“Worker_6”	“$Dw4vcC@”
007	“Worker_7”	“E(;5x*6K”
008	“Worker_8”	“.8UW#mb4”

Data Dictionaries

Table Name	MANAGERS		
Primary Key	ManagerID		
Foreign Keys			
Data Item	Data Type	Validation	Example Data
WorkerID	Integer	NotNull, =3 digits	001
Username	String	NotNull , =9 digits	“Manager_1”
Password	String	NotNull , =9 digits	“8_x4zAFBg”

Table Name	WORKERS		
Primary Key	WorkerID		
Foreign Keys			
Data Item	Data Type	Validation	Example Data
WorkerID	Integer	NotNull, =3 digits	001
Username	String	NotNull , =8 digits	“Worker_1”
Password	String	NotNull , =8 digits	“md7P@F)7”

SQL Statements

To create the tables:

CREATE TABLE Workers

(WorkerID **NOT NULL PRIMARY KEY**,

Username **NOT NULL** ,

Password **NOT NULL**;)

CREATE TABLE Managers

(ManagerID **NOT NULL PRIMARY KEY**,

Username **NOT NULL**,

Password **NOT NULL**;)

To enter the logins into the tables the INSERT INTO command is needed. For example to insert the details for the first worker, the SQL statement would look like:

INSERT INTO Workers(WorkerID,Username,Password)

VALUES (001,Worker_1, md7P@F)7)

The login details are provided to the users but in case they should forget their password they can request it. In real life situations for security reasons, this would not be a feature but to make it easier to use for now this is enabled. For example, the SQL statement to request the password for worker number 3 would be:

SELECT Password FROM Workers WHERE Username = Worker_3

Tasks Table – Post Normalisation (3NF)

There will also be a table with all tasks that have been assigned and completed within the database. For each task it will store the type of task, details about the task, the date it was completed and who it was completed by. The standard notation, format of the table, data dictionary for the table, explanation of the table attributes and SQL statements are shown here:

Standard Notation

Tasks (TaskID,TaskType,Object,Quantity,Info,DateAssigned,DateCompleted,WorkerID*,ManagerID*)

Key
Attribute – Primary Key Attribute
*Attribute** – Foreign Key Attribute

Table Format

TaskID	TaskType	Object	Quantity	Info	DateAssigned	DateCompleted	WorkerID*	ManagerID*
013	“Print”	“Documents”	30.0	“In colour ink”	12/03/21	14/03/21	004	002

An example record of data is shown in the table above to demonstrate exactly what the table would look like in a database.

Data Dictionary

Table Name	TASKS		
Primary Key	TaskID		
Foreign Keys	WorkerID , ManagerID		
Data Item	Data Type	Validation	Example Data
TaskID	Integer	NotNull	054
TaskType	String	NotNull	"Sort"
Object	String		"Pages"
Quantity	Real		20.0
Info	String		"A-Z"
DateAssigned	DateTime	NotNull	05/08/21
DateCompleted	DateTime		08/08/21
WorkerID	Integer	NotNull	017
ManagerID	Integer		009

Attributes

- TaskID – Unique number to identify each task
- TaskType – Shows what the task involves doing. Can be things such as printing, sorting or inputting data.
- Object – The actual item that the task needs to be performed on, for example the name of a document or a reference to any item
- Quantity – The number of things that the task needs to be performed on, for example printing 30 documents. (If left blank it will be assumed there is one thing)
- Info – Information about the task,for example which order to sort data in or which ink to use for printing
- DateAssigned – The date that the task was assigned on
- DateCompleted – The date that the task was marked as complete on
- WorkerID – The unique ID of the worker that the task was assigned to
- ManagerID – The unique ID of the manager that the task was assigned by

SQL Statements

All SQL statements needed for this table are:

To create the table:

```
CREATE TABLE Tasks
(TaskID NOT NULL  PRIMARY KEY,
TaskType NOT NULL,
Object,
Quantity,
Info,
DateAssigned NOT NULL,
DateCompleted,
WorkerID NOT NULL,
ManagerID;)
```

Tasks will be inserted into the table once completed using the INSERT INTO command. The exact SQL statement to insert the example record shown in the table format is shown below:

```
INSERT INTO Tasks (TaskID,TaskType,Object,Quantity,Info,DateAssigned,DateCompleted,
WorkerID,ManagerID)

VALUES (013,Print,30,Doc1,12/03/21,14/03/21,004,002)
```

The manager may request a list of all tasks that have been assigned and completed. The SQL statement for this is shown below:

```
SELECT * FROM tblTasks
```

The manager may request a list of all tasks assigned to and completed by a specific worker. For example the SQL statement to retrieve a list of all tasks assigned to the worker with ID 002 is shown below:

```
SELECT * FROM Tasks WHERE WorkerID = 002
```

The manager may want a list of only tasks assigned by them. The SQL statement for this situation for the manager with ID 001 would look like:

```
SELECT * FROM Tasks WHERE ManagerID = 001
```

Details Table – Post Normalisation (3NF)

Finally, there will be a table to store details about the workers which the algorithm uses to decide the best person for the task. For privacy reasons the data in this table would be collected only after gaining permission from the workers and it would be stored securely in the table and is only to be used for the purpose of the code, by the algorithm and by managers in order to decide who to assign a task to.

The standard notation, format of the table, data dictionary for the table, explanation of the table attributes and SQL statements for the table are shown below:

Standard Notation

Details (WorkerID,FirstName,LastName,Age,YearsWorked,JobRole)

Key
Attribute – Primary Key Attribute
*Attribute** – Foreign Key Attribute

Table Format

WorkerID	FirstName	LastName	Age	YearsWorked	JobRole
003	“Ben”	“Johnson”	43	12	“Receptionist”

Data Dictionary

Table Name	DETAILS		
Primary Key	WorkerID		
Foreign Keys			
Data Item	Data Type	Validation	Example Data
WorkerID	Integer	NotNull	003
FirstName	String	NotNull	Ben
LastName	String	NotNull	Johnson
Age	Integer	NotNull	43
YearsWorked	Integer	NotNull	12
JobRole	String	NotNull	Receptionist

Attributes

- WorkerID – The unique ID of the worker
- FirstName – The worker’s first name
- LastName – The worker’s last name or surname
- Age – The worker’s age
- YearsWorked – The number of years the worker has worked for the organisation (only full years are recognised so if somebody has worked 12 ½ years it will be recognised as 12 full years)
- JobRole – The job role the worker has within the organisation

SQL Statements

All SQL statements needed for this table are listed here:

To create the table:

```
CREATE TABLE Details
(WorkerID NOT NULL  PRIMARY KEY,
FirstName NOT NULL,
LastName NOT NULL,
Age NOT NULL,
YearsWorked NOT NULL,
JobRole NOT NULL;)
```

To insert the records of data into the table the insert command is required. The SQL statement to insert the example record shown in the table format is shown below:

```
INSERT INTO Details (WorkerID,FirstName,LastName,Age,YearsWorked,JobRole)
VALUES (003,Ben,Johnson,43,12,Receptionist)
```

Every year the age of the workers and the number of years they have worked at the company will change. Therefore the table will need to be updated. The SQL statements for this for the example record are:

```
UPDATE Details
SET Age = 44, YearsWorked = 13
WHERE WorkerID = 003
```

The WHERE condition ensures that only the chosen record gets updated and not any others.

The central algorithm requires access to information in this table to choose a worker best suited for each task. In order to retrieve data from the table the following SQL statement is used:

```
SELECT * FROM Details
```

In some cases the algorithm may use some attributes and not others from the table but it is useful and convenient to have access to everything so all attributes are selected.

The managers may want to manually assign a task and to help them decide they can retrieve details about the workers. Once again, all attributes may not be used each time but for convenience all attributes would be selected.

The manager may want details about all workers:

```
SELECT * FROM Details
```

Alternatively they may want only information for a specific worker. For example for worker with ID 007:

```
SELECT * FROM Details WHERE WorkerID = 007
```

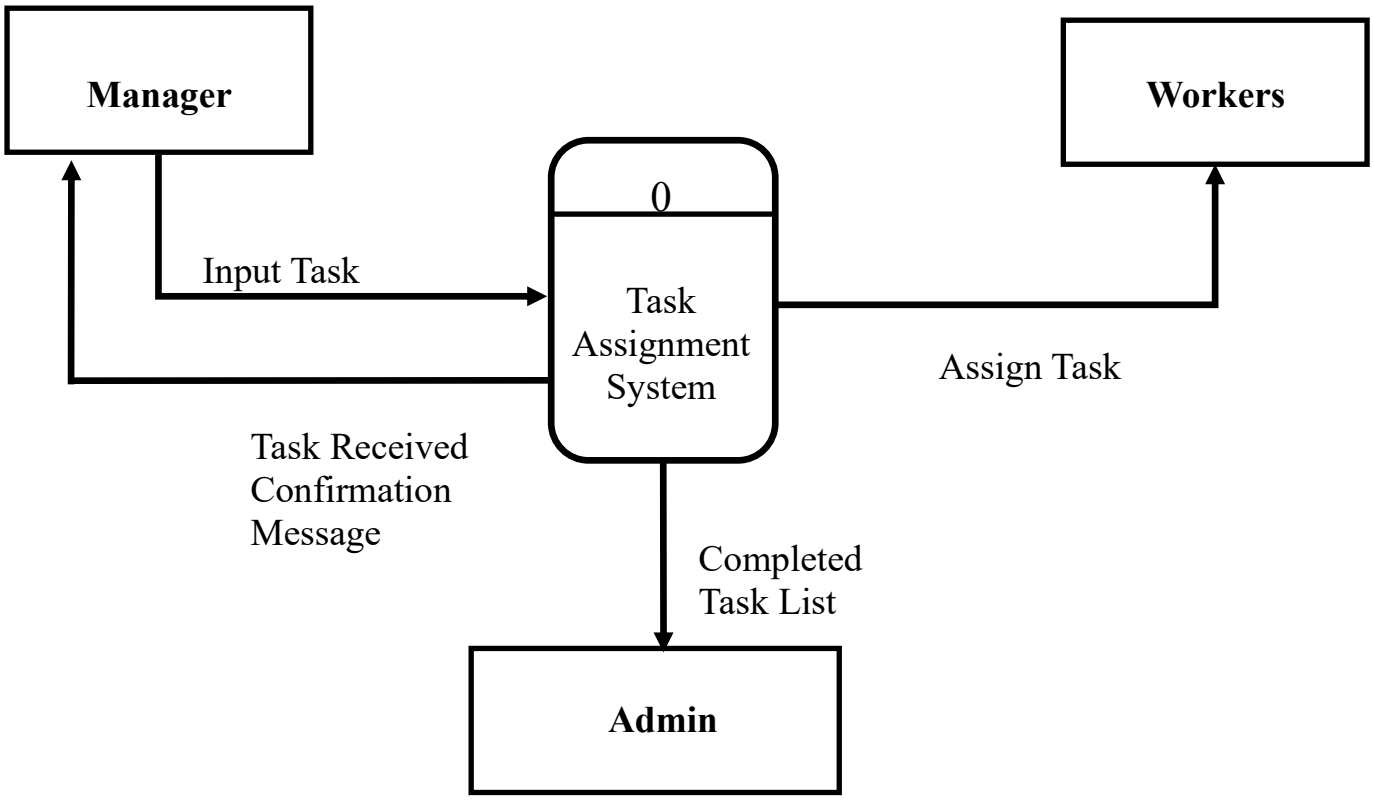
They could also want details about workers who have been working for a specific number of years. For example the SQL statement to retrieve the details of only workers who have worked at the organisation for at least 5 years would be:

```
SELECT * FROM Details WHERE YearsWorked >= 5
```

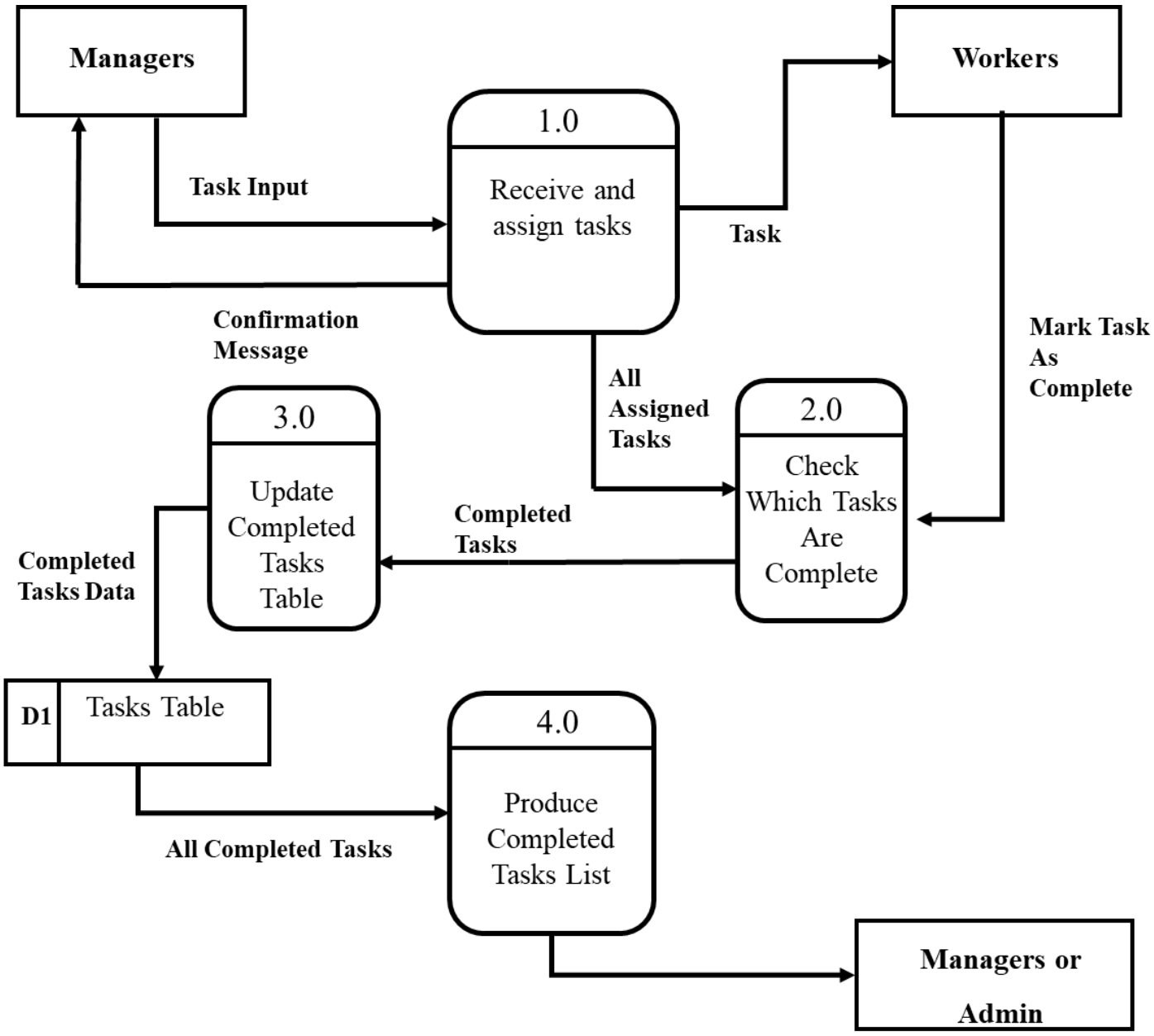
Data Flow Diagrams

Level 0 Context Diagram

The diagram showing the context of the overall system:



Level 1 Data Flow Diagram



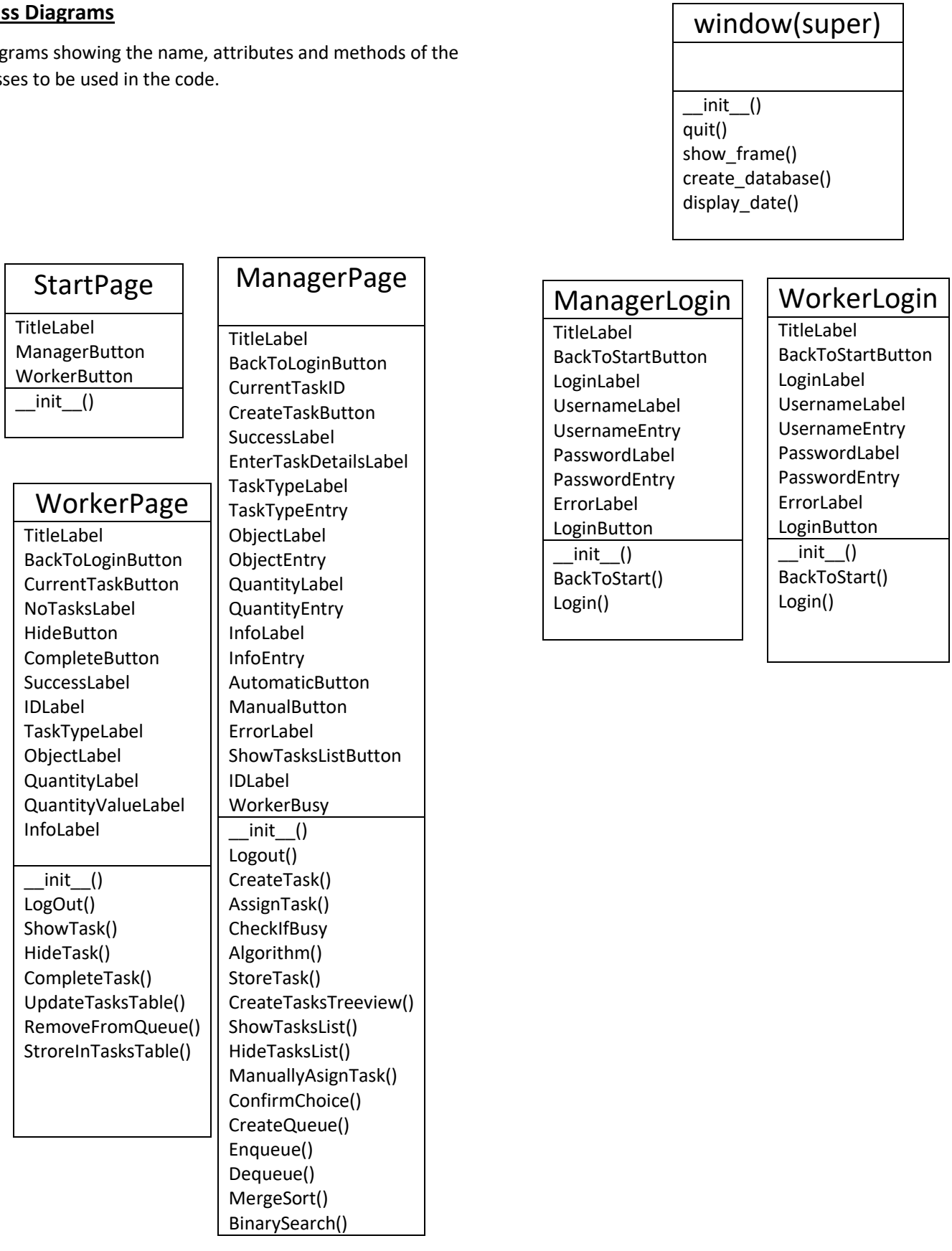
IPSO Chart

IPSO	Section of Program	Item
Input	Logging In	Worker Username Worker Password Manager Username Manager Password
	Inputting Task	Task Type Object Quantity Task Information
	Completing Task	Worker marks task as complete
	Request worker details	Manager chooses option to see all worker details for manual assignment
	Manual assignment	Manager manually selects a worker for the task
Process	Choosing worker for task	Algorithm decides most suitable worker for task
	Check if worker is available	Is the worker free?
	Add tasks to queue	Add the task to the queue if the worker chosen for the task is busy
	Manage queue	Check tasks in the queue and see if the workers chosen for them are free
	Assign task to worker	If worker is now free assign the task to them
	Dequeue items	Remove tasks from queue if they have been assigned
Storage	Storing Tasks in Table	Insert a new record into Tasks table when a task gets completed, and store: TaskID, TaskType, Quantity, Info, DateAssigned, DateCompleted, WorkerID, ManagerID.
Output	Assigning tasks to workers	Display task to chosen worker
	Completed tasks table	Output list of all completed tasks from Tasks table
	Worker details	Output list of details of workers from Details table when requested by manager
	Confirmation message	Message displayed to manager to confirm that task has been assigned

OOP Class Design

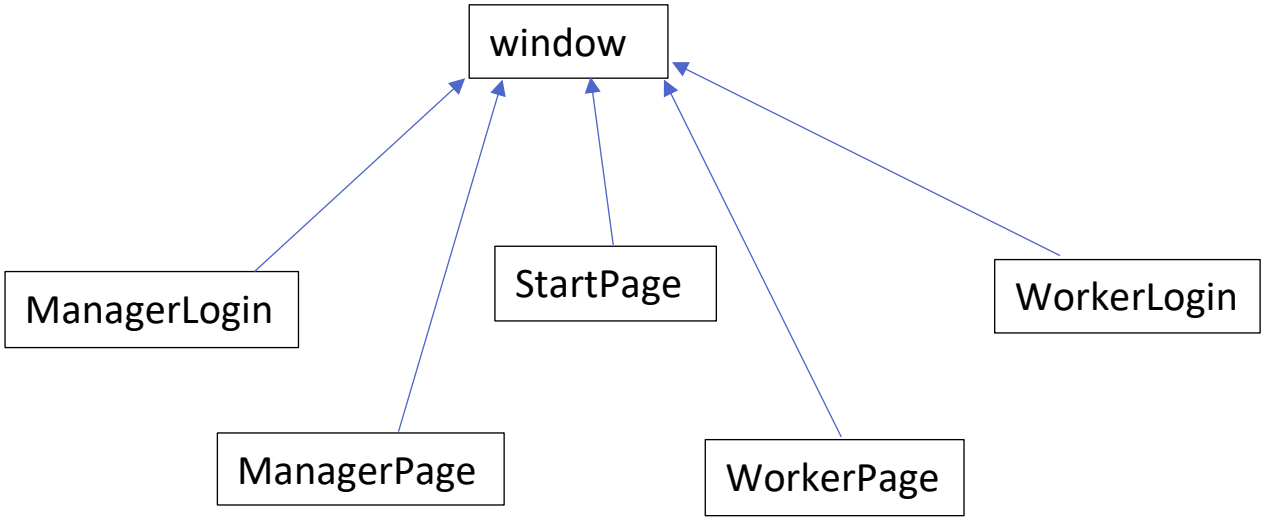
Class Diagrams

Diagrams showing the name, attributes and methods of the classes to be used in the code.



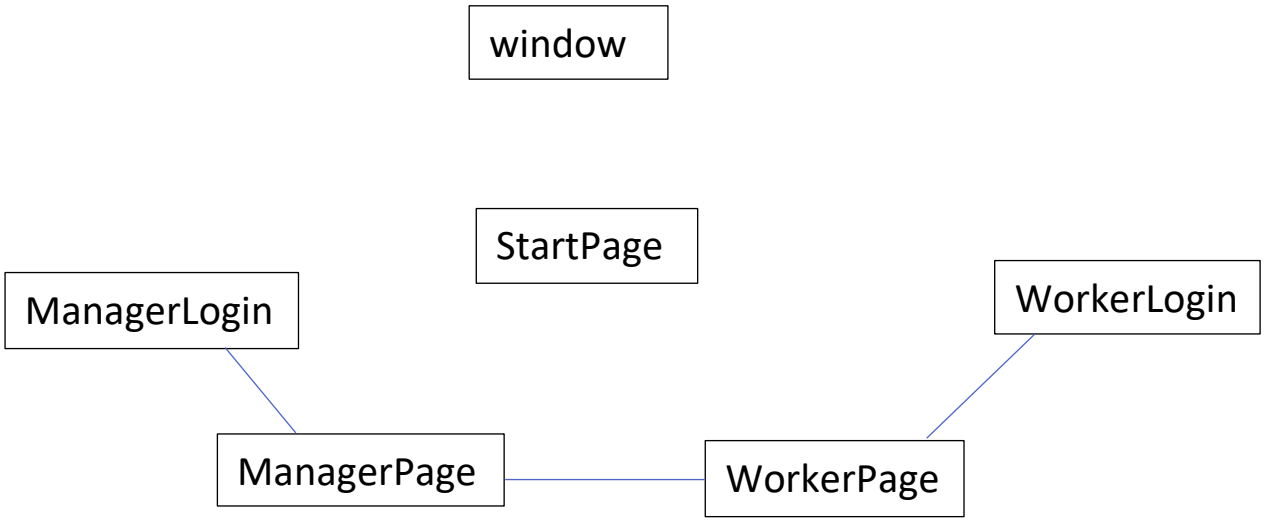
Inheritance Diagram

Shows inheritance between classes.

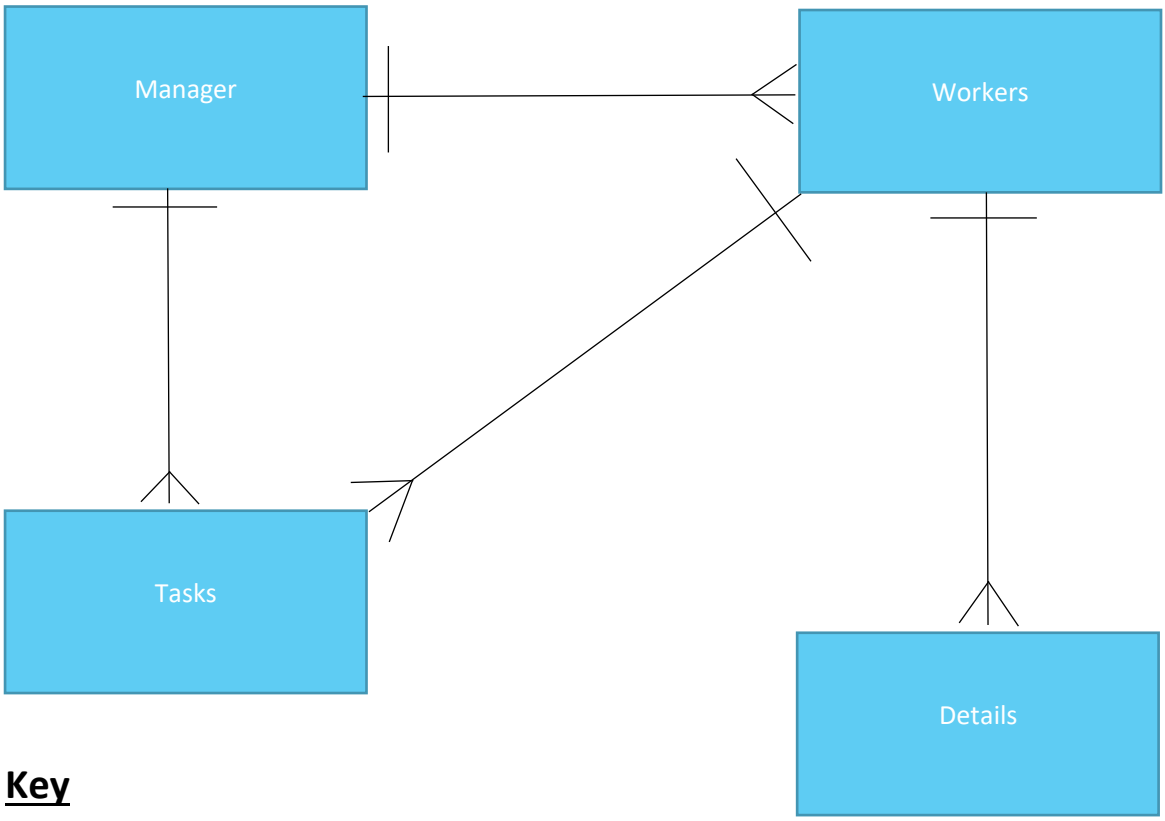


Relations Diagram

Shows which classes are related to each other (use each other's methods or attributes). All classes inherit from the window superclass and use its methods, so these relations aren't shown in this diagram.



Entity Relationship Diagram

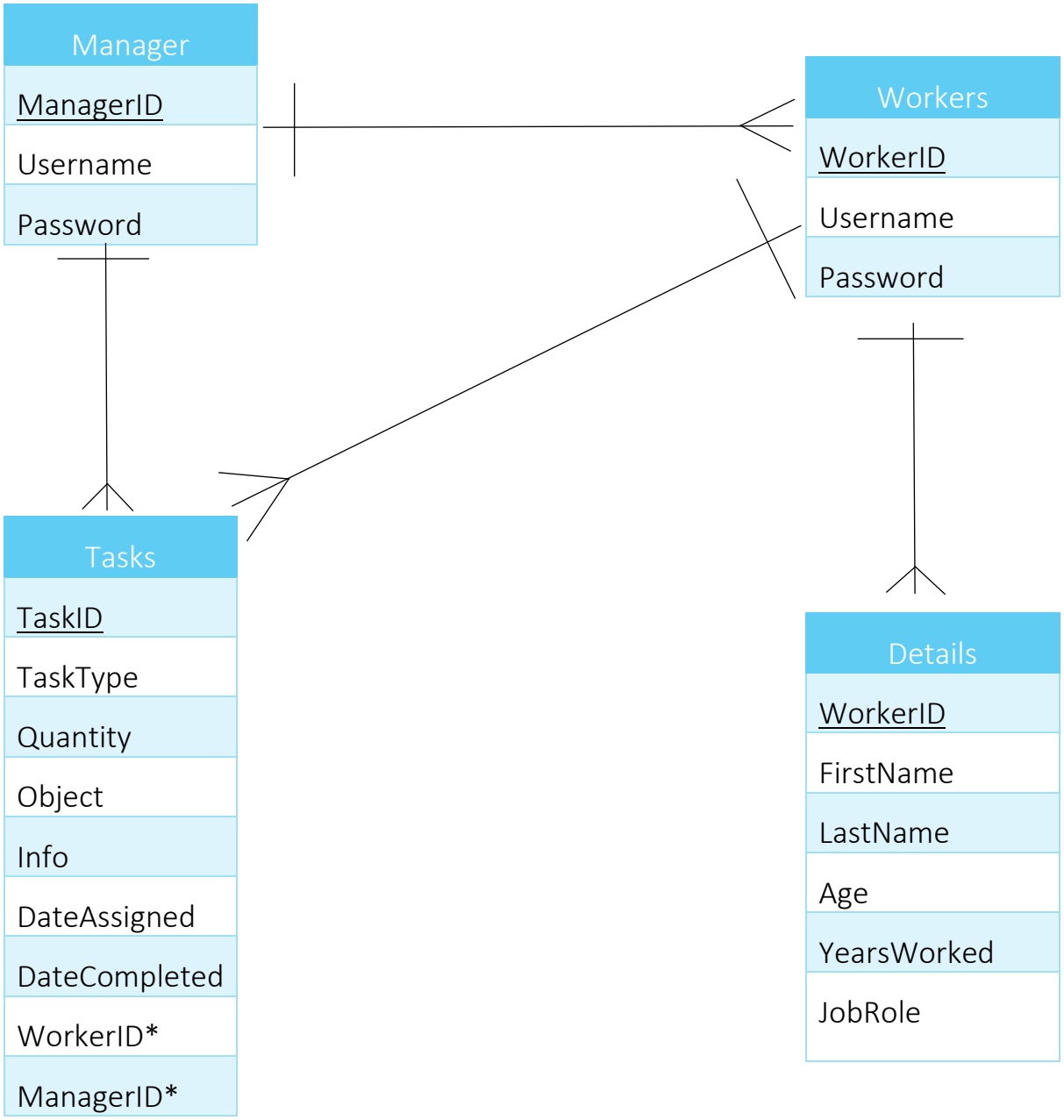


Key

One

Many

Entity Attribute Model



Key

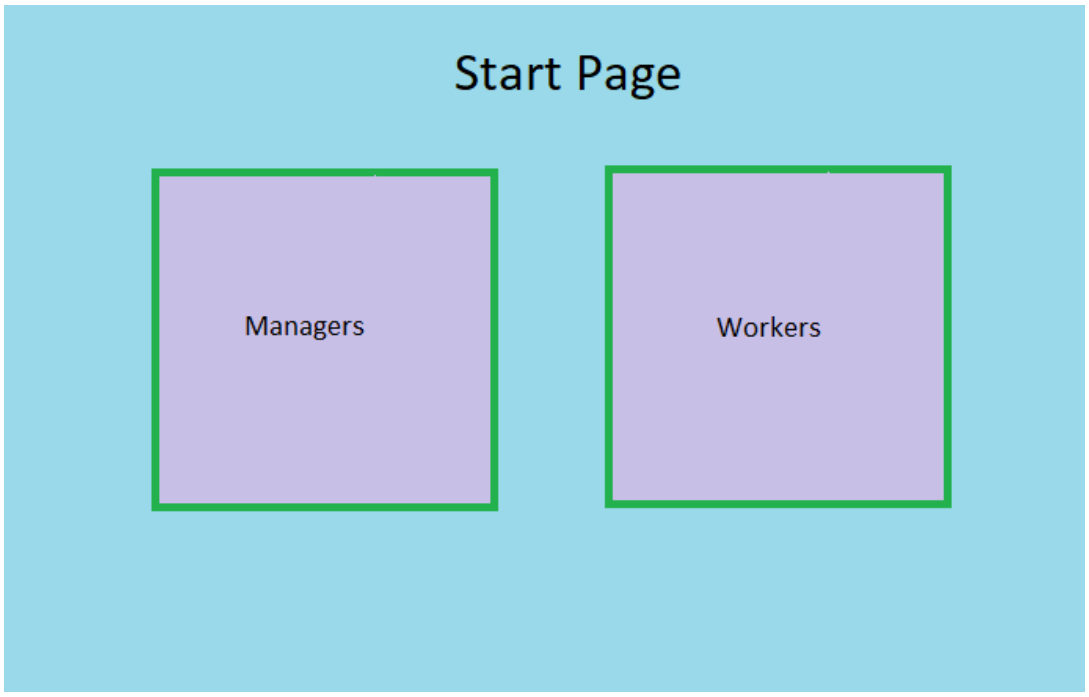
Key

Attribute – Primary Key Attribute

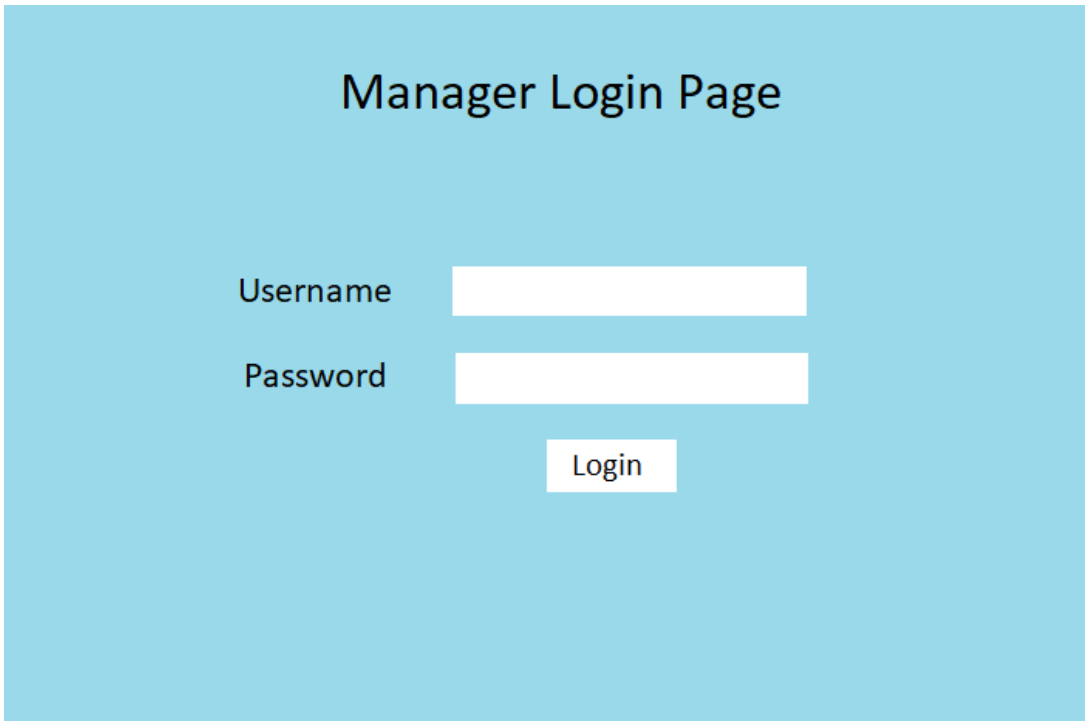
*Attribute** – Foreign Key Attribute

User interface design

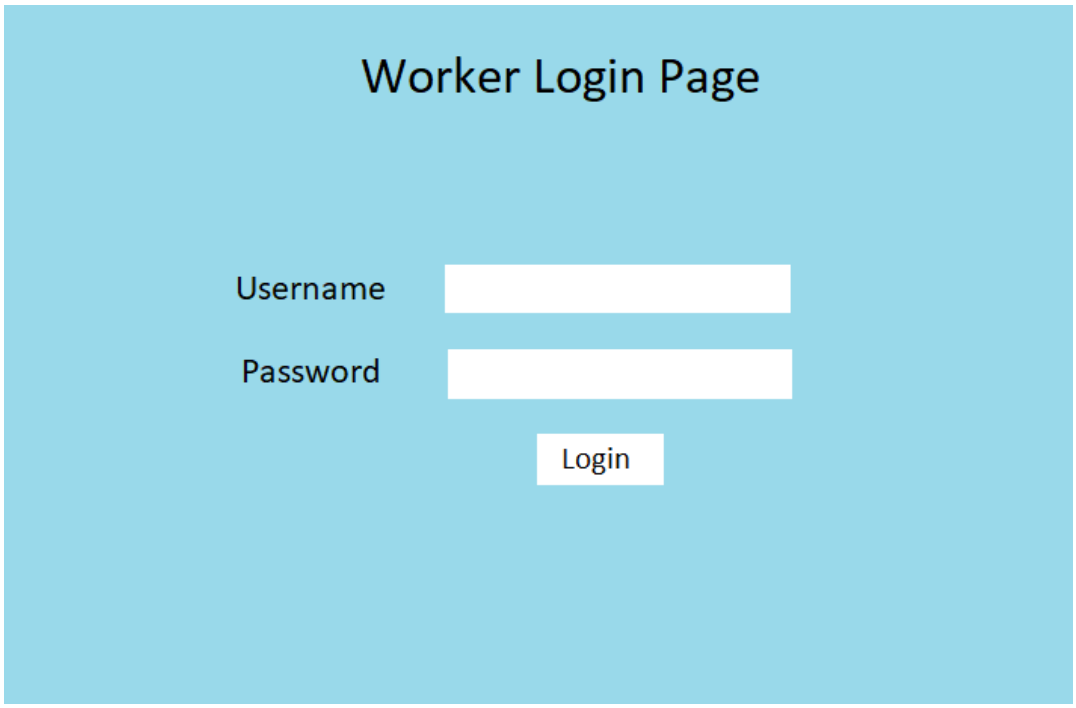
The interface for the program will consist of multiple pages. The first page will be the start page. It will have two buttons, one for managers and one for workers. Each button moves to separate login pages.



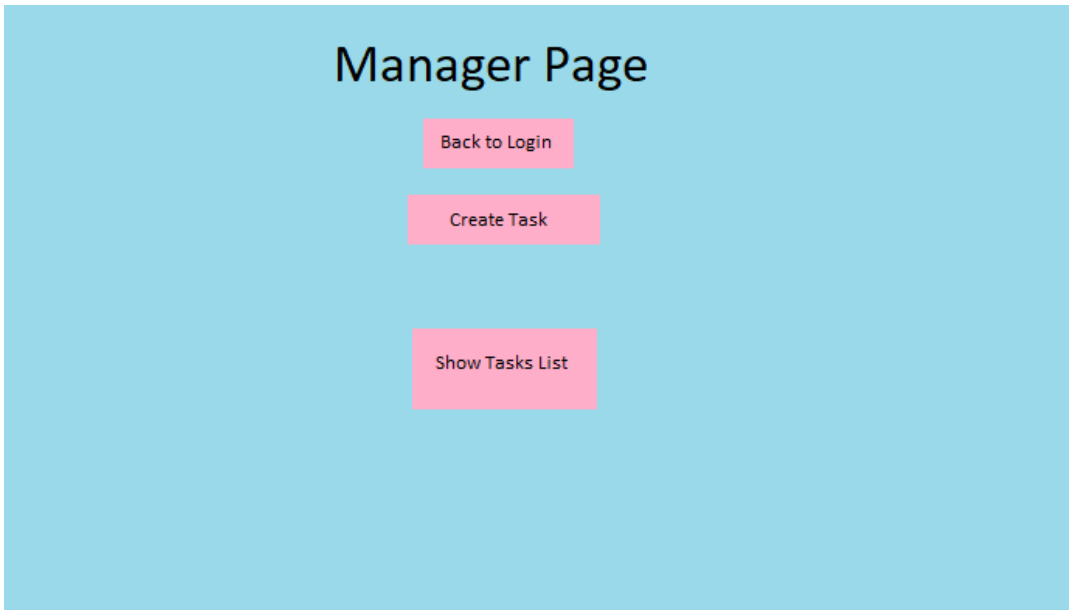
When the “Managers” button is clicked the user is taken to the Manager login page. They will need to enter their own username and password. If the login details are incorrect a message will be displayed and the user needs to try again. If the details are correct they will be taken to the “Manager Page”.



When the “Workers” button is clicked the user is taken to the Worker login page. They will need to enter their own username and password. If the login details are incorrect a message will be displayed and the user needs to try again. If the details are correct they will be taken to the “Worker Page”.

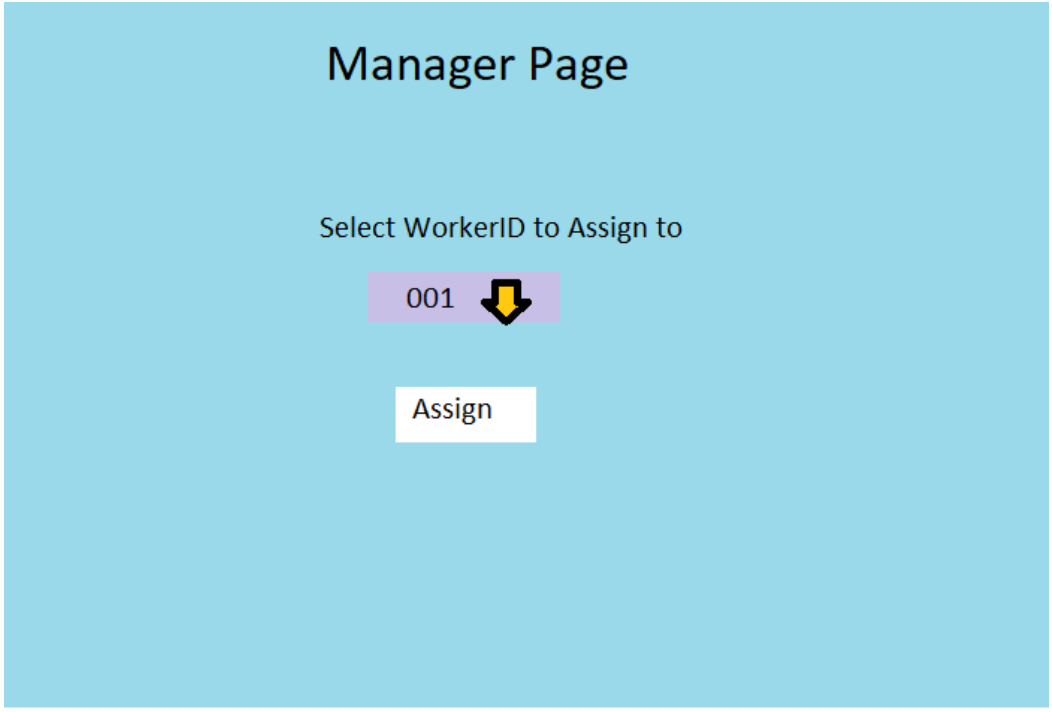


This is the “Manager Page”. There are initially 3 buttons. One of which allows the user to log out and go back to the “Manager Login Page”. There is a button to show the Tasks List which will display a list of all tasks that have been assigned. There is also a button to create a task which is to be assigned.

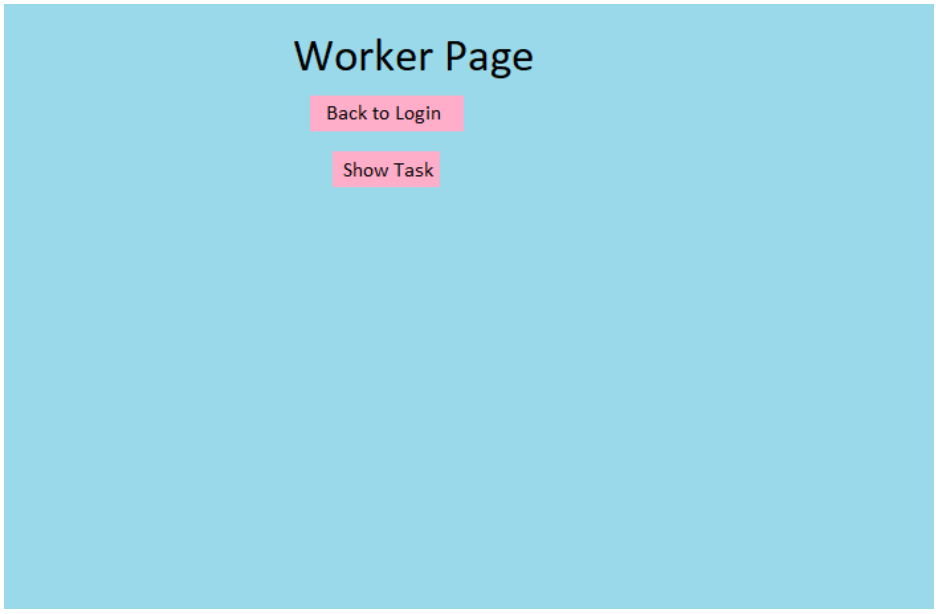


When the “Create Task” button is pressed this will show up on the page. The details of the task are entered and then one of the assign buttons is pressed. If the “Assign button is pressed then the task will automatically be assigned and then the page will return to normal.

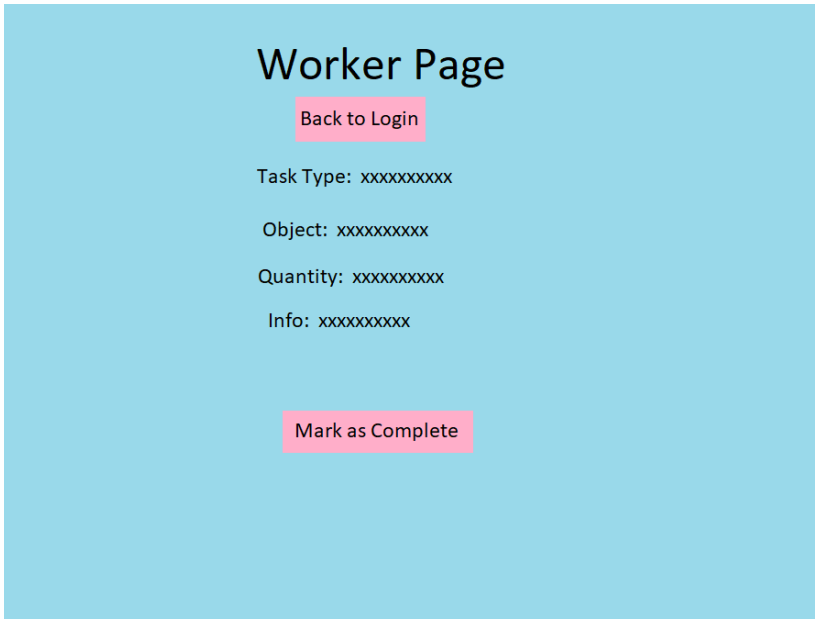
If the “Assign Manually” button was pressed then this will come up. The user should select a worker ID using the drop down menu and then press assign. The task will be assigned to the chosen worker and the page will return to normal.



When a worker logs in, this page is shown. There is a button to logout and go back to the login page. There is also a button to show the current task to be completed. When the “Show Task” button is pressed the details of the current task for the specific worker will be shown on the page. If there is no current task, a message stating that there is no task will be shown.



If the “Show Task” button is pressed and there is a task to show it will be displayed like this. A new button will appear to mark the task as complete. When this button is pressed the task will be marked as complete in the software and the page will be restored to how it was originally.

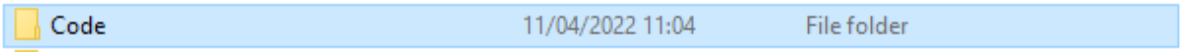


Security and Integrity of Data

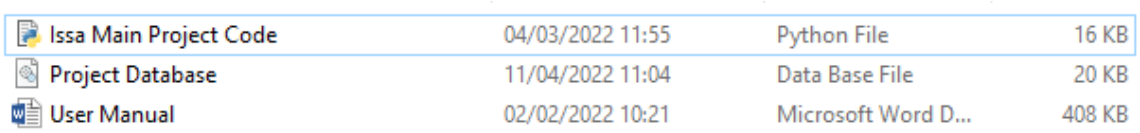
- All data used within the system is stored securely in the database and nowhere else
- All passwords are random collections of characters so they cannot easily be guessed by others
- Each user is provided with only their own username and password
- All information collected about users isn't collected until permission has been granted to collect and use their personal information and the information which is collected is stored securely, not to be seen by anyone else and is only to be used by the code

File Structure and Organisation

All the program files are stored securely and all together in a folder called "Code".



Within this folder are all necessary files to fully run and optimise the user's experience of the software.



There is the main python file which is where the whole code is stored. There is also the database for the project in case the users want to inspect and check up on their data stores. Finally there is a user manual to instruct and guide the users on how to use the code to its full potential and get the best out of it. The user manual also contains a copy of all their login details, in case they forget them.

Algorithms

Merge Sort

```
function MergeSort(self,List):
    if len(List) > 1 then
        mid = len(List) // 2
        LeftList = List[:mid]
        RightList = List[mid:]
        MergeSort(LeftList)
        MergeSort(RightList)
```

```
    #add elements from right and left into
    #merged list in order
    i = 0
    j = 0
    k = 0
    while i < len(LeftList) and j < len(RightList) then
      if LeftList[i] < RightList[j] then
        List[k] = LeftList[i]
        i = i + 1
      else:
        List[k] = RightList[j]
        j = j + 1
      k = k + 1
      endif
    endwhile
      #check if left list has elements not merged
    while i < len(LeftList) then
        List[k] = LeftList[i]
        i = i + 1
        k = k + 1
    endwhile
      #check if right list has elements not merged
    while j < len(RightList) then
        List[k] = RightList[j]
        j = j + 1
        k = k + 1
    endwhile
  return(List)
  endfunction
```

Binary Search

```
Function BinarySearch(List, item ,count,left,right):
  mid = (1+left+right)//2
  if count > (len(List)//2) then
```

```
        return FALSE
    endif
    if left < 0 then
        return FALSE
    endif
    if List[mid] == item then
        return TRUE
    elif item <List[mid] then
        return BinarySearch(List,item,(count+1),left,mid-1)
    elif item> List[mid] then
        return BinarySearch(List,item,(count+1),mid+1,right)
     endif
end Function
```

I used a Binary Search rather than a Linear Search because the time complexity of a Linear Search is O(n) and a Binary Search is O(log n).

I used a Merge Sort rather than a Bubble Sort because the time complexity for a Bubble Sort is $O(n^2)$ and a Merge Sort is O(nlog n)

I chose to code both of these recursively rather than iteratively because using recursion makes them both shorter, easier to code and reduces the time complexity for both.

Data structures

If a task gets assigned but the chosen person is busy or unavailable to carry out the task then it will be placed in a linear queue. When the chosen person's status becomes available again the task will be dequeued and assigned to then person. I chose to use a queue for this because it is the most suited as the first item to be put in will be the first to come out, allowing to tasks to reach the workers in the order they were assigned by the manager. A linear queue is more suitable than a priority queue because it is easier to implement and there is no need for a special priority system, the items leave the queue based on the order they entered it.

The structure of the queue is shown below:

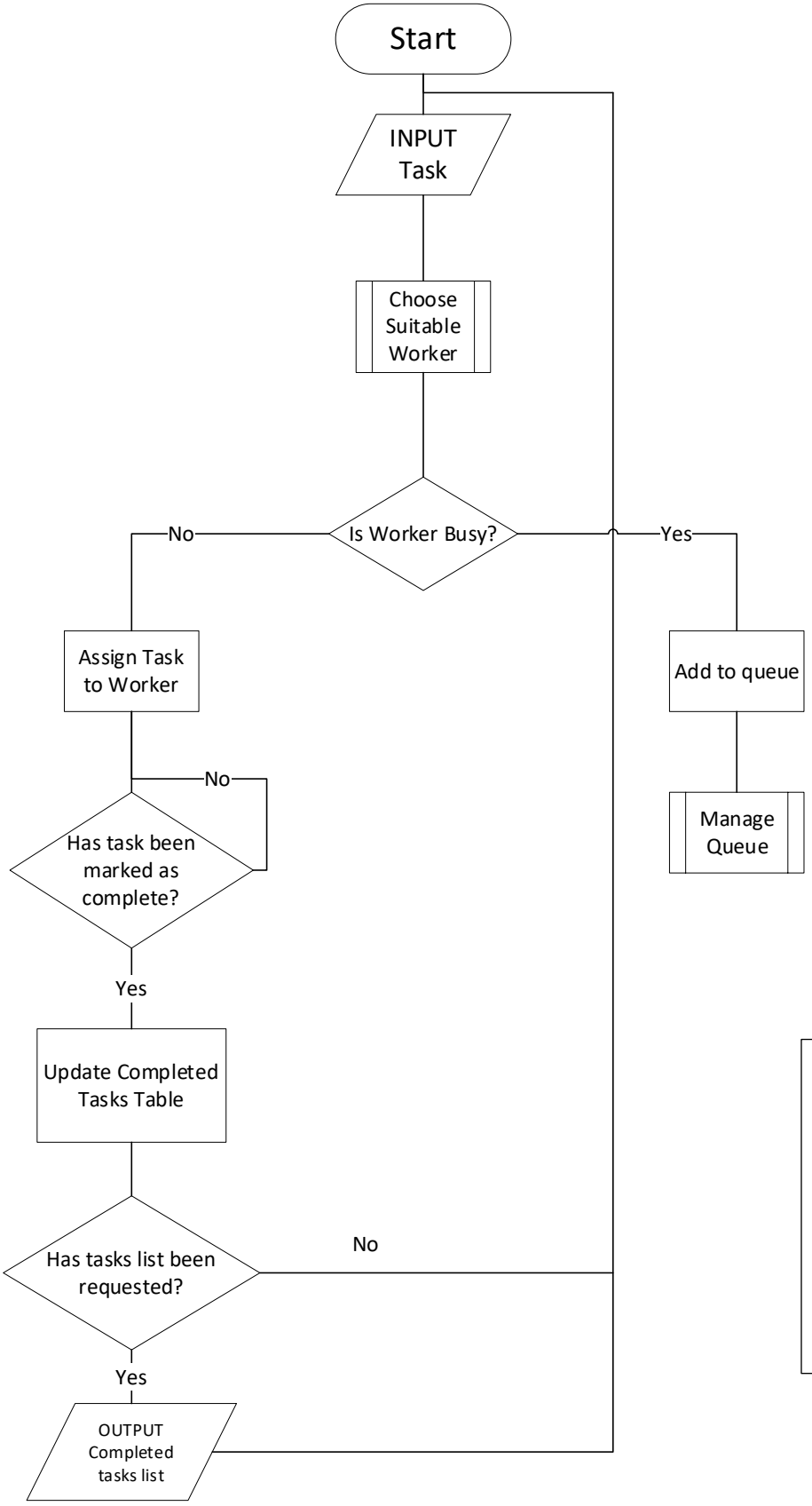
[0]	TaskID
[1]	
[2]	
[3]	
[4]	
[5]	
[6]	
[7]	
[8]	
[9]	

The task IDs are input into the table with their index position shown in the left column. The task at the top has the highest priority and will leave the queue first.

Hardware Selection

The code was created in Python 3.8.8 using built in libraries so there is no other requirements to run the system other than having any version of Python 3 installed. I used DB Browser for SQLite for checking and testing the code. This can be installed and used by the clients to view the database but this is not essential. The only hardware requirements are to have a functioning computer device with a connected keyboard and mouse to interact with the system. Any additional hardware is optional and not a requirement to run and use the system.

System Flowchart



The system flow chart shows an overview of all processes involved in the main functioning system of the code. It is a representation of the process of managing and assigning tasks, which is the fundamental part of the whole program.

Technical Solution

Below is all the code for the entire software. It is split up into sections, with annotations for each section. The code is from the version with minimal comments in to save space as these arent necessary to have along with the annotations. Only the really important comments have been left in.

Firstly I needed to import all the rquired built in libraries. These include tkinter for the GUI, SQLite3 for the database, datetime and random.

```
from tkinter import *
import tkinter as tk
from tkinter import ttk
import sqlite3
import random
from datetime import date, datetime , time
from tkinter import messagebox
```

I defined all constants which needed to be used within the code.

Firstly, here are the font type constants where I created constants for varying sizes of fonts so they can easily be used within the code by calling the constant name. Each one is the same font type but has a different size.

```
TITLE_FONT = ("Verdana",26)
LARGE_FONT = ("Verdana",20)
MEDIUM_FONT = ("Verdana",16)
SMALL_FONT = ("Verdana",14)
MINI_FONT = ("Verdana",12)
TINY_FONT = ("Verdana",10)
```

The second type of constant to be used within the code was the date. The ‘date_today’ variable was created and assigned today’s date using the function from the built-in library ‘datetime’. This was then changed into the correct day-month-year format and stored in the new variable ‘str_date_today’, which would be used as a constant throughout the code.

```
date_today = date.today()
str_date_today = date_today.strftime("%d/%m/%y")
```

Next, I created the window class. This was designed to act as a superclass where all other classes in the code would inherit from it so they could be set up the same way and use its methods and attributes.

The method and attribute names in this super class are all in Snake Case whereas the methods and attributes in the other classes are all in Camel Case, so they can be easily distinguished. This is the most important class so I have annotated it line by line.

```
class window(tk.Tk):
```

The constructor method initialises the whole window.

```
    def __init__(self):

        tk.Tk.__init__(self)

        tk.Tk.wm_title(self, "Issa's Project")
        tk.Tk.geometry(self,"800x600")
        tk.Tk.resizable(self,False,False)


        container = Frame(self)
        container.pack(side="top", fill="both", expand=False)


        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.create_database()

        self.frames = {}

        for F in (StartPage, ManagerLogin, WorkerLogin,
ManagerPage,WorkerPage):

            frame = F(container,self)

            self.frames[F] = frame

            frame.grid(row = 0, column = 0, sticky="nesw")

        self.show_frame(StartPage)

        self.set_up_menubar(container)


        self.display_date()
```

Here tkinter is initialised

The title is set here. The dimensions of the window are set. Resizable is False meaning the window can't be enlarged by the user.

A variable called container is created which is a frame that will hold everything in the window. container is packed to fit to the whole window.expand is set to False because the window wont expand anyway so it isn't needed.

The grid is configured for container. The minimum lengths for the column and row are both 0. weight = 1 gives both equal priority

The create_database method is called to create the database.

An empty library is created to store all the frames from each class.

The for loop iterates through a list of the classes and makes each one a frame based on the container variable. It also adds each one to the library of frames. And each frame is assigned to the grid, for now the row and column have been set to 0 as they aren't going to be used. sticky stretches everything equally to fit the window size.

StartPage is the first page to be showed straightaway.

The menubar is created and set up onto the container.

The date is displayed.

I created the 'quit' method to exit the system by using the option in the taskbar. This method displays a pop up message box onto the screen asking the user, if they are sure that they want to quit. If they say yes, the window gets destroyed.

```
    def quit(self):
        get_exit = messagebox.askyesno(title="Quit",message="Are you sure you
want to quit?")
        if get_exit > 0:
           tk.Tk.destroy(self)
```

This method creates the menubar. It sets up the 'menubar' attribute as a tkinter menu and then sets up the 'filemenu' attribute as a tkinter menu. Onto 'filemenu' a command is added which is labelled to the user as "Close Program" and when pressed it will call the 'quit' method. 'filemenu' is then added as a cascade which made it an option onto 'menubar'. Then 'menubar' was added onto the window.

```
    def set_up_menubar(self,container):
        self.menubar = tk.Menu(container)
        self.filemenu = tk.Menu(self.menubar, tearoff = False)
        self.filemenu.add_command(label="Close Program",command= self.quit)
        self.menubar.add_cascade(label="File",menu = self.filemenu)
        tk.Tk.config(self,menu = self.menubar)
```

The 'show_frame' method is very important throughout the code. The way the code is designed, is that multiple frames are created and each one acts as a page. This method brings a frame on top of the others so it can be used. A variable 'PageName' is passed in to the method, which is the name of the page that needs to be raised to the top. The method finds the page given by 'PageName' within the library of frames and then assigns it to a variable called 'frame'. 'frame' is then raised using the built in tkinter function 'tkraise()'.

```
    def show_frame(self,PageName):
        frame = self.frames[PageName]
        frame.tkraise()
```

The 'create_database' method creates the database using SQL Commands. It assigns the intended database name to the variable 'new_path'. Then a connection is created using the built in sqlite3 function 'connect( )' to the 'new_path'. The connection is assigned to the variable 'data'.

```
    def create_database(self):
        new_path = "Project Database.db"
        connection = sqlite3.connect(new_path)
        data = connection.cursor()
```

Using the 'execute' built in function I was able to write in my SQL statements. I used the 'CREATE TABLE' command to create the tables. I used the 'IF NOT EXISTS' line to ensure that the tables don't get repeated.

Firstly, I created the 'Managers' table which would store the Username and Password for managers along with the primary key 'ManagerID'. The primary key was 'NOT NULL', meaning it cannot be left blank. The data type for the primary key was 'TEXT' rather than 'INTEGER' because later in the code, the primary key needs to be manipulated, so it must be a string data type.

```
        data.execute("""CREATE TABLE IF NOT EXISTS "Managers"(
        "ManagerID" TEXT PRIMARY KEY NOT NULL,
        "Username" TEXT,
        "Password" TEXT
    );""")
```

Then I created the 'Workers' table. Similarly to the 'Managers' table it stored the Usernames and Passwords for the workers along with the primary key 'WorkerID' which was also 'NOT NULL' and the data type was 'TEXT' for the same reason as with the 'Managers' table.

```
        data.execute("""CREATE TABLE IF NOT EXISTS "Workers"(
        "WorkerID" TEXT PRIMARY KEY NOT NULL,
        "Username" TEXT,
        "Password" TEXT
    );""")
```

I created the 'Tasks' table with 'TaskID' as the primary key and all other task details as attributes of the 'TEXT' data type. There are also two foreign keys: 'ManagerID' from 'Managers' and 'WorkerID' from 'Workers'. These foreign keys built relations between the tables.

```
        data.execute("""CREATE TABLE IF NOT EXISTS "Tasks"(
        "TaskID" TEXT PRIMARY KEY NOT NULL,
        "TaskType" TEXT,
        "Object" TEXT,
        "Quantity" TEXT,
        "Info" TEXT,
        "DateAssigned" DATETIME,
        "DateCompleted" DATETIME,
        "WorkerID" TEXT,
        "ManagerID" TEXT,
        FOREIGN KEY("WorkerID") REFERENCES "Workers"("WorkerID"),
        FOREIGN KEY("ManagerID") REFERENCES "Managers"("ManagerID")
    );""")
```

The final table to be created was 'Details'. This stores details about the workers including their first name last name and job role, all attributes with the 'TEXT' data type and age and years worked as 'INTEGER'. The primary key for this table was 'WorkerID' which is also a foreign key from the 'Workers' table.

```
        data.execute("""CREATE TABLE IF NOT EXISTS "Details"(
        "WorkerID" TEXT PRIMARY KEY NOT NULL,
        "FirstName" TEXT,
        "LastName" TEXT,
        "Age" INTEGER,
        "YearsWorked" INTEGER,
        "JobRole" TEXT,
        FOREIGN KEY("WorkerID") REFERENCES "Workers"("WorkerID")
    );""")
```

The 'commit()' function was used to save the changes to the database.

```
        connection.commit()
```

I executed a SELECT statement to select all attributes from the 'Managers' table and assigned the results to a variable called 'my_list' using the 'fetchall()' function. I used an 'if' statement to check whether the table was empty or not and if it is then the whole database is assumed to be empty so all necessary data is inserted using the INSERT command. The manager an worker login details were inserted with their primary keys. The worker details would be inserted later by the user themselves and the tasks would be inserted after they get created and assigned later on.

```
        data.execute("""SELECT * FROM Managers""")
        my_list = data.fetchall()

        if len(my_list) == 0:

            data.execute("""INSERT INTO Managers(ManagerID,Username,Password)
    Values("001","Manager_1","8_x4zAFBg")
    """)

            data.execute("""INSERT INTO Managers(ManagerID,Username,Password)
    Values("002","Manager_2","{Cy:2?MC>")

    """)

            data.execute("""INSERT INTO Managers(ManagerID,Username,Password)
    Values("003","Manager_3","L4#g)tsD&")
    """)

            data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
    Values("001","Worker_1","md7P@F)7")""")
```

```
        data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("002","Worker_2","{mz&4=P&")""")

        data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("003","Worker_3","6^Te{`7D")""")

        data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("004","Worker_4","J>w$5Gjy")""")

        data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("005","Worker_5","Q#e<2Dda")""")

        data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("006","Worker_6","$Dw4vcC@")""")

        data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("007","Worker_7","E(;5x*6K")""")

        data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("008","Worker_8",".8UW#mb4")""")

        connection.commit()
```

This method displays the date onto the page by creating a label and adding the str_date_today constant as the text and setting the font size using the MEDIUM_FONT constant. The label is then placed onto the page using the coordinates which place it into the top left corner.

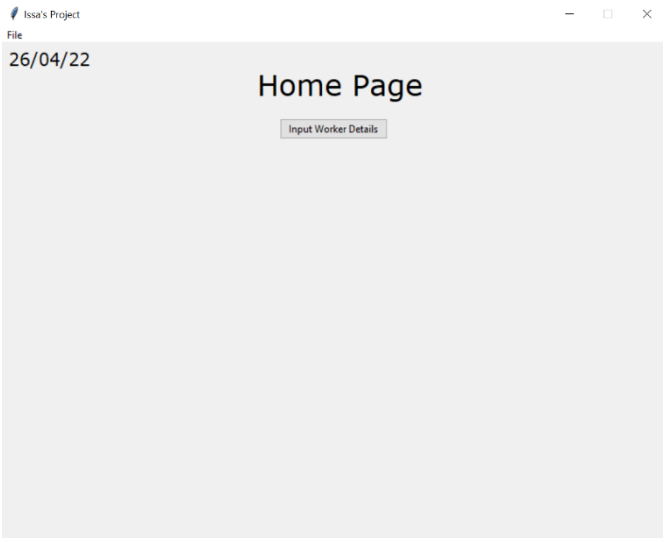
```
    def display_date(self):
        date_label = Label(self, text = str_date_today, font = MEDIUM_FONT)
        date_label.place(x=6,y=3)
```

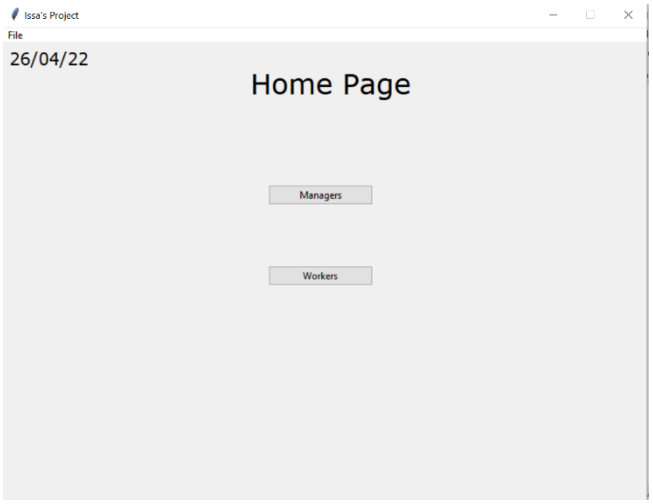
From this class a bas template for a window that all classes use is created. The base window would look something like this

The next class that is created is the StartPage.

The constructor method initialises the page, using the template from the parent class. It calls the SetCurrentWorkerID and DefineWidgets methods. Then it places the title label at the top middle section of the screen. Then it uses the CheckDetails method to check if the Details table in the database is full. If it isn't full it will pack the DetailsButton which allows the user to enter in details to complete the table. If it is already full, the ManagerButton and WorkerButton are placed. These buttons allow the user to access the login pages for both managers and workers.

```
class StartPage(tk.Frame):

    def __init__(self, parent, controller):
        Frame.__init__(self,parent)

        self.SetCurrentWorkerID()

        self.DefineWidgets(controller)

        self.TitleLabel.place(y=25,x=305)


        DetailsComplete = self.CheckDetails()
        if DetailsComplete == False:
            self.DetailsButton.pack(pady=90)
        else:
             self.ManagerButton.place(x=330,y=175)

             self.WorkerButton.place(x=330,y=275)
```





This method simply creates all the widgets such as labels and buttons that are to be used on this page. It also creates the entry boxes that will be used by the user to input details in and the variables to store the details in which initially get assigned blank strings. Finally it creates a list and puts in all the widgets that have been created. The list is to be used later in the code.

```
    def DefineWidgets(self,controller):
        self.TitleLabel = Label(self, text="Home Page", font=TITLE_FONT)

        self.DetailsButton = ttk.Button(self, text="Input Worker
Details",width = "20", command =lambda: self.GetDetails())

        self.ManagerButton = ttk.Button(self, text="Managers",width = "20",
command =lambda: controller.show_frame(ManagerLogin))

        self.WorkerButton = ttk.Button(self, text="Workers",width = "20",
command =lambda: controller.show_frame(WorkerLogin))

        FirstName =
""
        LastName = ""
        Age = ""
        YearsWorked = ""
        JobRole = ""

        self.EnterDetailsLabel = Label(self, text="Enter Details Below:",
font=MEDIUM_FONT)

        self.FirstNameLabel = Label(self, text="First Name", font=SMALL_FONT)
        self.FirstNameEntry = ttk.Entry(self, textvariable = FirstName)

        self.LastNameLabel = Label(self, text="Last Name", font=SMALL_FONT)
        self.LastNameEntry = ttk.Entry(self, textvariable = LastName)

        self.AgeLabel = Label(self, text="Age", font=SMALL_FONT)
        self.AgeEntry = ttk.Entry(self, textvariable = Age)

        self.YearsWorkedLabel = Label(self, text="Years Worked",
font=SMALL_FONT)
        self.YearsWorkedEntry = ttk.Entry(self, textvariable = YearsWorked)

        self.JobRoleLabel = Label(self, text="Job Role", font=SMALL_FONT)
        self.JobRoleEntry = ttk.Entry(self, textvariable = JobRole)

        self.SaveButton = ttk.Button(self, text="Save", command =lambda:
self.SaveDetails())
```

```
        self.ErrorLabel = Label(self, text="All Fields Must Be Filled In
",font=MINI_FONT, fg = "red")

        self.WidgetList =
[self.EnterDetailsLabel,self.FirstNameLabel,self.FirstNameEntry,self.LastNameL
abel,
                  self.LastNameEntry,self.AgeLabel,self.AgeEntry,self.YearsWor
kedLabel,self.YearsWorkedEntry,
                  self.JobRoleLabel,self.JobRoleEntry,self.SaveButton]
```

This method resets the page by removing every widget within the widget list. Rather than writing out each widget and using the forget() function I used iteration to go through the list and forget each item which is much quicker.

```
    def ResetPage(self):
        for x in self.WidgetList:
              x.forget()
```

This method calls to CheckDetails method to see if the Details table is full. If it isn't it calls the EnterDetails method. If it is full then the DetailsButton is removed and the MangagerButton and WorkerButton are placed onto the page.

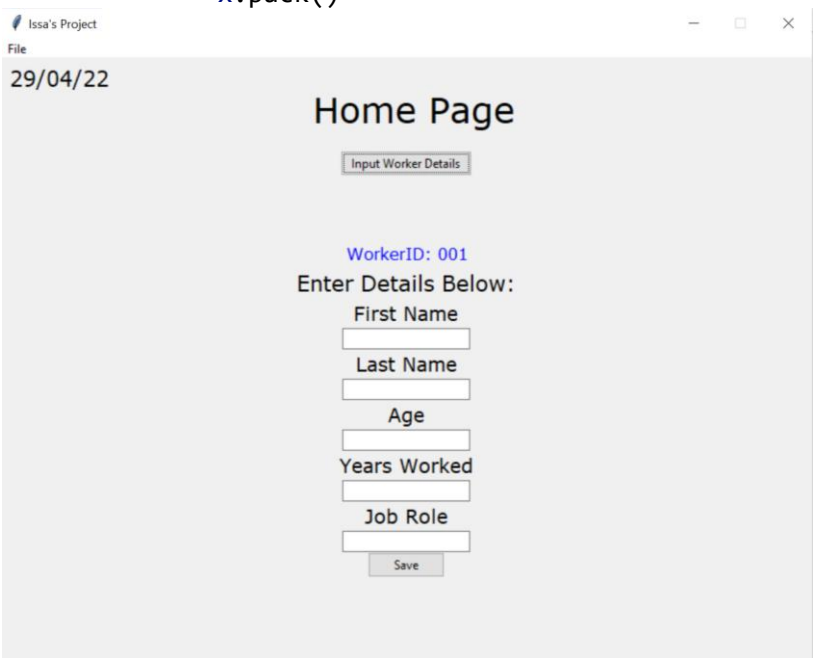
```
    def GetDetails(self):
        DetailsComplete = self.CheckDetails()
        if DetailsComplete == False:
           self.EnterDetails()
        else:
             self.ResetPage()
             self.DetailsButton.forget()
             self.ManagerButton.place(x=330,y=175)

             self.WorkerButton.place(x=330,y=275)
```

This method checks if the details table is full. It creates a variable 'Complete' and initially assigns True to it. In the try section the code tries to select all attributes from the last record in details where the WorkerID is 008. It doesn't need to check every record because each record cannot be filled without the previous one being filled so only the last one needs to be checked. A for loop is used to iterate through the whole record and if any field is empty then Complete is set to False, otherwise it will just stay True. If the try part fails it will be because there is no data in the table and the code still tries to select it, therefore exception handling has been used here so in case this error occurs, the code will not crash and instead the except part will be run which assumes that the table is empty so Complete is set to False. Finally Complete is returned, holding a Boolean value of either True or False.

```
    def CheckDetails(self):
        Complete = True
        try:
            new_path = "Project Database.db"
            connection = sqlite3.connect(new_path)
            data = connection.cursor()
            data.execute("""SELECT * FROM Details WHERE WorkerID LIKE "008"
""")
            details = data.fetchall()
            for i in range(0,6):
                  if details[0][i] == None:
                     Complete = False
        except:
               Complete = False

        return Complete
```

This method allows the user to input the details. Firstly a new label is created which will show the current Worker ID for each set of details that is being entered. The CurrentWorkerID attribute is set using the SetCurrentWorkerID method at the start of the code and is incremented using the IncrementCurrentWorkerID method when necessary. It is attached to the label and placed onto the screen. Then all of the labels and entries from the WidgetList are packed onto the screen using an iterative line of code.

```
    def EnterDetails(self):
        self.IDLabel = Label(self, text="WorkerID:
"+self.CurrentWorkerID,font=MINI_FONT, fg = "blue")
        self.IDLabel.place(x=338,y=180)

        for x in self.WidgetList:
              x.pack()
```

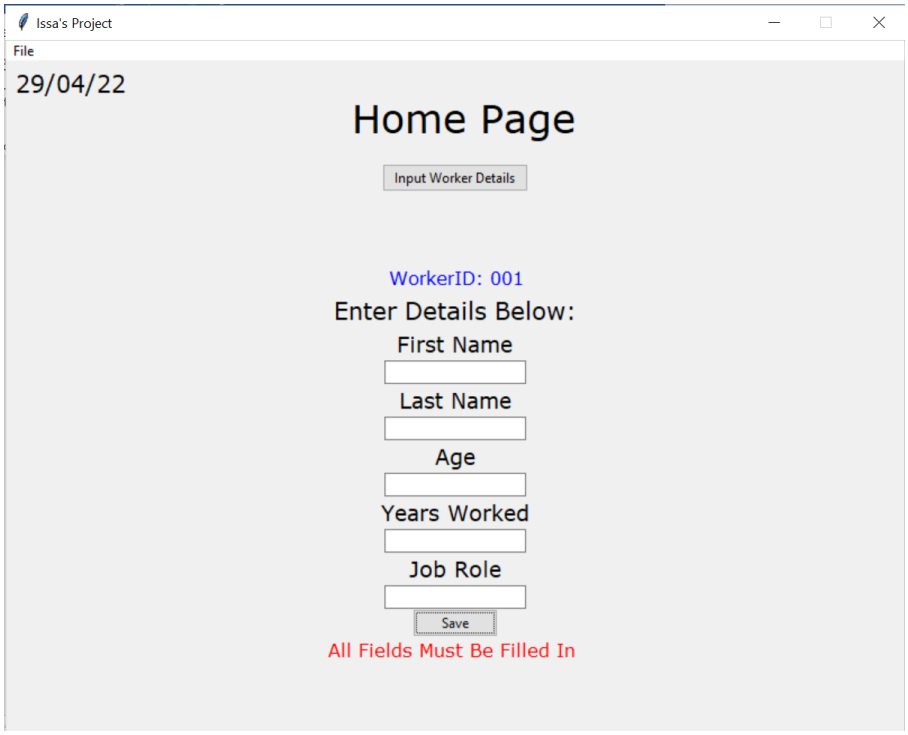


This method is called every time the Save button is pressed after all of a worker's details have been entered. The input from the entry boxes are fetched using the 'get()' function and they are assigned to the variables. Then the entries are deleted, ready for the next set of details to be entered. A new variable called Valid is created then an if statement is used to check if any of the entries are left blank. If any of them are Valid is set to False.

```
    def SaveDetails(self):
        FirstName =
self.FirstNameEntry.get()

        LastName = self.LastNameEntry.get()
        Age = self.AgeEntry.get()
        YearsWorked = self.YearsWorkedEntry.get()
        JobRole = self.JobRoleEntry.get()

        self.FirstNameEntry.delete(0,20)
        self.LastNameEntry.delete(0,20)
        self.AgeEntry.delete(0,20)
        self.YearsWorkedEntry.delete(0,20)
        self.JobRoleEntry.delete(0,20)

        Valid = True

        if FirstName == "" or LastName == "" or Age ==  "" or YearsWorked
==  "" or JobRole == "":
            Valid = False
```

If Valid is False the error label will be displayed at the bottom which states that all entries must be filled in. so the user will have to enter the details in again, making sure they fill in every field. If Valid was True, meaning that no entries are blank then the error label is removed(if it was there already), the ID Label is removed and the StoreDetails method is called and the variables holding all the user entries are passed into it.

```
        if Valid == False:
           self.ErrorLabel.pack()
        else:
             self.ErrorLabel.forget()
             self.IDLabel.place_forget()
             self.StoreDetails(FirstName,LastName,Age,YearsWorked,JobRole)
```



This method sets the current Worker ID. It selects the last Worker ID from the Details table. The SELECT statements always return a tuple so this is casted into a string and the individual characters which make up the Worker ID are taken from it and concatened together and stored as the CurrentWorkerID attribute. Then the IncrementCurrentWorkerID method is called to increase it by one because the last one was selected from the table so it must now be the next one which is one more than the last one. This is all coded using exception handling, in case the same error occurs where there is nothing to select if the table is empty. If this happens the except part assumes that the table is empty and sets the CurrentWorkerID attribute to “001” as the first worker because there are no others.

```
    def SetCurrentWorkerID(self):
        try:
            NewPath = "Project Database.db"
            connection = sqlite3.connect(NewPath)
            data = connection.cursor()
            data.execute("""SELECT WorkerID FROM Details ORDER BY WorkerID
DESC LIMIT 1""")
            self.CurrentWorkerID = str(data.fetchall())
            self.CurrentWorkerID = self.CurrentWorkerID[3] +
self.CurrentWorkerID[4] + self.CurrentWorkerID[5]
            self.IncrementCurrentWorkerID()
        except:
            self.CurrentWorkerID = "001"
```

This method increments the current WorkerID. It casts the CurrentWorkerID attribute as an integer (because it is a string) and adds 1 to it. It then casts it back into a string. The if statement checks that if the number is still a 1 digit number it will add “00” to the front of it, or if the number is a 2 digit number it will add “0” to it and if it is 3 digit it will leave it.

```
    def IncrementCurrentWorkerID(self):
        self.CurrentWorkerID = (int(self.CurrentWorkerID)+1)
        self.CurrentWorkerID = str(self.CurrentWorkerID)
        if len(self.CurrentWorkerID) == 1:
           self.CurrentWorkerID = "00" + self.CurrentWorkerID
        elif len(self.CurrentWorkerID) == 2:
             self.CurrentWorkerID = "0" + self.CurrentWorkerID
```

The final method in this class stores the details into the Details table in the database. The 4 variables are passed into the method. It connects to the database and inserts into the Details table using the variables and the CurrentWorkerID attribute. Then the IncrementCurrentWorkerID method is called to increment the CurrentWorkerID ready for the next WorkerID and the GetDetails method is called which will check if anymore details need to be entered in and repeat the cycle.

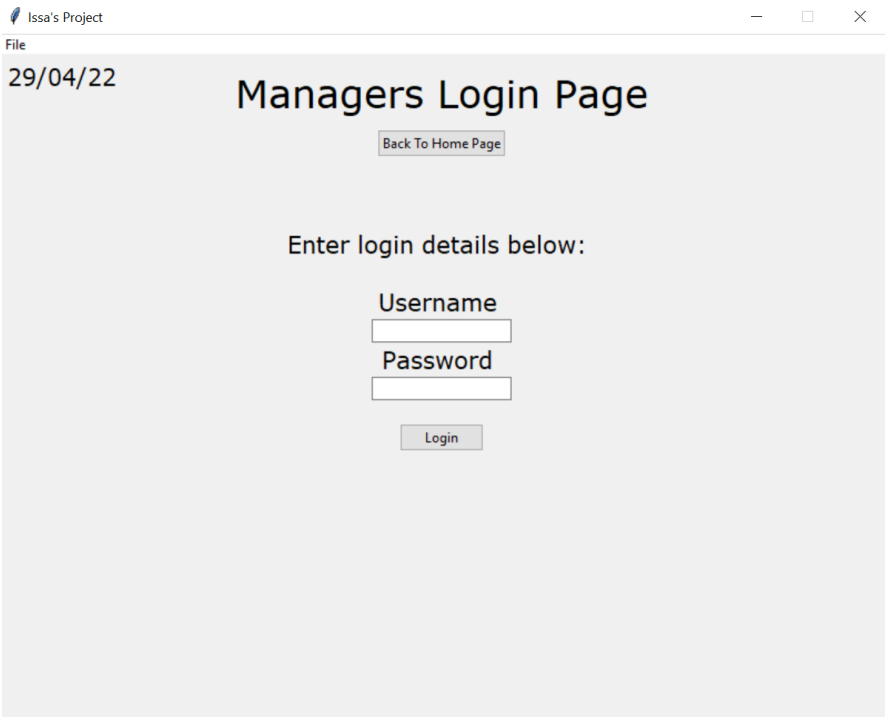
```
    def StoreDetails(self,FirstName,LastName,Age,YearsWorked,JobRole):
        NewPath = "Project Database.db"
        connection = sqlite3.connect(NewPath)
        data = connection.cursor()
        data.execute("""INSERT INTO
Details(WorkerID,FirstName,LastName,Age,YearsWorked,JobRole)
Values(?,?,?,?,?,?)"""
                ,(self.CurrentWorkerID,FirstName,LastName,Age,YearsWorked,Job
Role,))
        connection.commit()

        self.IncrementCurrentWorkerID()
        self.GetDetails()
```

This class is the page for the managers to log in.

```
class
ManagerLogin(tk.Frame):
```

The constructor method initialises the page by inheriting from the parent class. Then it calls the DefineWidgets method to create all the widgets. The title label is placed at the top of the page. The back to start button is packed which allows the user to go back to the Home Page. The other widgets are packed including the login label, which tells the user to enter their login details below, the Username and Password entries and labels which allows the user to enter in the login details and the login button which lets the user login when their username and password has been entered. There a few blank labels packed in between the widgets to create spaces between them.

```
    def __init__(self, parent, controller):
        Frame.__init__(self,parent)

        self.DefineWidgets(controller)

        self.TitleLabel.pack(pady=10,padx=10)


        self.BackToStartButton.pack()

        Label(self, text= " ").pack()
        Label(self, text= " ").pack()
        Label(self, text= " ").pack()

        self.LoginLabel.pack()

        Label(self, text= " ").pack()


        self.UsernameLabel.pack()
        self.UsernameEntry.pack()

        self.PasswordLabel.pack()
        self.PasswordEntry.pack()

        Label(self, text= " ").pack()

        self.LoginButton.pack()

        Label(self, text = "").pack()
```



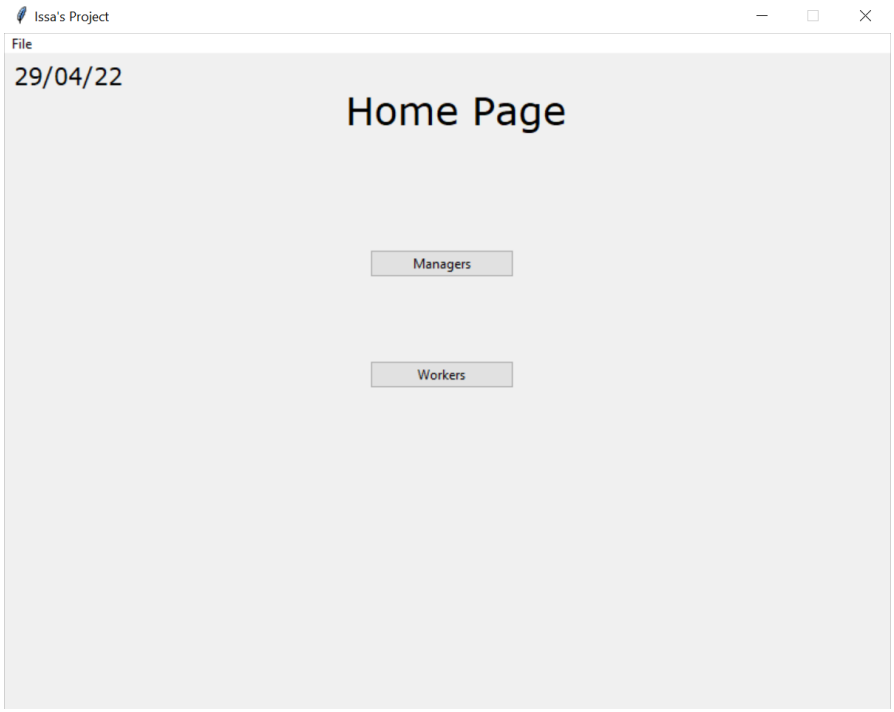
This method creates all the widgets such as the labels, buttons and entry boxes.

```
    def DefineWidgets(self,controller):
        self.TitleLabel = ttk.Label(self, text="Managers Login Page",
font=TITLE_FONT)

        self.BackToStartButton = ttk.Button(self, text="Back To Home Page",
command =lambda: self.BackToStart(controller))

        self.LoginLabel = Label(self, text="Enter login details below: ",
font=MEDIUM_FONT)

        username = ""
        password = ""

        self.UsernameLabel = Label(self, text="Username ", font=MEDIUM_FONT)
        self.UsernameEntry = ttk.Entry(self, textvariable = username)

        self.PasswordLabel = Label(self, text="Password ", font=MEDIUM_FONT)
        self.PasswordEntry = ttk.Entry(self, textvariable = password)

        self.ErrorLabel = Label(self, text="Incorrect Login Details
",font=MINI_FONT, fg = "red")
```

```
        self.LoginButton = ttk.Button(self, text = "Login",command =lambda:
self.Login(controller))
```

This method allows the user to go back to the Home Page when the button is pressed. It removes the error label off the screen (if it was there) and then uses the show_frame method from the window super class to Show the Start Page class, which is the Home Page.

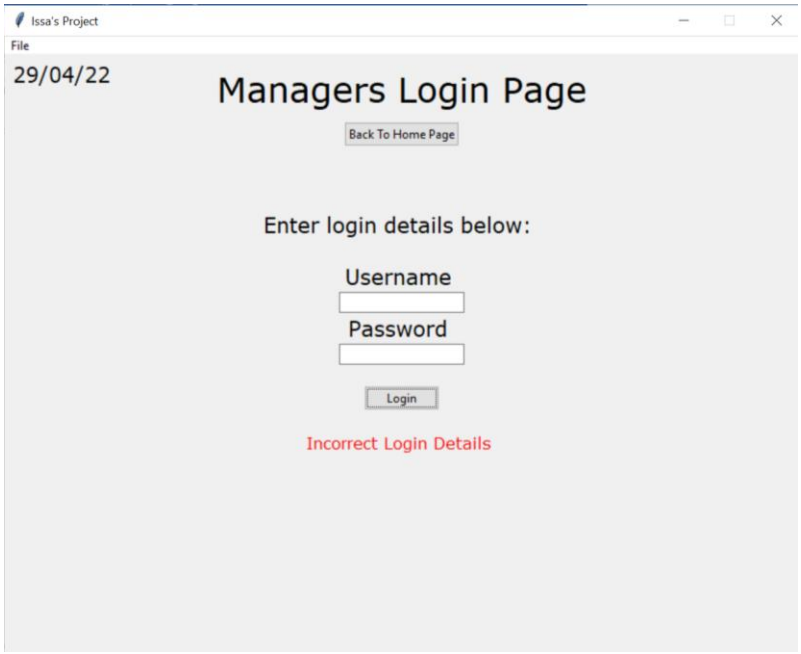
```
    def BackToStart(self,controller):
        self.ErrorLabel.forget()
        controller.show_frame(StartPage)
```

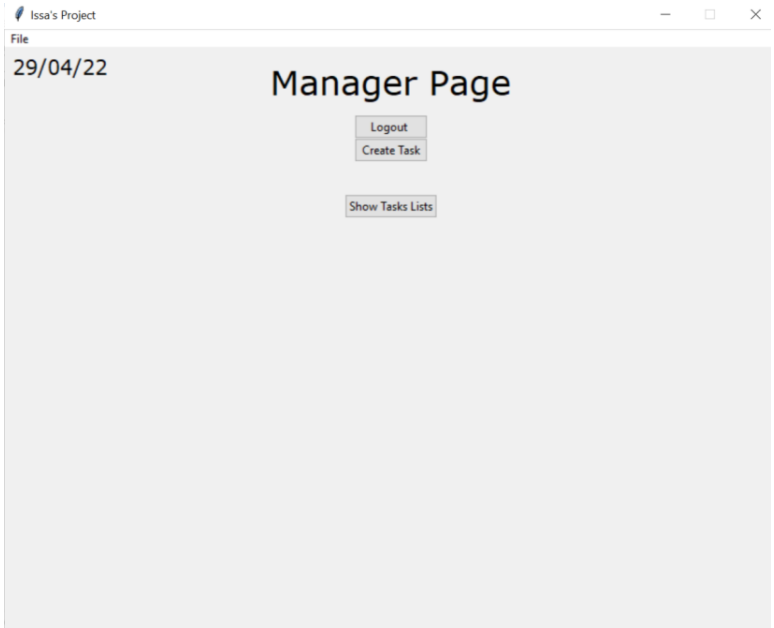


This method allows the user to login. It assigns the entries to the variables. It then calls the CheckLoginDetails method and assigns the return value to the variable Valid. The if statement then checks if Valid is True or False. If its True it will remove the error label if it was there and use the show_frame method to show the Manager Page. It also creates a global variable called CurrentManagerID. From username the 9th character is taken(which is the number part of the username) and adds “00” to the start of it to turn it into the Manager ID for the Manager with that username. This is a global variable because it will need to be used in lots of different parts of the code later on. If Valid was False the error label will appear which tells the user that the login details that they entered are incorrect. Afterwards the entry boxes are cleared.

```
    def Login(self,controller):
        username =  self.UsernameEntry.get()
        password = self.PasswordEntry.get()
```

```
        Valid =  True
self.CheckLoginDetails(username,password)
        if Valid == True:
            self.ErrorLabel.pack_forget()
            controller.show_frame(ManagerPage)
            global CurrentManagerID
            CurrentManagerID = "00" + str(username[8])
        else:
              self.ErrorLabel.pack()
        self.UsernameEntry.delete(0,10)
        self.PasswordEntry.delete(0,10)
```





This method checks the login details that the user entered and determines if they are valid or not. The variable Valid is created and set to False. The username and password are selected from the Managers table where the username is the one that was entered. The username and password that are selected are assigned into the variables CheckUsername and Check Password. If CheckUsername matches the username that the user entered and CheckPassword matches the password that the user entered then Valid will be set to True, otherwise it will be left as False. Exception handling is used here in case the username entered is not correct so it will cause an error when trying to select from the table something which is not there. The except part will assume the username is wrong so Valid remains False. Finally Valid is returned.

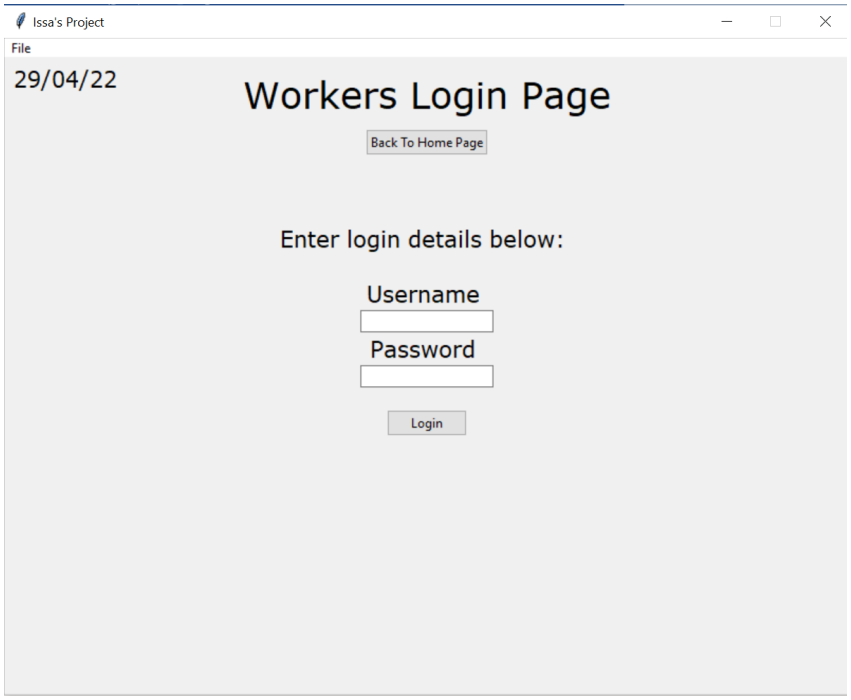
```
    def CheckLoginDetails(self,username,password):
        Valid =  False
        try:
             NewPath = "Project Database.db"
             connection = sqlite3.connect(NewPath)
             data = connection.cursor()
             data.execute("""SELECT Username, Password FROM Managers WHERE
Username LIKE ? """,(username,))
             Login = data.fetchall()
             CheckUsername = Login[0][0]
             CheckPassword = Login[0][1]
             if CheckUsername == username and CheckPassword == password:
                Valid = True
        except:
             Valid =  False
        return Valid
```

This class is the page for the workers to log in.

```
class WorkerLogin(tk.Frame):
```

The constructor method initialises the page by inheriting from the parent class. Then it calls the DefineWidgets method to create all the widgets. The title label is placed at the top of the page. The back to start button is packed which allows the user to go back to the Home Page. The other widgets are packed including the login label, which tells the user to enter their login details below, the Username and Password entries and labels which allows the user to enter in the login details and the login button which lets the user login when their username and password has been entered. There a few blank labels packed in between the widgets to create spaces between them.

```
    def __init__(self, parent, controller):
        Frame.__init__(self,parent)

        self.DefineWidgets(controller)

        self.TitleLabel.pack(pady=10,padx=10)


        self.BackToStartButton.pack()

        Label(self, text= " ").pack()
        Label(self, text= " ").pack()
        Label(self, text= " ").pack()

        self.LoginLabel.pack()

        Label(self, text= " ").pack()


        self.UsernameLabel.pack()
        self.UsernameEntry.pack()

        self.PasswordLabel.pack()
        self.PasswordEntry.pack()

        Label(self, text= " ").pack()

        self.LoginButton.pack()

        Label(self, text = "").pack()
```

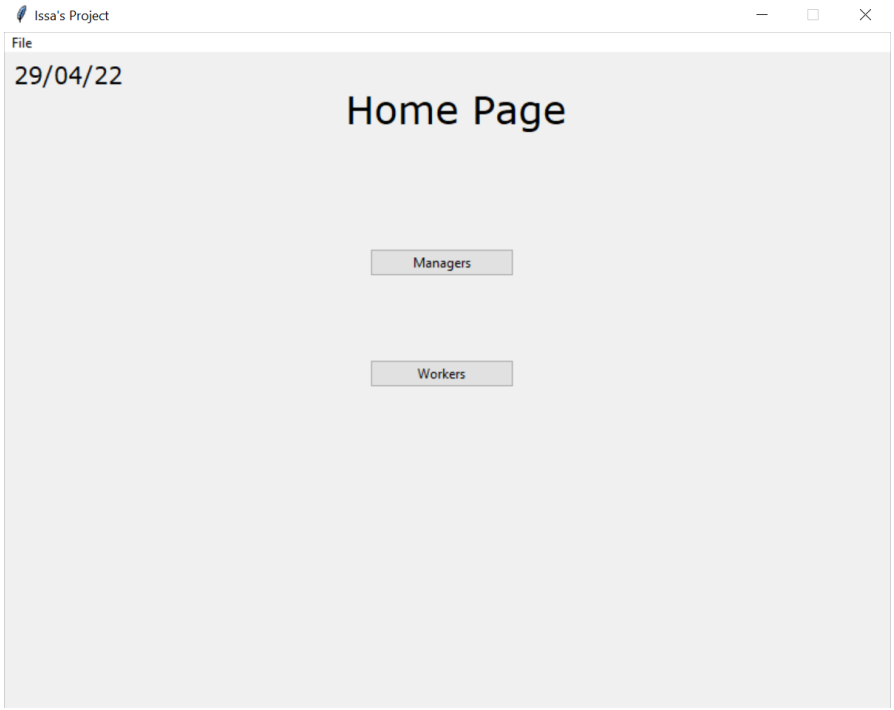


This method creates all the widgets such as the labels, buttons and entry boxes.

```
    def DefineWidgets(self,controller):
        self.TitleLabel = ttk.Label(self, text="Workers Login Page",
font=TITLE_FONT)

        self.BackToStartButton = ttk.Button(self, text="Back To Home Page",
command =lambda: self.BackToStart(controller))

        self.LoginLabel = Label(self, text="Enter login details below: ",
font=MEDIUM_FONT)

        username = ""
        password = ""

        self.UsernameLabel = Label(self, text="Username ", font=MEDIUM_FONT)
        self.UsernameEntry = ttk.Entry(self, textvariable = username)

        self.PasswordLabel = Label(self, text="Password ", font=MEDIUM_FONT)
        self.PasswordEntry = ttk.Entry(self, textvariable = password)

        self.ErrorLabel = Label(self, text="Incorrect Login Details
",font=MINI_FONT, fg = "red")

        self.LoginButton = ttk.Button(self, text = "Login",command =lambda:
self.Login(controller))
```

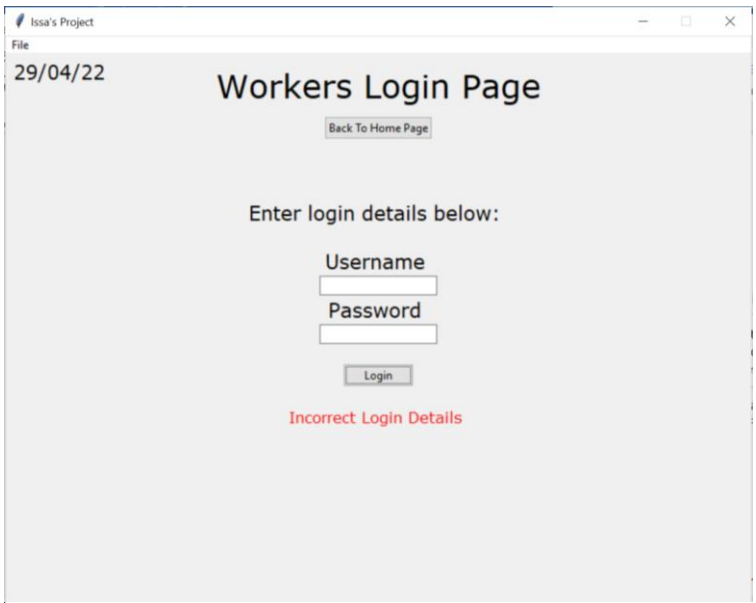
This method allows the user to go back to the Home Page when the button is pressed. It removes the error label off the screen (if it was there) and then uses the show_frame method from the window super class to Show the Start Page class, which is the Home Page.

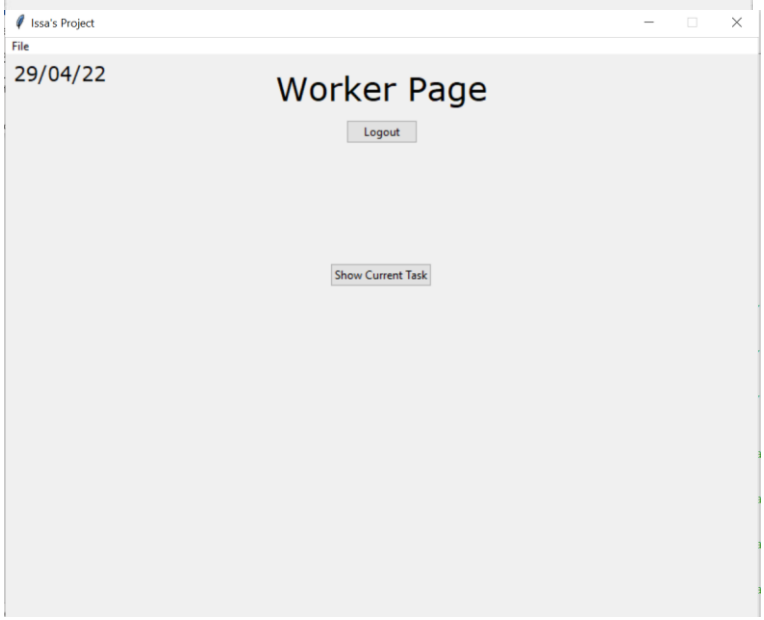
```
    def BackToStart(self,controller):
        self.ErrorLabel.forget()
        controller.show_frame(StartPage)
```



This method allows the user to login. It assigns the entries to the variables. It then calls the CheckLoginDetails method and assigns the return value to the variable Valid. The if statement then checks if Valid is True or False. If its True it will remove the error label if it was there and use the show_frame method to show the Worker Page. It also creates a global variable called CurrentWorkerID. From username the 8th character is taken(which is the number part of the username) and adds “00” to the start of it to turn it into the Worker ID for the Worker with that username. This is a global variable because it will need to be used in lots of different parts of the code later on. If Valid was False the error label will appear which tells the user that the login details that they entered are incorrect. Afterwards the entry boxes are cleared.

```
    def Login(self,controller):
        username =  self.UsernameEntry.get()
        password = self.PasswordEntry.get()
        Valid =
self.CheckLoginDetails(username,password)
        if Valid == True:
           self.ErrorLabel.pack_forget()
           controller.show_frame(WorkerPage)
           global CurrentWorkerID
           CurrentWorkerID = "00" + str(username[7])
        else:
             self.ErrorLabel.pack()
        self.UsernameEntry.delete(0,10)
        self.PasswordEntry.delete(0,10)
```





This method checks the login details that the user entered and determines if they are valid or not. The variable Valid is created and set to False. The username and password are selected from the Workers table where the username is the one that was entered. The username and password that are selected are assigned into the variables CheckUsername and Check Password. If CheckUsername matches the username that the user entered and CheckPassword matches the password that the user entered then Valid will be set to True, otherwise it will be left as False. Exception handling is used here in case the username entered is not correct so it will cause an error when trying to select from the table something which is not there. The except part will assume the username is wrong so Valid remains False. Finally Valid is returned.

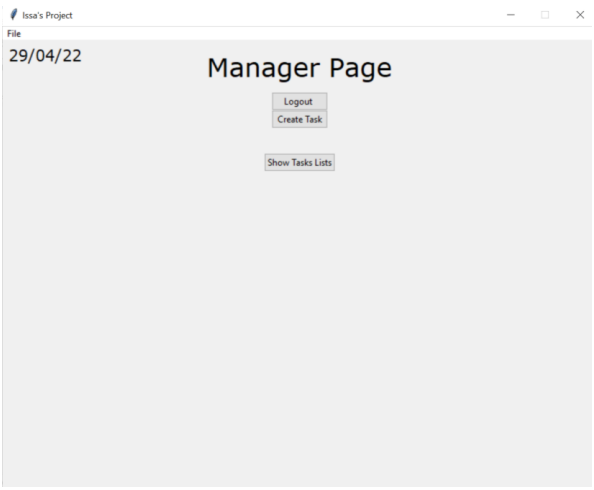
```
    def CheckLoginDetails(self,username,password):
        Valid =  False
        try:
             NewPath = "Project Database.db"
             connection = sqlite3.connect(NewPath)
             data = connection.cursor()
             data.execute("""SELECT Username, Password FROM Workers WHERE
Username LIKE ? """,(username,))
             Login = data.fetchall()
             CheckUsername = Login[0][0]
             CheckPassword = Login[0][1]
             if CheckUsername == username and CheckPassword == password:
                Valid = True
        except:
             Valid =  False
        return Valid
```

This class is the main page for all managers. It is the longest and most complex class in the whole code. Like all other classes it inherits from the window parent class.

```
class ManagerPage(tk.Frame):
```

The constructor method initialises the frame for the page. Then it calls the DefineWidgets method to create all the widgets for the page. The CreateQueue method is called to create the tasks queue, which will initially be empty. The CreateTasksTreeview method is called to define the Treeview widget from tkinter. The SetCurrentTaskID method is called to set the current task ID by looking at the last task ID in the Tasks table from the database. The title label is packed to the top of the screen. The back to login button is also packed. The create task button and show task list button are both placed onto the page.

```
    def __init__(self, parent, controller):
        Frame.__init__(self,parent)
        self.DefineWidgets(controller)
        self.CreateQueue()
        self.CreateTasksTreeview()
        self.SetCurrentTaskID()
        self.TitleLabel.pack(pady=10,padx=10)

        self.BackToLoginButton.pack()

        self.CreateTaskButton.place(x=362,y=92)

        self.ShowTasksListButton.place(x=352,y=150)
```



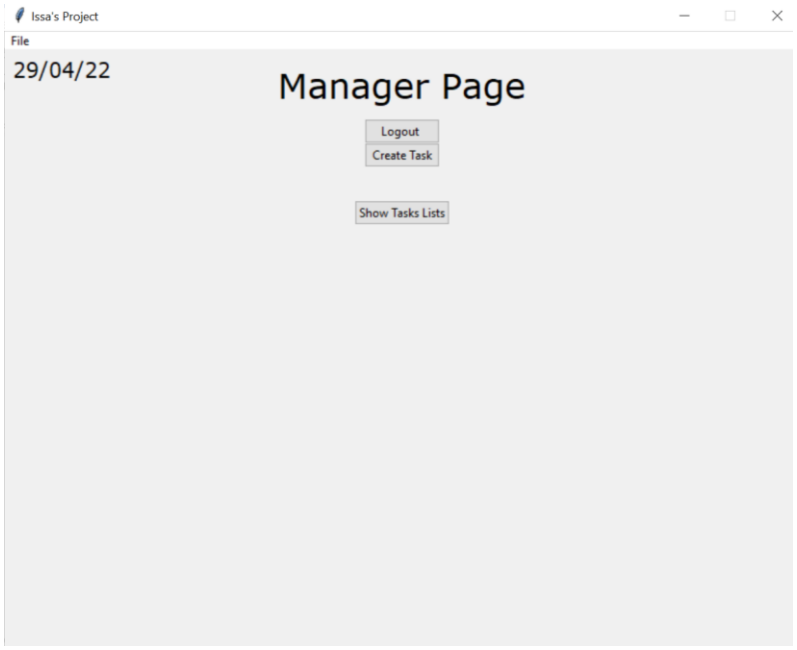
This method simply defines all the widgets to be used in this class. This includes buttons, labels and entry boxes.

```
    def DefineWidgets(self,controller):
        self.TitleLabel = Label(self, text="Manager Page", font=TITLE_FONT)

        self.BackToLoginButton = ttk.Button(self, text="Logout ", command
=lambda: self.LogOut(controller))

        self.CreateTaskButton = ttk.Button(self, text="Create Task", command
= lambda: self.CreateTask(controller))

        self.SuccessLabel = Label(self,text="Task Assigned! ",font =
MEDIUM_FONT,fg="green")

        TaskType =
""
        Object = ""
        Quantity = ""
        Info = ""

        self.EnterTaskDetailsLabel = Label(self, text="Enter Task Details
Below:", font=MEDIUM_FONT)

        self.TaskTypeLabel = Label(self, text="Task Type ", font=SMALL_FONT)
        self.TaskTypeEntry = ttk.Entry(self, textvariable = TaskType)

        self.ObjectLabel = Label(self, text="Object (Optional) ",
font=SMALL_FONT)
        self.ObjectEntry = ttk.Entry(self, textvariable = Object)

        self.QuantityLabel = Label(self, text="Quantity (Optional) ",
font=SMALL_FONT)
        self.QuantityEntry = ttk.Entry(self, textvariable = Quantity)

        self.InfoLabel = Label(self, text="Extra Information (Optional) ",
font=SMALL_FONT)
        self.InfoEntry = ttk.Entry(self, textvariable = Info)

        self.AutomaticButton = ttk.Button(self, text = "Assign Task", command
=lambda: self.ManageTask("A"))

        self.ManualButton = ttk.Button(self, text = "Assign Task Manually",
command =lambda: self.ManageTask("M"))

        self.ErrorLabel1 = Label(self, text="Task Type cannot be left blank",
font = MINI_FONT,fg="red")
```

```
        self.MyTree = ttk.Treeview(self)

        self.ShowTasksListButton = ttk.Button(self, text = "Show Tasks
Lists", command = lambda: self.ShowTasksList())

        self.HideTasksListButton = ttk.Button(self,text="Hide Tasks List",
command = lambda: self.HideTasksList())
```

This method resets the page and leaves it exactly how it looked when it was first created. It defines a list of widgets and iterates through the list, removing each one. It also removes the HideTasksListButton and SuccessLabel. The CreateTaskButton and ShowTasksListButton are placed onto the page. There is also the use of exception handling to try to remove some widgets, however this method might be called before these widgets have been defined which would cause an error so to prevent this exception handling was used. The except part simply returns the None value which does nothing.

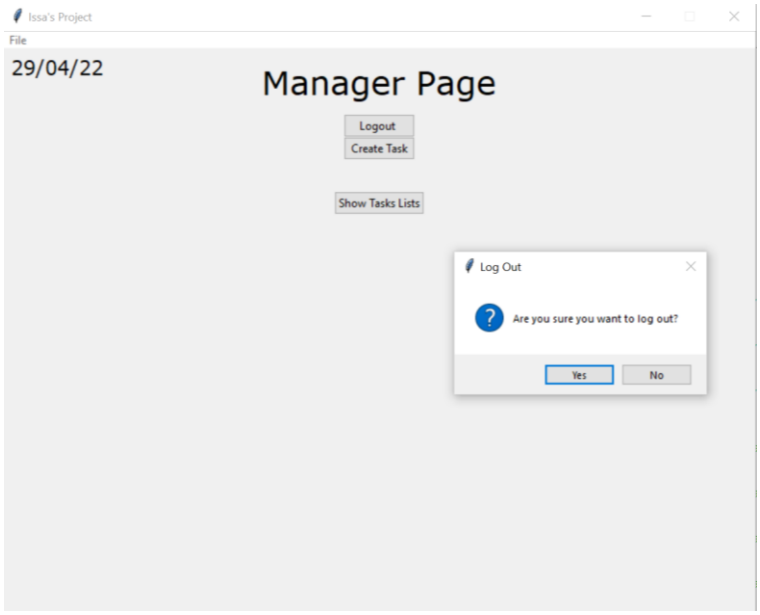
```
    def ResetPage(self):
        WidgetList =
[self.EnterTaskDetailsLabel,self.TaskTypeLabel,self.ObjectLabel,self.QuantityL
abel,self.InfoLabel,
                 self.AutomaticButton,self.ManualButton,
                 self.TaskTypeEntry,self.ObjectEntry,self.QuantityEntry,self.
InfoEntry,
                 self.SuccessLabel,self.ErrorLabel1,self.MyTree]
        for x in WidgetList:
            x.forget()

        self.HideTasksListButton.place_forget()
        self.SuccessLabel.place_forget()
        self.CreateTaskButton.place(x=362,y=92)
        self.ShowTasksListButton.place(x=352,y=150)
        try:
            self.IDLabel.forget()
            self.ConfirmButton.forget()
            self.dropmenu.forget()
            self.SelectWorkerIDLabel.forget()
        except:
            return None
```

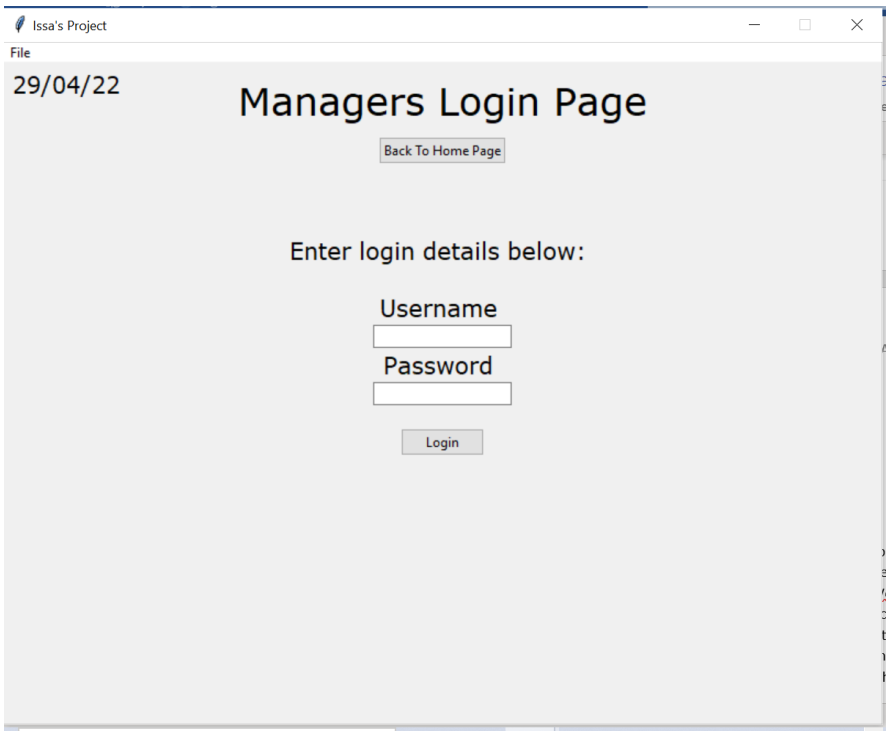


The ResetPage method leaves the page looking exactly like this.

This method allows the user to logout and be returned to the login page. It starts off by sending a pop up message box onto the screen, asking the user if they are sure they want to log out. If they press yes the show_frame method is used to show the Manager Login page. The ResetPage method is called so that the next time the user logs back into the Manager Page, it looks how it's supposed to, to start off with.

```
    def LogOut(self,controller):
        Confirm = messagebox.askyesno(title="Log Out",message="Are you sure
you want to log out?")
        if Confirm > 0:
            controller.show_frame(ManagerLogin)
            self.ResetPage()
```



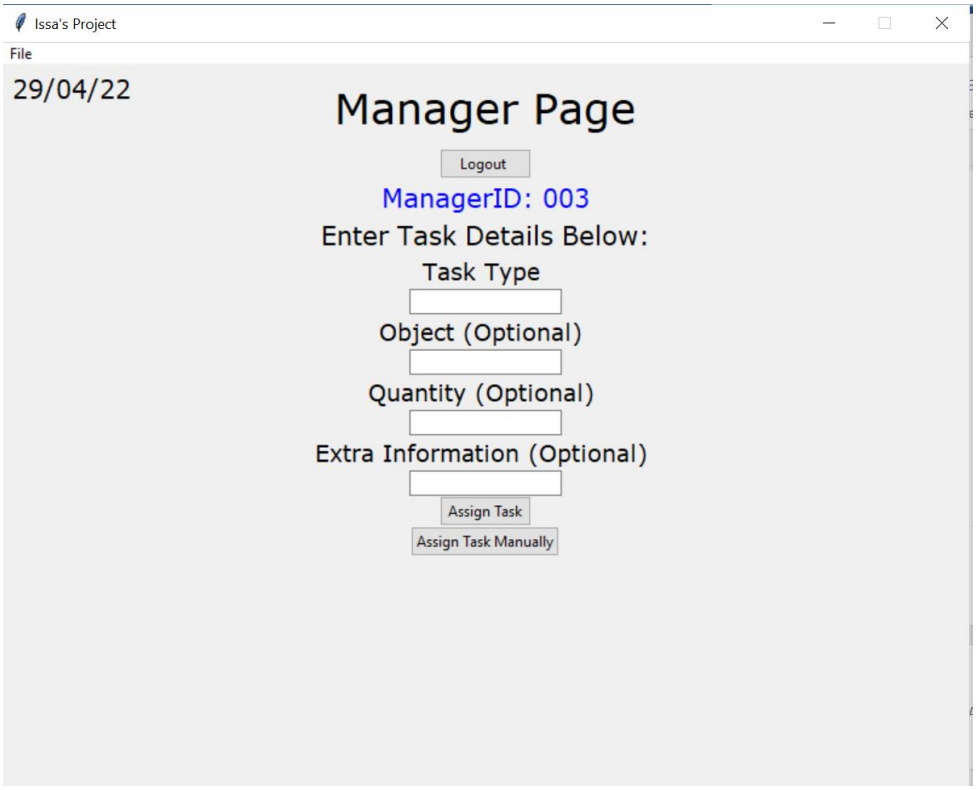


This method sets the current Task ID. It selects the last Task ID from the Details table. The SELECT statements always return a tuple so this is casted into a string and the individual characters which make up the Task ID are taken from it and concatened together and stored as the CurrentTaskID attribute. Then the IncrementCurrentTaskID method is called to increase it by one because the last one was selected from the table so it must now be the next one which is one more than the last one. This is all coded using exception handling, in case the error occurs where there is nothing to select if the table is empty. If this happens the except part assumes that the table is empty and sets the CurrentTaskID attribute to “001” as the first worker because there are no others.

```
    def SetCurrentTaskID(self):
        try:
            NewPath = "Project Database.db"
            connection = sqlite3.connect(NewPath)
            data = connection.cursor()
            data.execute("""SELECT TaskID FROM Tasks ORDER BY TaskID DESC
LIMIT 1""")
            self.CurrentTaskID = str(data.fetchall())
            self.CurrentTaskID = self.CurrentTaskID[3] +
self.CurrentTaskID[4] + self.CurrentTaskID[5]
            self.IncrementCurrentTaskID()
        except:
               self.CurrentTaskID = "001"
```

This method is very important as it allows the user to create a task and is called when the user presses the create task button. It creates a label which tells the user what their manager ID is by use of the CurrentManagerID global variable. This label is packed onto the screen. Then a few buttons (including the create task button), along with the treeview and the success label are removed from the page to ensure it is clear. The variables are defined which will hold the task details that are to be entered by the user. A label to is packed to tell the user to enter the task details in. This is followed by the TaskType label and entry box, the Object label and entry box, the Quantity label and entry box and the Info label and entry box being packed all in that order. Finally the automatic and manual assignment buttons are packed. The user needs to enter in all the details and press one of the buttons to assign the task.

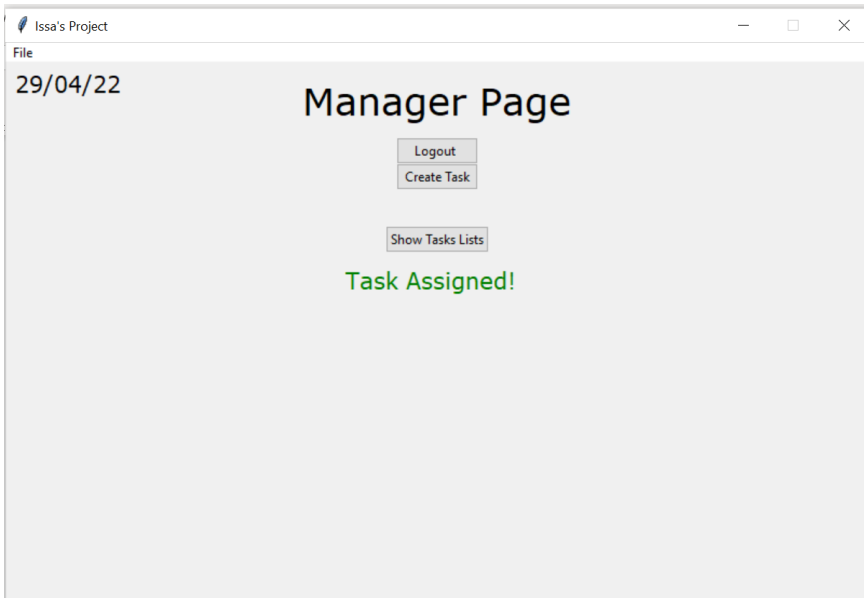
```
    def CreateTask(self,controller):

        self.IDLabel = Label(self, text="ManagerID: "+CurrentManagerID,
font=MEDIUM_FONT,fg="blue")
        self.IDLabel.pack()

        self.CreateTaskButton.place_forget()

        self.ShowTasksListButton.place_forget()
        self.HideTasksListButton.place_forget()
        self.MyTree.forget()


        self.SuccessLabel.place_forget()

        TaskType =
""
        Object = ""
        Quantity = ""
        Info = ""

        self.EnterTaskDetailsLabel.pack()

        self.TaskTypeLabel.pack()
        self.TaskTypeEntry.pack()

        self.ObjectLabel.pack()
        self.ObjectEntry.pack()

        self.QuantityLabel.pack()
        self.QuantityEntry.pack()

        self.InfoLabel.pack()
        self.InfoEntry.pack()


        self.AutomaticButton.pack()

        self.ManualButton.pack()
```



This method takes the inputs and decides what to do with them. It firstly removes the error label off the page in case it was there before. Then it creates new attributes and assigns the entries, using the 'get()' function for each entry box. The entry boxes are then cleared. The if statement checks whether the TaskType entry was left blank or not. If it was the error label is packed onto the screen, telling the user that "TaskType cannot be left blank" and nothing else happens after this. However if it was not blank then the error label is removed in case it was there before and the ResetPage method I called to reset the page. Then the if statement checks the variable called 'AutomaticOrManual' which is passed in as an argument into the method. If the automatic button was pressed to assign the task by the user then "A" would be passed in. If the manual button was pressed then "M" would be passed in. The if statement shows that if "A" was passed in, the 'Algorithm' method is called and the return value is assigned to the variable called 'WorkerID'. The AssignTask method is called and WorkerID is passed in as an argument. Then the success label is packed which informs the user that their task has been assigned. If "M" was passed in, the two buttons are removed from the page and the ManualChoice method is called.

```
    def ManageTask(self,AutomaticOrManual):
        self.ErrorLabel1.forget()
        self.TaskType = self.TaskTypeEntry.get()        #fetch the data
inputted into the entry boxes
        self.Object = self.ObjectEntry.get()
        self.Quantity = self.QuantityEntry.get()
        self.Info = self.InfoEntry.get()
```

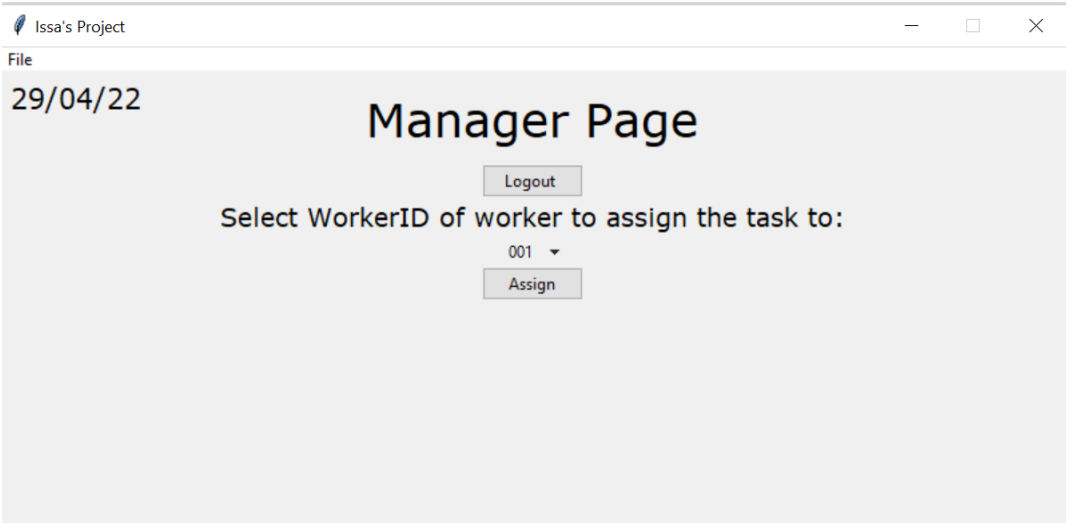
```
        self.TaskTypeEntry.delete(0,20)             #clear the entry boxes
        self.ObjectEntry.delete(0,20)
        self.QuantityEntry.delete(0,20)
        self.InfoEntry.delete(0,100)

        if self.TaskType == "" or self.TaskType == " " or self.TaskType ==
"  ":
            self.ErrorLabel1.pack()
        else:
            self.ErrorLabel1.forget()

            self.ResetPage()

            if AutomaticOrManual == "A":
                WorkerID = self.Algorithm()
                self.AssignTask(WorkerID)
                self.SuccessLabel.place(x=312,y=185)
            else:
                self.CreateTaskButton.place_forget()
                self.ShowTasksListButton.place_forget()
                self.ManualChoice()
```



This method is to allow the user to manually select a worker to assign the task to. Firstly a label is created which tells the user to select the worker ID of the worker to assign the task to. The label is packed onto the screen. A list of options is created containing all possible worker IDs. A variable called ID is defined as a StringVar data type and is set to the first item in the options list which is “001”. A tkinter drop down menu is created with the options list passed in as the possible choices and ID as the initial choice to be shown on the screen. The drop down menu is packed onto the screen.

A new button is created for the user to confirm their choice. The button's command is to call the ConfirmChoice method and passes in whatever ID was from what the user chose for it to be. This button is packed onto the screen.

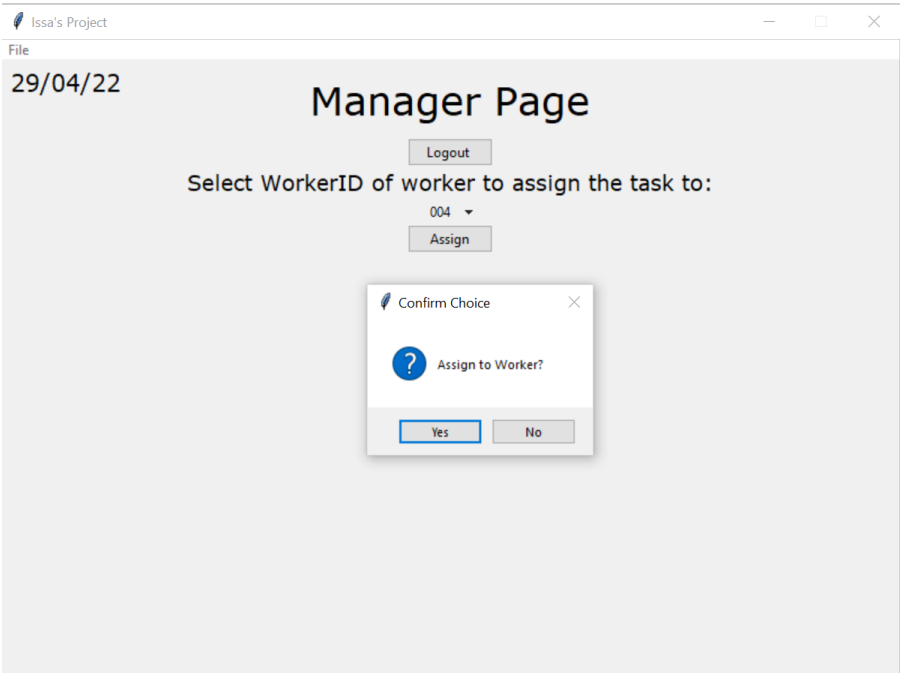
```
    def ManualChoice(self):
        self.SelectWorkerIDLabel = Label(self,text = "Select WorkerID of
worker to assign the task to:",font=SMALL_FONT)
        self.SelectWorkerIDLabel.pack()

        options = ["001", "002", "003", "004", "005", "006", "007", "008"]

        ID = StringVar()
        ID.set(options[0])

        self.dropmenu = ttk.OptionMenu(self,ID,*options)
        self.dropmenu.pack()

        self.ConfirmButton = ttk.Button(self,text="Assign",command = lambda:
self.ConfirmChoice(ID.get()))
        self.ConfirmButton.pack()
```

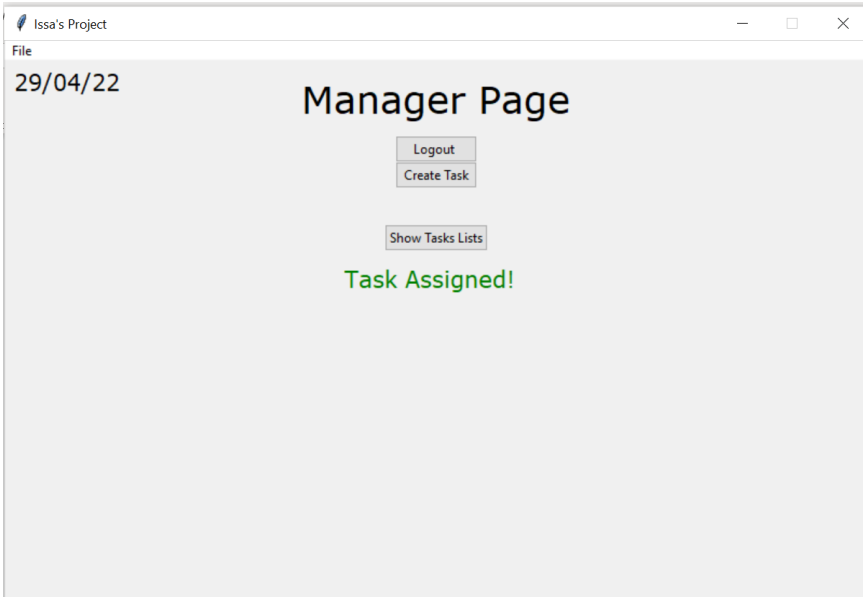


This method gets the user to confirm their worker ID choice. It takes in WorkerID as an argument which was passed in from the ManualChoice method as the user's ID choice. A message box is shown to the user asking them to confirm that they want to assign the task to the chosen worker. If the user presses yes, the SelectWorkerID label, the drop down menu and the confirm button are all removed off the page. The ResetPage method is also called to set the page back to exactly how it was at the start. The success label is placed onto the page to tell the user that their task has been assigned. The AssignTask method is called and WorkerID is passed in.

```
    def ConfirmChoice(self,WorkerID):
        Confirm = messagebox.askyesno(title="Confirm Choice",message=("Assign
to Worker?"))
        if Confirm > 0:
```

```
            self.SelectWorkerIDLabel.forget()
            self.dropmenu.forget()
            self.ConfirmButton.forget()
            self.ResetPage()
            self.SuccessLabel.place(x=312,y=185)
            self.AssignTask(WorkerID)
```





This method assigns the task to the worker. WorkerID is passed in as an argument, which was either created automatically by the algorithm or manually by the user. A new attribute WorkerBusy is created and the CheckIfBusy method is called with WorkerID passed in and the return value is stored in the attribute. This method checks if the worker with that particular worker ID is currently busy with another task. If WorkerBusy is true, the current task ID will be added to the queue. Finally the StoreTask method is called which will store all the details of the task into the Tasks table within the database. WorkerId is passed into this method.

```
    def AssignTask(self,WorkerID):
        self.WorkerBusy = self.CheckIfBusy(WorkerID)
        if self.WorkerBusy == True:
           self.AddToQueue(self.CurrentTaskID)
        self.StoreTask(WorkerID)
```

This method increments the current Task ID. It casts the CurrentTaskID attribute as an integer (because it is a string) and adds 1 to it. It then casts it back into a string. The if statement checks that if the number is still a 1 digit number it will add “00” to the front of it, or if the number is a 2 digit number it will add “0” to it and if it is 3 digits long it will leave it.

```
    def IncrementCurrentTaskID(self):
        self.CurrentTaskID = (int(self.CurrentTaskID)+1)
        self.CurrentTaskID = str(self.CurrentTaskID)
        if len(self.CurrentTaskID) == 1:
           self.CurrentTaskID = "00" + self.CurrentTaskID
        elif len(self.CurrentTaskID) == 2:
             self.CurrentTaskID = "0" + self.CurrentTaskID
```

This method checks if a worker is currently busy with another task. It takes in WorkerID as an argument which is the WorkerID of the worker that needs to be checked. The SQL statement selects the last TaskID from the Tasks table where the Worker ID for the task is WorkerID(which has been passed in as an argument). If this TaskID is a blank string then the WorkerBusy attribute is set to False, otherwise it is set to True. Exception handling is used here in case an error occurs due to there being nothing to select from the table. If this error happens the except part assumes the worker to not be busy and sets WorkerBusy to False. The WorkerBusy attribute is returned.

```
    def CheckIfBusy(self,WorkerID):
        try:
            NewPath = "Project Database.db"
            connection = sqlite3.connect(NewPath)
            data = connection.cursor()
            data.execute("""SELECT TaskID FROM Tasks WHERE WorkerID LIKE ?
AND DateCompleted IS NULL ORDER BY TaskID DESC LIMIT 1""",(WorkerID,))
            ID = data.fetchall()
            if ID[0][0] == "":
               self.WorkerBusy = False
            else:
```

```
                self.WorkerBusy = True
        except:
                self.WorkerBusy = False
        return self.WorkerBusy
```

This method is vital to the whole program and it is the central algorithm which decides the best worker for each task. It connects to the database and selects all YearsWorked fields from the Details table and assigns these to the Years variable. Then an empty list, YearsWorkedList is created. The for loop iterates through the tuple of the selected items from the database and casts each item as a string, uses an if statement to check the number of digits of each Years number and then selects the appropriate characters from the string, converts this into an integer and appends it into the YearsWorkedList. This eventually turns YearsWorkedList into a list of all the years worked values, all as integers. A new list called SortedYearsWorkedList is created and assigned the return value of the MergeSort method. YearsWorkedList is passed into this method and it returns the list after merge sorting it.

Two new variables are created: most which is assigned the length of the SortedYearsWorkedList minus 1 because index positions in python always start from 0 so the index position of the last item in the list will be one less and found which is assigned the value False.

```
    def Algorithm(self):
        NewPath = "Project Database.db"
        connection = sqlite3.connect(NewPath)
        data = connection.cursor()
        data.execute("""SELECT YearsWorked FROM Details""")
        Years = data.fetchall()
        YearsWorkedList = []
        for i in range(0,8):
            Years1 = str(Years[i])
            if len(Years1) == 4:
               Years1 = Years1[1]
            elif len(Years1) == 5:
                 Years1 = Years1[1] + Years1[2]
            Years1 = int(Years1)
            YearsWorkedList.append(Years1)

        SortedYearsWorkedList = self.MergeSort(YearsWorkedList)

        most = len(SortedYearsWorkedList) - 1
        found = False
        count = 0
```

The while loop condition is that while found is False WorkerID of the worker that has the YearsWorked which is the item in the SortedYearsWorkedList with the index position of most. This effectively selects the WorkerID of the worker with the largest YearsWorked from the Details table. The tuple is casted as a string and then the appropriate characters are taken and concatenated and stored as the variable WorkerID. Then the if statement uses the CheckIfBusy method to see if the worker which has the WorkerID as the variable WorkerID is busy. If they aren't busy, found is set to True to break the while loop and stop it looping again. If the worker is busy most is decreased by 1 and found is left as False so the while loop iterates again, and most is now the second last item in the SortedYearsWorkedList so the second largest YearsWorked number. The while loop repeats until a worker that isn't busy is found. WorkerID is returned at the end of the method.

```
        while found == False:
              data.execute("""SELECT WorkerID FROM Details WHERE YearsWorked
LIKE ?""",(SortedYearsWorkedList[most],))
              ID = str(data.fetchall())
              WorkerID = ID[3] + ID[4] + ID[5]
              busy = self.CheckIfBusy(WorkerID)
              if busy == False:
                 found = True
              else:
                   most = most - 1
        return WorkerID
```

This method stores everything about the current task into the Tasks table in the database. It stores these using the TaskType,Object,Quantity,Info attributes along with the CurrentTaskID attribute, the WorkerID variable which is passed in as an argument to this method, the global variable CurrentManagerID and the str_date_today constant. It also stores these all into the tasks treeview, and stores the string value "None" into the DateCompleted column as the task hasn't been completed yet. Finally the IncrementCurrentTaskID method is called so the CurrentTaskID attribute can be incremented ready for the next task to be assigned.

```
    def StoreTask(self,WorkerID):
        NewPath = "Project Database.db"
        connection = sqlite3.connect(NewPath)
        data = connection.cursor()
        data.execute("""INSERT INTO
Tasks(TaskID,TaskType,Object,Quantity,Info,DateAssigned,WorkerID,ManagerID)
  Values(?,?,?,?,?,?,?,?)
  """,(self.CurrentTaskID,self.TaskType,self.Object,self.Quantity,self.Info,st
r_date_today,WorkerID,CurrentManagerID,))
        connection.commit()

        self.MyTree.insert(parent='',index='end',text="",values=(self.Current
TaskID,self.TaskType,self.Object,self.Quantity,self.Info,
```

```
                                                                    str_date_today
,"None",WorkerID,CurrentManagerID))
            self.IncrementCurrentTaskID()
```

This method creates a tkinter treeview for the managers to view the tasks table from their page. MyTree is created as a Treeview widget from ttk. From the database the wildcard character '*' is used to select everything from the Tasks table. This gets assigned to the variable TasksList. Then the treeview's columns are all defined. The columns are the same as all the field names from the Tasks table because the treeview is to be an exact copy of the table. Then the anchors and widths are set for each column. The anchor is where the text is positioned in the column so the numerical values had "CENTER" as the anchor, whereas everything else had "W" which means West.

```
    def CreateTasksTreeview(self):
        self.MyTree = ttk.Treeview(self)
        NewPath = "Project Database.db"
        connection = sqlite3.connect(NewPath)
        data = connection.cursor()
        data.execute("""SELECT * FROM Tasks""")
        TasksList = data.fetchall()


        self.MyTree['columns'] =
("TaskID","TaskType","Object","Quantity","Info","DateAssigned","DateCompleted"
,"WorkerID","ManagerID")  #define columns

        self.MyTree.column("#0", width=0,minwidth=0)
```

This is the extra column that always has to be defined with treeviews. The width of it was set to 0 so it doesn't appear.

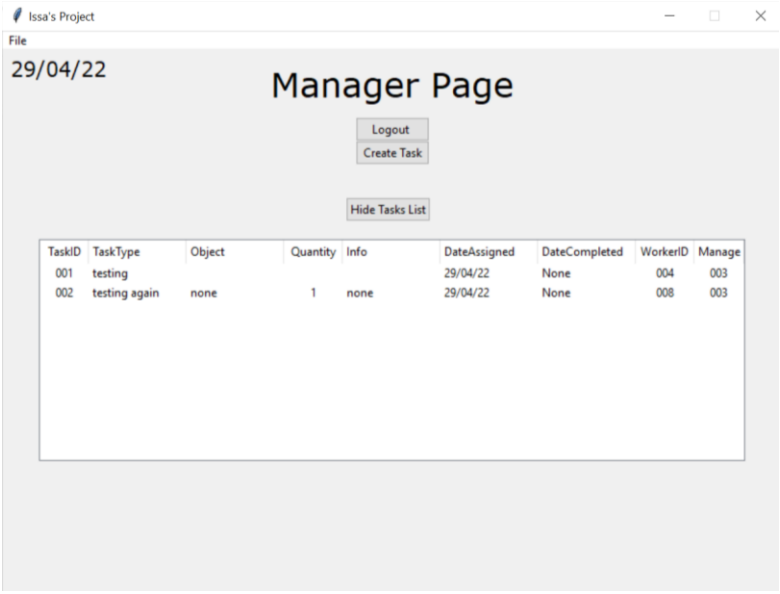
```
        self.MyTree.column("TaskID",anchor=CENTER,width=50)
        self.MyTree.column("TaskType",anchor=W,width=100)
        self.MyTree.column("Object",anchor=W,width=100)
        self.MyTree.column("Quantity",anchor=CENTER,width=60)
        self.MyTree.column("Info",anchor=W,width=100)
        self.MyTree.column("DateAssigned",anchor=W,width=100)
        self.MyTree.column("DateCompleted",anchor=W,width=100)
#format columns
        self.MyTree.column("WorkerID",anchor=CENTER,width=60)
        self.MyTree.column("ManagerID",anchor=CENTER,width=50)
```

The heading names for the columns were given. Finally to insert data there is a for loop that iterates through every record and inserts each field from TasksList into the treeview.

```
        self.MyTree.heading("#0",text="", anchor=W)
        self.MyTree.heading("TaskID",text="TaskID", anchor=CENTER)
        self.MyTree.heading("TaskType",text="TaskType", anchor=W)
        self.MyTree.heading("Object",text="Object", anchor=W)
        self.MyTree.heading("Quantity",text="Quantity", anchor=CENTER)
        self.MyTree.heading("Info",text="Info",
anchor=W)                                #create headings
        self.MyTree.heading("DateAssigned",text="DateAssigned", anchor=W)
        self.MyTree.heading("DateCompleted",text="DateCompleted", anchor=W)
        self.MyTree.heading("WorkerID",text="WorkerID", anchor=CENTER)
        self.MyTree.heading("ManagerID",text="ManagerID", anchor=CENTER)

        for record in TasksList:                    #add data
            self.MyTree.insert(parent='',index='end',text="",values=(record[
0],record[1],record[2],record[3],record[4],record[5],record[6],record[7],recor
d[8]))
```

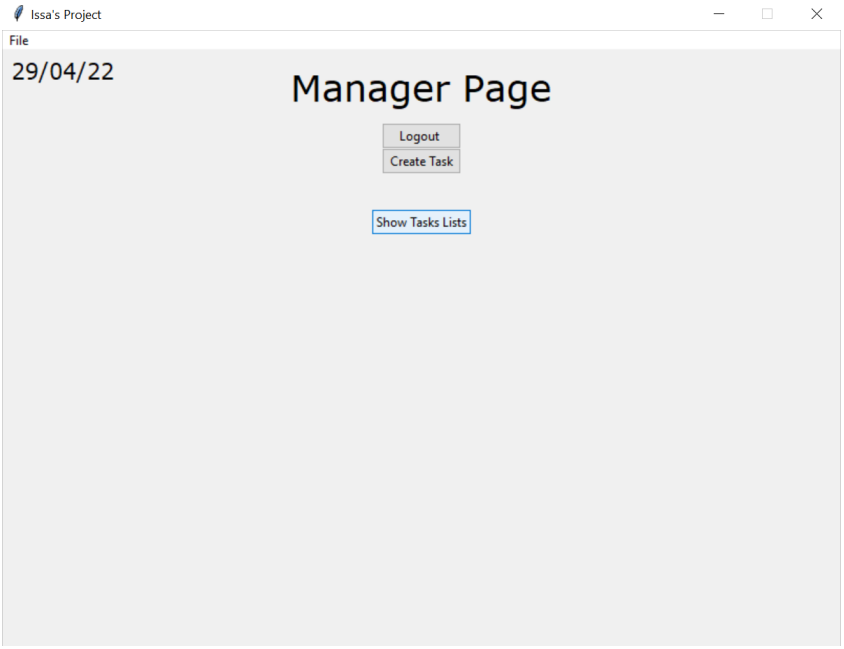
This method simply shows the treeview onto the page. The method is called when the ShowTasksList button is pressed. The success label and ShowTasksList button are removed from the page. The HideTasksList is placed in the position that the ShowTasksList was. Finally the treeview MyTree is packed onto the page.

```
    def ShowTasksList(self):
        self.SuccessLabel.place_forget()
        self.ShowTasksListButton.place_forget()
        self.HideTasksListButton.place(x=352,y=150)
        self.MyTree.pack(pady=100)
```



This method hides the treeview from the page. It is called when the HideTasksList button is pressed. It removes the treeview MyTree off the page. Then it removes the HideTasksList button and places the ShowTasksList button in its place so the treeview can be shown again.

```
def HideTasksList(self):
    self.MyTree.forget()
    self.HideTasksListButton.place_forget()
    self.ShowTasksListButton.place(x=352,y=150)
```



This method creates the Queue. It is called in the constructor method of this class so the Queue is ready straightaway. A limiting factor that I came across is that Queues can't be coded directly into Python, so the Queue is implemented using an array. The attribute Queue is created as an empty array

```
def CreateQueue(self):
    self.Queue = []
```

This method adds a TaskID to the Queue. The TaskID that needs to be added gets passed into the method as an argument, 'item'. Item is appended into the Queue which is a built in function to add an item into the array.

```
def AddToQueue(self,item):
    self.Queue.append(item)
```

This method removes a TaskID from the Queue. Once again the TaskID is passed in as an argument called item. The built in 'remove' function removes the item from the array.

```
def RemoveFromQueue(self,item):
    self.Queue.remove(item)
```

This method is to merge sort any list. The list that needs to be sorted is passed in as a parameter 'List'. This method is a recursive subroutine, meaning it calls itself. If the length of List is greater than 1 it assigns into a variable 'mid' half of the length of the list. Any items before the 'mid' position of the list are assigned into a new list called LeftList. Any items after the 'mid' position are assigned into RightList. It then recursively calls itself to merge sort both LeftList and Right List. This will call itself again and again until LeftList and RightList are just a single item. At this ppoint the stopping condition has been met and the recursive calls will start to unwind and go back. Then it will use the while loops to iteratively merge the lists back together. Finally the sorted list is returned.

```
def MergeSort(self,List):
    if len(List) > 1:
        mid = len(List) // 2
        LeftList = List[:mid]
        RightList = List[mid:]
        self.MergeSort(LeftList)
        self.MergeSort(RightList)
        #add elements from right and left into merged list in order
        i = 0
        j = 0
        k = 0
        while i < len(LeftList) and j < len(RightList):
            if LeftList[i] < RightList[j]:
                List[k] = LeftList[i]
                i = i + 1
            else:
                List[k] = RightList[j]
                j = j + 1
            k = k + 1
            #check if left list has elements not merged
        while i < len(LeftList):
                List[k] = LeftList[i]
                i = i + 1
                k = k + 1
            #check if right list has elements not merged
        while j < len(RightList):
                List[k] = RightList[j]
                j = j + 1
                k = k + 1
        return(List)
```

This method recursively binary searches a list for an item. The list to search, the item to search for and also the count , leftmost index position and the rightmost index position are passed in as arguments 'List','item', 'left', 'count' and 'right'. The binary search requires that the list is sorted.
The mid is found by adding together right, left and 1. If 'left' is less than 0 then the item cannot be in the list.
The stopping condition is defined as if count is less than half of the length of the list then False is returned. This is because with a binary search, the maximum number of searches that happens is half the number of items in the list. If it reaches this number of searches then the item cannot be in the list.
If the item with index position 'mid' in the list is the item that is being searched for then the item has been found and it returns 'mid' + 1 because index positions start from 0 in Python and mid is an index position so it will be one less. If the item is less than the item in index position mid it will recursively call itself and pass in the list, the item and left as they were. It will pass in mid – 1 as the new right because anything greater than the mid cannot be the item. count is incremented and passed in. If the item is greater than the item in index position mid it will recursively call itself and pass in the list, the item and right as they were. It will pass in mid +1 as the new right because anything less than the mid cannot be the item. count is incremented and passed in. This repeats until the stopping condition is met where the recursive calls will start to unwind. Eventually the position of the item will be returned if it is in the list or "False" will be returned if it is not in the list.

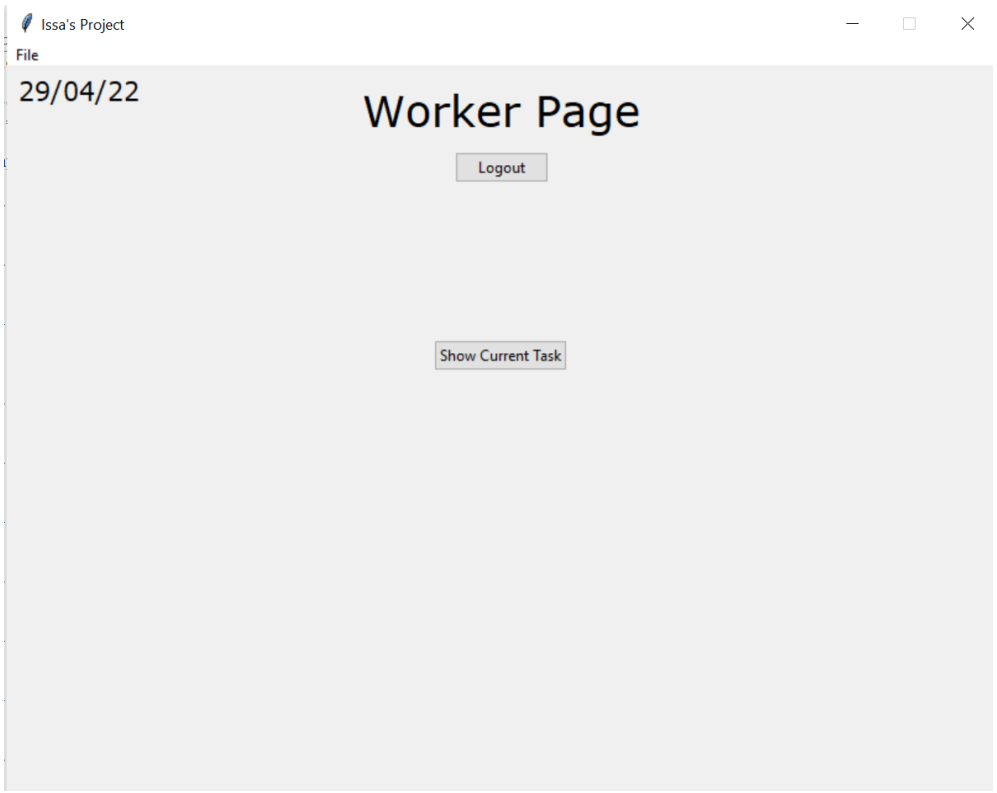
```
def BinarySearch(self,List,item,count,left,right):
    mid = (1+left+right)//2
    if count > (len(List)//2):
        return False
    if left < 0:
        return False
    if List[mid] == item:
        return mid + 1
    elif item <List[mid]:
        return self.BinarySearch(List,item,(count+1),left,mid-1)
    elif item> List[mid]:
        return self.BinarySearch(List,item,(count+1),mid+1,right)
```

This class is the final class and it is for the workers main page.

```
class WorkerPage(tk.Frame):
```

The constructor method sets up the page. It initialises the frame from the parent class. The DefineWidgets method is called to create all widgets for the page. The title label is packed at the top of the screen. The back to login button is packed onto the screen and the current task button is placed onto the screen.

```
    def __init__(self, parent, controller):
        Frame.__init__(self,parent)

        self.DefineWidgets(controller)

        self.TitleLabel.pack(pady=10,padx=10)

        self.BackToLoginButton.pack()

        self.CurrentTaskButton.place(x=345,y=220)
```



This method creates the widgets for the page. The title label, no tasks label and success label are defined. The back to login button, current task button, hide button and complete button are also defined here.

```
    def DefineWidgets(self,controller):
        self.TitleLabel = Label(self, text="Worker Page", font=TITLE_FONT)

        self.BackToLoginButton = ttk.Button(self, text="Logout", command
=lambda: self.LogOut(controller))

        self.CurrentTaskButton = ttk.Button(self, text = "Show Current Task",
command = lambda: self.ShowTask())

        self.NoTasksLabel = Label(self, text = "No tasks have been currently
assigned", font = SMALL_FONT, fg="red")

        self.HideButton = ttk.Button(self, text = "Hide", command = lambda:
self.HideTask())

        self.CompleteButton = ttk.Button(self, text = "Mark Task As
Complete", command = lambda: self.CompleteTask())

        self.SuccessLabel = Label(self, text = "Task Completed!", font =
MINI_FONT,fg = "green")
```

This method lets the user logout of the page and be returned to the login page. It is called when the BackToLogin button is pressed. A pop up message box appears asking the user if they are sure they want to logout. If they press yes, the ResetPage method is called which restores the page how it was initially and then uses the show_frame method to show the Worker Login Page.

```
    def LogOut(self,controller):
        Confirm = messagebox.askyesno(title="Log Out",message="Are you sure
you want to log out?")
        if Confirm > 0:
            self.ResetPage()
            controller.show_frame(WorkerLogin)
```

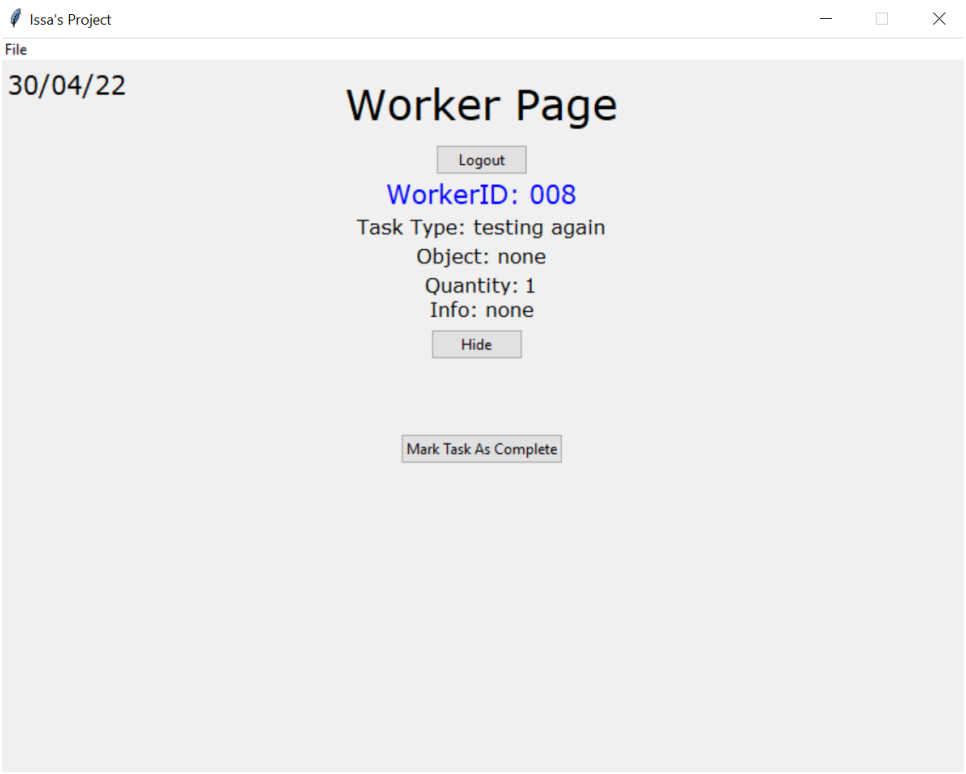
This method resets the page to restore it to look like how it was initially. It creates a list of all widgets that are packed onto the screen and another one with all widgets that are placed onto the screen. Then the method attempts to forget all of the extra widgets which are all defined outside of the DefineWidgets method therefore exception handling is used to remove them to prevent an error occurring in case they haven't been defined when this method is called. Then the for loop iterates through the first widget list and forgets them. It then uses another for loop to iterate through the second widget list and forgets everything in there. Finally the current task button is placed onto the screen.

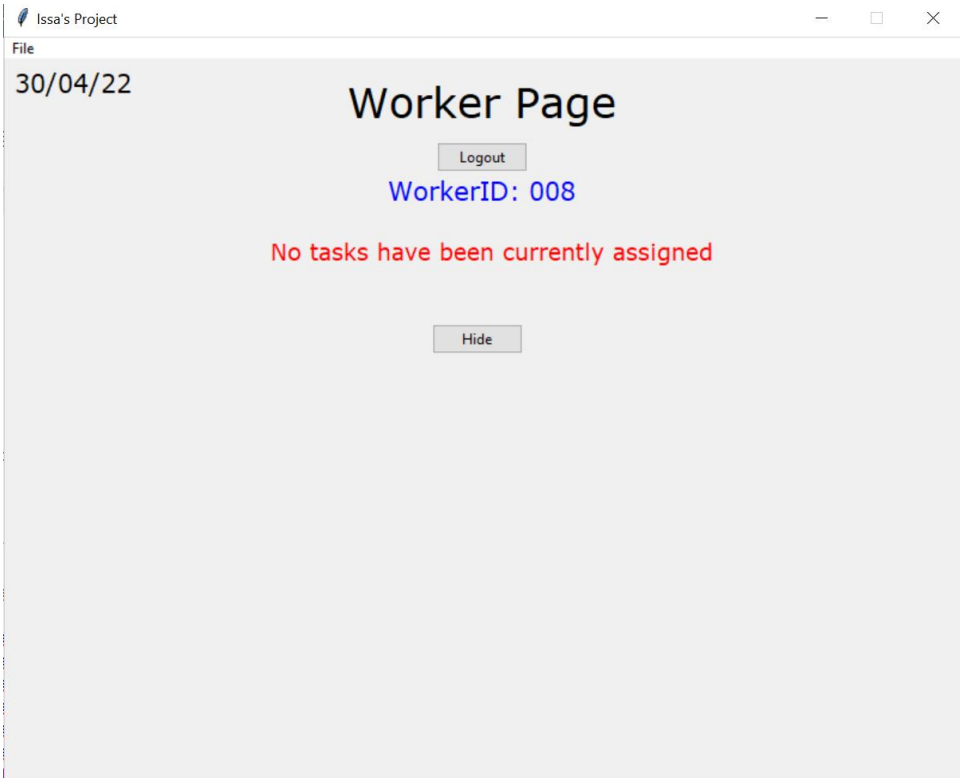
```
    def ResetPage(self):
        WidgetList1 =
[self.CompleteButton,self.HideButton,self.NoTasksLabel,self.SuccessLabel]
        WidgetList2 = [self.HideButton,self.NoTasksLabel]
        try:
           self.IDLabel.forget()
           self.TaskTypeLabel.forget()
           self.ObjectLabel.forget()
           self.InfoLabel.forget()
           self.QuantityLabel.place_forget()
           self.QuantityValueLabel.place_forget()
           for x in WidgetList1:
                x.pack_forget()
        except:
               for x in WidgetList1:
                   x.pack_forget()
        for x in WidgetList2:
            x.place_forget()
        self.CurrentTaskButton.place(x=345,y=220)
```

This method is to show the current task on the screen for the worker. It is called when the current task button is pressed. The ID label is created to show the user their worker ID and makes use of the global variable CurrentWorkerID to do this. This ID label is packed onto the screen. The current task button is removed off the screen. The hide button is placed where the current task button was. The SQL statement is executed to select the TaskType,Object,Quantity and Info for the current worker ID from the Tasks table. The WHERE condition o the statement makes sure that DateCompleted is null so it selects the task that hasn't been completed so is it current task. The "ORDER BY TaskID ASC LIMIT 1" selects the first record in the table that meets the WHERE condition. Then the labels are created to be able to show these task details and are displayed onto the page. This is all done using exception handling in case there is nothing to select from the table so the error can be prevented. The except part would assume nothing was selected from the table and place the error label onto the page which tells the user that there is currently no task assigned to them.

```
    def ShowTask(self):
        self.IDLabel = Label(self, text="WorkerID: "+CurrentWorkerID,
font=MEDIUM_FONT,fg="blue")
        self.IDLabel.pack()
        self.NoTasksLabel.place_forget()
        self.CurrentTaskButton.place_forget()
        self.SuccessLabel.forget()
```

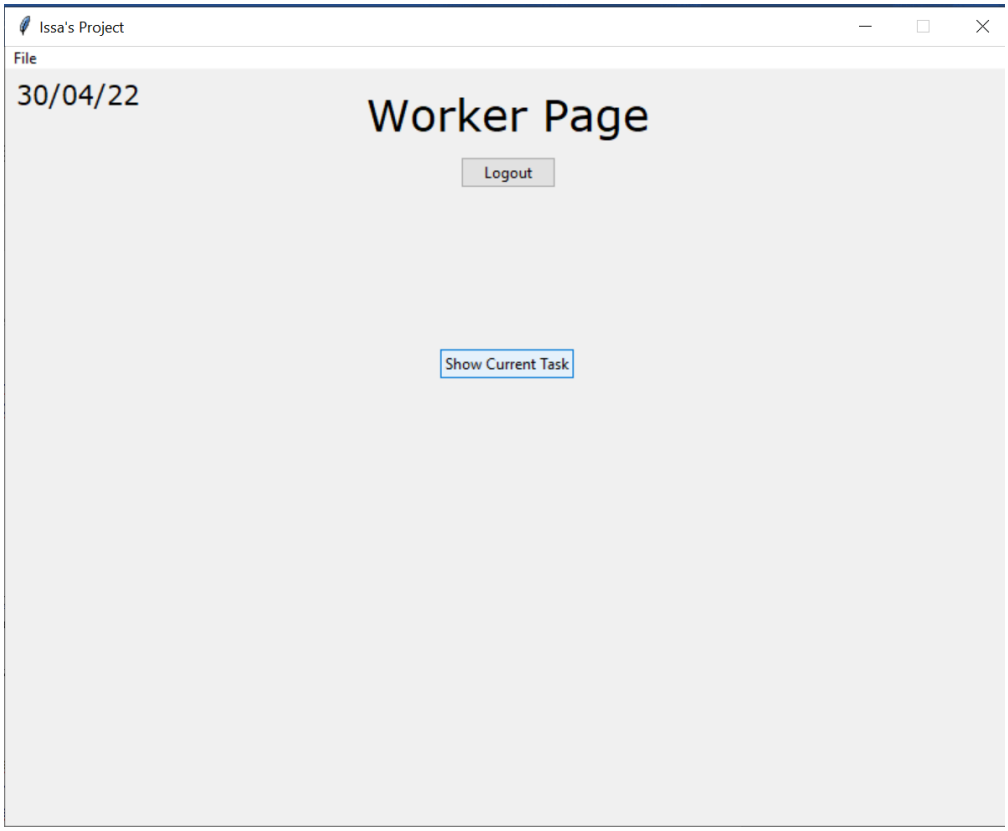
```
        self.HideButton.place(x=358,y=220)
        try:
            NewPath = "Project Database.db"
            connection = sqlite3.connect(NewPath)
            data = connection.cursor()
            data.execute("""SELECT TaskID,TaskType,Object,Quantity,Info FROM
Tasks WHERE WorkerID LIKE ? AND DateCompleted IS NULL ORDER BY TaskID ASC
LIMIT 1""",(CurrentWorkerID,))
            task = data.fetchall()

            self.CurrentTaskID = task[0][0]

            self.TaskTypeLabel = Label(self, text = "Task Type: "+task[0][1],
font = MINI_FONT)
            self.TaskTypeLabel.pack()

            self.ObjectLabel = Label(self, text = "Object: "+task[0][2], font
= MINI_FONT)
            self.ObjectLabel.pack()

            self.QuantityLabel = Label(self, text = "Quantity: ", font =
MINI_FONT)
            self.QuantityLabel.place(x=350,y=172)

            self.QuantityValueLabel = Label(self, text = task[0][3], font =
MINI_FONT)
            self.QuantityValueLabel.place(x=432,y=172)

            self.InfoLabel = Label(self, text = "Info: "+task[0][4], font =
MINI_FONT)
            self.InfoLabel.pack(pady=20)

            self.CompleteButton.pack(pady=70)

        except:
              self.NoTasksLabel.place(x=220,y=145)
```


Issa's Project
File
30/04/22
Worker Page
Logout
WorkerID: 008
Task Type: testing again
Object: none
Quantity: 1
Info: none
Hide
Mark Task As Complete


Issa's Project
File
30/04/22
Worker Page
Logout
WorkerID: 008
No tasks have been currently assigned
Hide

This method hides the current task from the page. It is called when the hide button is pressed. It uses exception handling to try to remove all of the TaskType, Object, Quantity and Info labels. In case these labels haven't been defined yet, the use of exception handling prevents an error and the except part is run which removes the error label off the page. Then the ID label is removed along with the hide button. The current task button is placed where the hide button was.

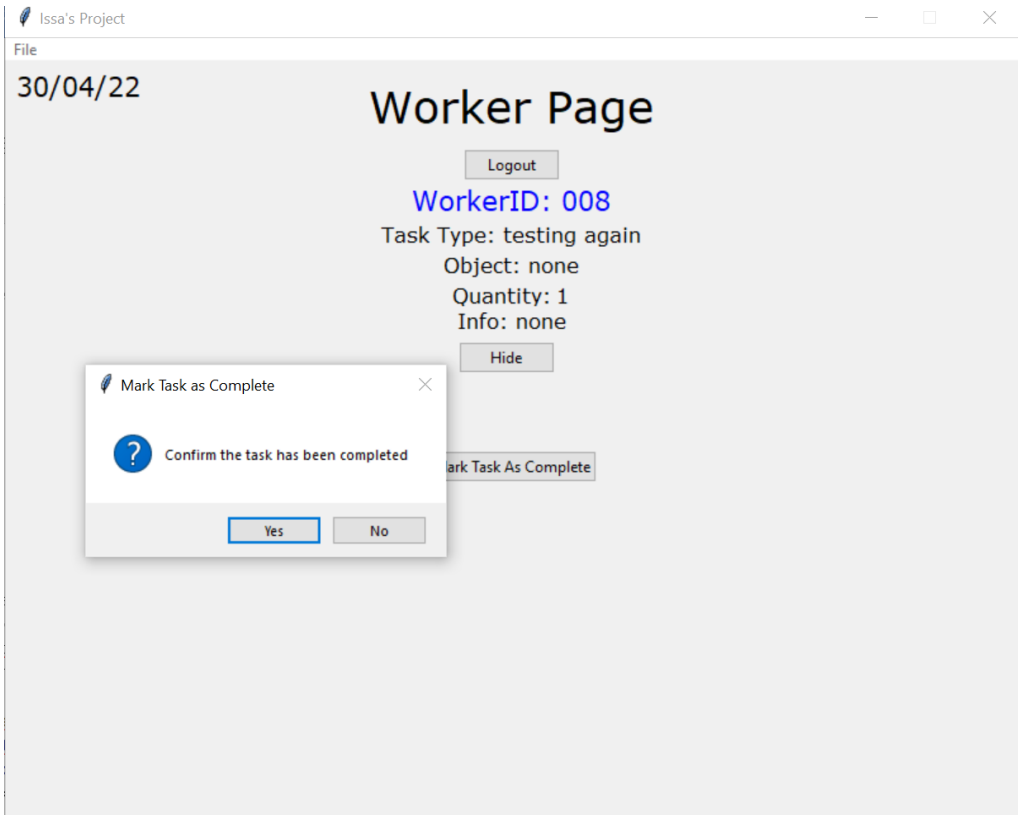
```
def HideTask(self):
    try:
        self.TaskTypeLabel.forget()
        self.ObjectLabel.forget()
        self.QuantityLabel.place_forget()
        self.QuantityValueLabel.place_forget()
        self.InfoLabel.forget()
        self.CompleteButton.forget()
    except:
            self.NoTasksLabel.forget()
    self.IDLabel.forget()
    self.NoTasksLabel.place_forget()
    self.HideButton.place_forget()
    self.CurrentTaskButton.place(x=345,y=220)
```

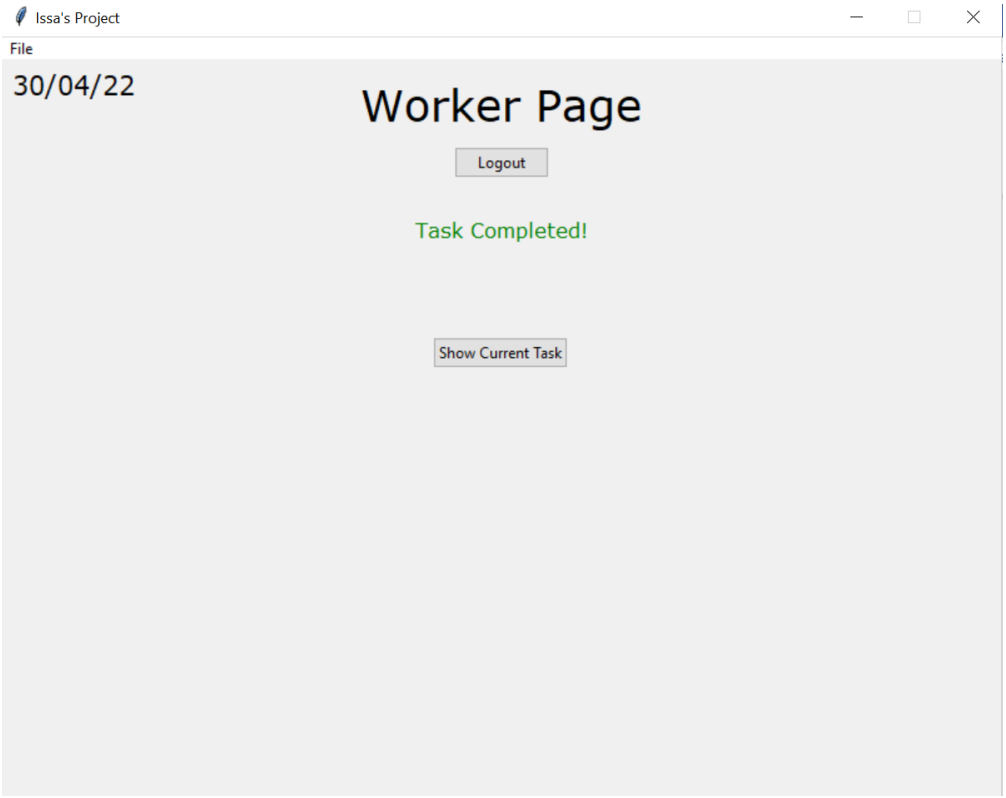


This method allows the user to mark their current task as complete. It is called when the complete button is pressed. A pop up message box is displayed on the page asking the user to confirm that they have completed the current task. If they press yes it runs the UpdateTasksTable method and the UpdateQueue

method. It then calls the ResetPage method to bring the page back to how it initially looked and then it places the success label on the screen to tell the user that the current task has been marked as complete.

```
    def CompleteTask(self):
        Confirm = messagebox.askyesno(title="Mark Task as
Complete",message="Confirm the task has been completed")
        if Confirm > 0:
            self.UpdateTasksTable()
            self.UpdateQueue()
            self.ResetPage()
            self.SuccessLabel.pack(pady=30)
```





This method updates the Tasks table in the database after a task has been completed by filling in the DateCompleted field. The SQL statement uses “UPDATE” to set the DateCompleted field, using the constant str_date_today. The WHERE part of the statement ensures that only the TaskID which matches the attribute CurrentTaskID is updated in the table.

```
    def UpdateTasksTable(self):
        NewPath = "Project Database.db"
        connection = sqlite3.connect(NewPath)
        data = connection.cursor()
        data.execute("""UPDATE Tasks SET DateCompleted = ? WHERE TaskID LIKE
?""",(str_date_today,self.CurrentTaskID,))
        connection.commit()
```

This method updates the Queue from the Manager Page after a task has been completed. It uses the CheckIfBusy method from the Manager Page to check if the current worker is still busy after completing the current task with another task that was assigned after the current one. If the worker is not busy the method will attempt to select the TaskID of the first task that has been assigned to the CurrentWorkerID that has not yet been completed. Then the RemoveQueueMethod is called from the Manager Page to remove this TaskID from the Queue.

This is all coded using exception handling because many errors could occur. One error that could occur is that there is nothing to select from the tasks table. Another is that the TaskID may not be in the Queue. Therefore exceptionhandling prevents these errors from occurring and the except part just returns the None value which does nothing.

```
    def UpdateQueue(self):
        busy = ManagerPage.CheckIfBusy(ManagerPage,CurrentWorkerID)
        if busy == False:
            try:
                NewPath = "Project Database.db"
                connection = sqlite3.connect(NewPath)
                data = connection.cursor()
                data.execute("""SELECT TaskID FROM Tasks WHERE WorkerID
LIKE ? AND DateCompleted IS NULL ORDER BY TaskID ASC LIMIT
1""",(CurrentWorkerID,))
                ID = data.fetchall()
                ManagerPage.RemoveFromQueue(ManagerPage,ID[0])

            except:
                  return None
```

The final part of the code creates an instance of the window class called app. Then it uses the mainloop function to make the whole program run.

```
app = window()                    #creating an instance of the window class
app.mainloop()                    #runs app
```

Testing

Test Strategy

I am going to test my code using two different types of testing: Functional Testing (Black Box Testing) and Objective Testing.

Functional Testing (Black Box Testing) will be carried out as I am coding the software and its purpose is to ensure that the code functions properly and that the user interface is valid and can be used by all users. It is important to do it whilst I am coding so I can find any possible errors or bugs in the code and am able to correct them and progress, therefore showing the development of the code.

Objective Testing will be carried out after fully completing the code. Its purpose is to test the code against the objectives I had set earlier in the Analysis section to ensure that my code functions as it was supposed to and that it solves the current problem. It is important to do it after completing the code in order to get the best possible outcomes from the code which can then be used to conclude whether the objectives have been met and that the current problem has been solved. For Objective Testing I will use End-User Testing to allow the potential users of the system to test the code themselves, so I can evaluate how well the code will work in a real life working environment and also to determine whether my objectives are met or not.

Test Plan

All tests are shown in the table. Underneath the table there is test evidence and additional annotations for each test shown.

Functional Testing

Test Number	Test Description	Test Data	Expected outcome	Actual outcome	Pass?	Corrective action	Refe-rence	Comments
1	Testing that the “close program” option in the menubar successfully closes the window		Window closes	Window closes	Yes	None	Test 1	Test was a success and the window closed exactly as expected.
2	Incorrectly logging in on the Manager Login Page	Erroneous (Login Details)	Login fails and error message is displayed	Login fails and error message is displayed	Yes	None	Test 2	Test was a success and the error message was shown exactly as expected

3	Correctly logging in on the Manager Login Page	Normal (Login Details)	Login is successful and the Manager Page appears	Login is successful and the Manager Page appears	Yes	None	Test 3	Test was a success and the Manager Page is shown, exactly as expected
4	Incorrectly logging in on the Worker Login Page	Erroneous (Login Details)	Login is not successful and the error message	Login is not successful and the error message	Yes	None	Test 4	Test was a success and the error message was shown exactly as expected
5	Correctly logging in on the Worker Login Page	Normal (Login Details)	Login is successful and the Worker Page appears	Login is successful and the Worker Page appears	Yes	None	Test 5	Test was a success and the Worker Page is shown, exactly as expected
6	Logging out from the Manager Page and checking that it returns to the Manager Login Page and also resets the Manager Page for when I log in again	Normal (Login Details)	Goes back to Manager Login Page and when logs back in the Manager Page has been reset to how it was originally	Goes back to Manager Login Page and when logs back in the Manager Page has been reset to how it was originally	Yes	None	Test 6	Test was a success, when logging out I was returned to the Manager Login Page and when I logged back in the page had been reset. All was as expected to be
7	Logging out from the Worker Page and checking that it returns to the Worker Login Page and also resets the Worker Page for when I log in again	Normal (Login Details)	Goes back to Worker Login Page and when logging back in the Worker Page has been reset to how it was originally	Goes back to Worker Login Page and when logging back in the Worker Page has been reset to how it was originally	Yes	None	Test 7	Test was a success, when logging out I was returned to the Worker Login Page and when I logged back in the page had been reset. All was as expected to be
8	Manually assigning a task from Manager ID 001's page to Worker ID 004 and checking it	Normal (TaskType Object Quantity Info)	Message stating that task has been assigned should appear on the page. The entered task details should	Message stating that task has been assigned appears on the page. The entered	No	Using the Increment Current Task ID method in the wrong place caused a logic error	Test 8	The test was mostly successful however not completely successful meaning it failed. The task had been assigned and a message to confirm this was displayed. In the database the task

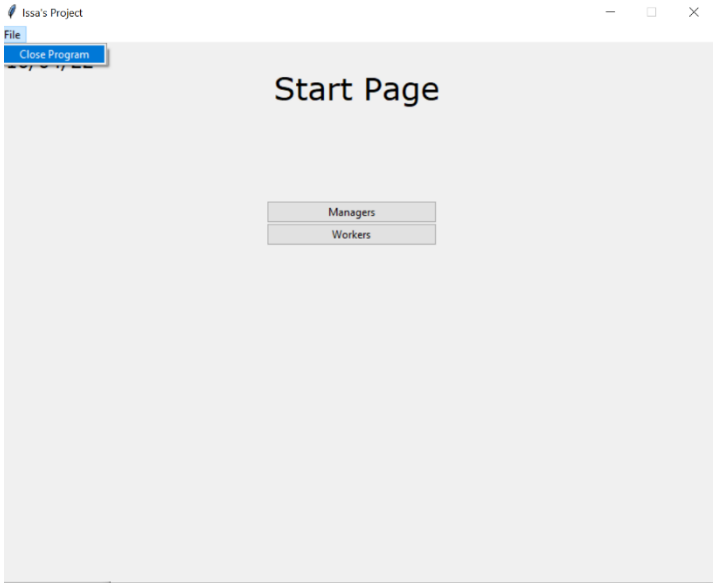
	gets assigned to them		be stored in the database	task details have been stored in the database however the TaskID was stored as 002, but it was supposed to be 001		by increm-enting before the Task had been stored in the database so it needs to be used somew-here else in the code.		details I had entered were stored correctly. The DateAssigned was today's date which is correct. The DateCompleted was null which is correct as the task hadn't been completed. The Manager ID was 001 and the WorkerID was 004 which were both correct. However the TaskID was 002 which was incorrect because it was the first task in the table, so it should be TaskID 001.This is a logic error because it didn't cause the code to crash but it did not function as intended.
9	Logging in as Worker_4 and checking that the task assigned in Test 8 can be viewed	Normal (Task Details)	When pressing the Show Current Task button in the Worker Page the TaskType, Object, Quantity and Info should appear on the page	When pressing the Show Current Task button in the Worker Page the TaskType, Object, Quantity and Info appears on the page	Yes	None	Test 9	The test was a success. The exact details of the assigned task appeared on the page.
10	Testing the Hide Task button function		When pressing the Hide Task button in the Worker Page all the task details should disappear aswell as the Hide Task button itself. The Show Current Task Button should then appear	All the task details and the Hide Task button disappear-ed and the Show Current Task button appeared.	Yes	None	Test 10	The test was successful. The details and Hide button disappeared and the show button appeared, exactly as expected.

11	Logging into 2 other worker accounts and checking that they can't view the task assigned in test 8 because it wasn't assigned to them	Normal (Login Details)	When pressing the Show Current Task button, the task details of the task assigned in test 8 should not appear	The details of the task from test 8 don't appear. No task has been assigned to either worker so in both a message appears stating that there is no task that has been currently assigned to them	Yes	None	Test 11	The test was successful. The task didn't appear to them and a suitable message was displayed. This was exactly as expected.
12	Marking the task from test 8 as complete on Worker_4's page		A message stating that the task has been completed should appear on the page. In the database the DateCom-pleted field should for this task should be filled in with today's date.	The message was displayed on the page and the DateCom-pleted was updated in the database.	Yes	None	Test 12	The test was successful. The message appeared and the database was correctly updated, exactly as expected.
13	Checking the task from Test 8 doesn't show up again once it has been completed when the Show Current Task button is pressed		The task from Test 8 doesn't show up when the Show Current Task button is pressed on the Worker_4 page. The no tasks message should show up	The task from Test 8 doesn't show up. The no tasks message does show up	Yes	None	Test 13	The test was a success. The task didn't show up and the no tasks message showed up, exactly as expected.
14	Assign two tasks and check that	Normal (Task Details)	Both the tasks appear in the	Both tasks appeared, exactly as	Yes	None	Test 14	The test was successful. The tasks appeared correctly on

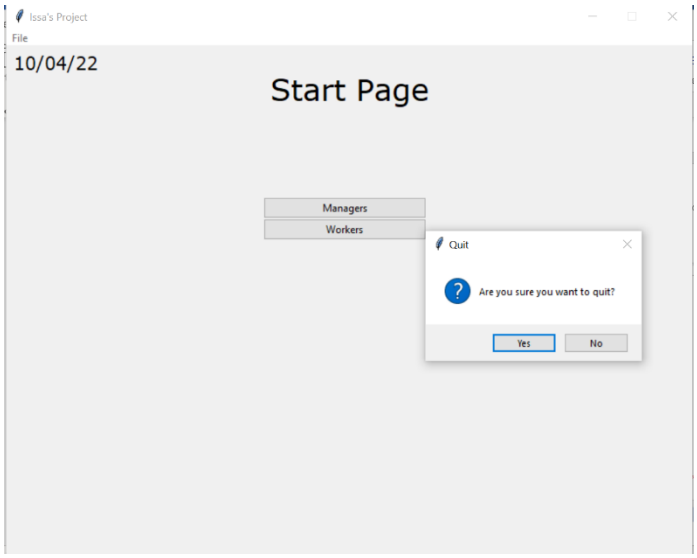
	these appear and are correct when the Show Tasks List button is pressed in the Manager Page		treeview, exactly as they were entered	they were entered				the page, in the treeview exactly as expected.
15	Testing the check if a user is busy method to ensure it correctly recognises whether a user is busy or not.	Normal (Task Details)	When the first task is assigned the shell should print "Not busy" and when the second task is assigned it should print "Busy"	"Busy" is printed when the first task is assigned and "Not Busy" is printed when the second task is assigned	Yes	None	Test 15	Test was a success. When the first task was assigned the worker's status was "Not Busy". Since that task got assigned to them and hadn't been completed, when the second task was assigned the worker's status was "Busy".
16	Logging out after trying to manually assign a task to see if the page resets	Normal (Login Details)	When logging out I should be returned to the Manager Login Page and when I log back in the Manager Page should be reset to how it is originally	When logged out I was returned to the Manager Login Page. But when logging back in the page hadn't properly been reset.	No	The Reset Page method needs to be changed so that it removes everything off the page.	Test 16	The test was a fail. The code didn't crash but there was a logic error in the code as the page didn't reset properly.
17	Assigning two tasks to the same worker and checking it shows up for the worker in the correct order	Normal (Test Details)	When the worker views their current task, the task that was assigned first should show up. After this is marked as complete, the worker views their current task again and the second task should show up.	The first task shows up as the worker's current task. After marking it as complete the second task shows up as the current task.	Yes	None	Test 17	The test was a success. The first task that was assigned by the manager showed up when the Show Current Task button was pressed in the Worker Page. This task was marked as complete and then the Show Current Task button was pressed again and the second task assigned by the manager showed up. Both tasks were displayed

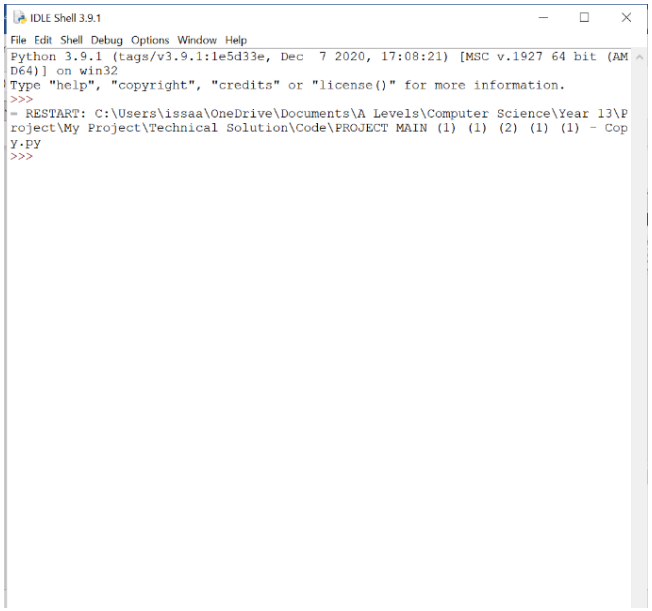
								correctly and in the order that they were assigned, exactly as expected.
18	Assigning two tasks to the same worker and checking that the second one gets added to the queue	Normal (Test Details)	When the first task is assigned, the queue will be printed and it should be empty. When the second task is assigned, the queue should be printed again but it should have TaskID '002' in it.	When the first task was assigned, the empty queue was printed. When the second task was assigned the queue containing TaskID '002' was printed.	Yes	None	Test 18	The test was successful. When the first task is assigned, the worker is free so the task will go straight to them and won't be added to the queue. When the second task is assigned, the worker is still busy with the first task so the second task which has a TaskID '002' is added to the queue. This happened exactly as expected.

Test 1

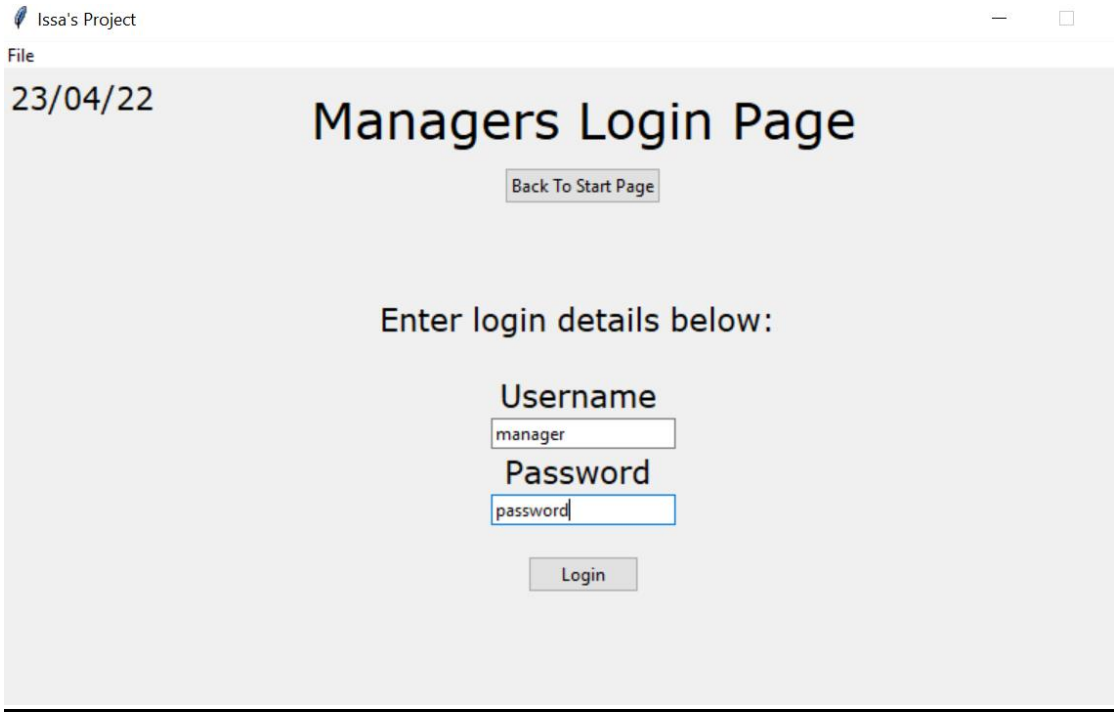


When pressing the file button in the taskbar in the top left of the window, the close program option comes up. When this is pressed a pop up message is displayed asking the user if they are sure that they want to quit. If they press 'Yes' the window closes, leaving only the Python IDLE Shell.

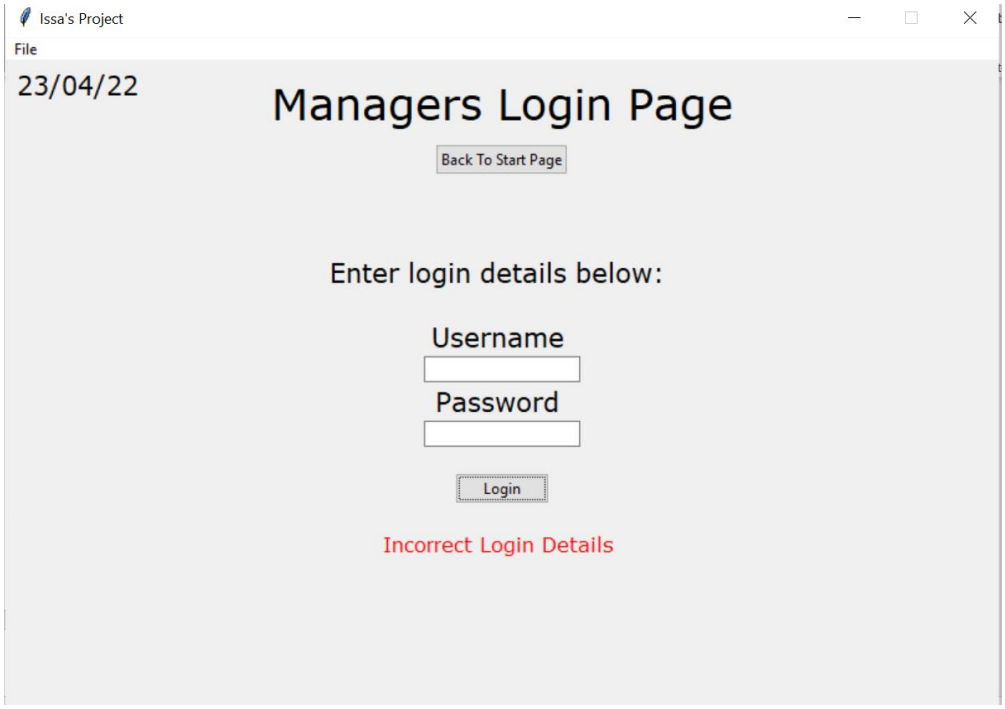

Issa's Project
File
10/04/22
Start Page
Managers
Workers
Quit
Are you sure you want to quit?
Yes
No


IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\issaa\OneDrive\Documents\A Levels\Computer Science\Year 13\Project\My Project\Technical Solution\Code\PROJECT MAIN (1) (1) (2) (1) (1) - Copy.py
>>>

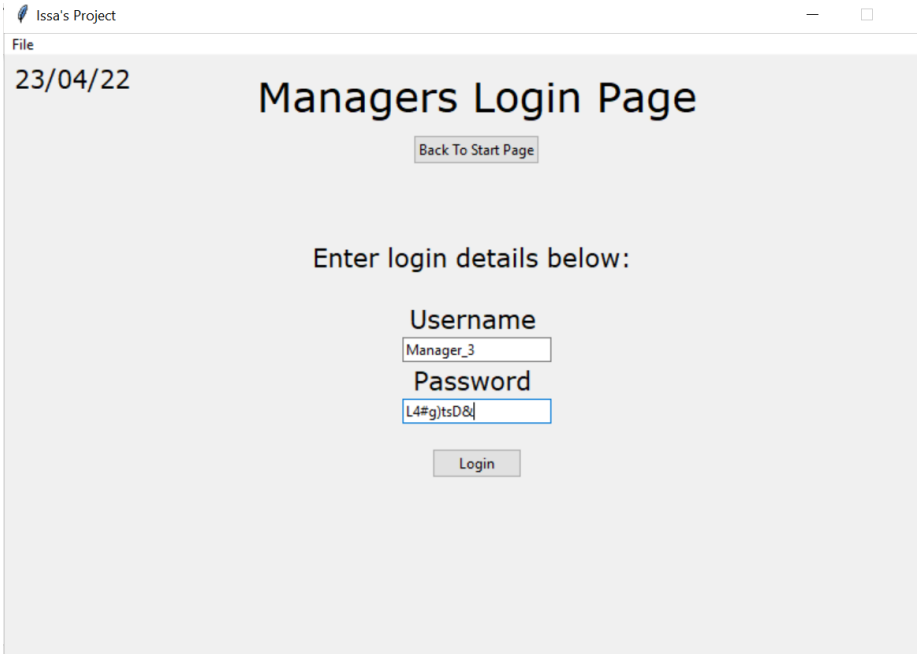
Test 2



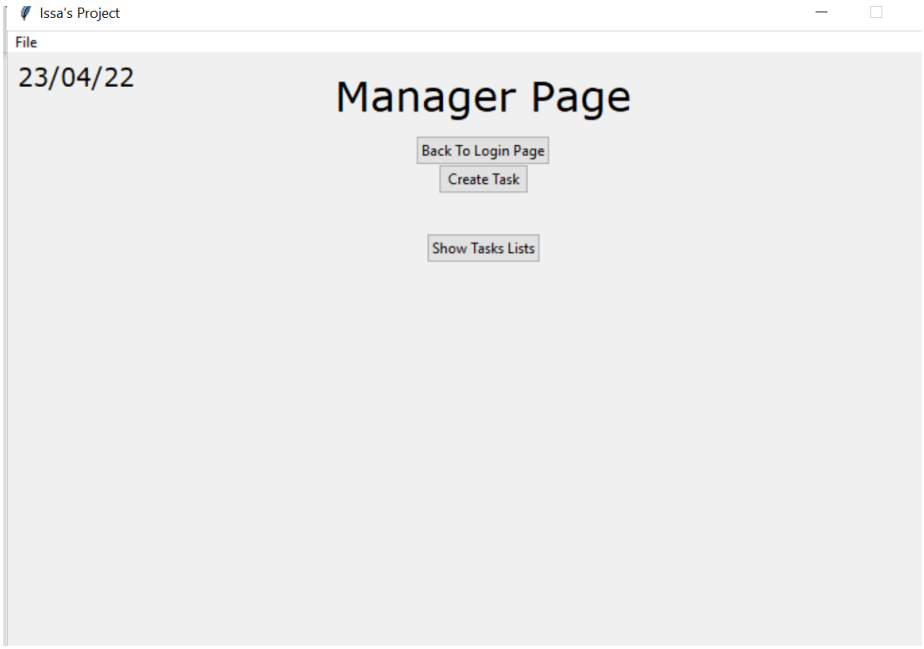
The incorrect login details are entered in and the login button is pressed. As the details are incorrect, the entry boxes are cleared and the error message is displayed on the screen stating that the login details are incorrect.



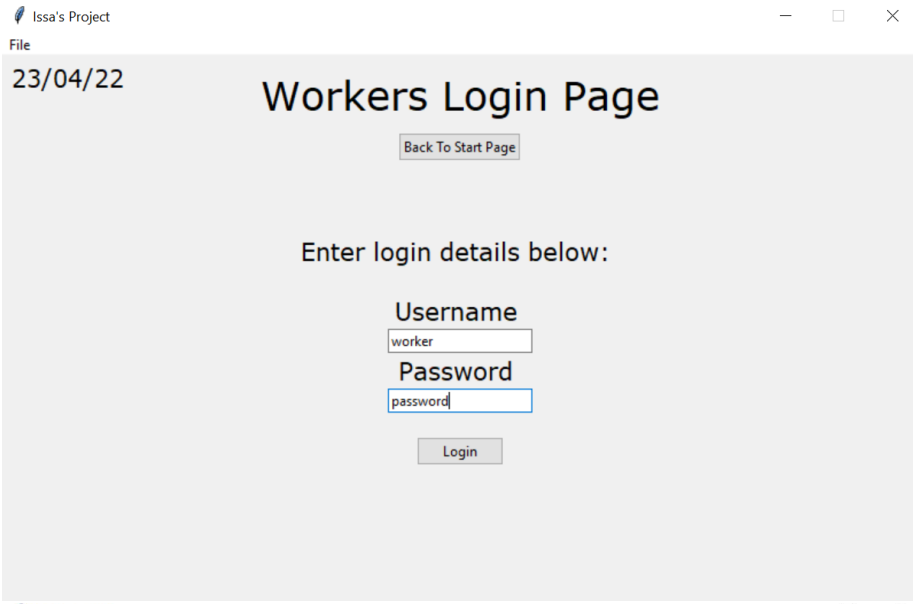
Test 3

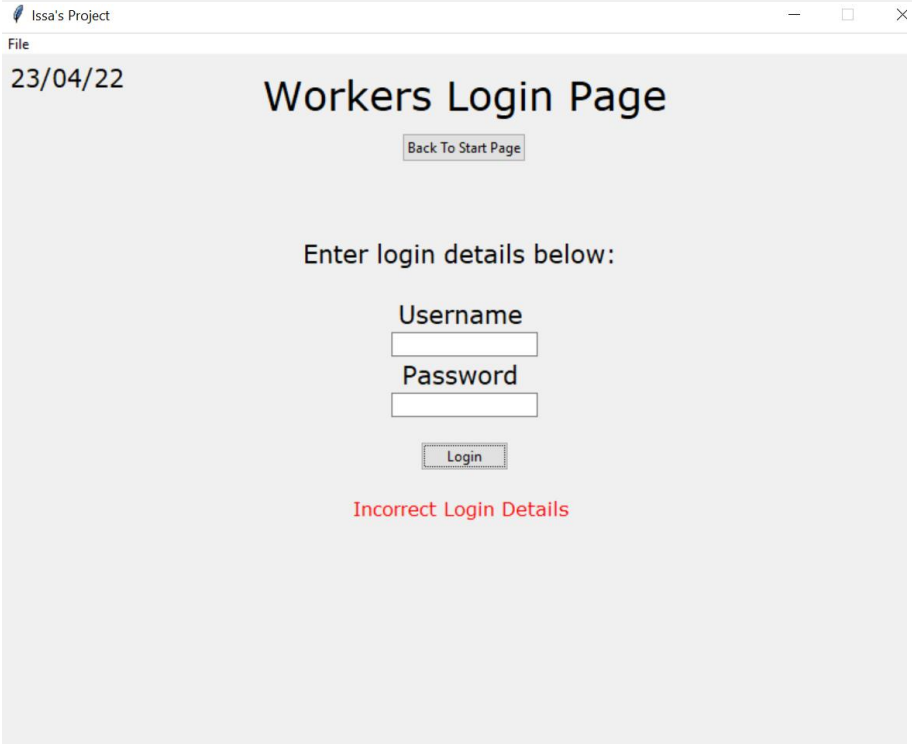


The correct login details are entered in and the login button is pressed. As the details are correct, the entry boxes are cleared and the Manager Page is displayed



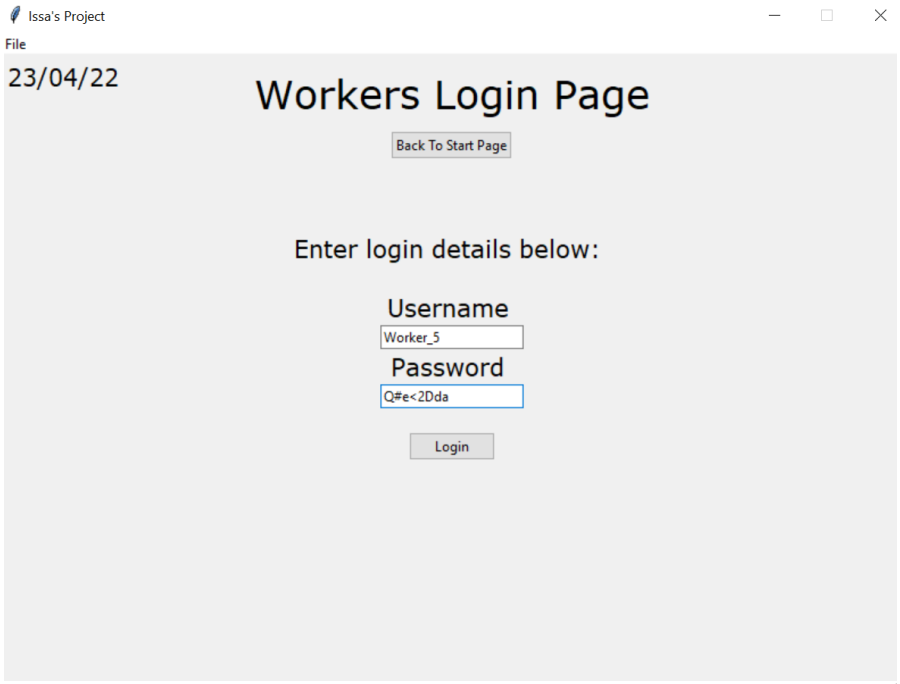
Test 4

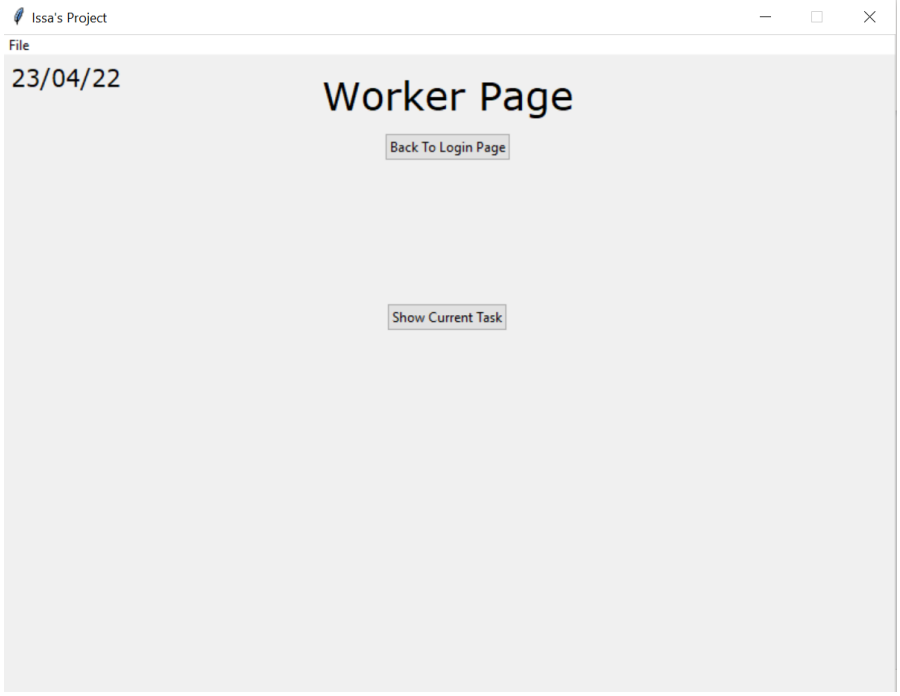




The incorrect login details are entered in and the login button is pressed. As the details are incorrect, the entry boxes are cleared and the error message is displayed on the screen stating that the login details are incorrect.

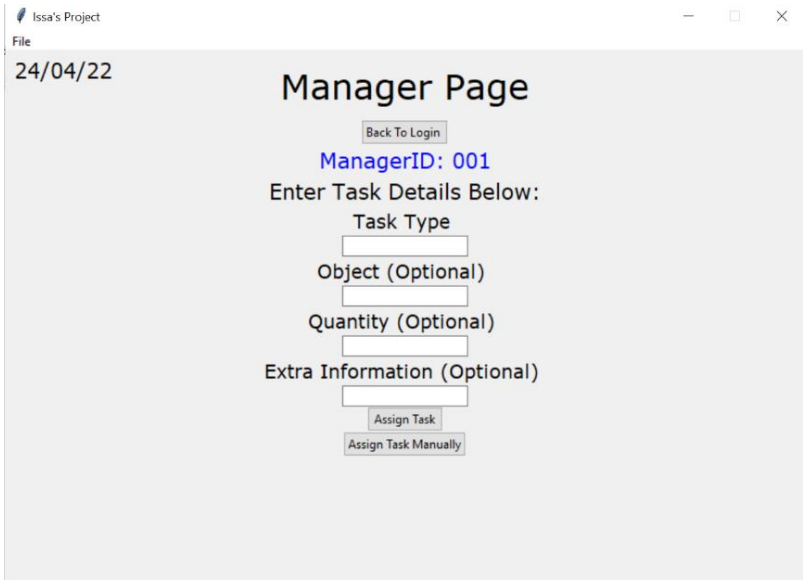
Test 5

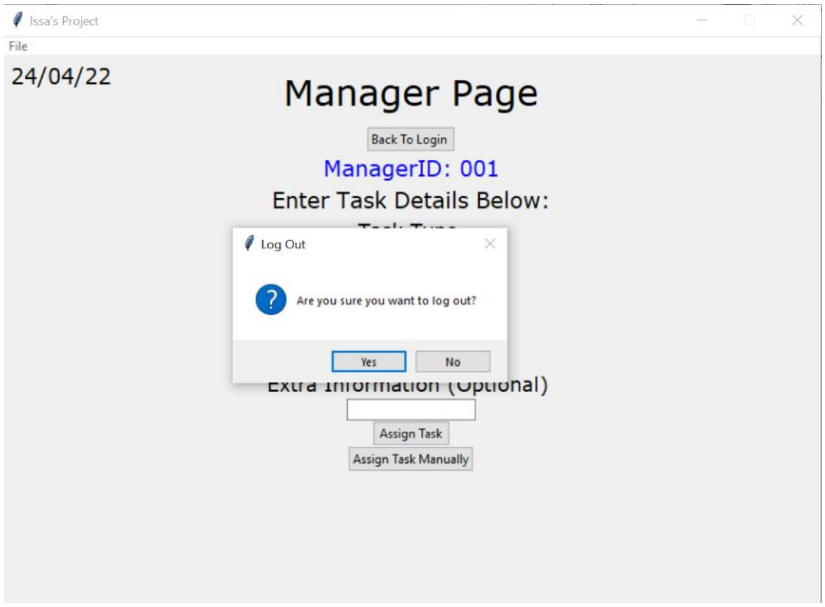


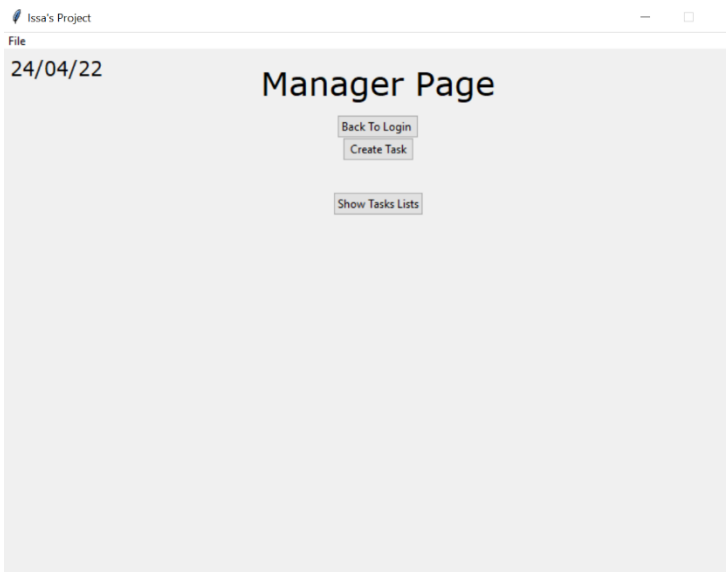


The correct login details are entered in and the login button is pressed. As the details are correct, the entry boxes are cleared and the Worker Page is displayed

Test 6

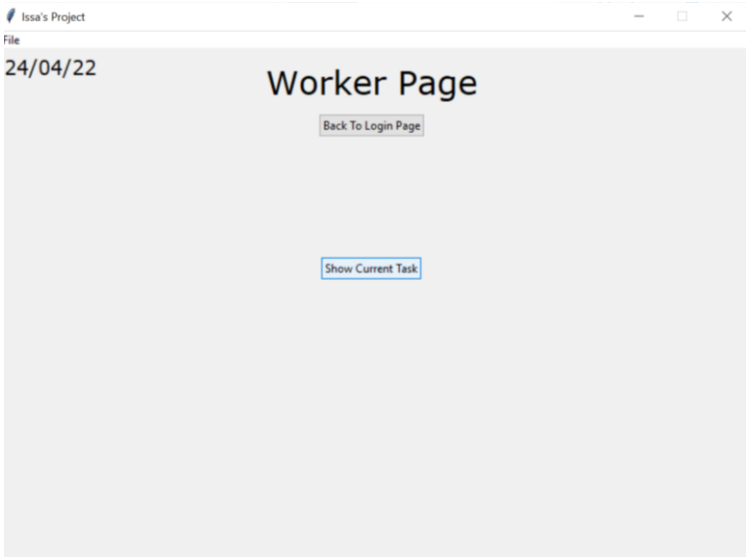


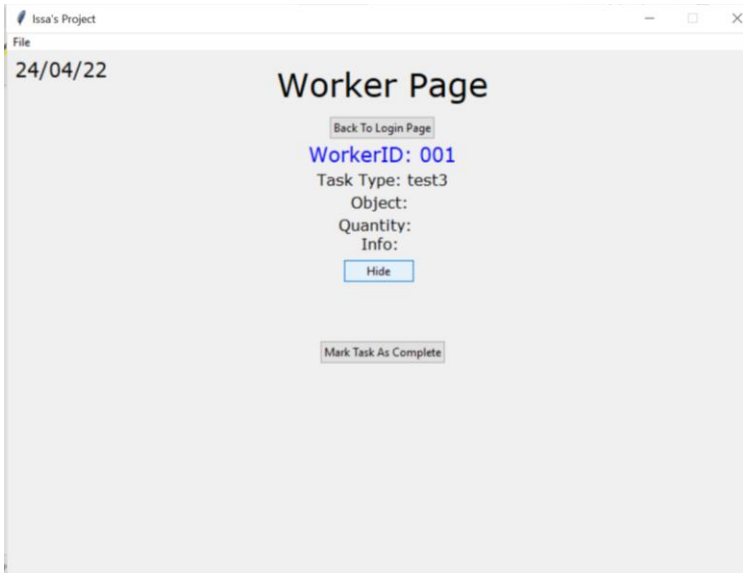




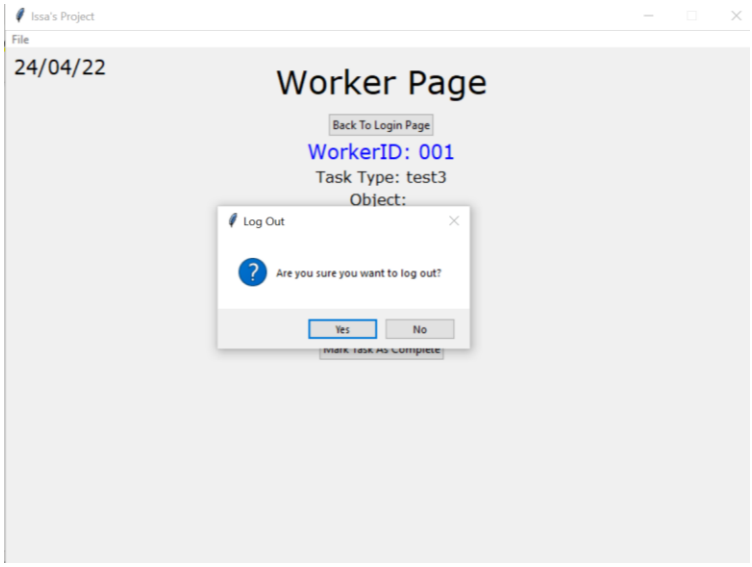
I logged into a Manager Page as normal then clicked the Create Task button to bring lots of extra widgets onto the page. Then I logged out using the Back To Login button. It successfully went back to the Manager Login page. I then logged back in to the same account and the page was reset as suspected, all the widgets from the Create Task function were gone and the original buttons were back on the screen.

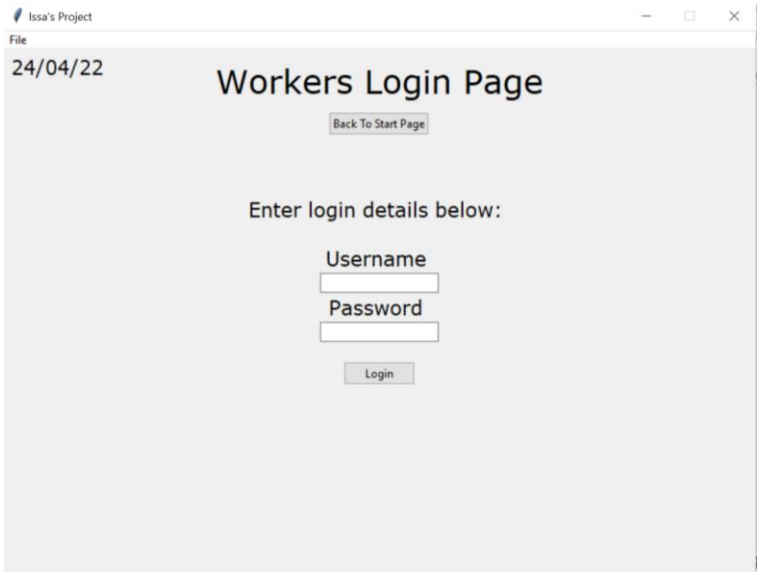
Test 7

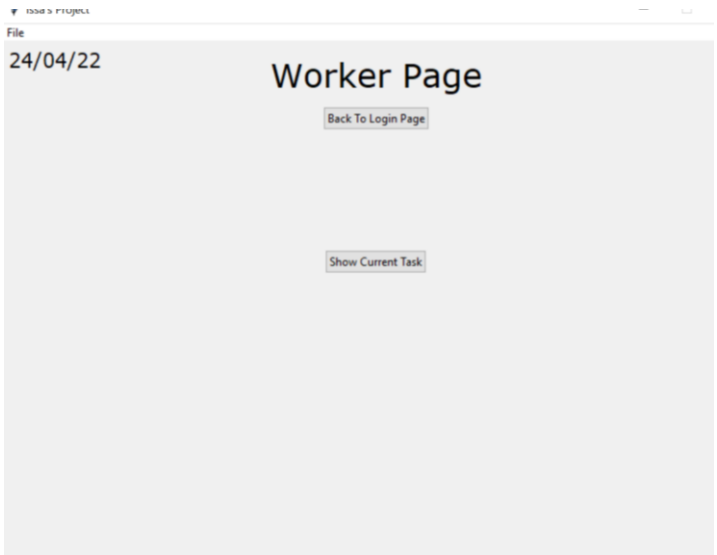




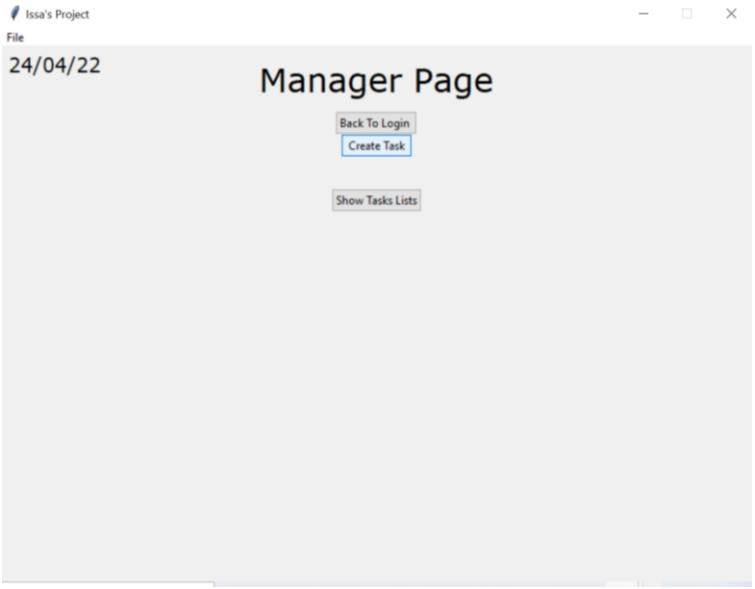
I logged into a Worker Page as normal then clicked the Show Current Task button to bring lots of extra widgets onto the page. Then I logged out using the Back To Login Page button. It successfully went back to the Worker Login Page. I then logged back in to the same account and the page was reset, all the widgets from the Show Current Task function were gone and the original buttons were back on the screen.

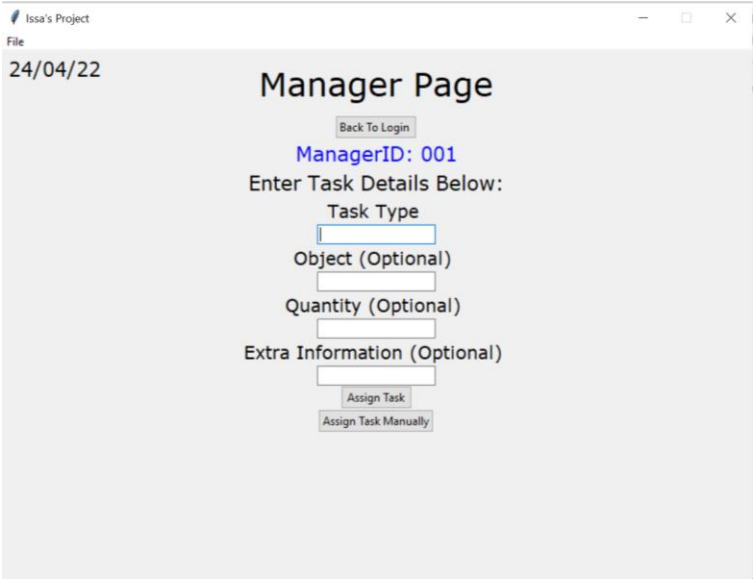

Issa's Project
File
24/04/22
Worker Page
Back To Login Page
WorkerID: 001
Task Type: test3
Object:
Log Out
Are you sure you want to log out?
Yes
No

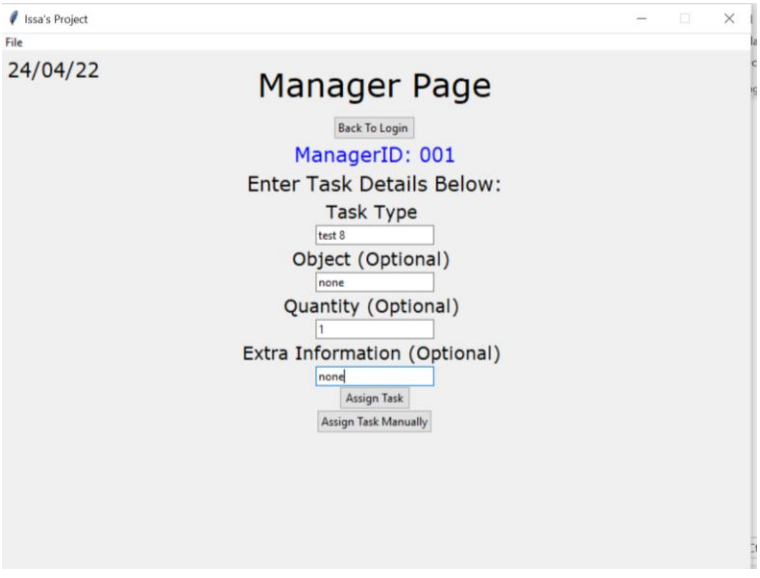

Issa's Project
File
24/04/22
Workers Login Page
Back To Start Page
Enter login details below:
Username
Password
Login


File
24/04/22
Worker Page
Back To Login Page
Show Current Task

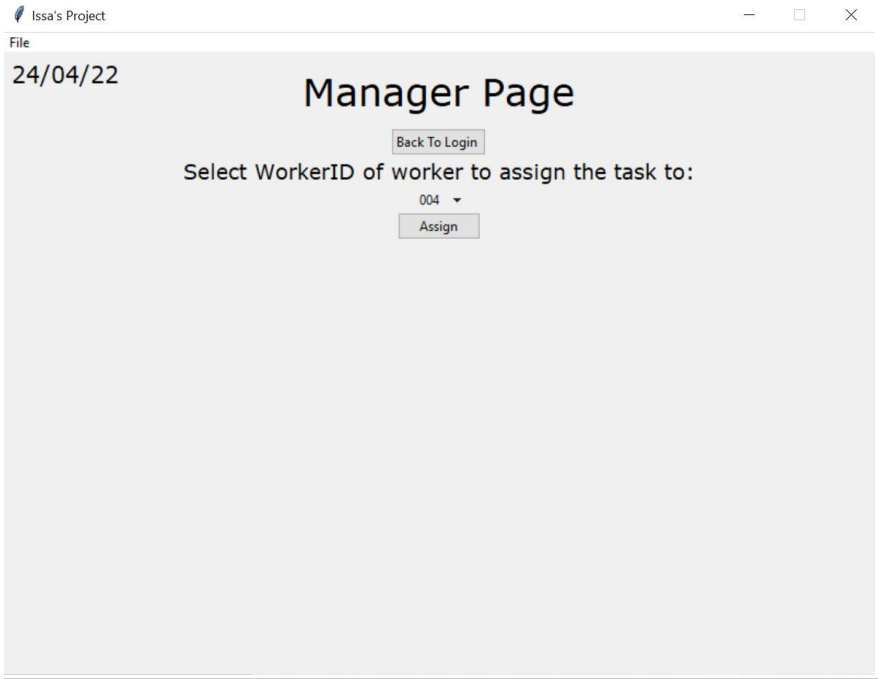
Test 8

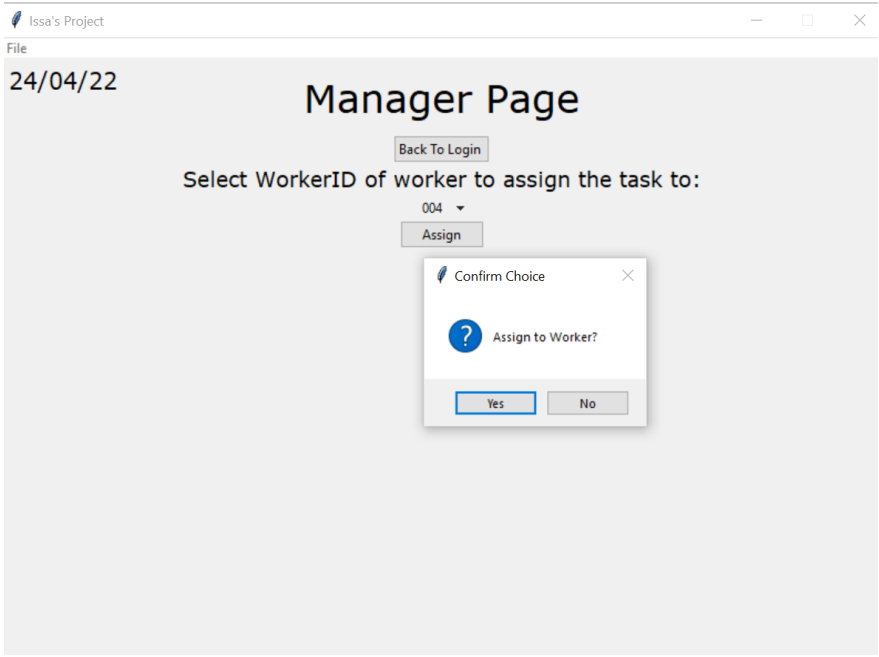


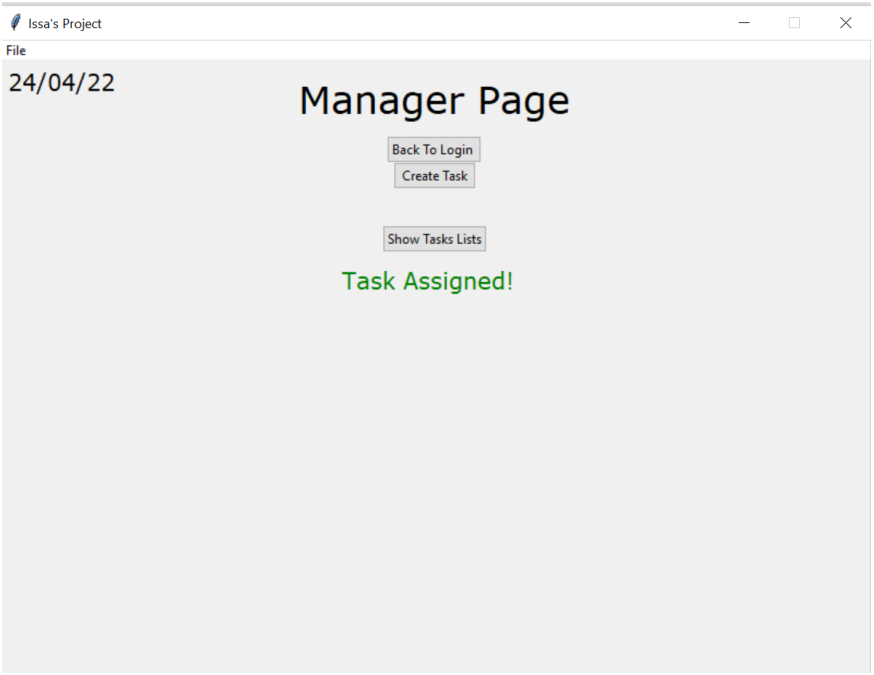


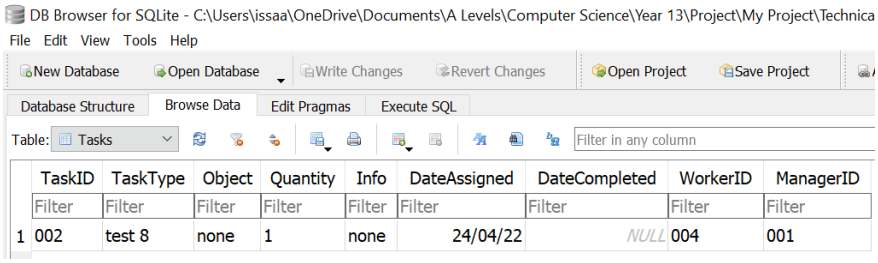


I logged into the Manager_1 page as normal then created a task. I entered in the details and pressed the "Assign Manually" button. From the drop down menu I selected Worker ID 004 and pressed assign task then confirmed it with the pop up window message. The page displayed the message saying that the task had been assigned. I opened my database using DB Browser and looked at the Tasks table and everything was there, however there was a logic error with the TaskID.


Issa's Project
File
24/04/22
Manager Page
Back To Login
Select WorkerID of worker to assign the task to:
004
Assign


Issa's Project
File
24/04/22
Manager Page
Back To Login
Select WorkerID of worker to assign the task to:
004
Assign
Confirm Choice
Assign to Worker?
Yes
No





Corrective Action(Test 8)

```
def ManageTask(self,AutomaticOrManual):    #method to
    self.ErrorLabel1.forget()
    self.TaskType = self.TaskTypeEntry.get()
    self.Object = self.ObjectEntry.get()
    self.Quantity = self.QuantityEntry.get()
    self.Info = self.InfoEntry.get()

    self.TaskTypeEntry.delete(0,20)              #cle
    self.ObjectEntry.delete(0,20)
    self.QuantityEntry.delete(0,20)
    self.InfoEntry.delete(0,100)

    if self.TaskType == "":
        self.ErrorLabel1.pack()
    else:
        self.ErrorLabel1.forget()

        self.ResetPage()

        if AutomaticOrManual == "A":
            WorkerID = self.Algorithm()
            self.AssignTask(WorkerID)
            self.SuccessLabel.place(x=312,y=185)
        else:
            self.CreateTaskButton.place_forget()
            self.ShowTasksListButton.place_forget()
            self.ManualChoice()

        self.IncrementCurrentTaskID()
```

```
def ManageTask(self,AutomaticOrManual):    #method to
    self.ErrorLabel1.forget()
    self.TaskType = self.TaskTypeEntry.get()
    self.Object = self.ObjectEntry.get()
    self.Quantity = self.QuantityEntry.get()
    self.Info = self.InfoEntry.get()

    self.TaskTypeEntry.delete(0,20)              #cle
    self.ObjectEntry.delete(0,20)
    self.QuantityEntry.delete(0,20)
    self.InfoEntry.delete(0,100)

    if self.TaskType == "":
        self.ErrorLabel1.pack()
    else:
        self.ErrorLabel1.forget()

        self.ResetPage()

        if AutomaticOrManual == "A":
            WorkerID = self.Algorithm()
            self.AssignTask(WorkerID)
            self.SuccessLabel.place(x=312,y=185)
        else:
            self.CreateTaskButton.place_forget()
            self.ShowTasksListButton.place_forget()
            self.ManualChoice()
```

```
    def StoreTask(self,WorkerID):                    #method which stores the current task into the databas
        NewPath = "Project Database.db"
        connection = sqlite3.connect(NewPath)
        data = connection.cursor()
        data.execute("""INSERT INTO Tasks(TaskID,TaskType,Object,Quantity,Info,DateAssigned,WorkerID,ManagerID)
Values(?,?,?,?,?,?,?,?)
""",(self.CurrentTaskID,self.TaskType,self.Object,self.Quantity,self.Info,str_date_today,WorkerID,CurrentManagerID,))
        connection.commit()

        self.MyTree.insert(parent='',index='end',text="",values=(self.CurrentTaskID,self.TaskType,self.Object,self.Qu
                                                    str_date_today,"None",WorkerID,CurrentManagerID))
```

The logic error was caused by the calling of the IncrementCurrentTaskID method within the ManageTask method because it was causing the Current Task ID to be incremented before the current task was stored in the database so each TaskID for every task would be stored as 1 more than it should be. To fix this I removed the call to the IncrementCurrentTaskID method from the ManageTask method and inserted it at the end of the StoreTask method so storing happens before incrementing as it is supposed to. I deleted the database and did the test again and it was all successful and the task was stored in the database with TaskID 001 as it should.

```
    def StoreTask(self,WorkerID):                          #method which stores the current task into the database in the task table
        NewPath = "Project Database.db"
        connection = sqlite3.connect(NewPath)
        data = connection.cursor()
        data.execute("""INSERT INTO Tasks(TaskID,TaskType,Object,Quantity,Info,DateAssigned,WorkerID,ManagerID)
Values(?,?,?,?,?,?,?,?)
""",(self.CurrentTaskID,self.TaskType,self.Object,self.Quantity,self.Info,str_date_today,WorkerID,CurrentManagerID,))
        connection.commit()

        self.MyTree.insert(parent='',index='end',text="",values=(self.CurrentTaskID,self.TaskType,self.Object,self.Quantity,self.Info,
                                                                 str_date_today,"None",WorkerID,CurrentManagerID))

        self.IncrementCurrentTaskID()
```

DB Browser for SQLite - C:\Users\issaa\OneDrive\Documents\A Levels\Computer Science\Year 13\Project\My Project\Technical S

File Edit View Tools Help

New Database | Open Database | Write Changes | Revert Changes | Open Project | Save Project | Att

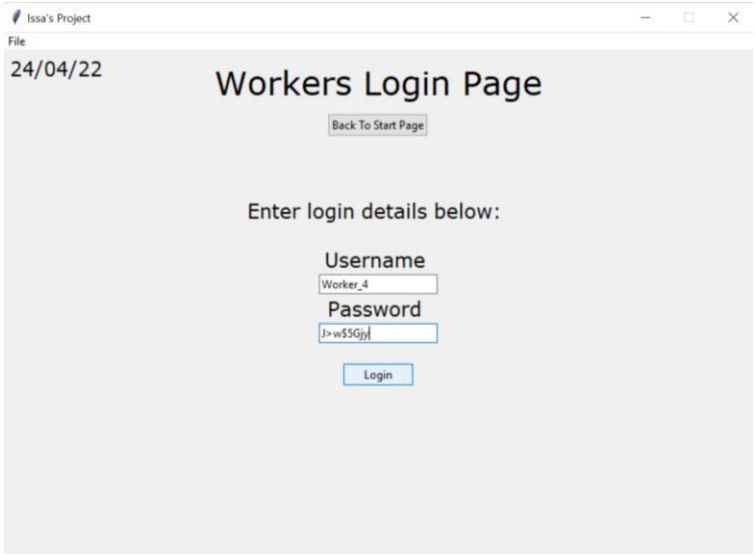
Database Structure | Browse Data | Edit Pragmas | Execute SQL

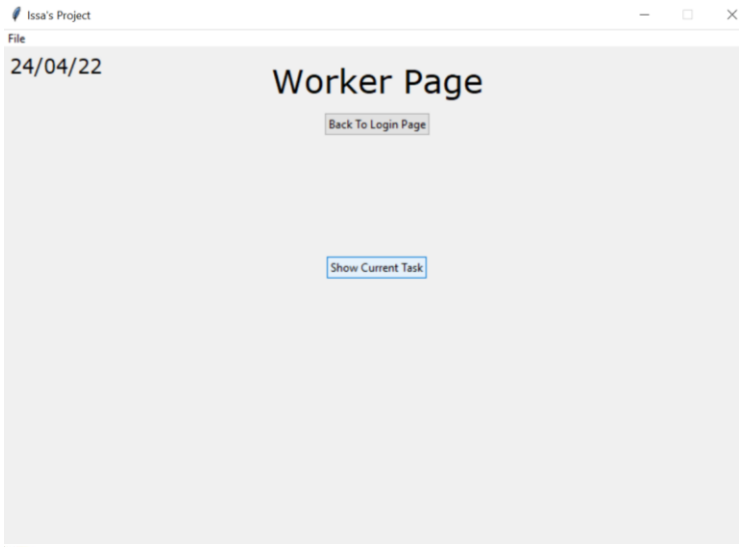
Table: Tasks | Filter in any column

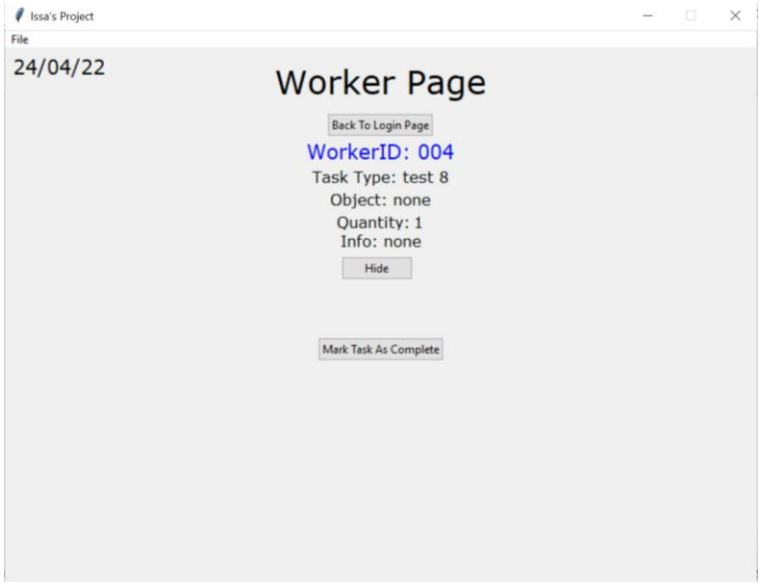
	TaskID	TaskType	Object	Quantity	Info	DateAssigned	DateCompleted	WorkerID	ManagerID
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	001	test 8	none	1	none	24/04/22	*NULL*	004	001

Test 9

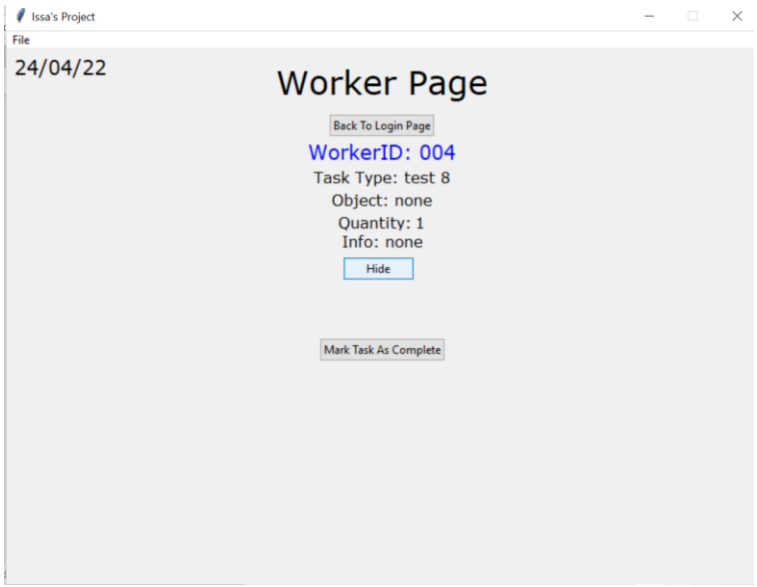
I logged in to the Worker_4 page and pressed the Show Current Task button. This displayed the details from the task that I had assigned in test 8.

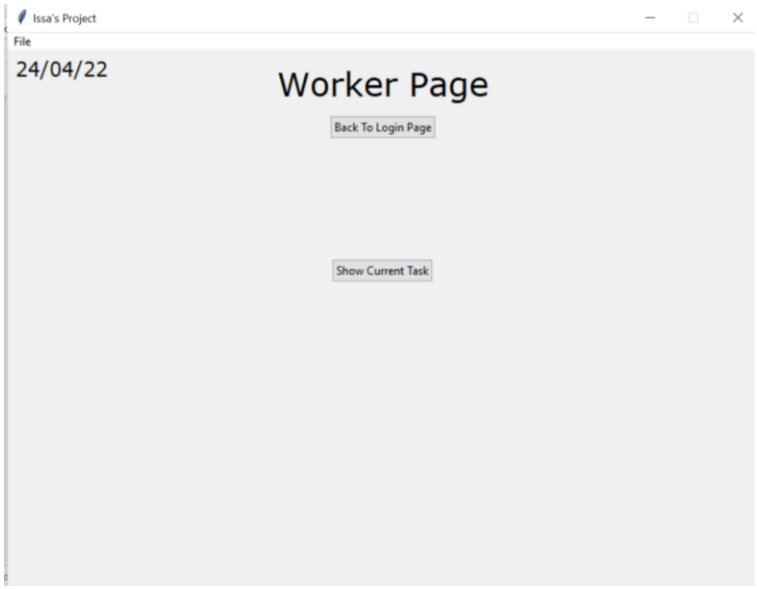






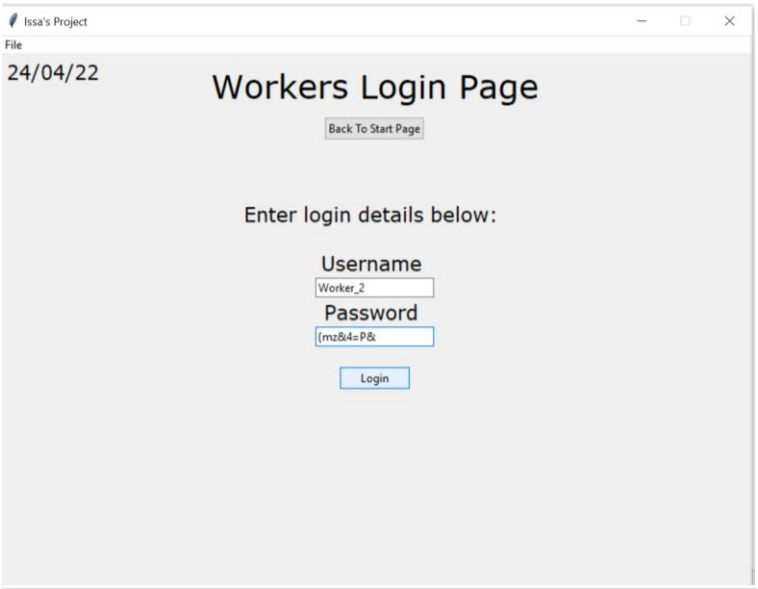
Test 10

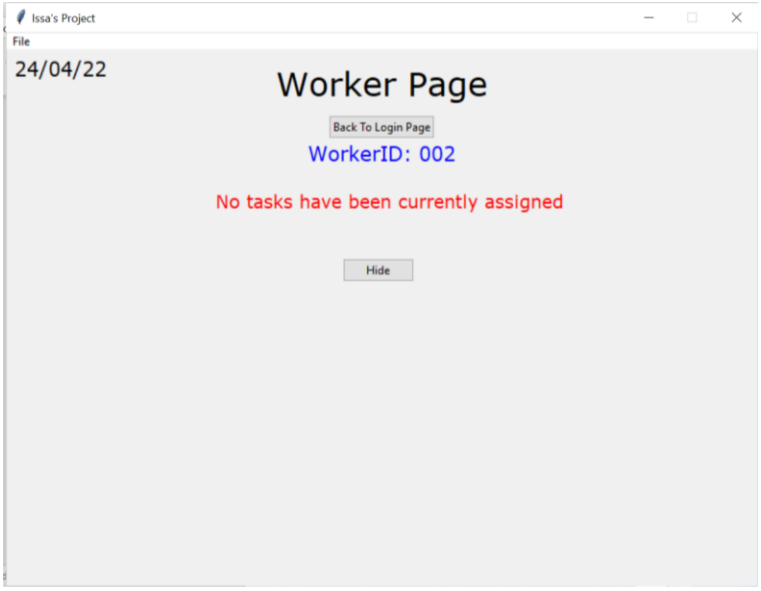




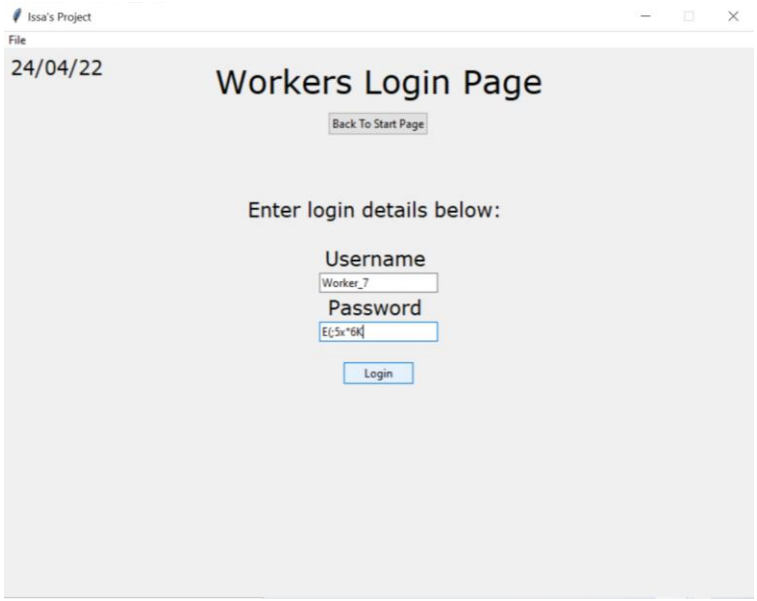
When the Hide Task Button is pressed, the task details and the Hid Task button itself disappear. The Show Current Task button appears in it's place.

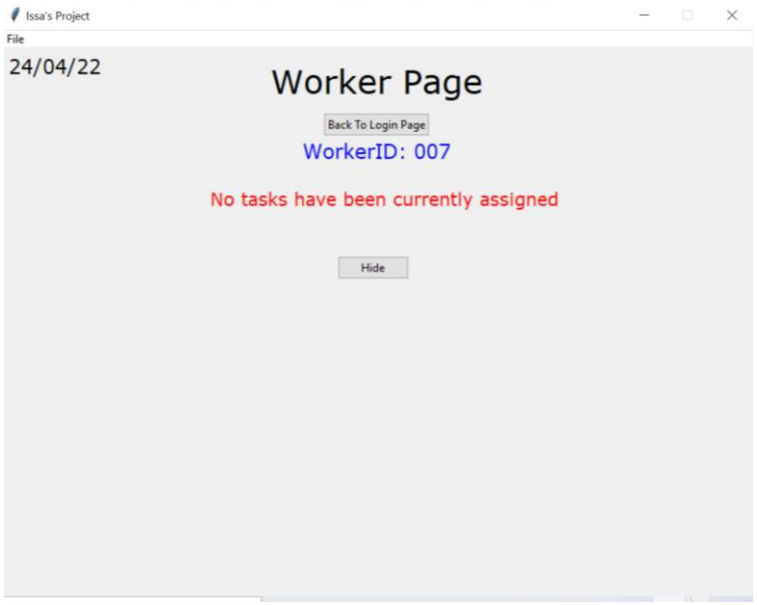
Test 11



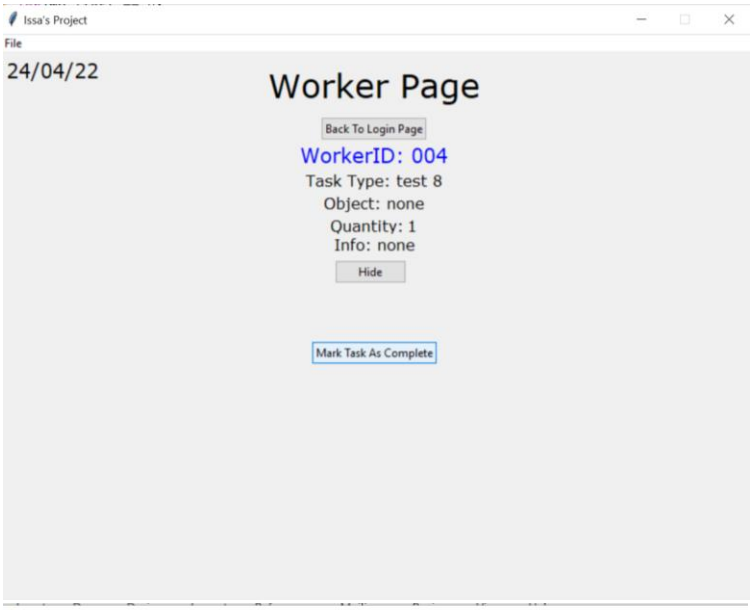


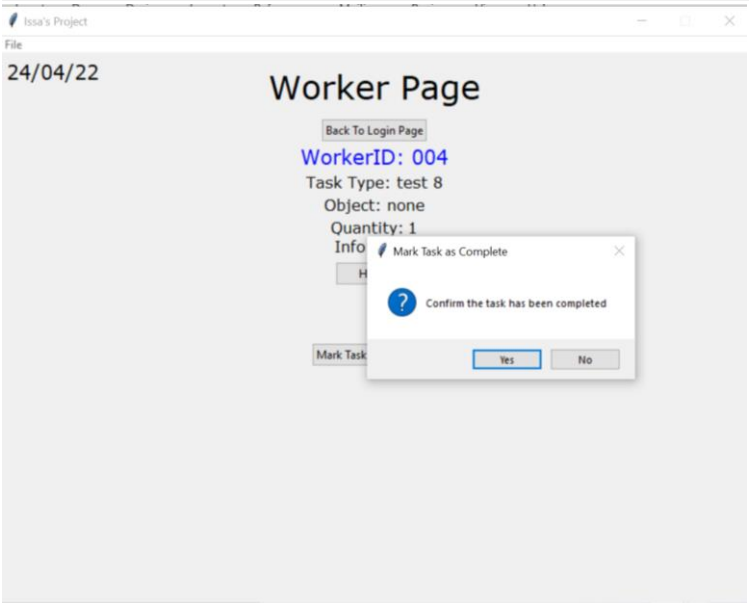
I logged in to the Worker_2 page and pressed the Show Current Task button and only a message stating that there are no current tasks came up. I repeated the test with Worker_7 and it had the same outcome.

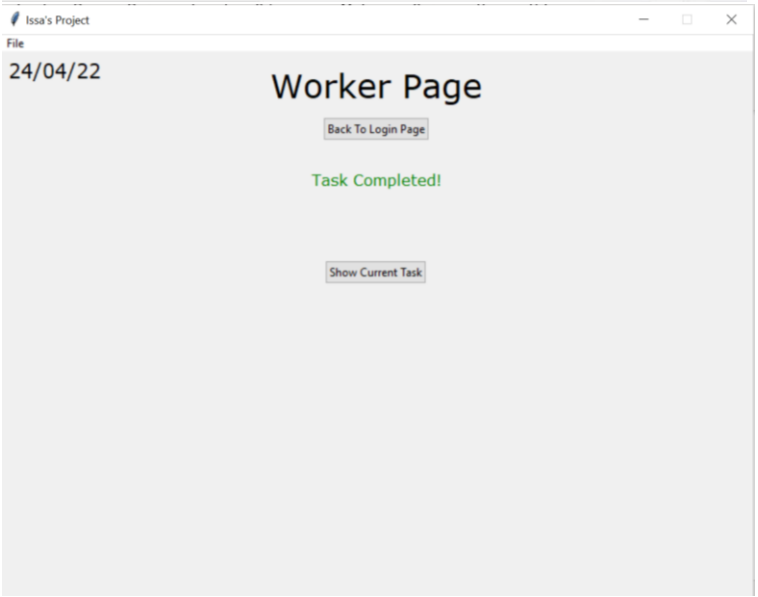

Issa's Project
File
24/04/22
Workers Login Page
Back To Start Page
Enter login details below:
Username
Worker_7
Password
E(;5x*6K
Login


Issa's Project
File
24/04/22
Worker Page
Back To Login Page
WorkerID: 007
No tasks have been currently assigned
Hide

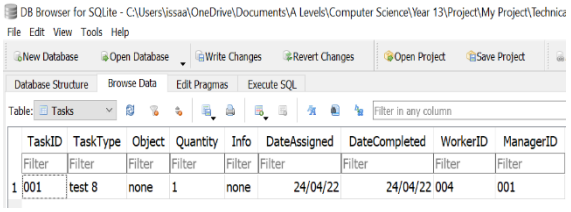
Test 12



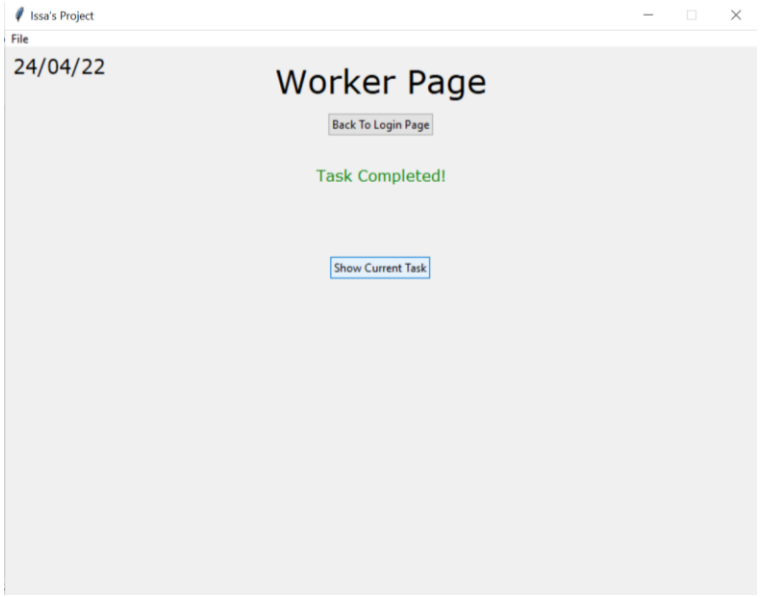


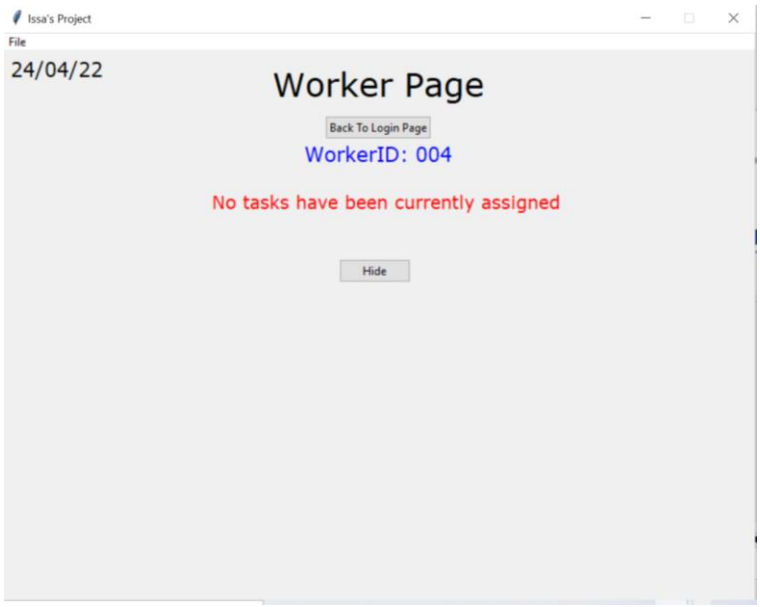


I logged into to the Worker_4 page as normal, pressed the Show Current Task button and pressed the Mark Task As Complete button. I confirmed that I wanted to complete it and the message appeared on the screen. I check DB Browser and today's date had been inserted into the DateCompleted field.



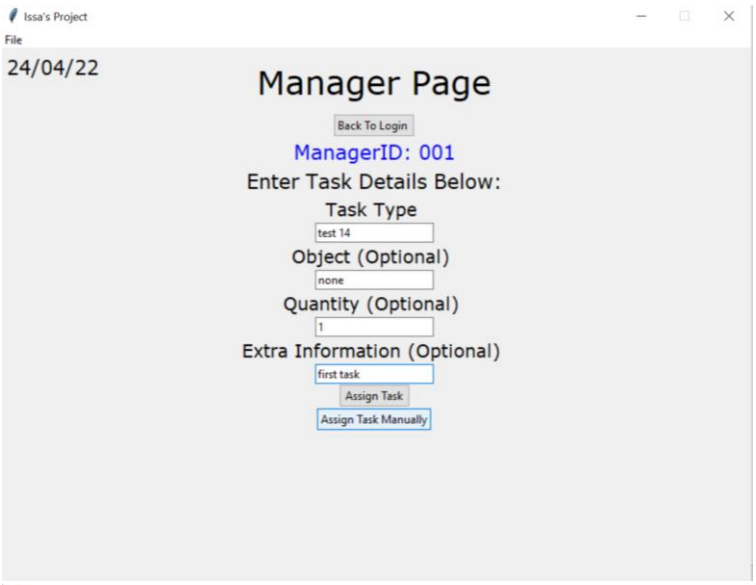
Test 13

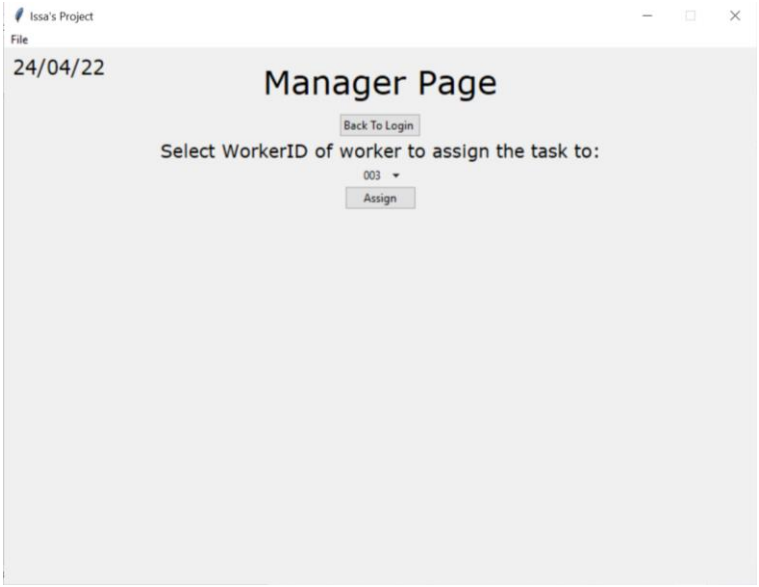


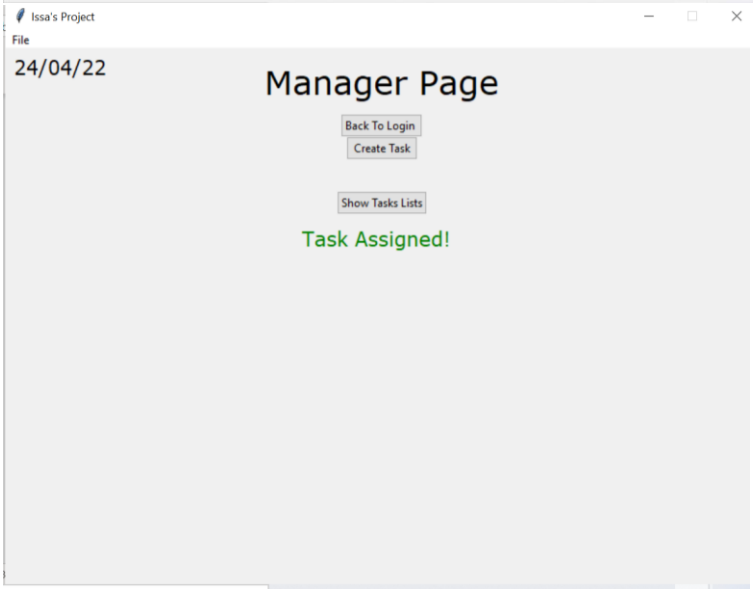


I logged into the Worker_4 page as normal and pressed the Show Current Task button. The task didn't show up and the no tasks message showed up.

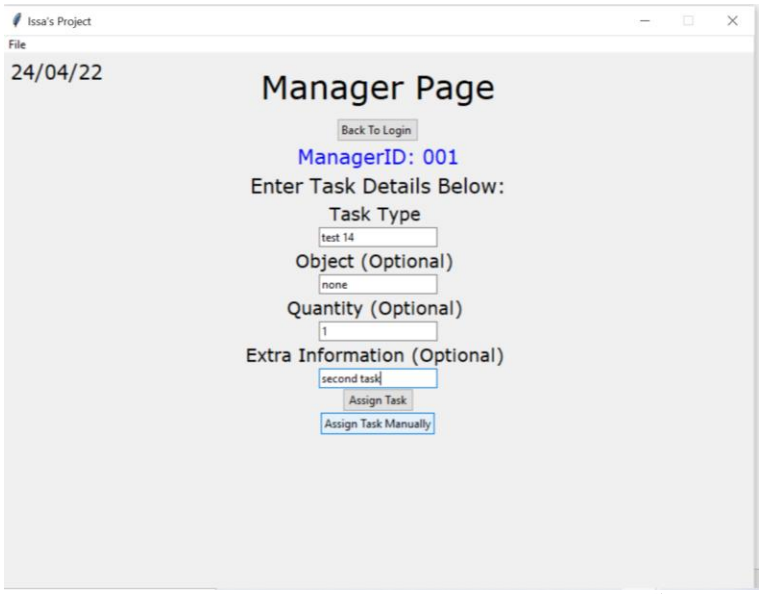
Test 14

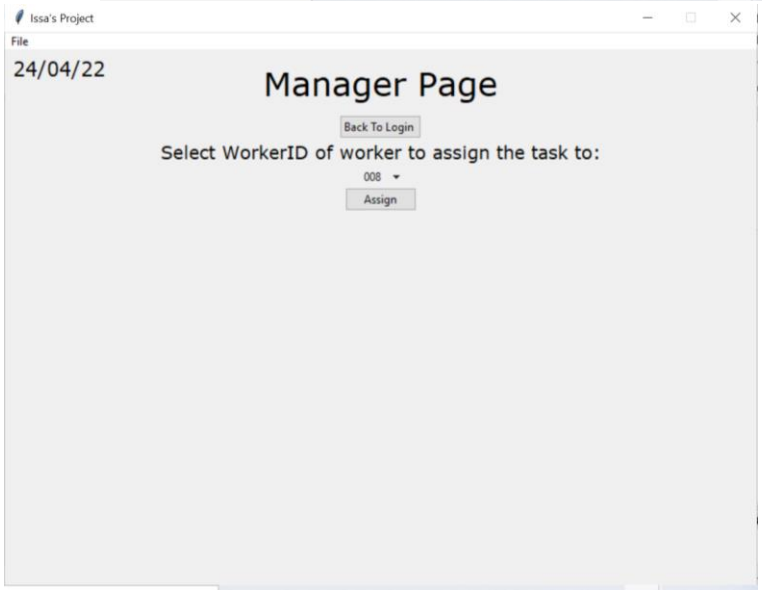


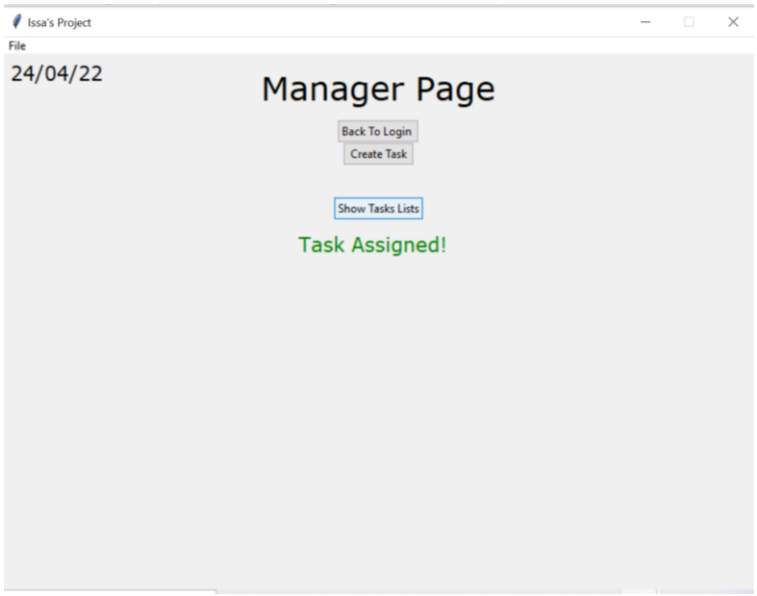


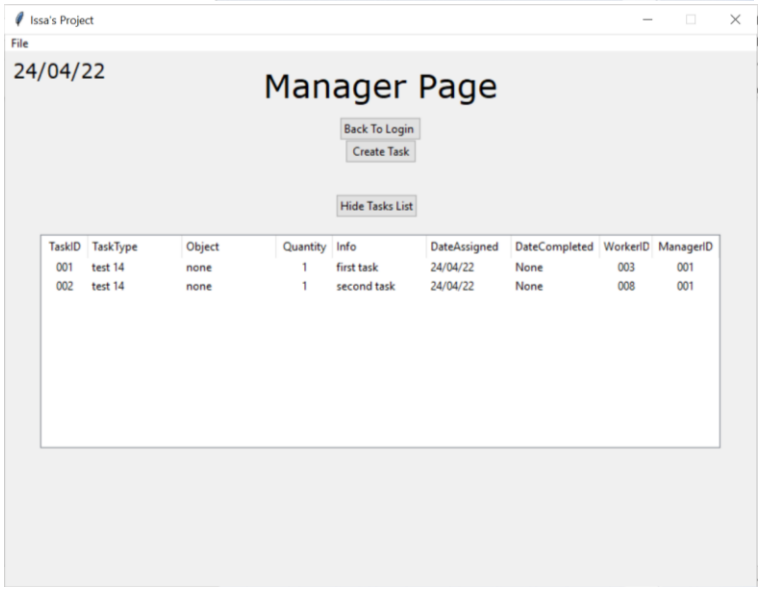


I logged into a Manager Page and assigned a task to Worker ID 003 and one to Worker ID 008. I pressed the Show Tasks List button and the treeview appeared with the two tasks correctly in it.


Issa's Project
File
24/04/22
Manager Page
Back To Login
ManagerID: 001
Enter Task Details Below:
Task Type
test 14
Object (Optional)
none
Quantity (Optional)
1
Extra Information (Optional)
second task
Assign Task
Assign Task Manually


Issa's Project
File
24/04/22
Manager Page
Back To Login
Select WorkerID of worker to assign the task to:
008
Assign

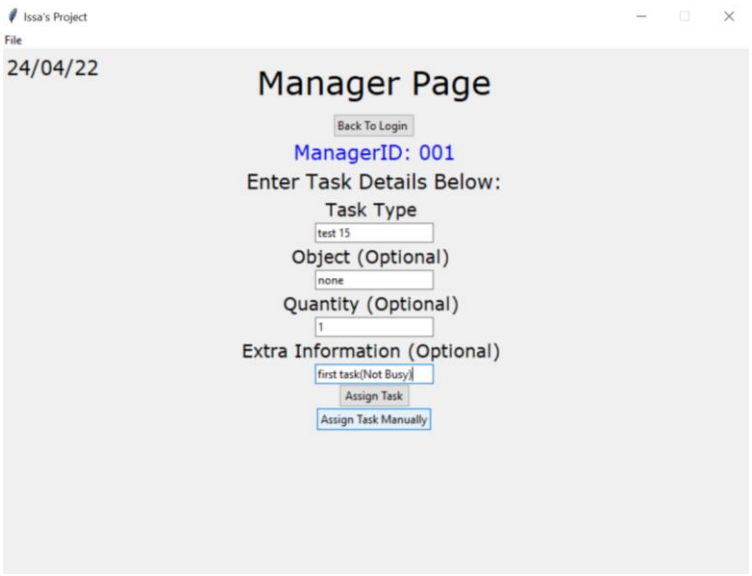

Issa's Project
File
24/04/22
Manager Page
Back To Login
Create Task
Show Tasks Lists
Task Assigned!


Issa's Project
File
24/04/22
Manager Page
Back To Login
Create Task
Hide Tasks List
TaskID TaskType Object Quantity Info DateAssigned DateCompleted WorkerID ManagerID
001 test 14 none 1 first task 24/04/22 None 003 001
002 test 14 none 1 second task 24/04/22 None 008 001

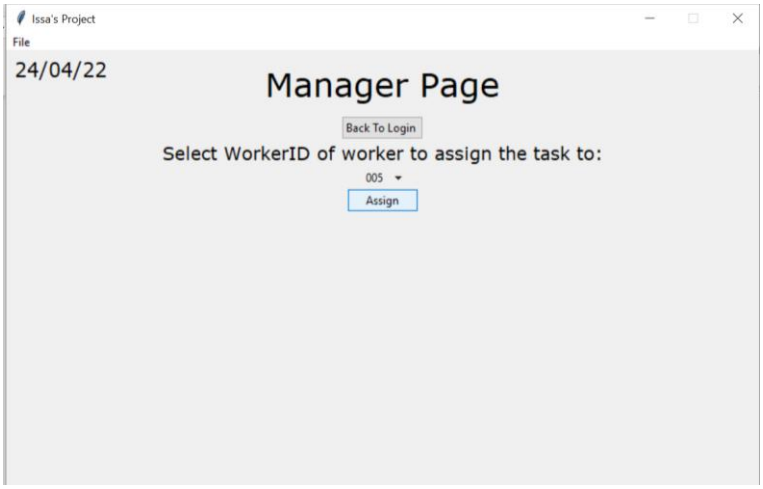
Test 15

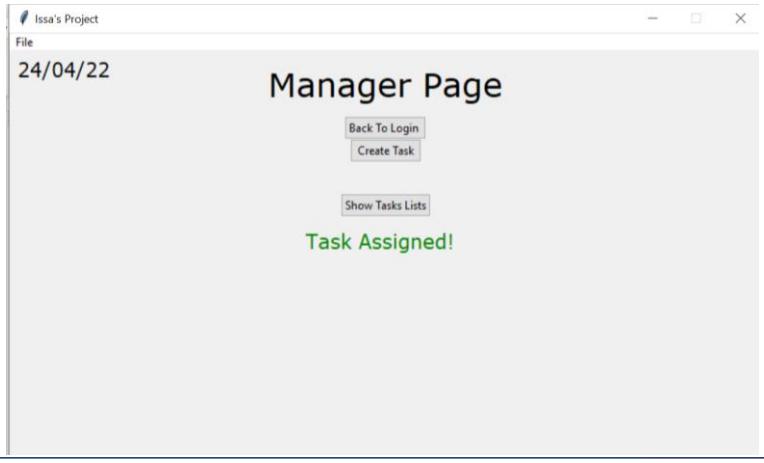
```
def CheckIfBusy(self,WorkerID):
    try:
        NewPath = "Project Database.db"
        connection = sqlite3.connect(NewPath)
        data = connection.cursor()
        data.execute("""SELECT TaskID FROM Tasks WHERE WorkerID LIKE ? AND DateCompleted IS NULL ORDER BY TaskID DESC LIMIT 1"""
                     ,(WorkerID,))
        ID = data.fetchall()
        if ID[0][0] == "":
            self.WorkerBusy = False
        else:
            self.WorkerBusy = True
    except:
        self.WorkerBusy = False
    return self.WorkerBusy


    def AssignTask(self,WorkerID):
        self.WorkerBusy = self.CheckIfBusy(WorkerID)
        if self.WorkerBusy == True:
            self.AddToQueue(self.CurrentTaskID)
            print("Busy")
        else:
            print("Not Busy")
        self.StoreTask(WorkerID)
```

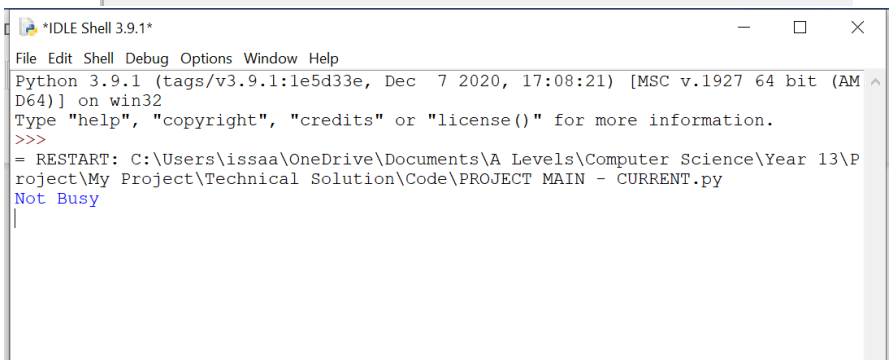
The CheckIfBusy method is being tested in this test. The AssignTask calls the CheckIfBusy method to see if the worker for the current task is already busy with another task. For this test I added in print statements to print “Busy” if the worker is busy and “Not Busy” if they aren’t.



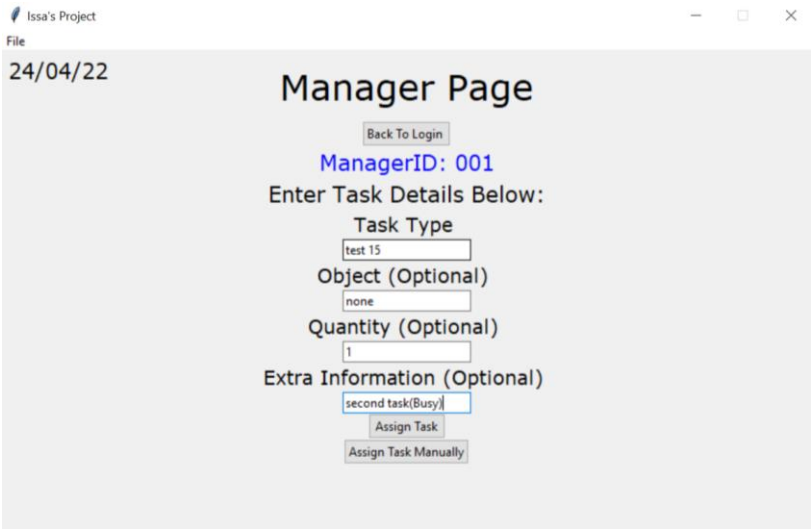
I assigned the first task with these details to Worker ID 005.



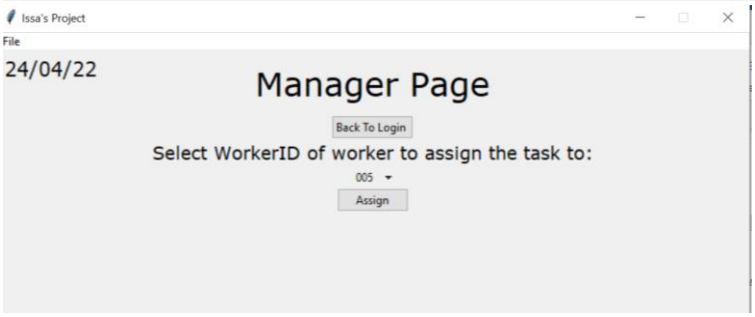


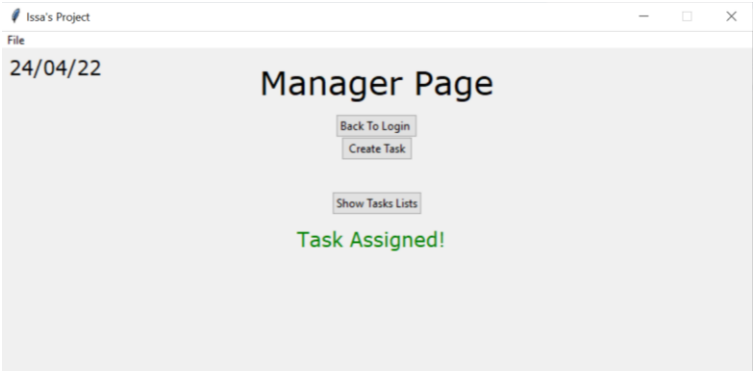


The output on the shell was "Not Busy" because at the time of assigning this task, Worker 005 had no other tasks currently assigned to them so they were not busy.



I assigned the second task with these details to Worker ID 005 aswell.

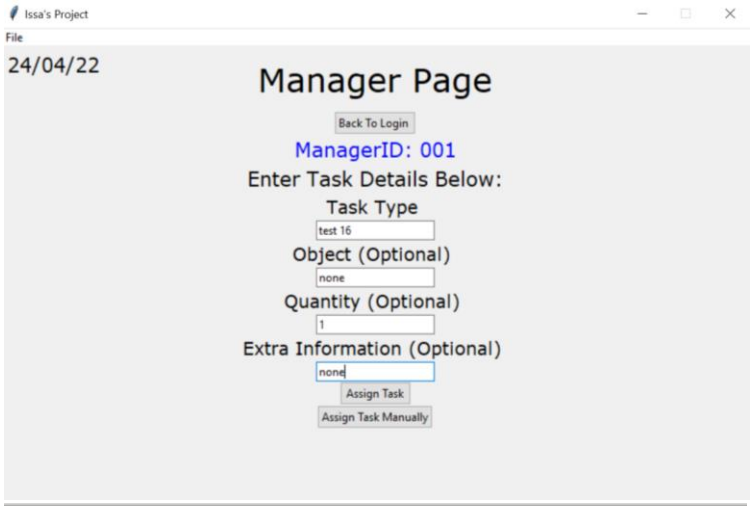


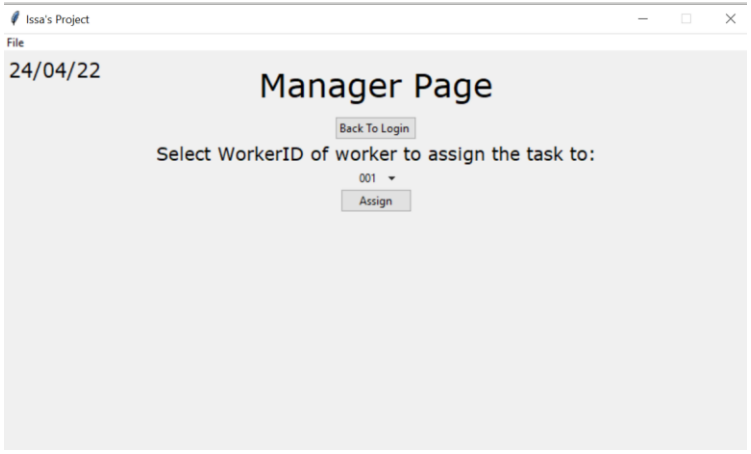


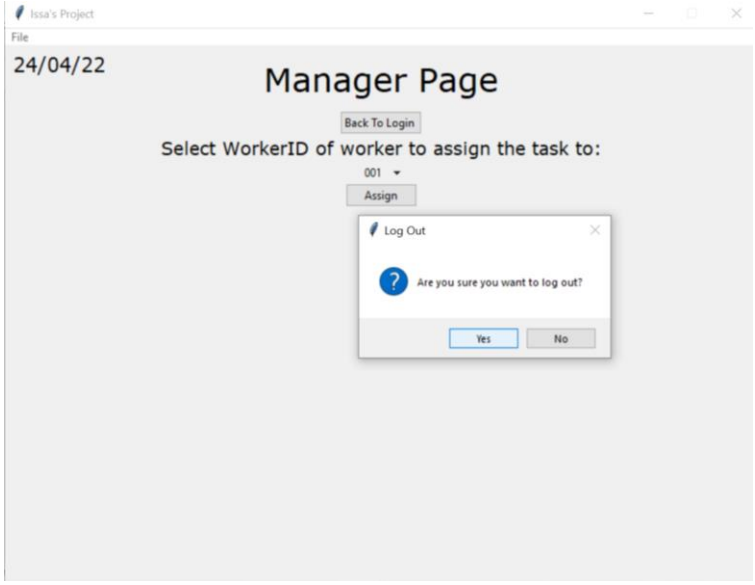

```
>>>
= RESTART: C:\Users\issaa\OneDrive\Documents\A Levels\Computer Science\Year 13\P
roject\My Project\Technical Solution\Code\PROJECT MAIN - CURRENT.py
Busy
```

The output on the shell was “Busy” because at the time of assigning this task, Worker 005 had the first task assigned to them so they were busy.

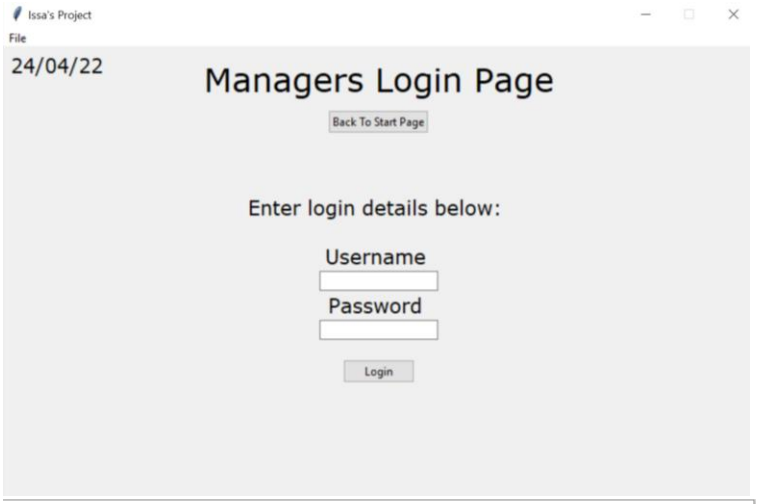
Test 16

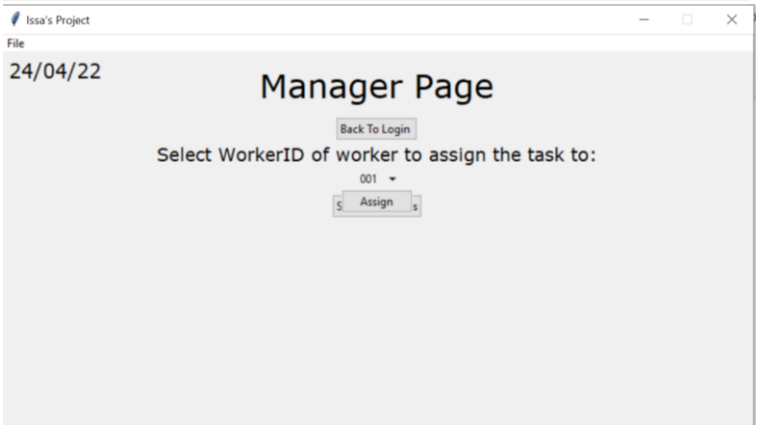






I created this task then pressed the Assign Manually button to bring up the drop menu screen. Then I logged out of the page, without assigning the task.





I was brought back to the login page as expected. Then when I logged back in the page appeared wrongly. It hadn't been reset properly

Corrective Action (Test 16)

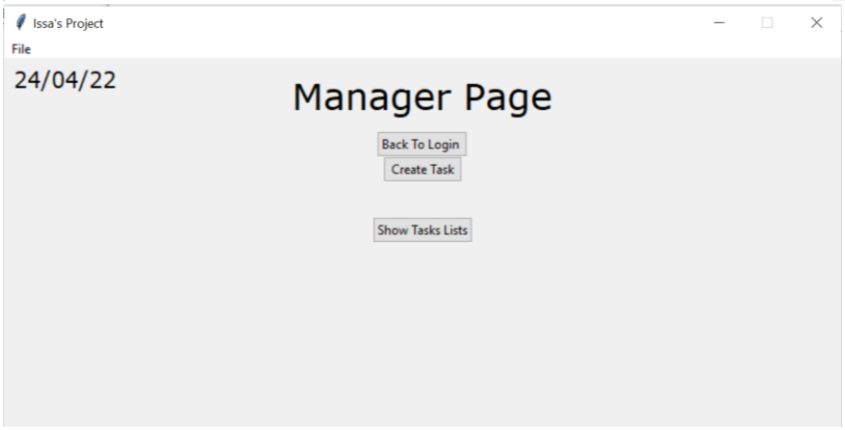
```
    def ResetPage(self):
        WidgetList = [self.EnterTaskDetailsLabel,self.TaskTypeLabel,self.ObjectLabel,self.QuantityLabel,self.InfoLabel,
                 self.AutomaticButton,self.ManualButton,
                 self.TaskTypeEntry,self.ObjectEntry,self.QuantityEntry,self.InfoEntry,
                 self.SuccessLabel,self.ErrorLabel1,self.MyTree]
        for x in WidgetList:
             x.forget()

        self.HideTasksListButton.place_forget()
        self.SuccessLabel.place_forget()
        self.CreateTaskButton.place(x=362,y=92)
        self.ShowTasksListButton.place(x=352,y=150)
```

The error is in the ResetPage method as it doesn't remove the drop menu, assign button or the Select Worker ID label.

```
    def ResetPage(self):
        WidgetList = [self.EnterTaskDetailsLabel,self.TaskTypeLabel,self.ObjectLabel,self.QuantityLabel,self.InfoLabel,
                self.AutomaticButton,self.ManualButton,
                self.TaskTypeEntry,self.ObjectEntry,self.QuantityEntry,self.InfoEntry,
                self.SuccessLabel,self.ErrorLabel1,self.MyTree]
        for x in WidgetList:
            x.forget()

        self.HideTasksListButton.place_forget()
        self.SuccessLabel.place_forget()
        self.CreateTaskButton.place(x=362,y=92)
        self.ShowTasksListButton.place(x=352,y=150)
        try:
            self.IDLabel.forget()
            self.ConfirmButton.forget()
            self.dropmenu.forget()
            self.SelectWorkerIDLabel.forget()
        except:
            return None
```

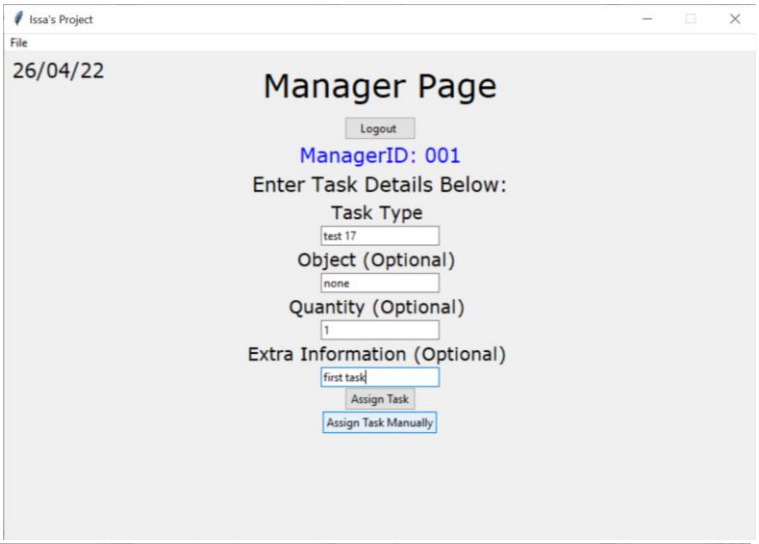
I amended the ResetPage method to remove the extra widgets. I had to add these lines in using exception handling because the code was crash as the ResetPage method was being called and it was trying to remove these widgets before they had been defined so to prevent the code from crashing I needed to use try/except.

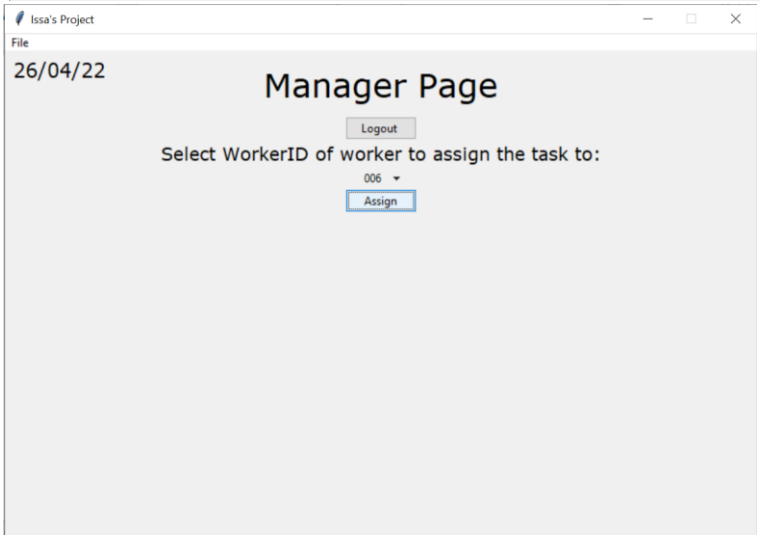
```
>>>
= RESTART: C:\Users\issaa\OneDrive\Documents\A Levels\Computer Science\Year 13\P
roject\My Project\Technical Solution\Code\PROJECT MAIN - CURRENT.py
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Windows.old\Users\issaa\AppData\Local\Programs\Python\Python39\lib\tk
inter\__init__.py", line 1884, in __call__
    return self.func(*args)
  File "C:\Users\issaa\OneDrive\Documents\A Levels\Computer Science\Year 13\Proj
ect\My Project\Technical Solution\Code\PROJECT MAIN - CURRENT.py", line 458, in
<lambda>
    self.ManualButton = ttk.Button(self, text = "Assign Task Manually", command
=lambda: self.ManageTask("M"))
  File "C:\Users\issaa\OneDrive\Documents\A Levels\Computer Science\Year 13\Proj
ect\My Project\Technical Solution\Code\PROJECT MAIN - CURRENT.py", line 573, in
ManageTask
    self.ResetPage()
  File "C:\Users\issaa\OneDrive\Documents\A Levels\Computer Science\Year 13\Proj
ect\My Project\Technical Solution\Code\PROJECT MAIN - CURRENT.py", line 476, in
ResetPage
    self.dropmenu.forget()
AttributeError: 'ManagerPage' object has no attribute 'dropmenu'
```



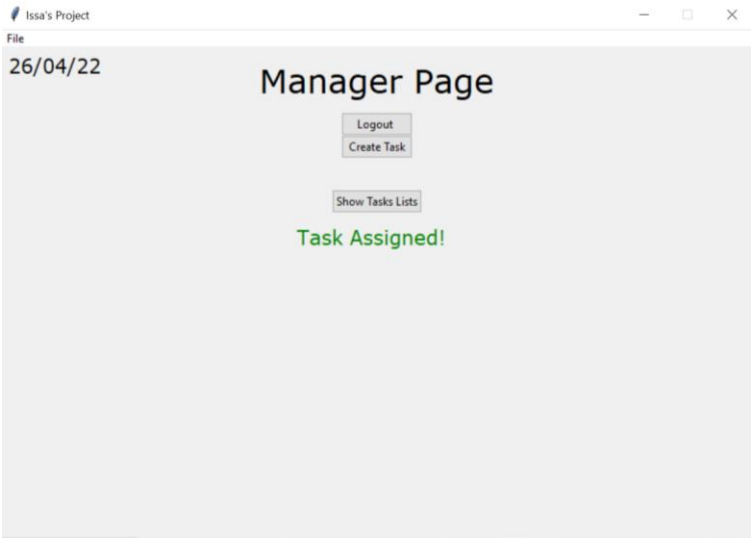
After fixing the ResetPage method I redid the test and it was successful. The Manager Page was reset to how it should be when I logged back in.

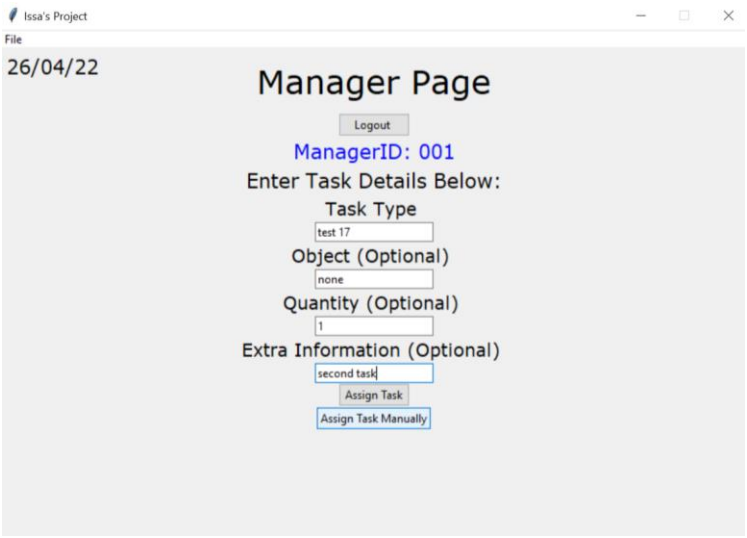
Test 17

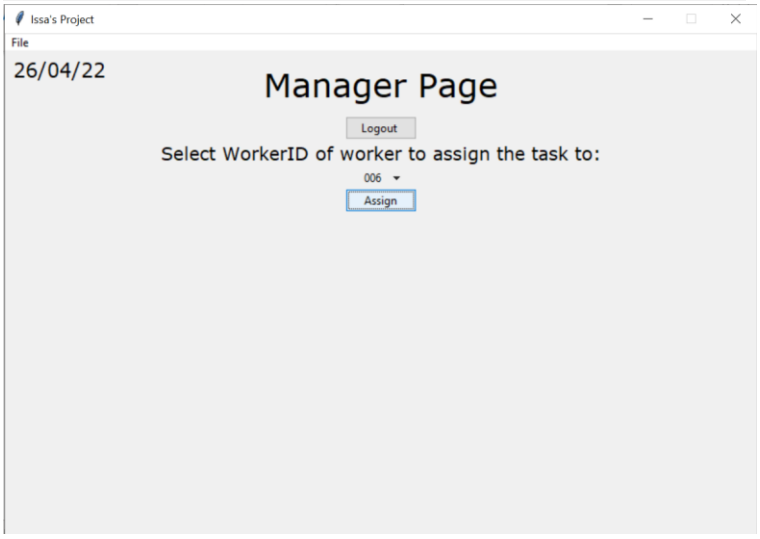


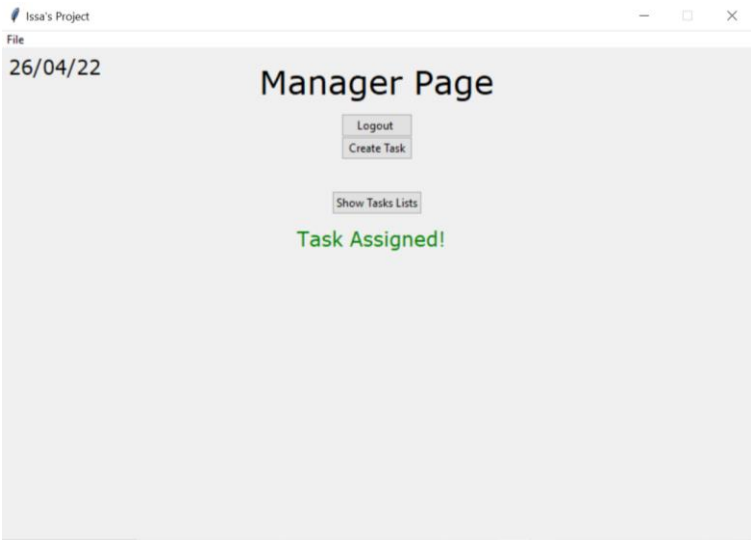


I logged into the Manager_1 Page and created the first task with these details. Then I assigned it to Worker ID 006

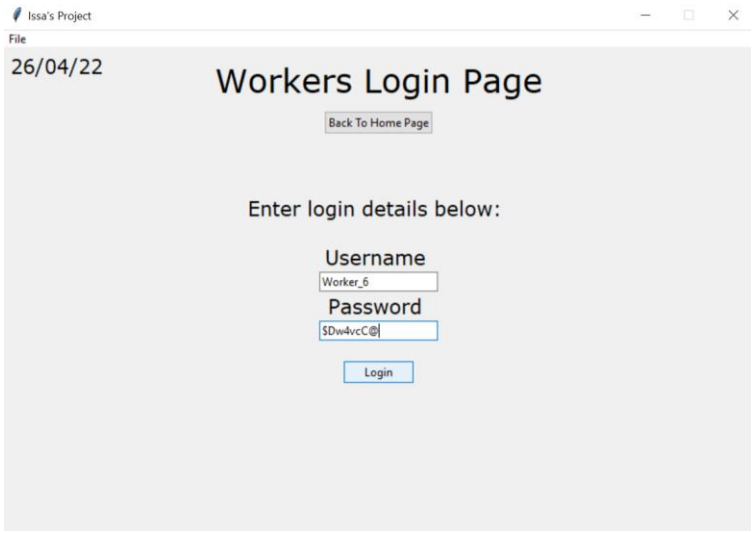


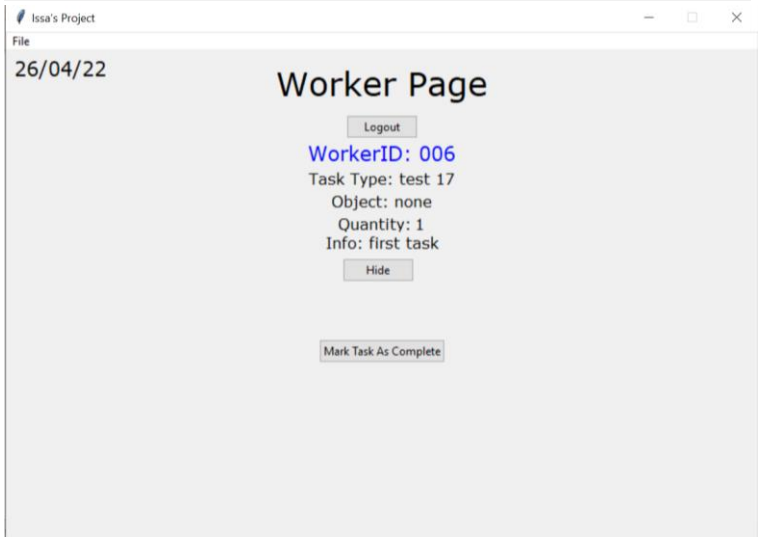




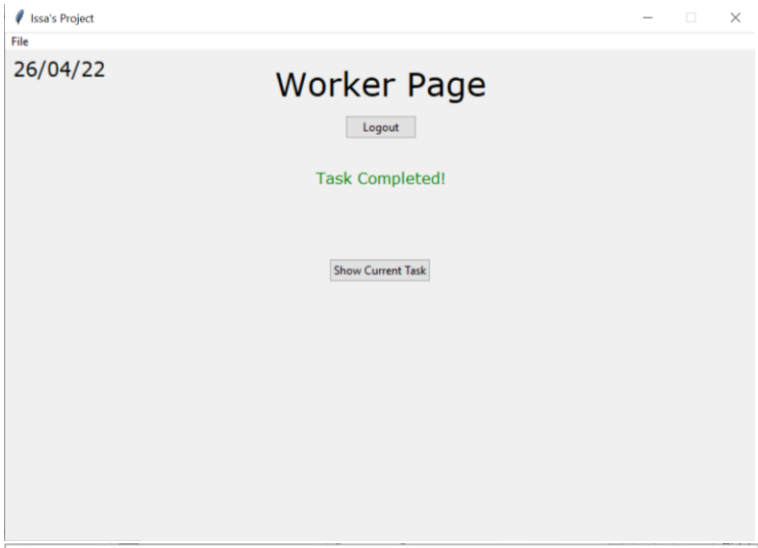


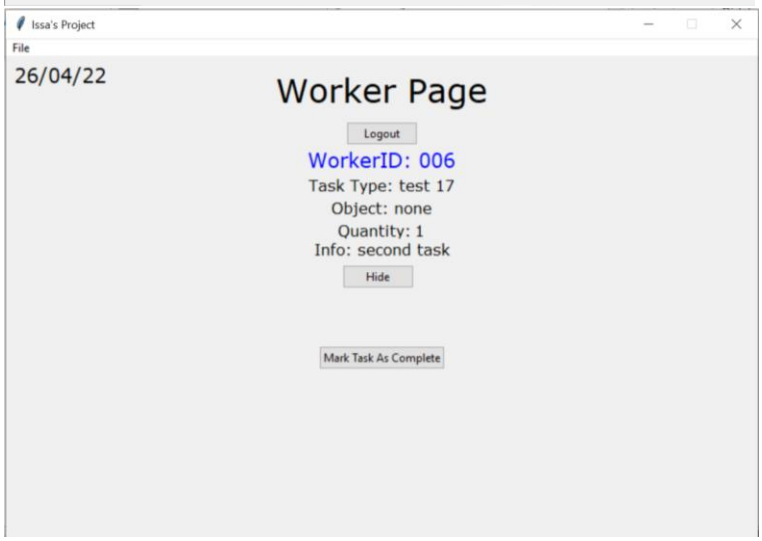
Still in the Manager_1 Page, I created the second task with these details. Then I assigned it to Worker ID 006 again.





I logged into Worker_6's page and pressed the Show Current Task button. The first task was displayed.



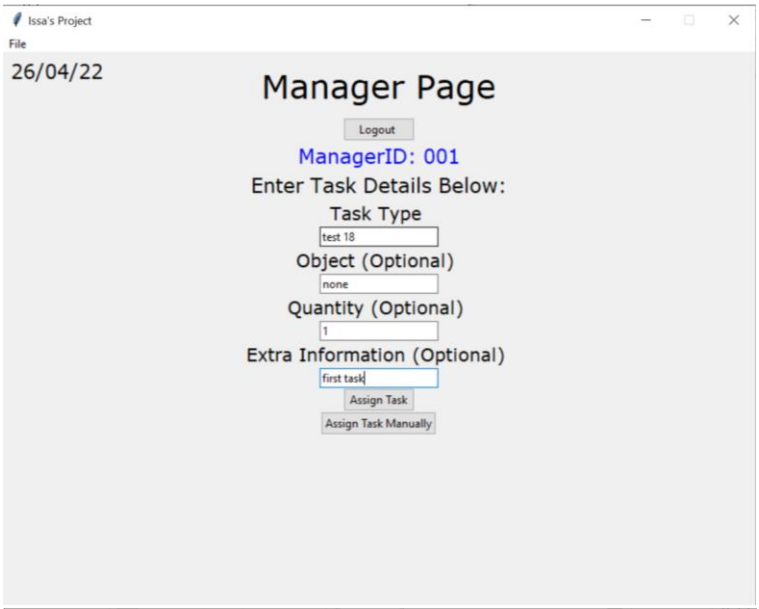


I marked the first task as complete and pressed the Show Current Task button again. The second task was displayed as expected.

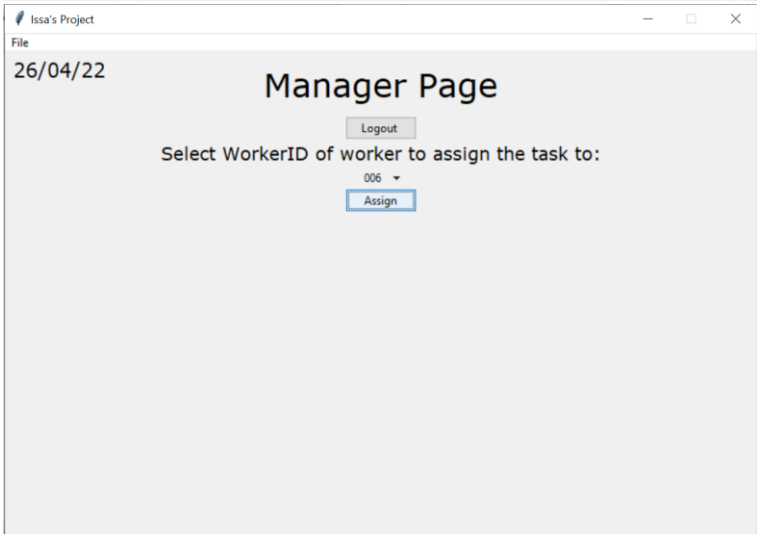
Test 18

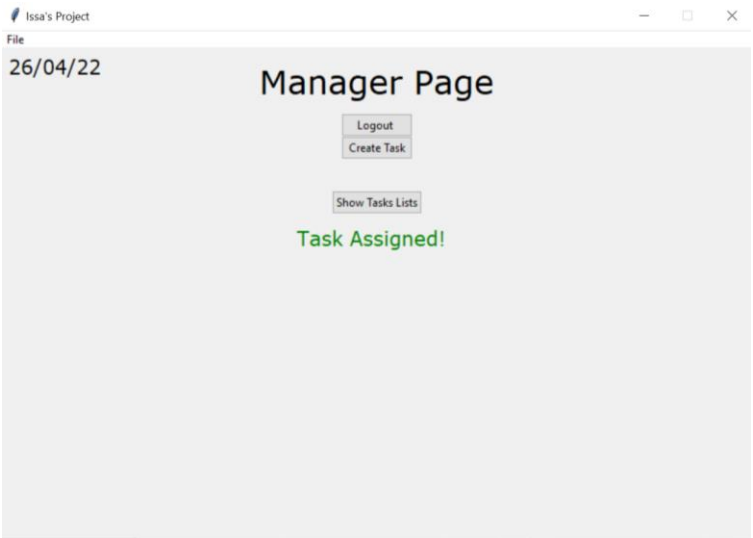
```
def AssignTask(self,WorkerID):
    self.WorkerBusy = self.CheckIfBusy(WorkerID)
    if self.WorkerBusy == True:
        self.AddToQueue(self.CurrentTaskID)
        print(self.Queue)
    else:
        print(self.Queue)
    self.StoreTask(WorkerID)
```

In the AssignTask method the CheckIfBusy method is called. If the worker is busy the TaskID of the current task gets added to the queue. If not nothing happens and then afterwards the task is stored in the Tasks table. For the test I have added in print commands to print the queue at both outcomes so we can see what it looks like at each stage.



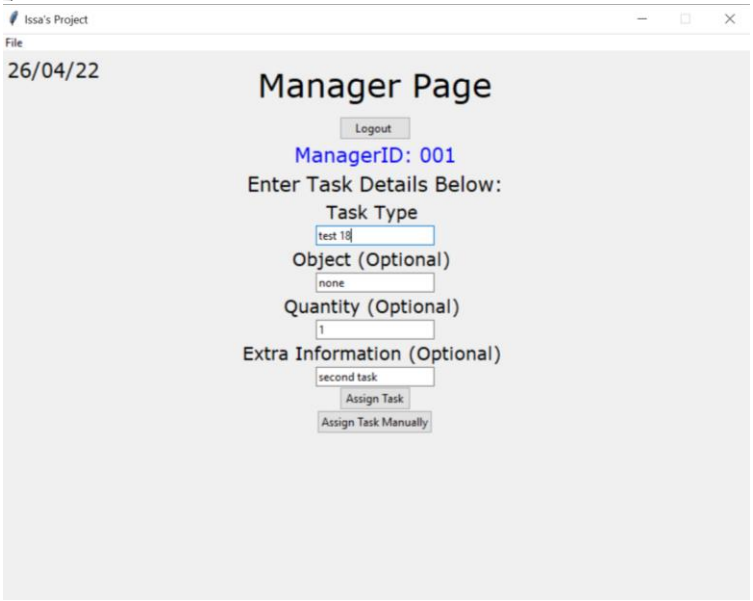
I logged into the Manager_1 Page and created the first task with these details. Then I assigned it to Worker ID 006



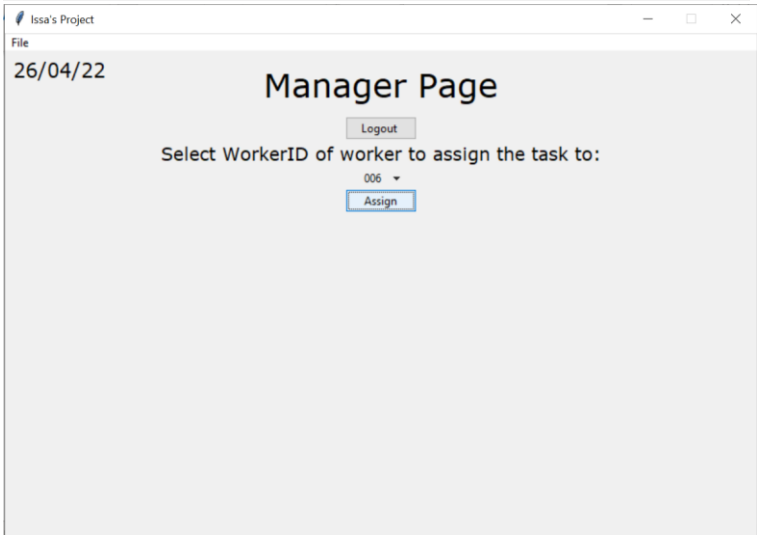


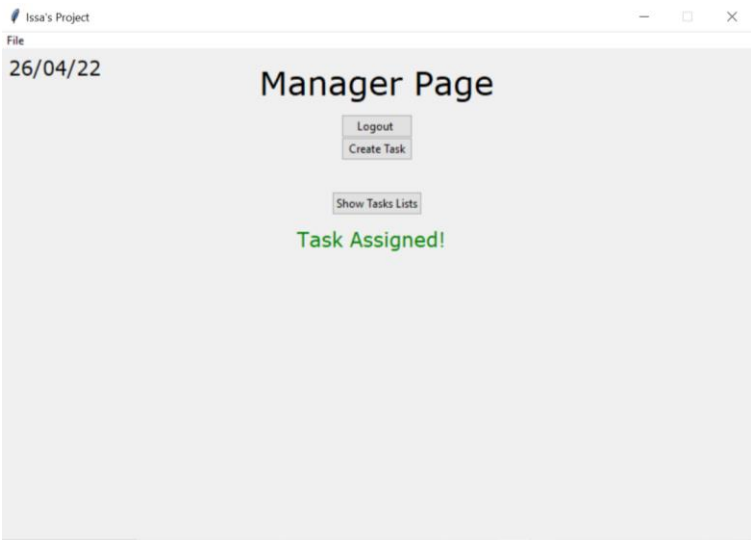

```
= RESTART: C:\Users\issaa\OneDrive\Documents\A Levels\Computer Science\Year 13\P
roject\My Project\Technical Solution\Code\PROJECT MAIN - CURRENT.py
[]
```

Here the empty queue was printed as Worker 006 is currently not busy



Still in the Manager_1 Page, I created the second task with these details. Then I assigned it to Worker ID 006 again.






```
>>>
= RESTART: C:\Users\issaa\OneDrive\Documents\A Levels\Computer Science\Year 13\P
roject\My Project\Technical Solution\Code\PROJECT MAIN - CURRENT.py
[]
['002']
```

Here the queue was printed again, this time containing TaskID '002' which is the current task and it is in the queue because the worker is still busy with task '001' so this is in the queue, as shown.

Objective Testing

Below is a copy of my 5 objectives from the Analysis section. These are being tested here.

1.The code utilises a proficient algorithm which analyses any relevant task and correctly assigns it to the most suitable person

2.The Graphical User Interface is clear and easy to use whilst also being complex enough to be used professionally

3.Contains a user login system where each user has their own personal login details to access their own page which looks and acts uniquely for each user

4.Stores user login details within a table in SQL

5.Stores information about the employees within a table in SQL which is used by the algorithm

These tests involve End-User testing, where the potential users of the system have tested the final finished code themselves in order to determine whether the objectives have been met.

The code requires that the worker details must be entered in before any other part of the code can be used. Objective 5 is about entering the worker details and having them stored in the database so this objective was done before the others. The tests were NOT carried out in order based on the Objective number.

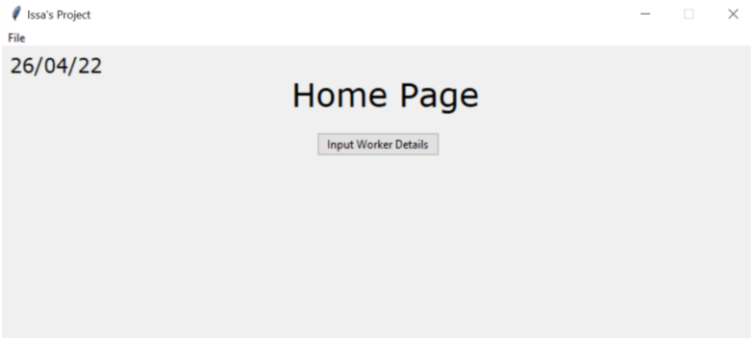
Objective Number	Test Description	Test Data	Expected outcome	Actual outcome	Pass?	Corrective action	Refe-rence	Comments
5	The code is run and details about the workers are entered in.	Normal (Worker Details)	All details get entered and are stored in the Details table in the database.	All details are easily entered and are found to be stored in the Details table in the database.	Yes	None	Test 1	The code was run and the manager entered each worker's details one by one. Then the database was inspected using DB Browser and was found to have all the details correctly stored.
3	A manager logs in and tests out their page. Then they assign a task to each different worker. Each worker logs in and tests out their page and views the current task to confirm that their	Normal Erroneous (Login Details)	Each user's login details should allow them to login to their own pages. Each worker should receive a different task on their page.	Every user was able to correctly login to their own page and each worker had a different task appear on their page.	Yes	None	Test 2	The manager successfully logged in. The manager manually assigned a different task to each worker. Every worker logged in and viewed their current task and all had a different one, showing that their pages are unique for them.

	own page is different from the others.							
4	Testing the create database method which creates the tables "Workers" and "Managers" and inserts the user login details into them		The tables are created and the correct login details are inserted into them	The tables are created and the correct login details are inserted into them	Yes	None	Test 3	The create_database method creates all the tables for the database including "Managers" and "Workers". It also inserts the data into them. Using DB Browser, the tables were checked to ensure they are correct and they were.
2	One manager and one worker completely test out everything within the program		Everything in the interface works as expected with no errors, bugs or crashes in the code.	Everything in the interface worked as expected with no errors, bugs or crashes in the code.	Yes	None	Test 4	Manager_3 and Worker_7 tested everything out in the interface and found it all worked.
1	Creating a task then automatically assigning sing the algorithm. Then get a manager to assign the same task manually. Check that the Worker chosen by the manager is the same as the one chosen by the algorithm	Normal (Task Details)	The Worker chosen by the algorithm matches with the one chosen by the manager.	The algorithm and manager chose the same worker for the task.	Yes	None	Test 5	The algorithm assigned the task to Worker ID 008. The manager also chose Worker ID 008 because that worker has been at the company the longest so has the most experience and is the most trusted with an important task.

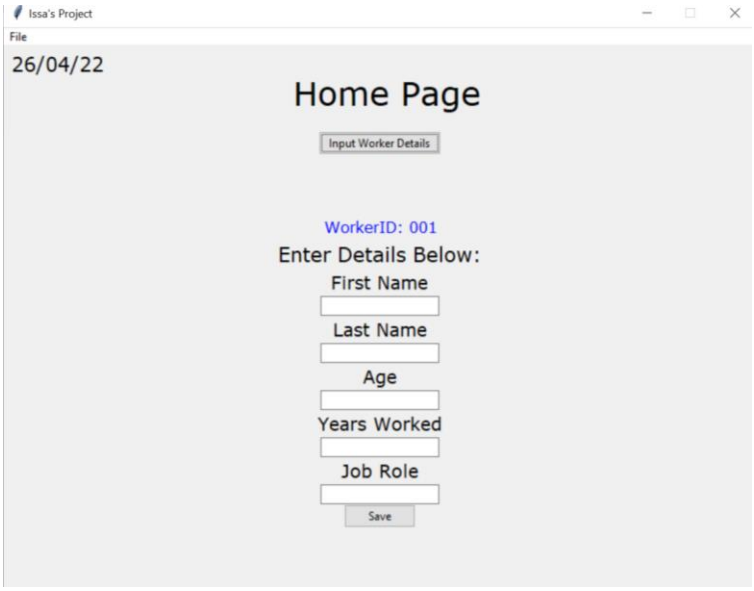
Test 1

The GetDetails method calls the CheckDetails method to see if the Details table within the database is full. If it isn't full it calls the EnterDetails method which allows the user to enter details into the

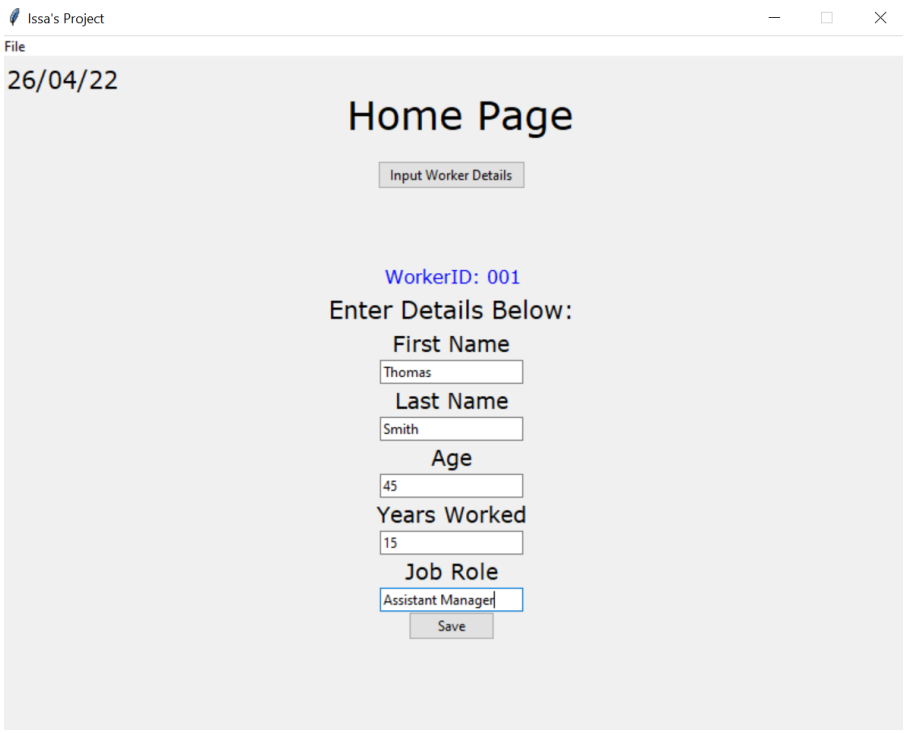
```
def GetDetails(self):
     DetailsComplete = self.CheckDetails()
     if DetailsComplete == False:
         self.EnterDetails()
     else:
          self.ResetPage()
          self.DetailsButton.forget()
          self.ManagerButton.place(x=330,y=175)
          self.WorkerButton.place(x=330,y=275)
```



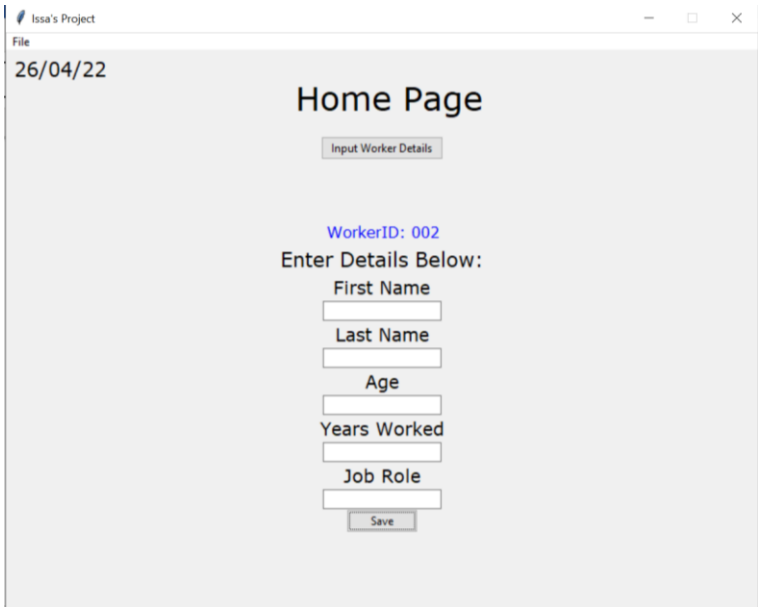
When running the code for the first time, the Details table was of course empty so the Manager has been prompted to enter worker details in. Firstly they pressed the button.



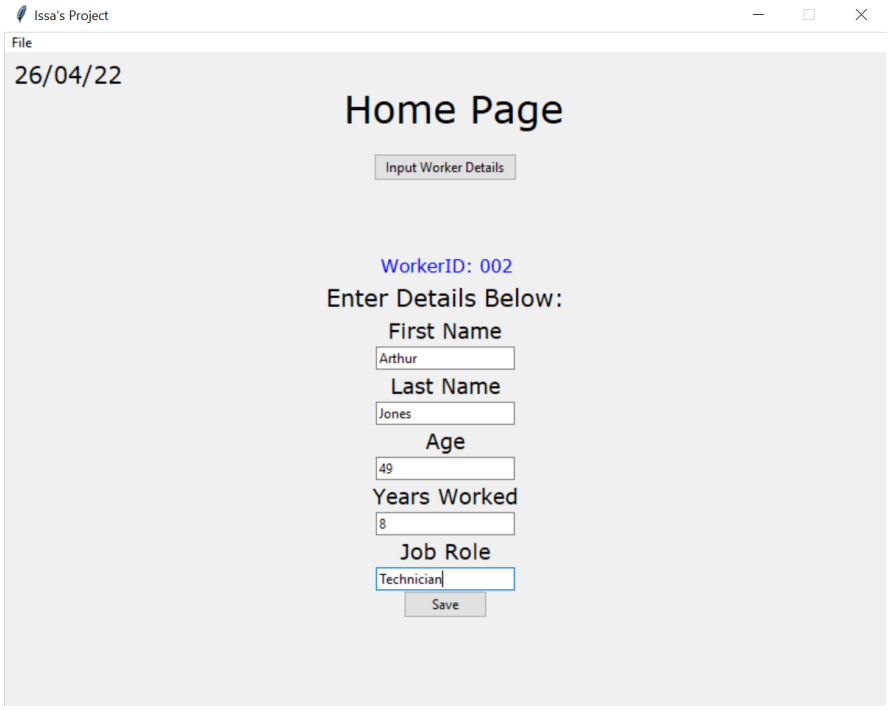
Then this page showed up. The manager had to enter the details for Worker ID 001. The manager was informed that it didn't matter which order that the worker's details needed to be entered so any worker could be Worker ID 001.



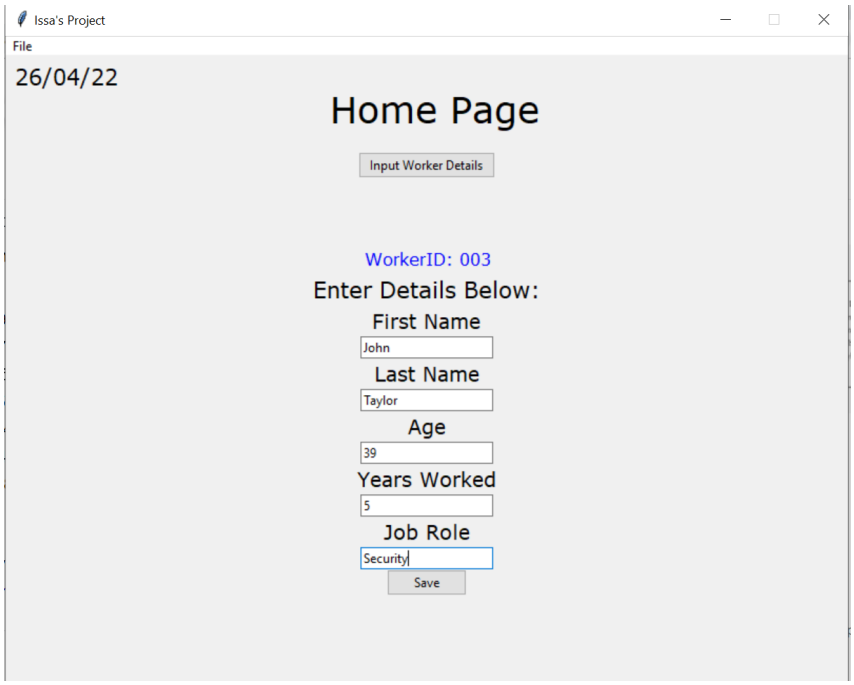
The manager entered the details of the first worker.

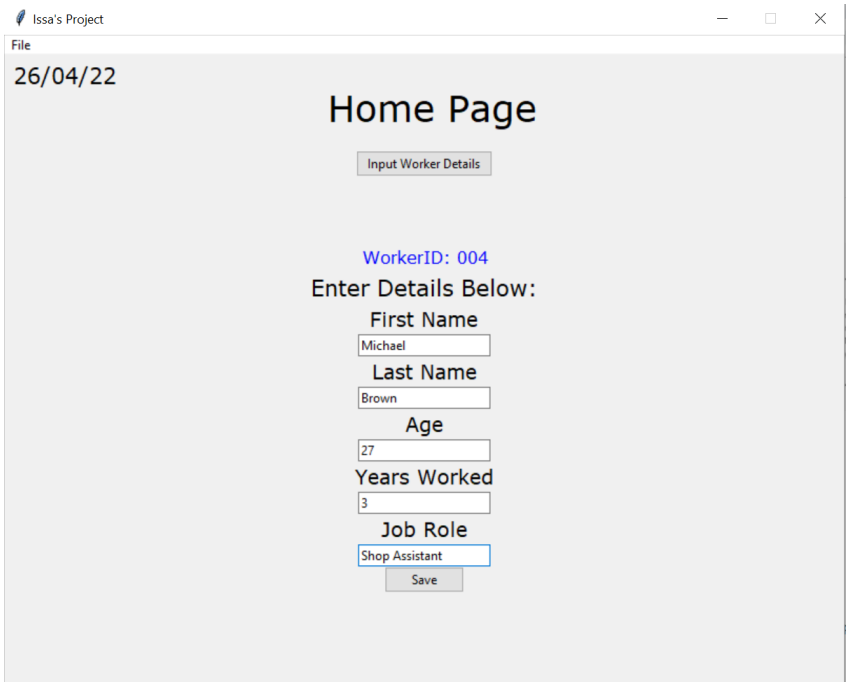


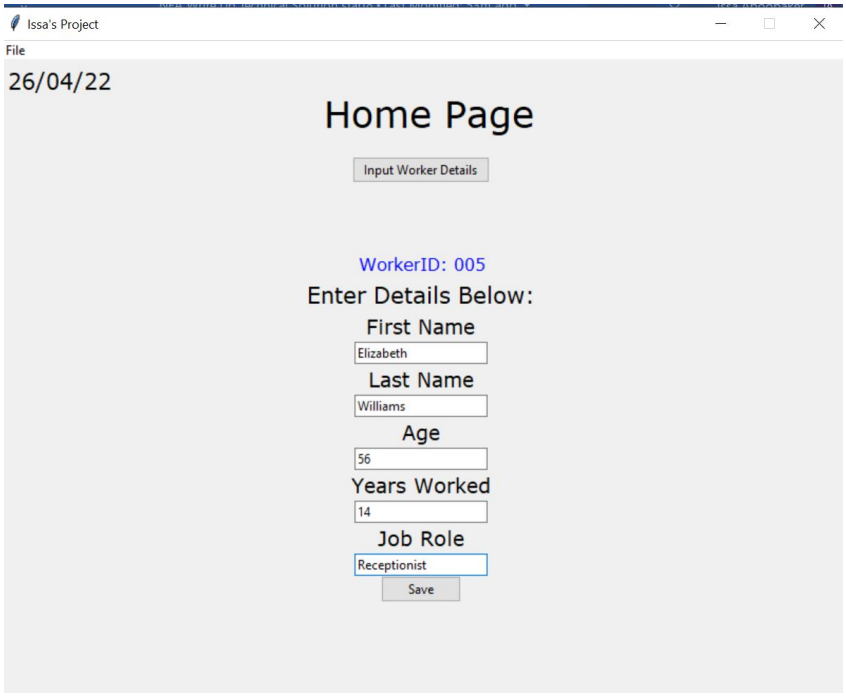
The entries were cleared and the WorkerID changed to show Worker ID 002. It was made apparent that this would happen 8 times, once for each worker so the manager needed to enter the details for each person and press save.

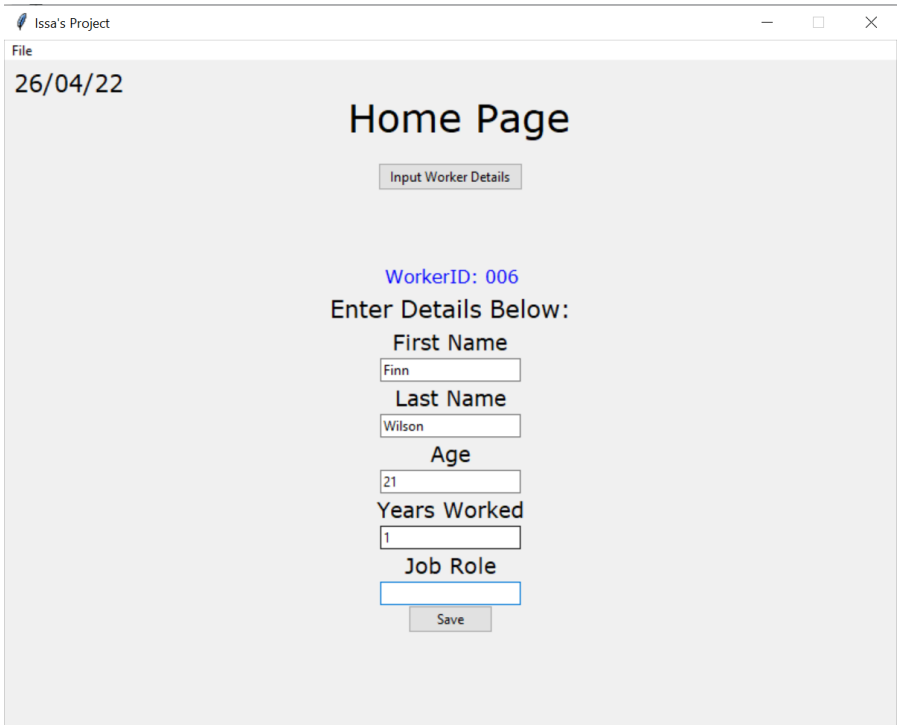


The manager entered the rest of the details in for all workers

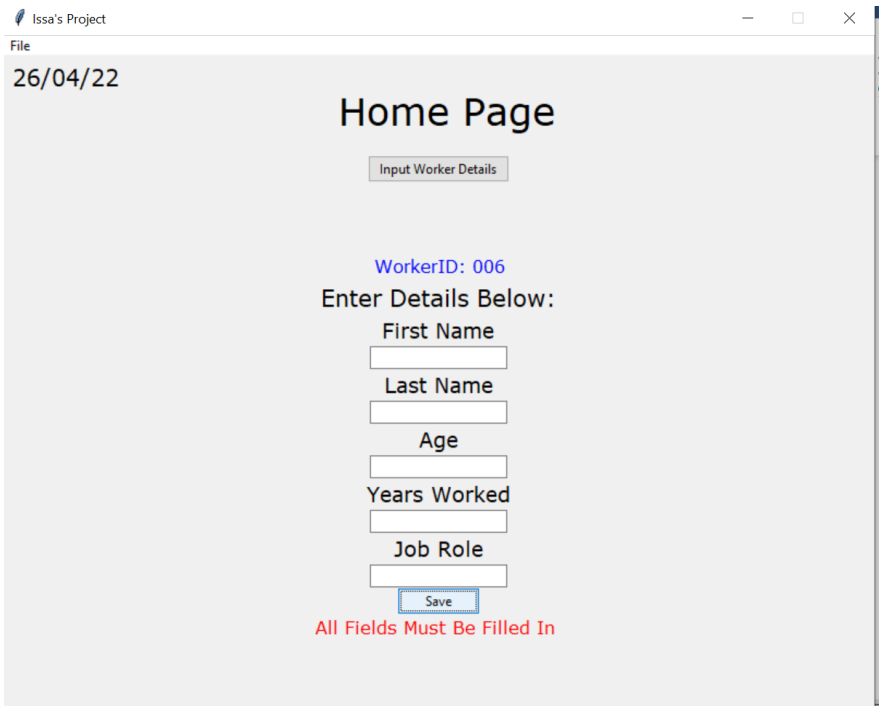



Issa's Project
File
26/04/22
Home Page
Input Worker Details
WorkerID: 004
Enter Details Below:
First Name
Michael
Last Name
Brown
Age
27
Years Worked
3
Job Role
Shop Assistant
Save

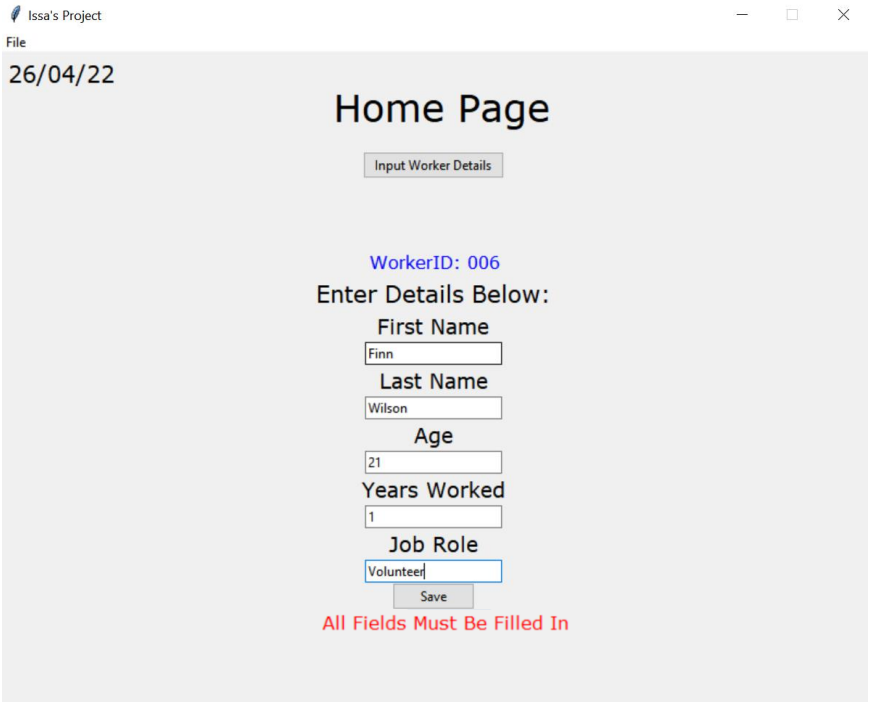

Issa's Project
File
26/04/22
Home Page
Input Worker Details
WorkerID: 005
Enter Details Below:
First Name
Elizabeth
Last Name
Williams
Age
56
Years Worked
14
Job Role
Receptionist
Save



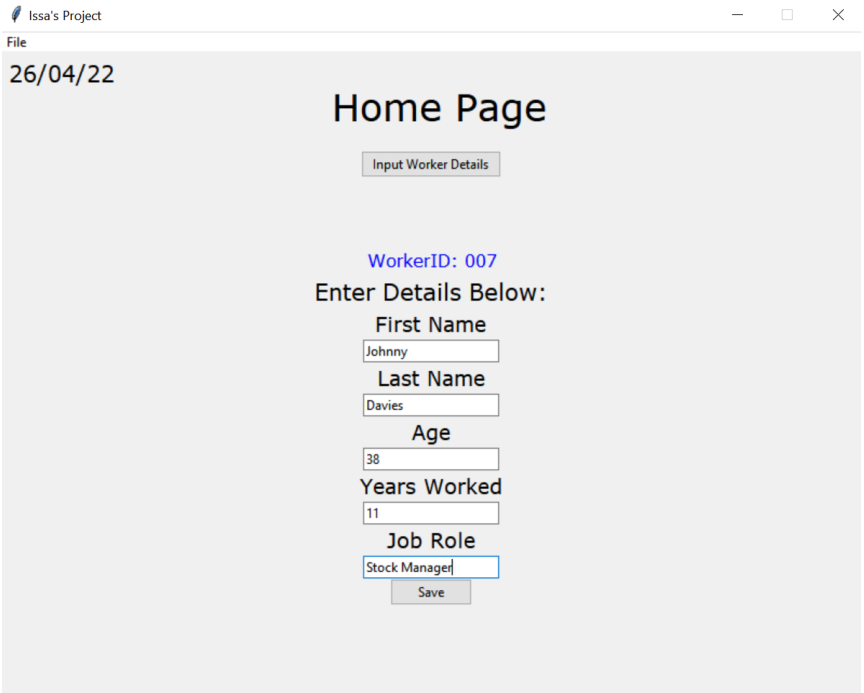
For WorkerID 006 the manager tried to enter the details but didn't know what to enter into the Job Role field as the employee is doing voluntary work at the company and doesn't have a job title yet. Then the manager pressed Save.



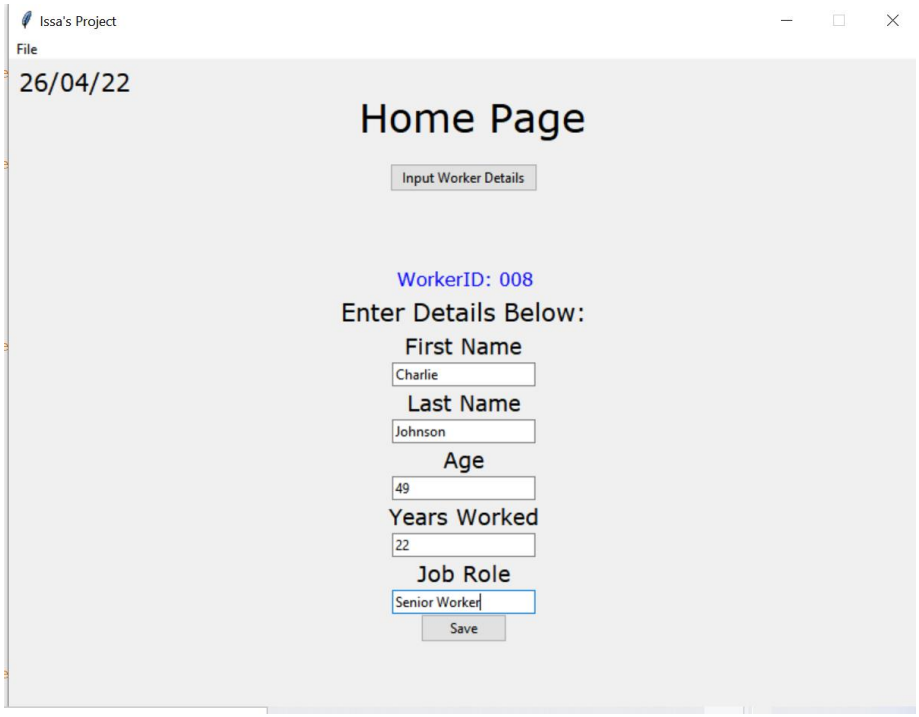
This was the outcome of that. The entries were cleared and a message was displayed on the screen saying that all fields must be filled in. The WorkerID remained at 006 because the details weren't correctly entered



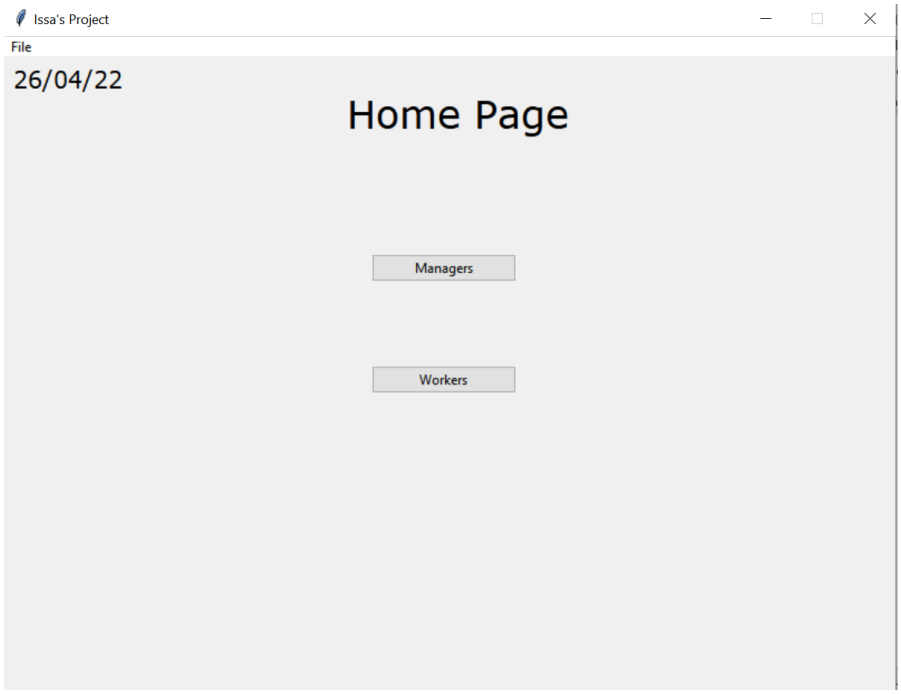
The Manager was told to re enter the details in and put "Volunteer" in the Job Role field. The manager did this and pressed Save



The message disappeared and the manager continued to enter the worker details as normal.



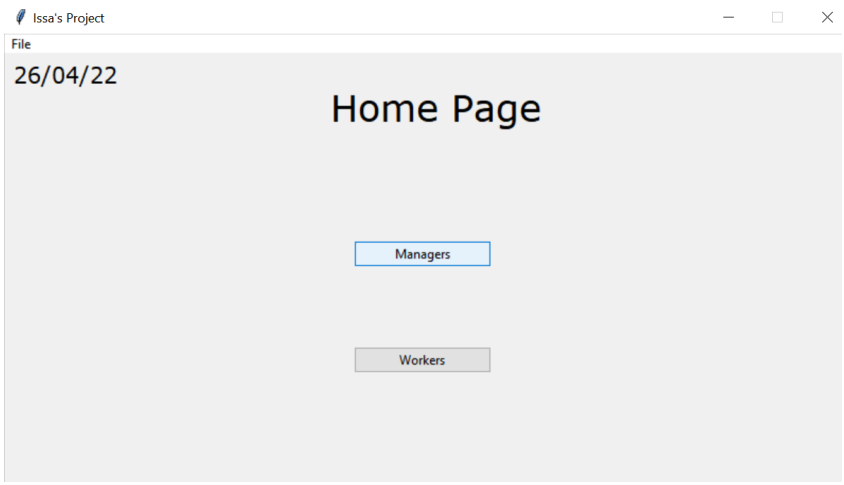
The final worker's details were entered and the Home Page appeared as normal showing the rest of the program is ready to be used.



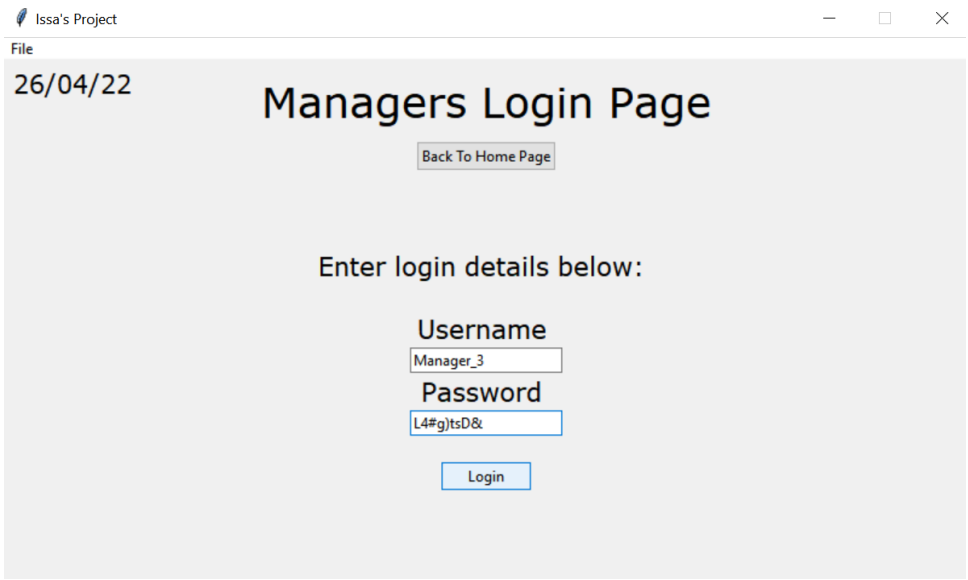
	WorkerID	FirstName	LastName	Age	YearsWorked	JobRole
	Filter	Filter	Filter	Filter	Filter	Filter
1	001	Thomas	Smith	45	15	Assistant Manager
2	002	Arthur	Jones	49	8	Technician
3	003	John	Taylor	39	5	Security
4	004	Michael	Brown	27	3	Shop Assistant
5	005	Elizabeth	Williams	56	14	Receptionist
6	006	Finn	Wilson	21	1	Volunteer
7	007	Johnny	Davies	38	11	Stock Manager
8	008	Charlie	Johnson	49	22	Senior Worker

The database was checked using DB Browser and the manager confirmed that all details that had been entered had been correctly stored in the database.

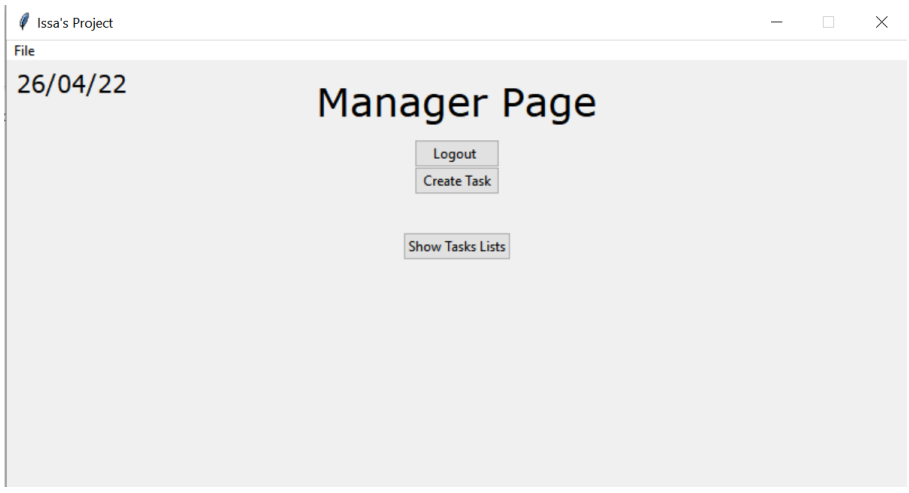
Test 2



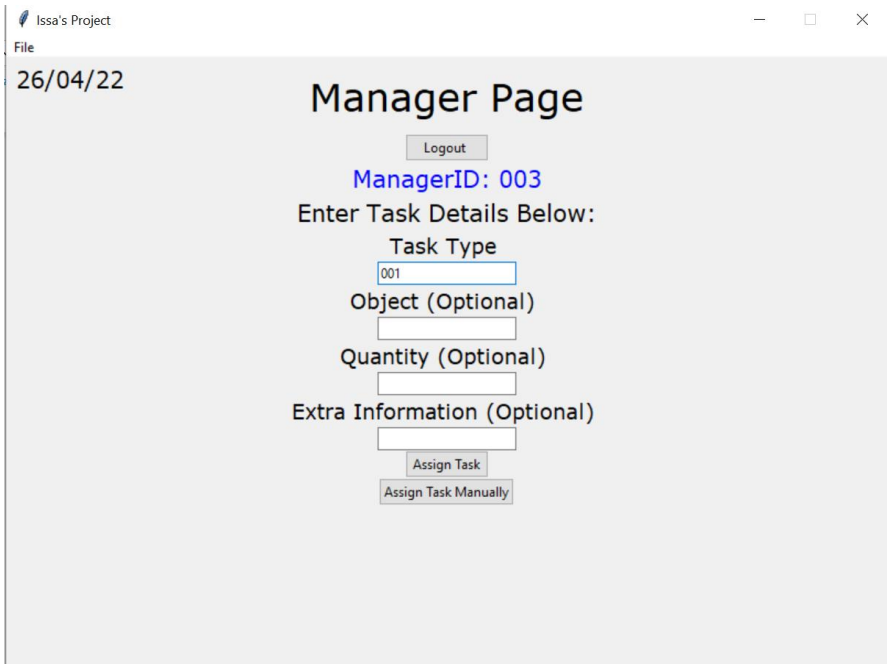
Firstly the manager logged in so the Managers button was pressed.

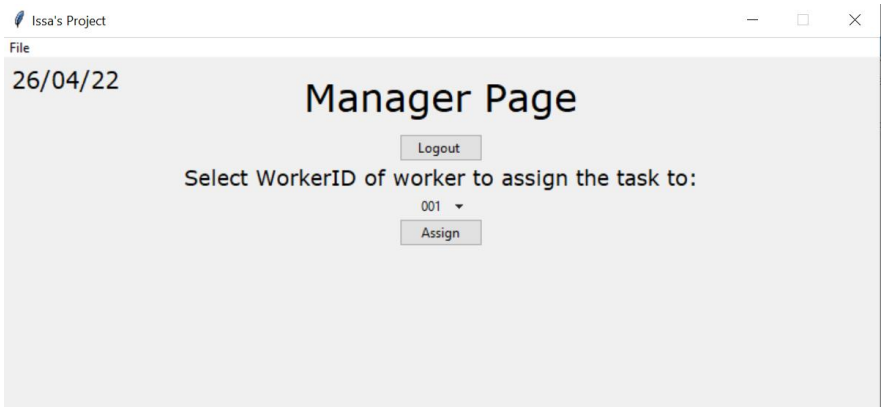


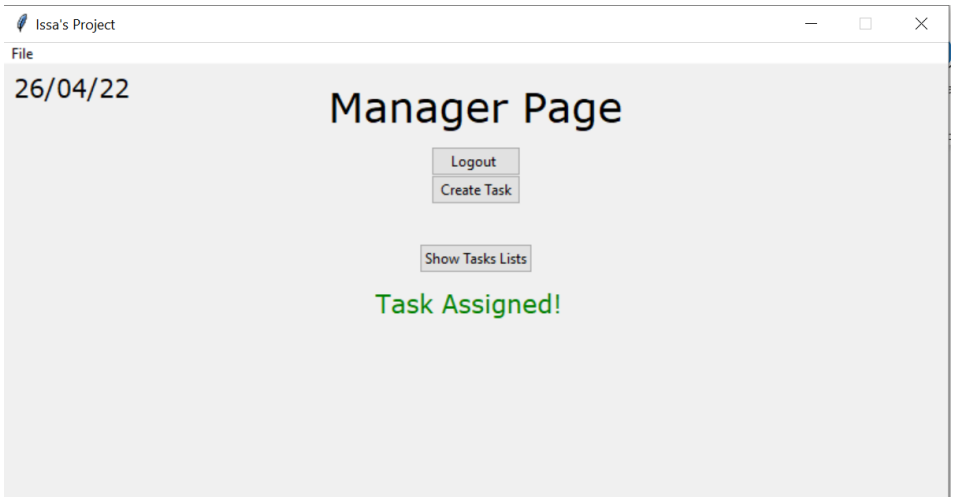
The manager logged in using their details.



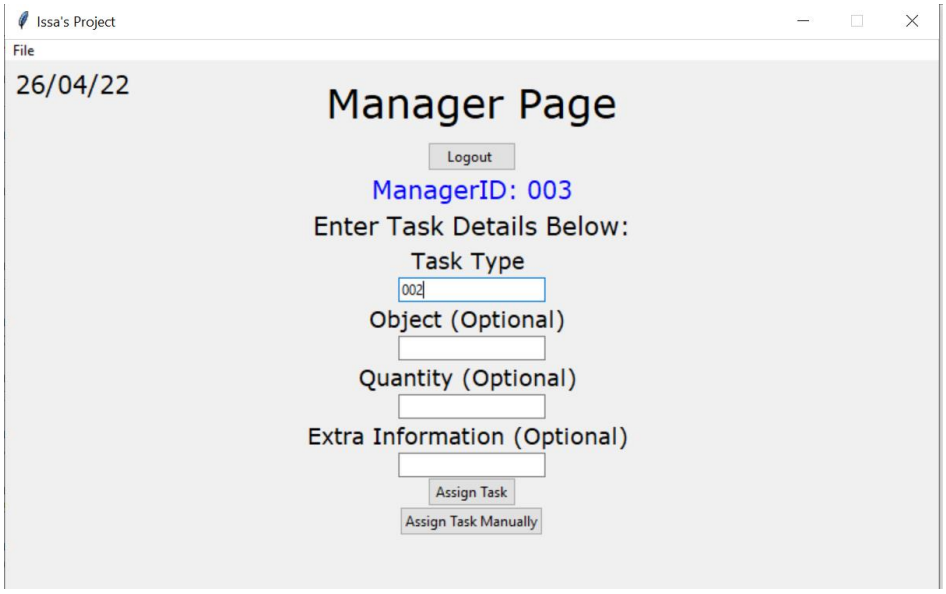
The details worked and it entered the manager into their page.

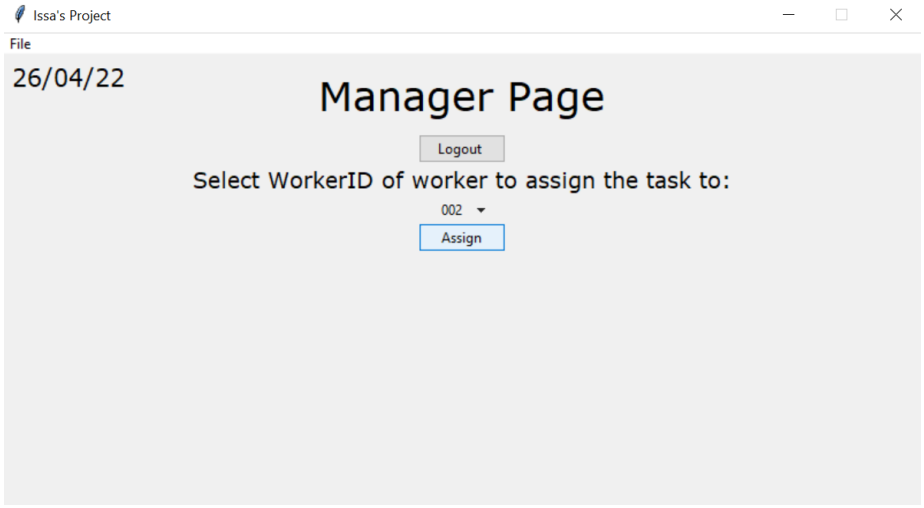


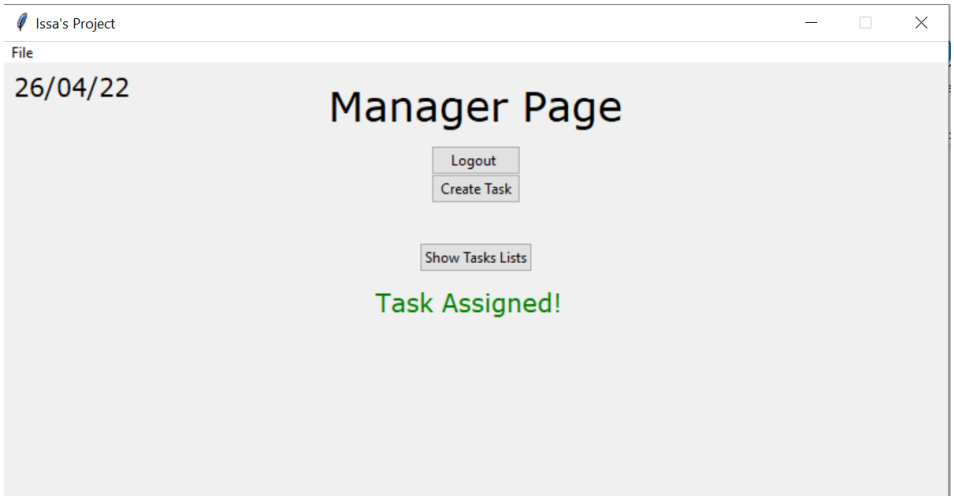


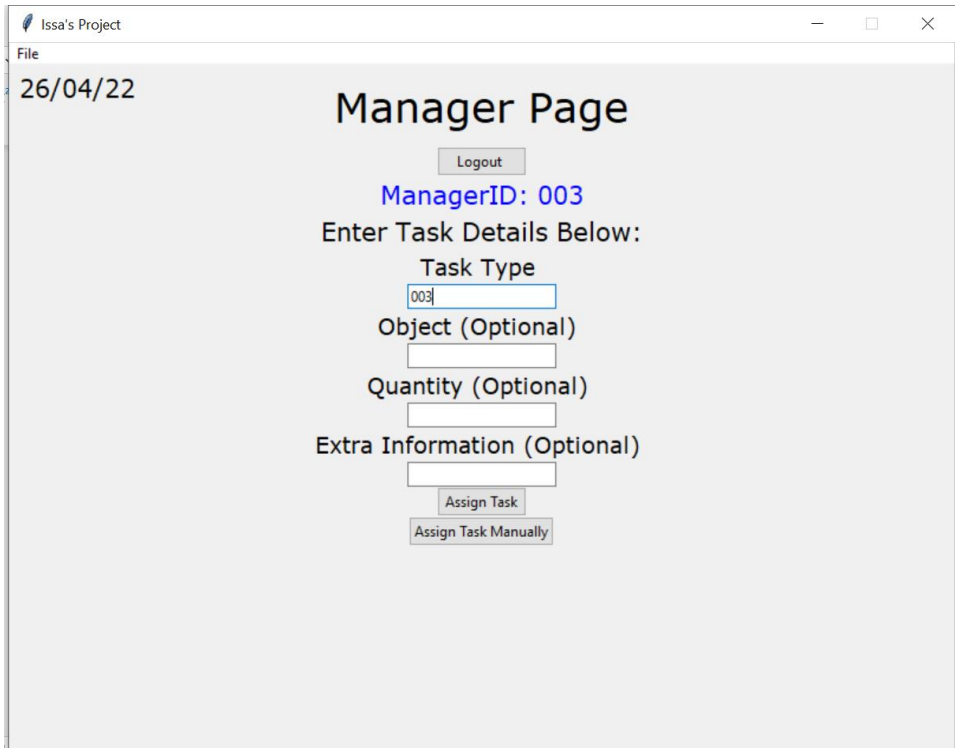


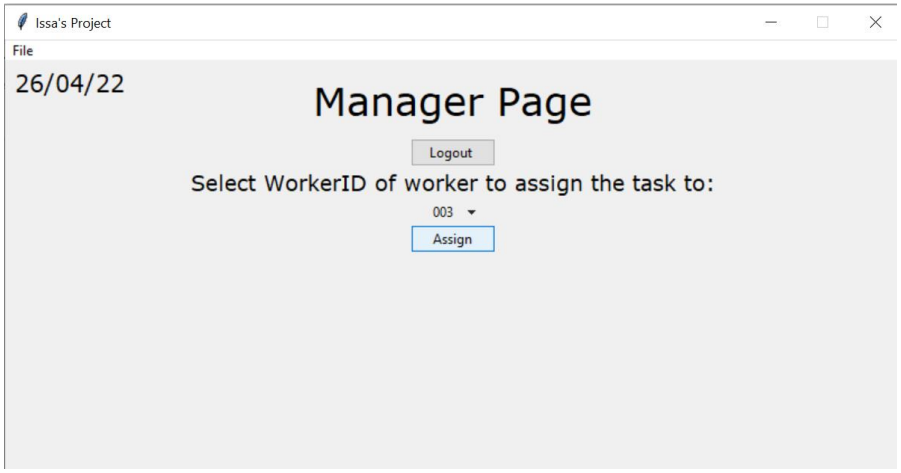
The manager created a task here. Since the other 3 fields are not required to be filled in, the manager only entered in the Task Type to keep the test shorter. Each task has to be different from each other so for the Task Type for each one, the manager just put in the Worker ID of the worker that the task was going to be assigned to

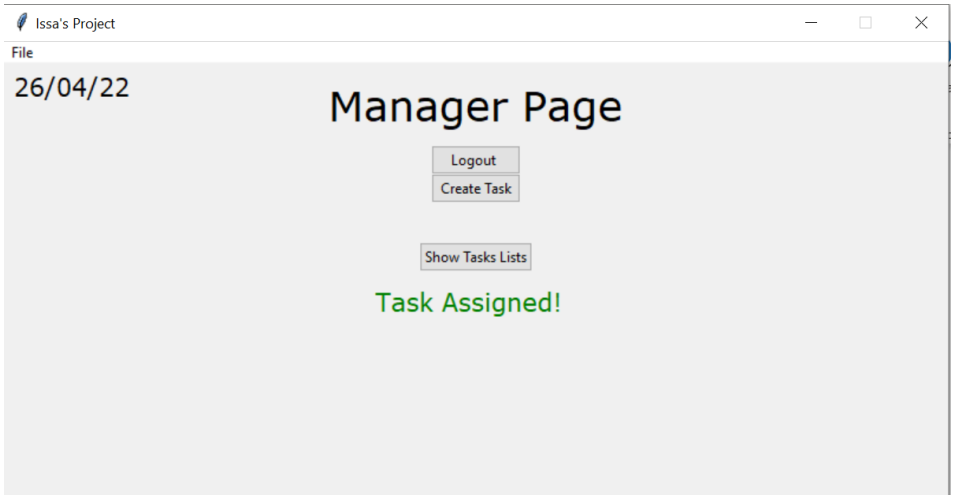

Issa's Project
File
26/04/22
Manager Page
Logout
ManagerID: 003
Enter Task Details Below:
Task Type
002
Object (Optional)
Quantity (Optional)
Extra Information (Optional)
Assign Task
Assign Task Manually

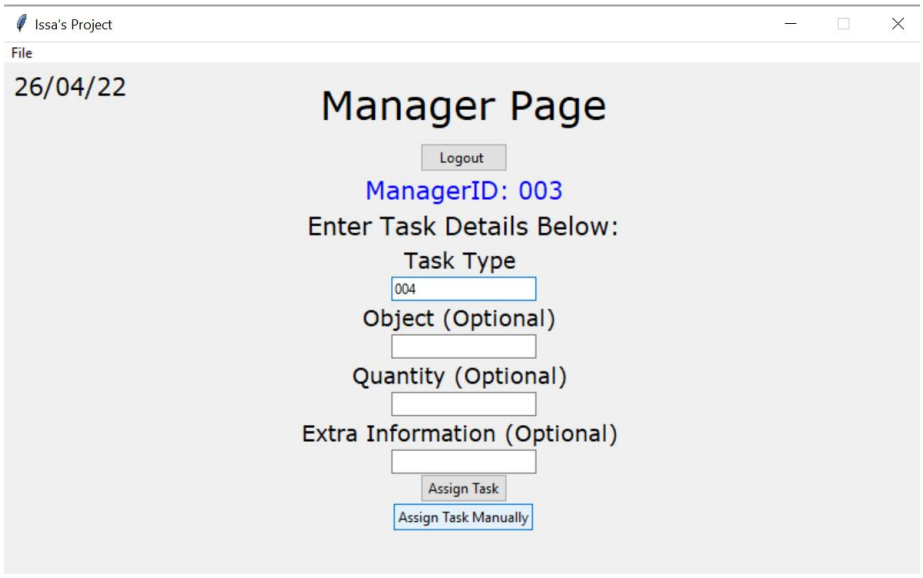

Issa's Project
File
26/04/22
Manager Page
Logout
Select WorkerID of worker to assign the task to:
002
Assign

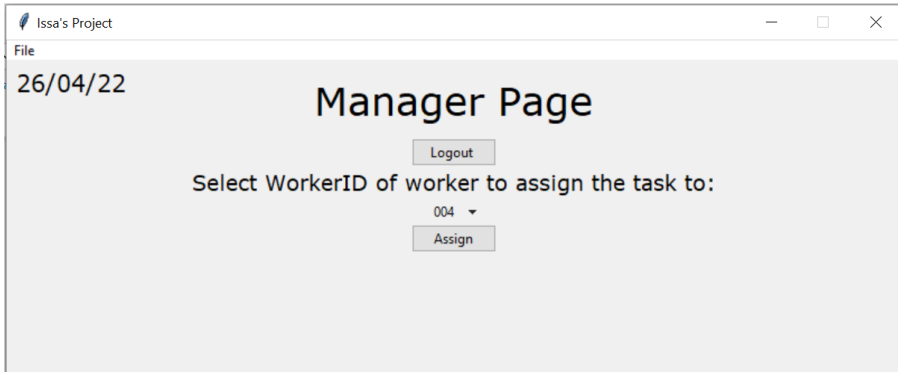

Issa's Project
File
26/04/22
Manager Page
Logout
Create Task
Show Tasks Lists
Task Assigned!

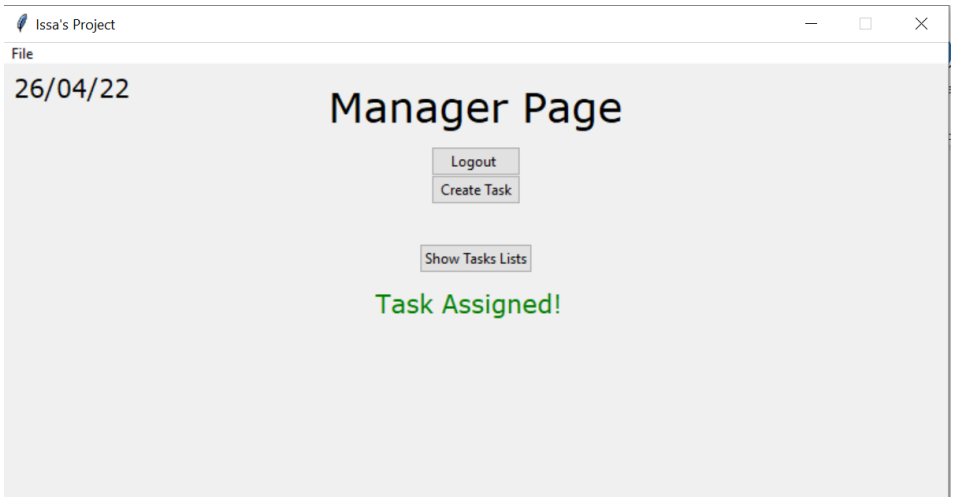

Issa's Project
File
26/04/22
Manager Page
Logout
ManagerID: 003
Enter Task Details Below:
Task Type
003
Object (Optional)
Quantity (Optional)
Extra Information (Optional)
Assign Task
Assign Task Manually

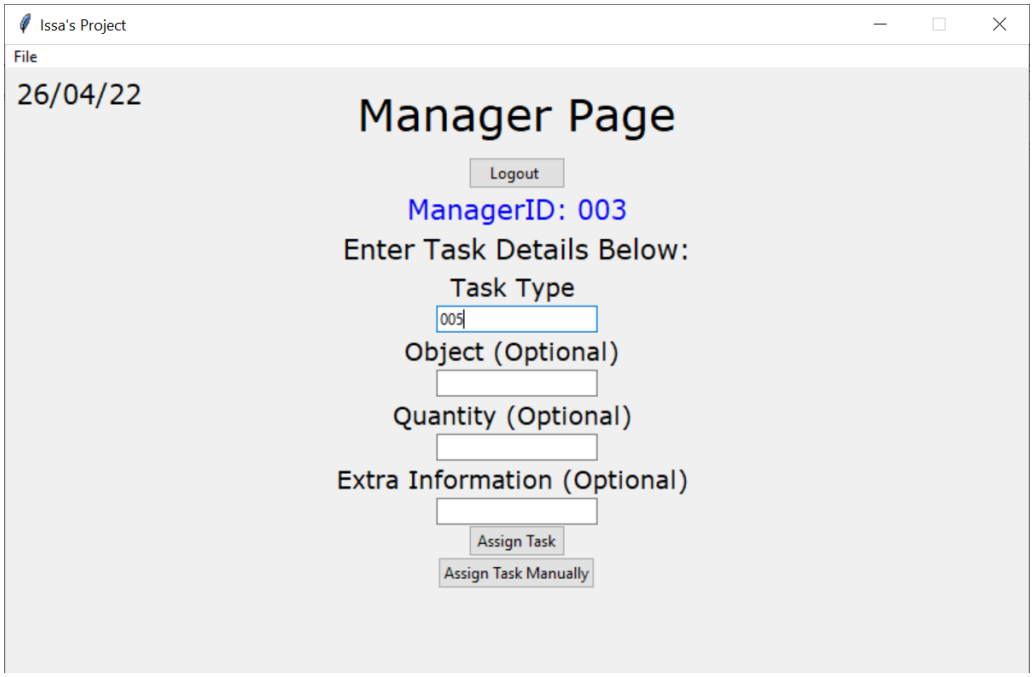

Issa's Project
File
26/04/22
Manager Page
Logout
Select WorkerID of worker to assign the task to:
003
Assign

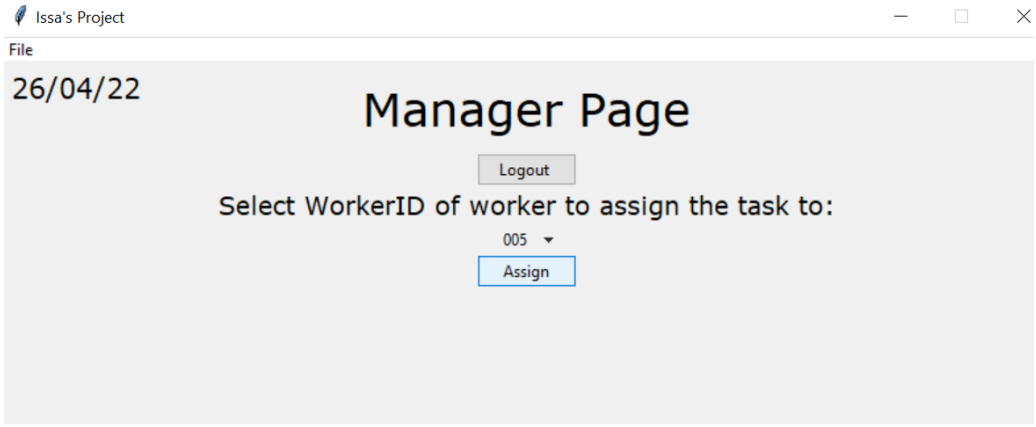

Issa's Project
File
26/04/22
Manager Page
Logout
Create Task
Show Tasks Lists
Task Assigned!

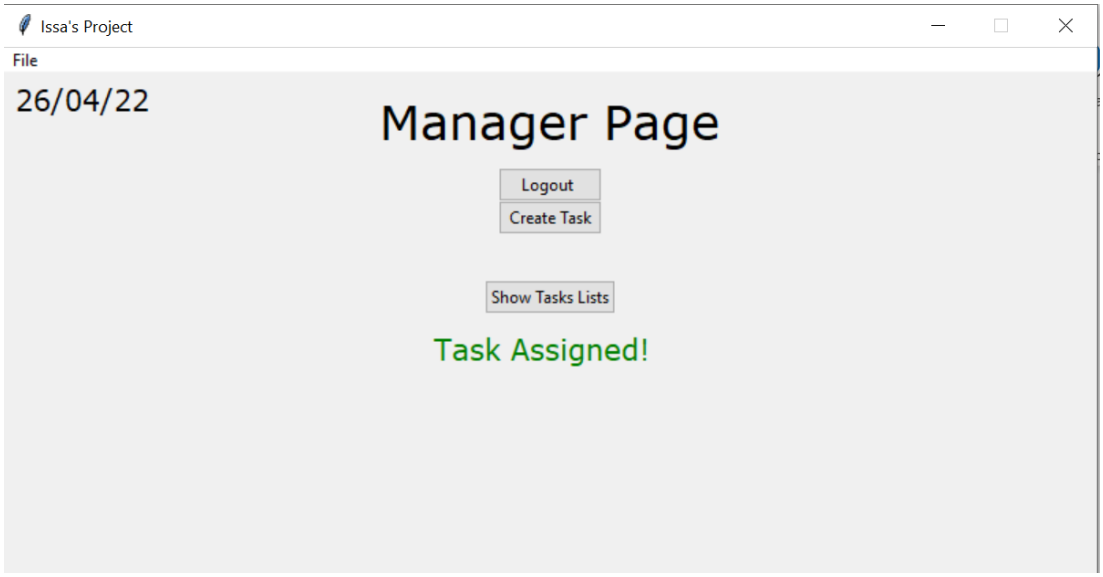

Issa's Project
File
26/04/22
Manager Page
Logout
ManagerID: 003
Enter Task Details Below:
Task Type
004
Object (Optional)
Quantity (Optional)
Extra Information (Optional)
Assign Task
Assign Task Manually


Issa's Project
File
26/04/22
Manager Page
Logout
Select WorkerID of worker to assign the task to:
004
Assign

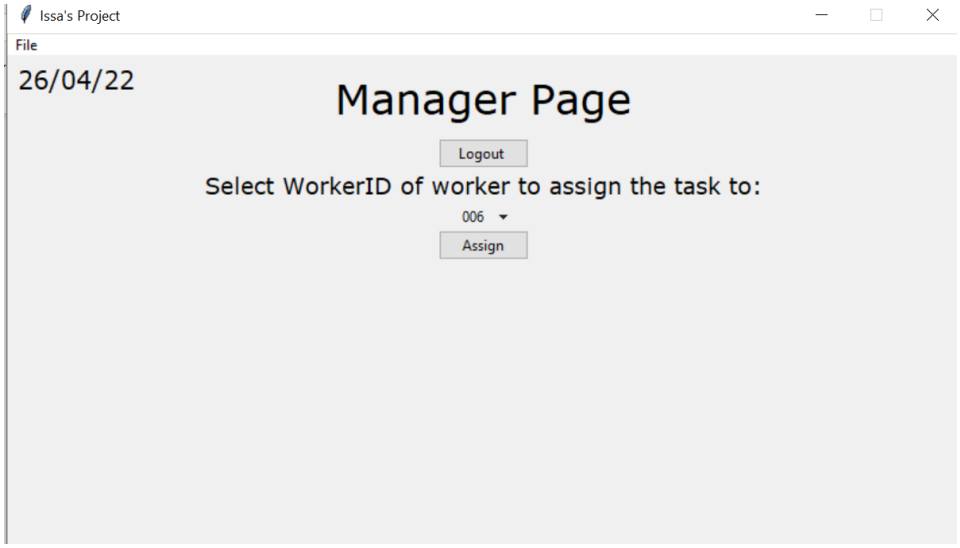

Issa's Project
File
26/04/22
Manager Page
Logout
Create Task
Show Tasks Lists
Task Assigned!

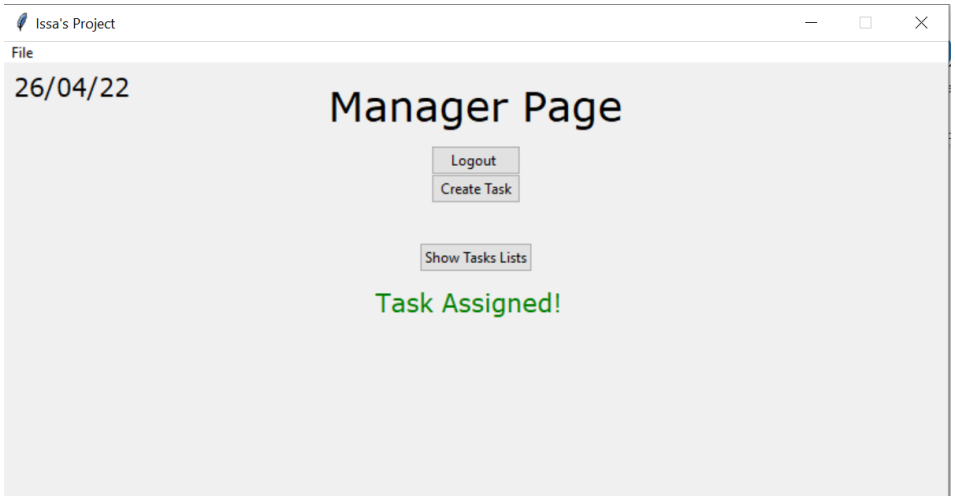

Issa's Project
File
26/04/22
Manager Page
Logout
ManagerID: 003
Enter Task Details Below:
Task Type
005
Object (Optional)
Quantity (Optional)
Extra Information (Optional)
Assign Task
Assign Task Manually


Issa's Project
File
26/04/22
Manager Page
Logout
Select WorkerID of worker to assign the task to:
005
Assign

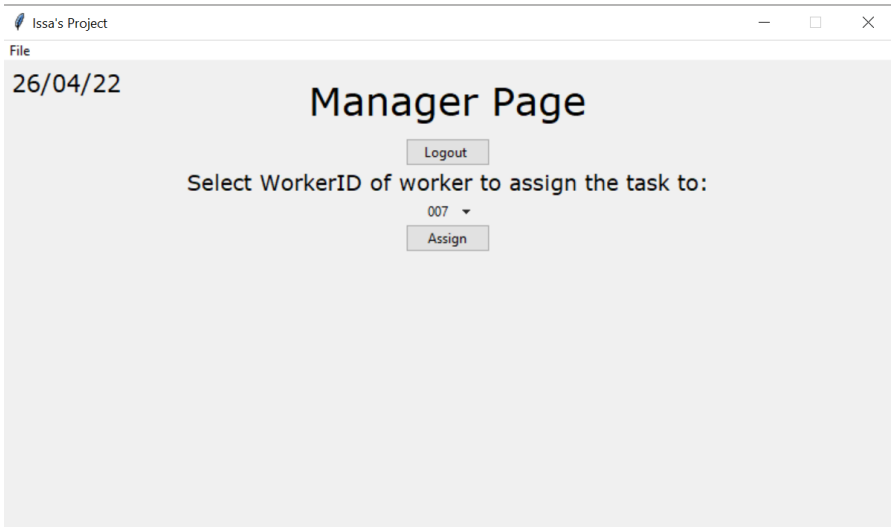

Issa's Project
File
26/04/22
Manager Page
Logout
Create Task
Show Tasks Lists
Task Assigned!

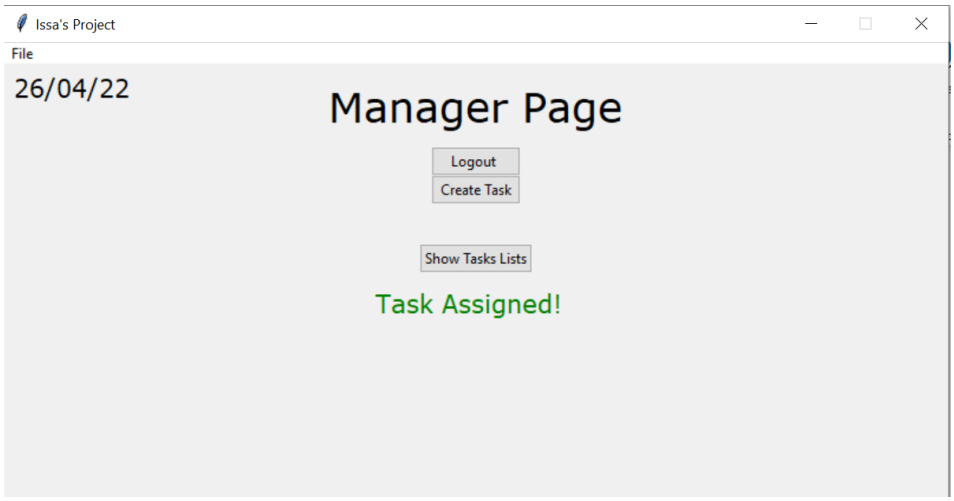

Issa's Project
File
26/04/22
Manager Page
Logout
ManagerID: 003
Enter Task Details Below:
Task Type
006
Object (Optional)
Quantity (Optional)
Extra Information (Optional)
Assign Task
Assign Task Manually

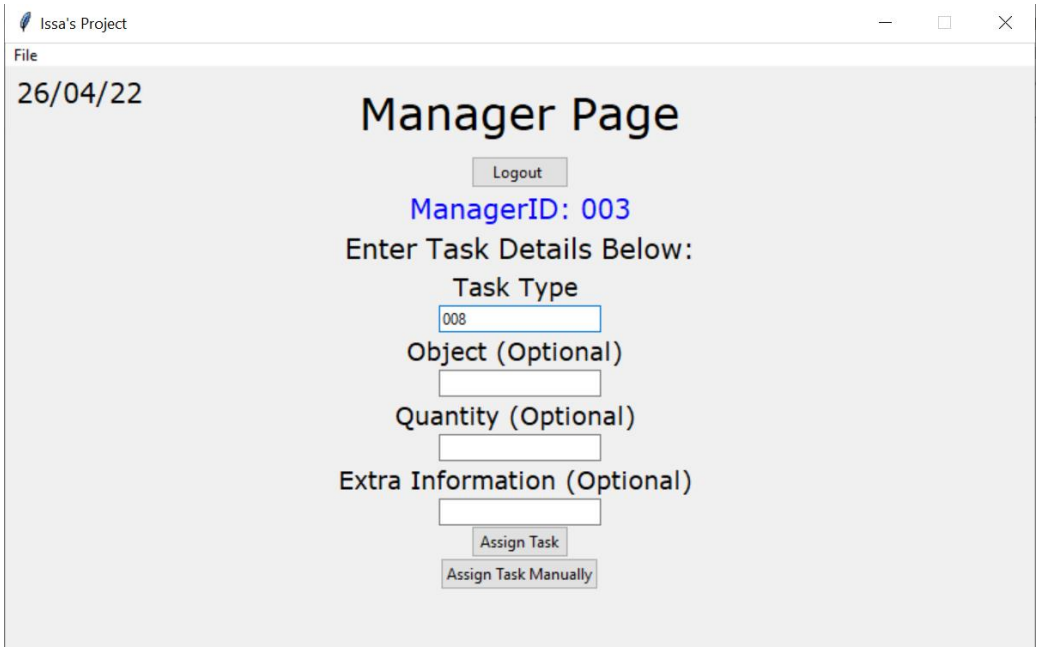

Issa's Project
File
26/04/22
Manager Page
Logout
Select WorkerID of worker to assign the task to:
006
Assign

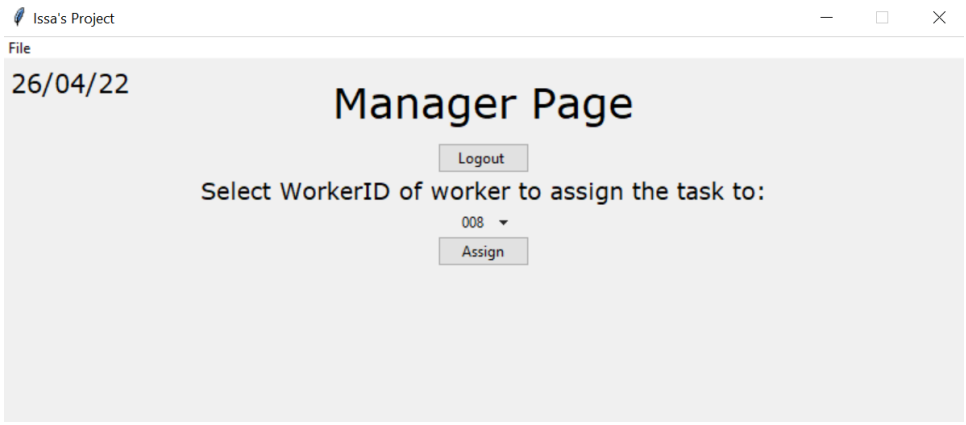

Issa's Project
File
26/04/22
Manager Page
Logout
Create Task
Show Tasks Lists
Task Assigned!

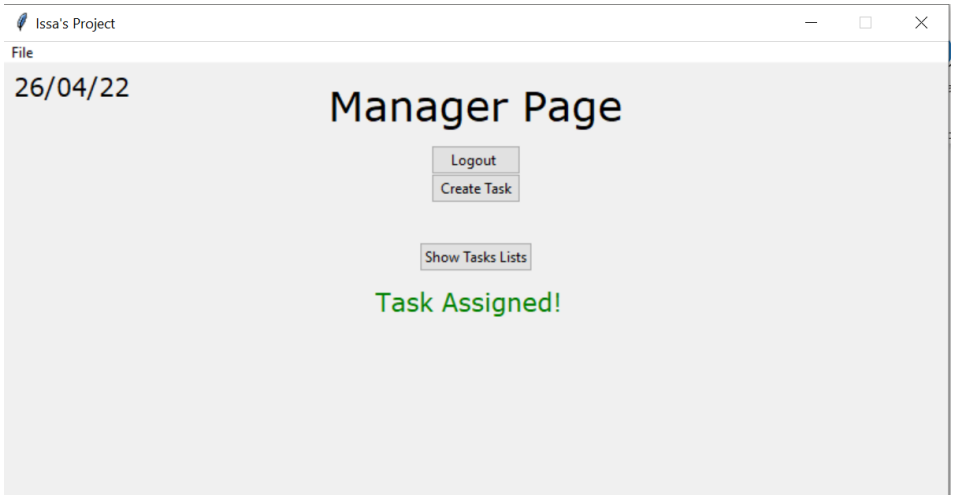

Issa's Project
File
26/04/22
Manager Page
Logout
ManagerID: 003
Enter Task Details Below:
Task Type
007
Object (Optional)
Quantity (Optional)
Extra Information (Optional)
Assign Task
Assign Task Manually

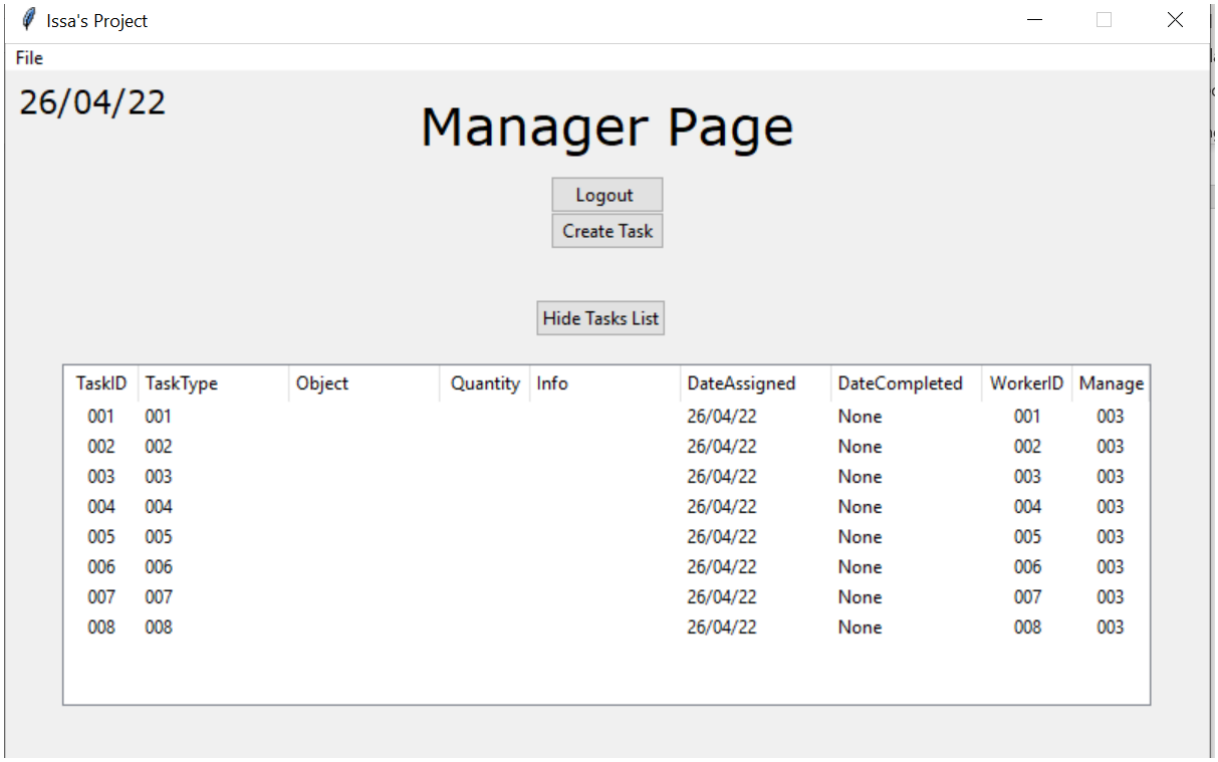

Issa's Project
File
26/04/22
Manager Page
Logout
Select WorkerID of worker to assign the task to:
007
Assign


Issa's Project
File
26/04/22
Manager Page
Logout
Create Task
Show Tasks Lists
Task Assigned!

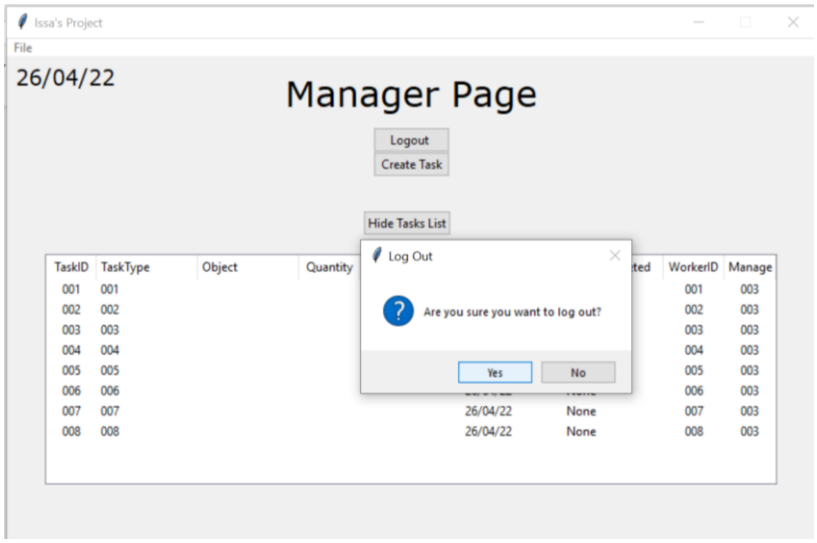

Issa's Project
File
26/04/22
Manager Page
Logout
ManagerID: 003
Enter Task Details Below:
Task Type
008
Object (Optional)
Quantity (Optional)
Extra Information (Optional)
Assign Task
Assign Task Manually


Issa's Project
File
26/04/22
Manager Page
Logout
Select WorkerID of worker to assign the task to:
008
Assign

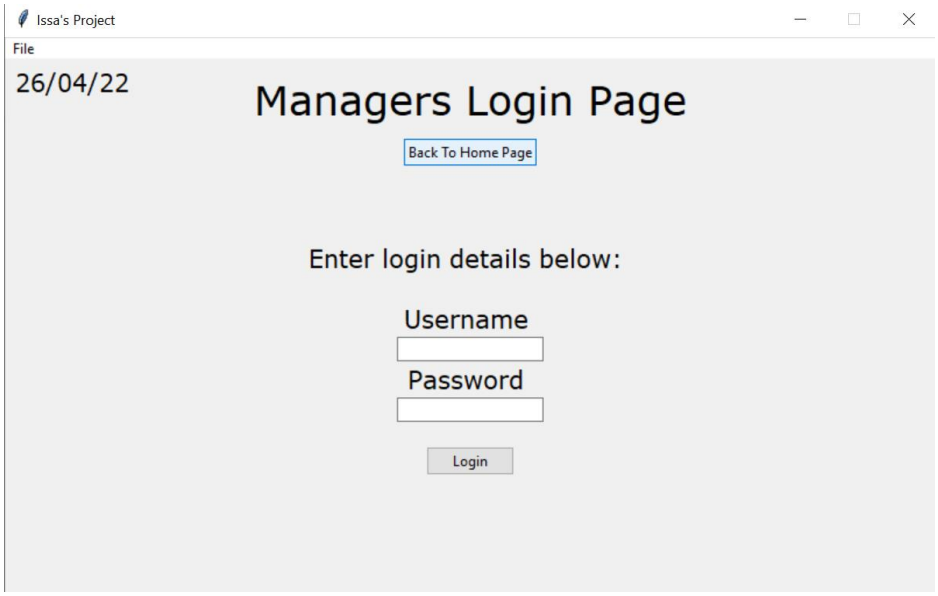

Issa's Project
File
26/04/22
Manager Page
Logout
Create Task
Show Tasks Lists
Task Assigned!

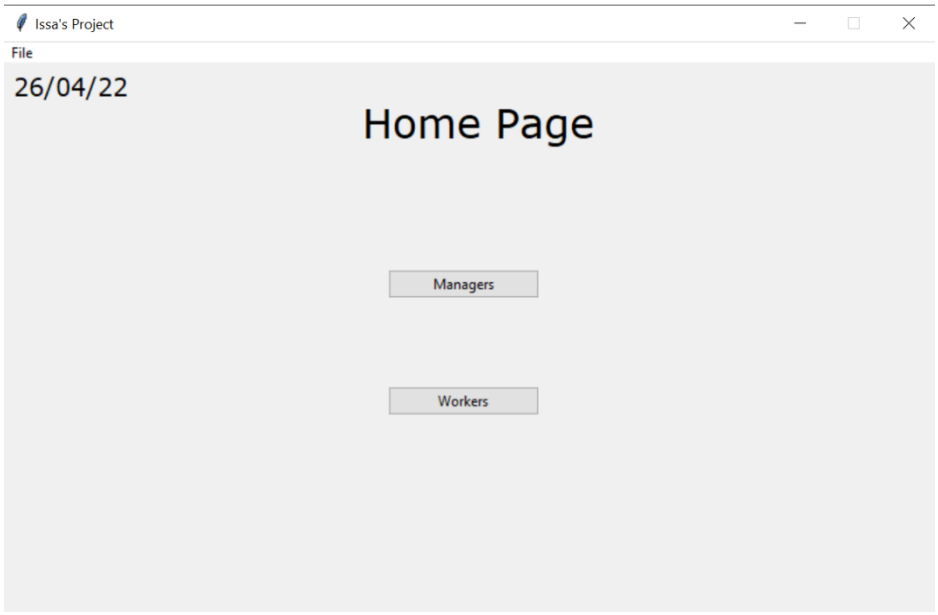


The manager checked that all the tasks had been assigned by using the Show Tasks List button which displayed the treeview, showing the Tasks table.

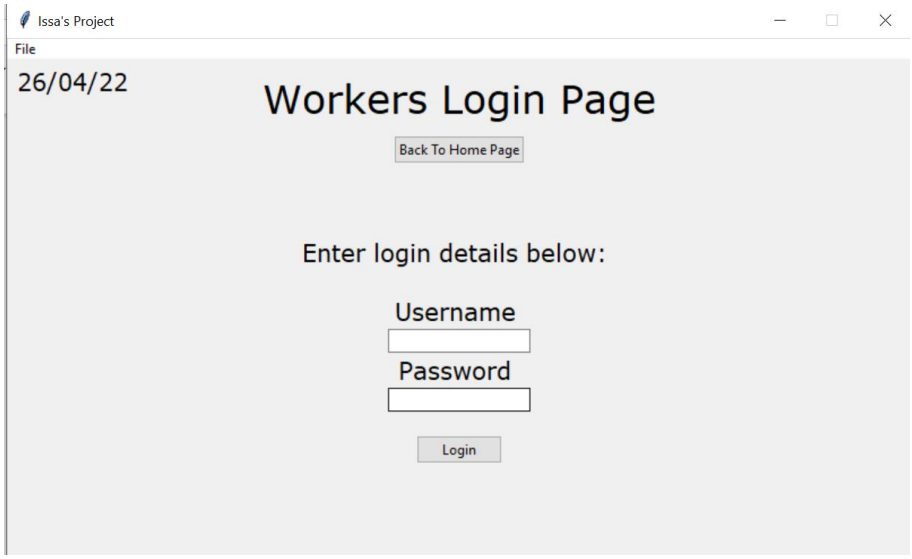


The manager used the Logout button to logout of the page.



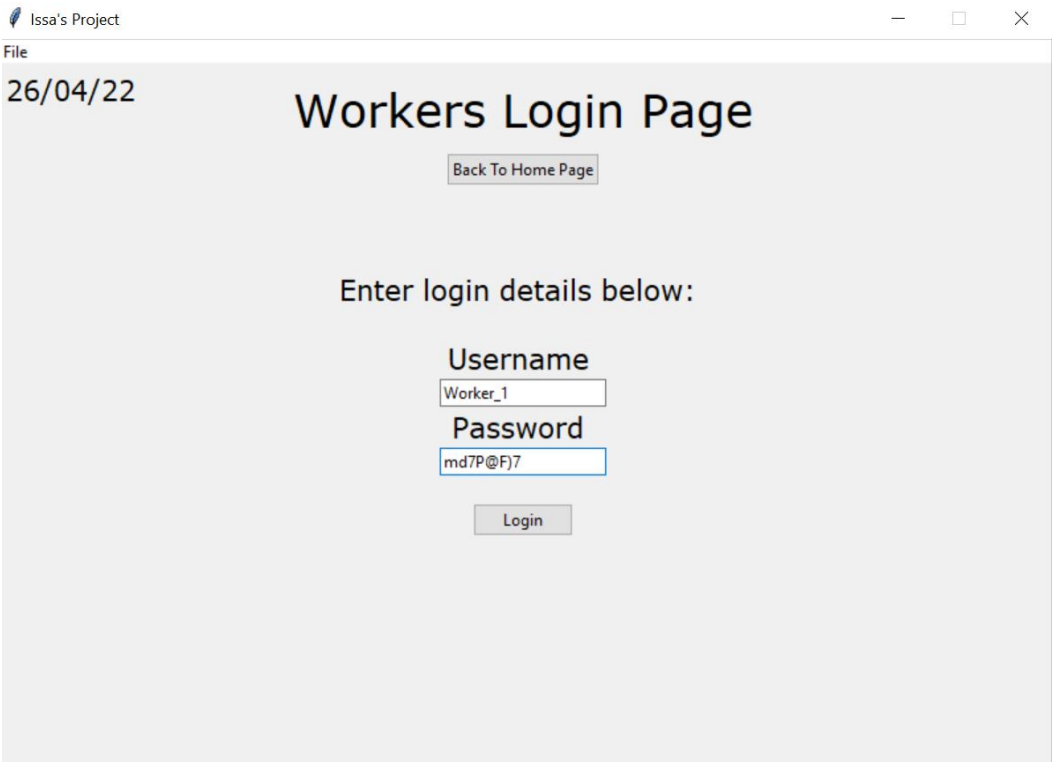


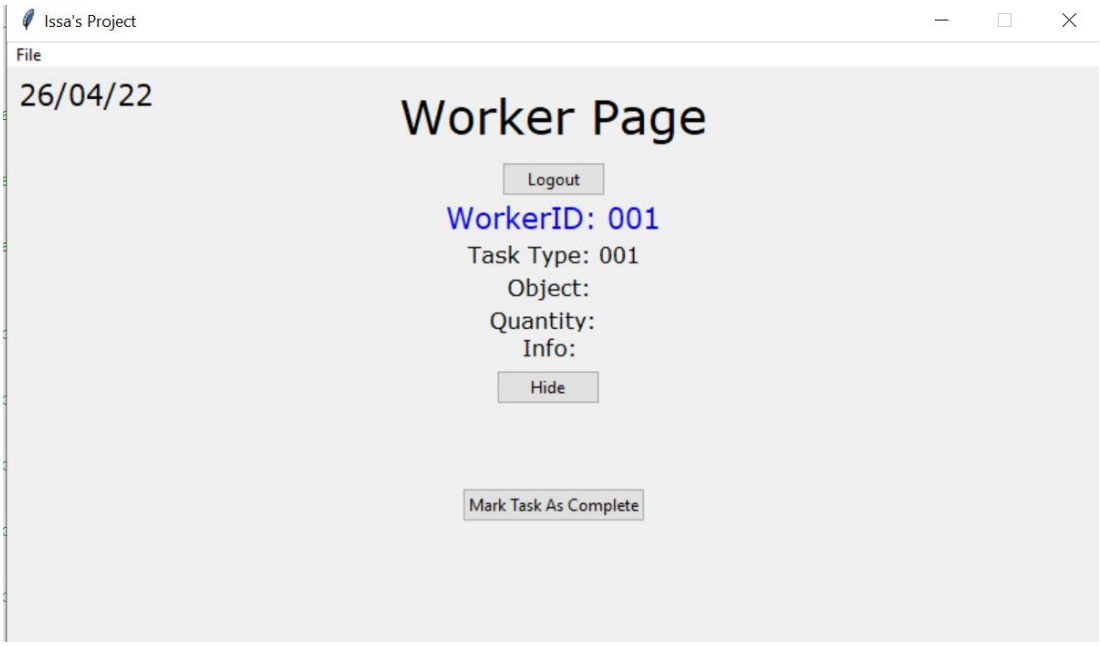
The manager used the Back To Home Page button to return to the home page.

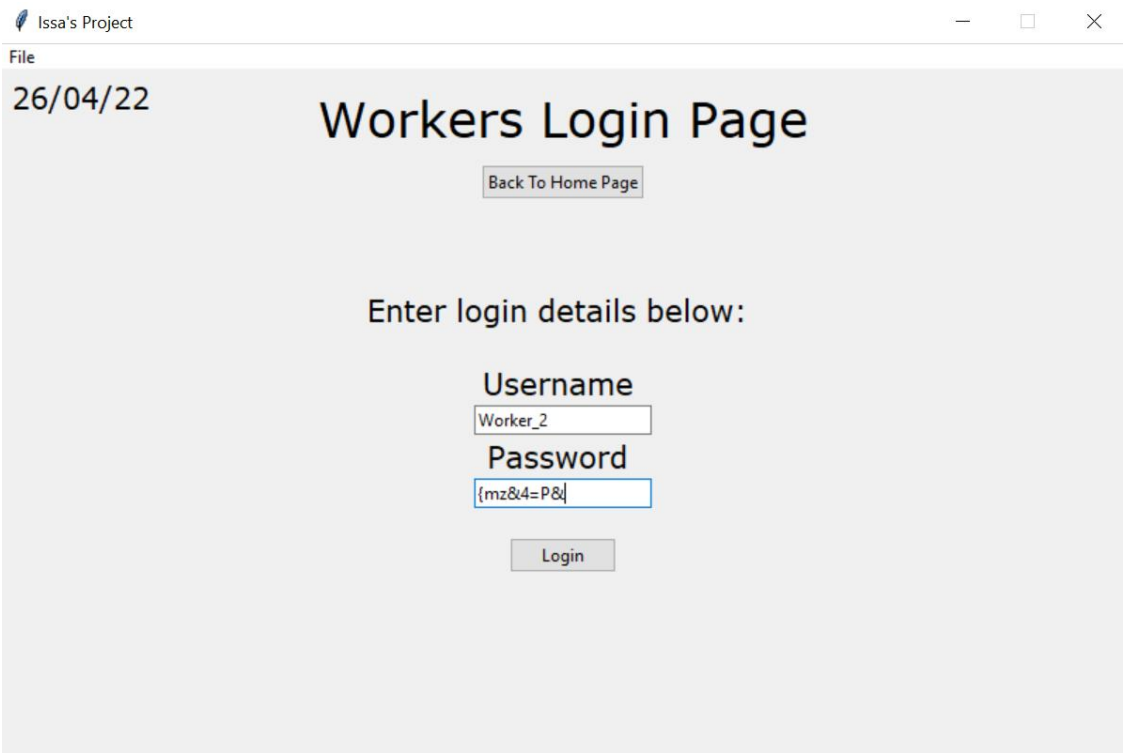


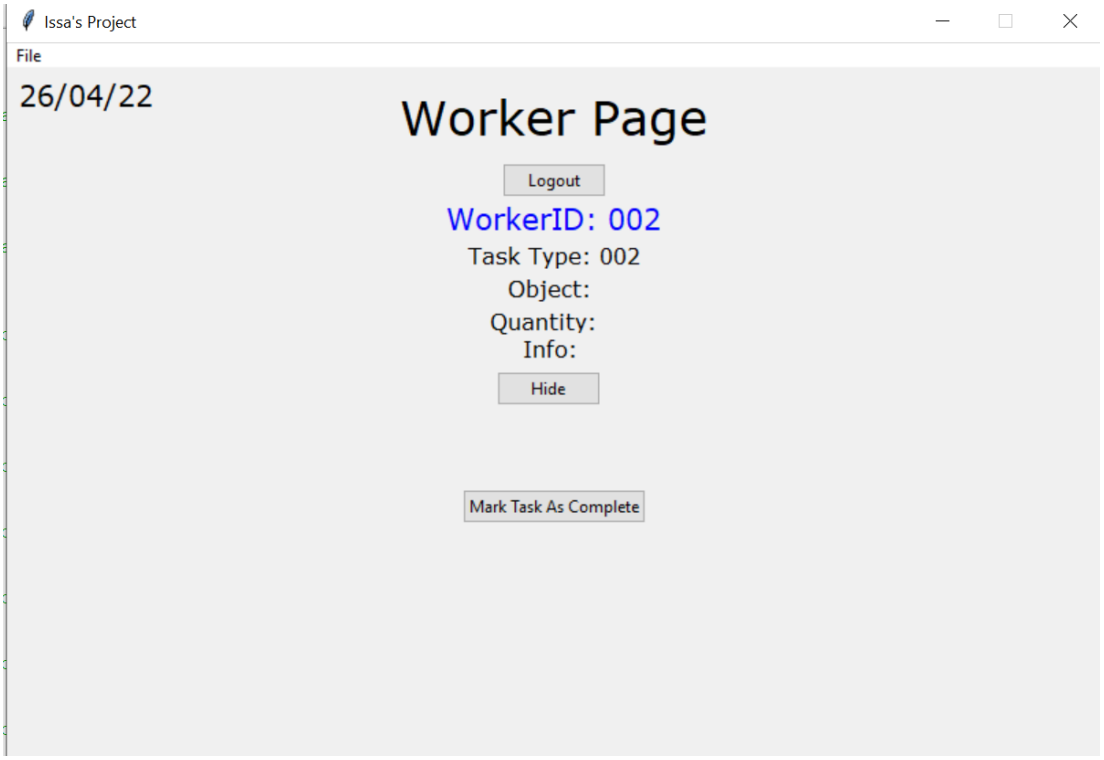
The worker login page was loaded using the Worker button

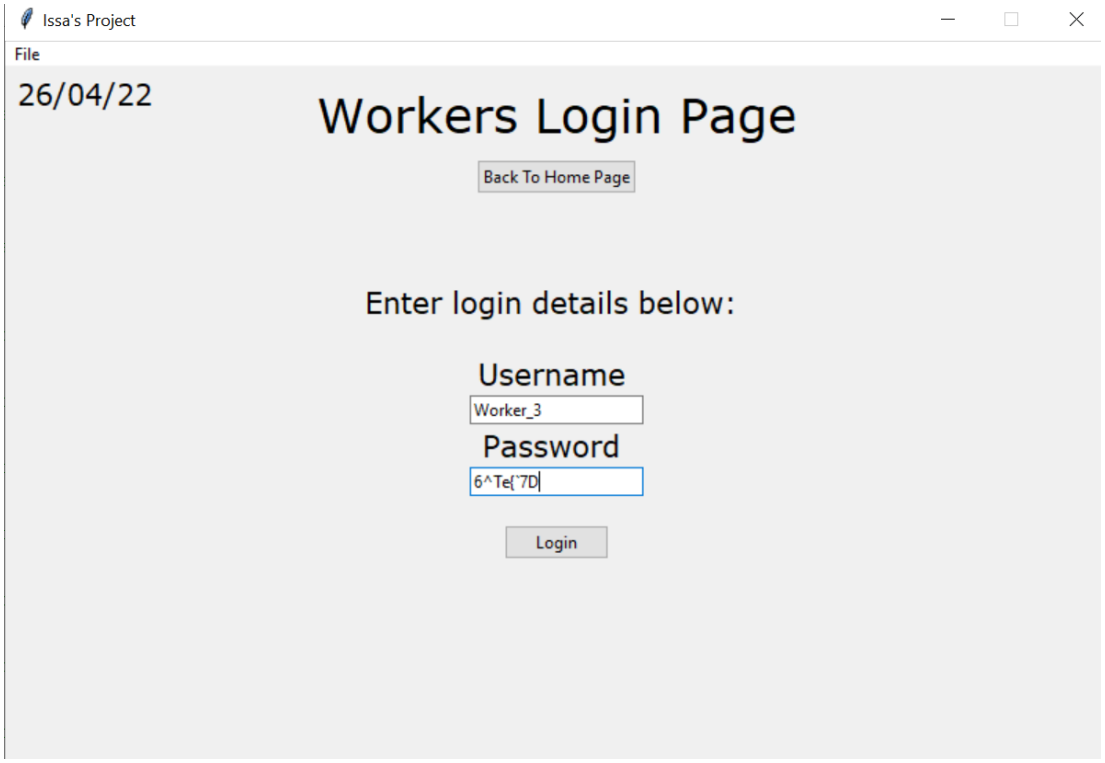
Each worker logged into their own page using their own details and pressed the Show Current Task button and they all had a different task.

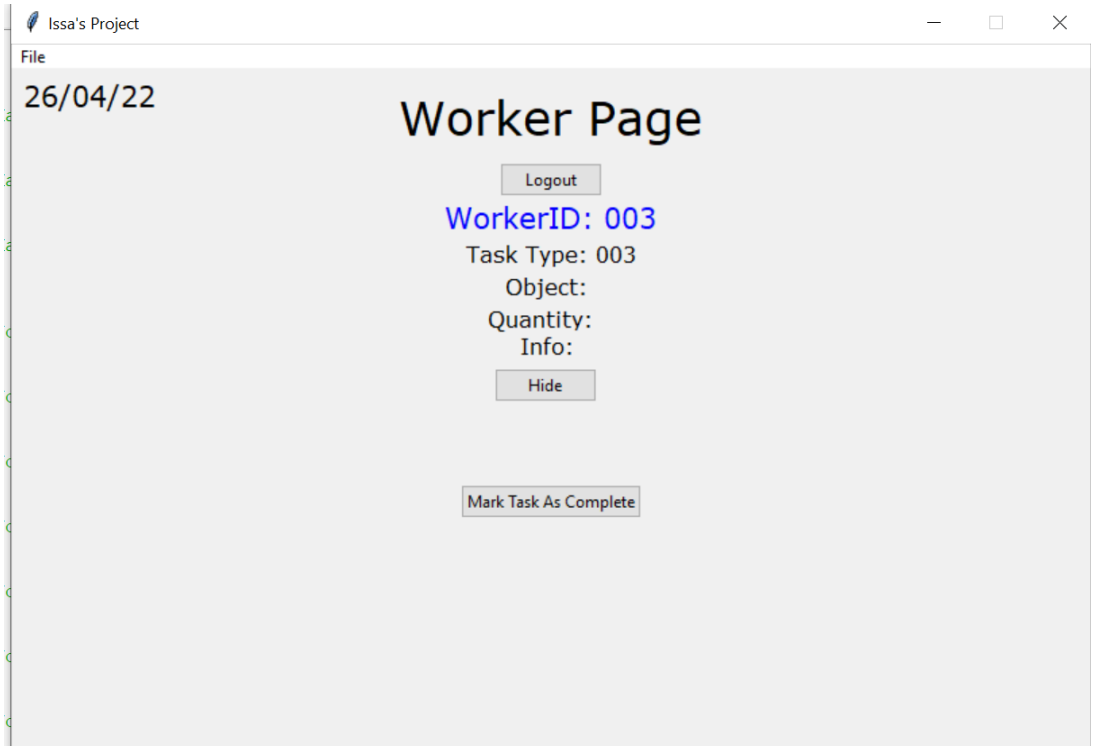


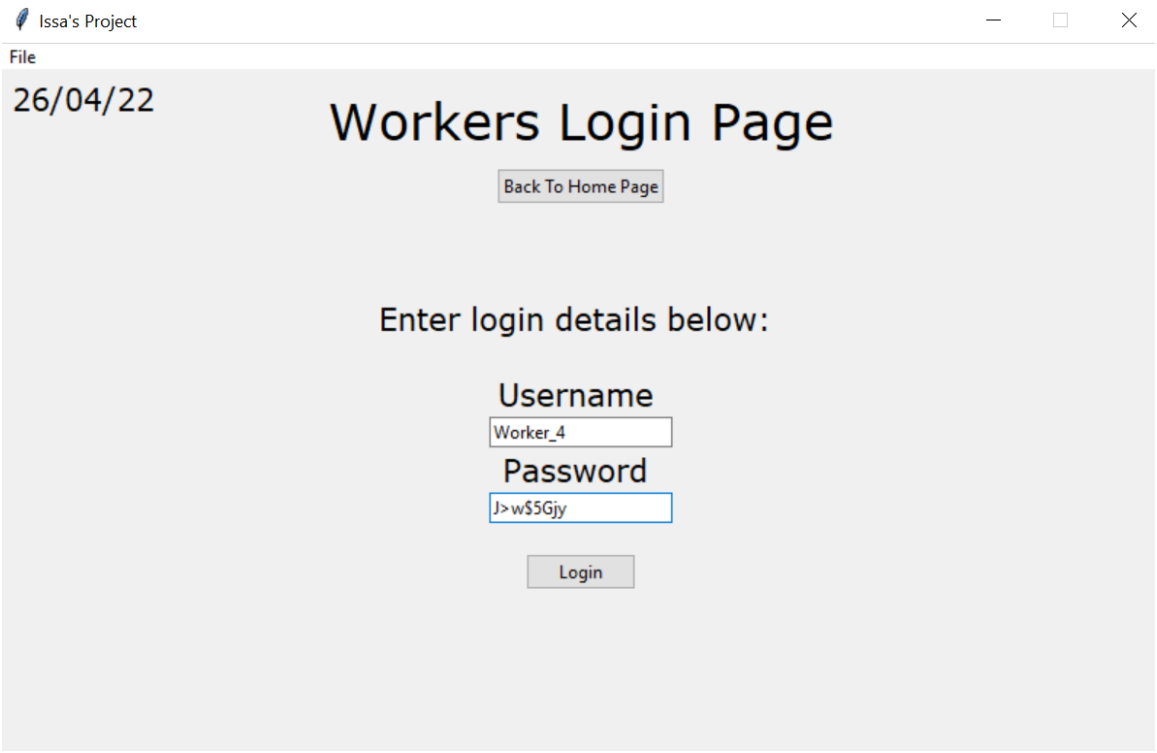


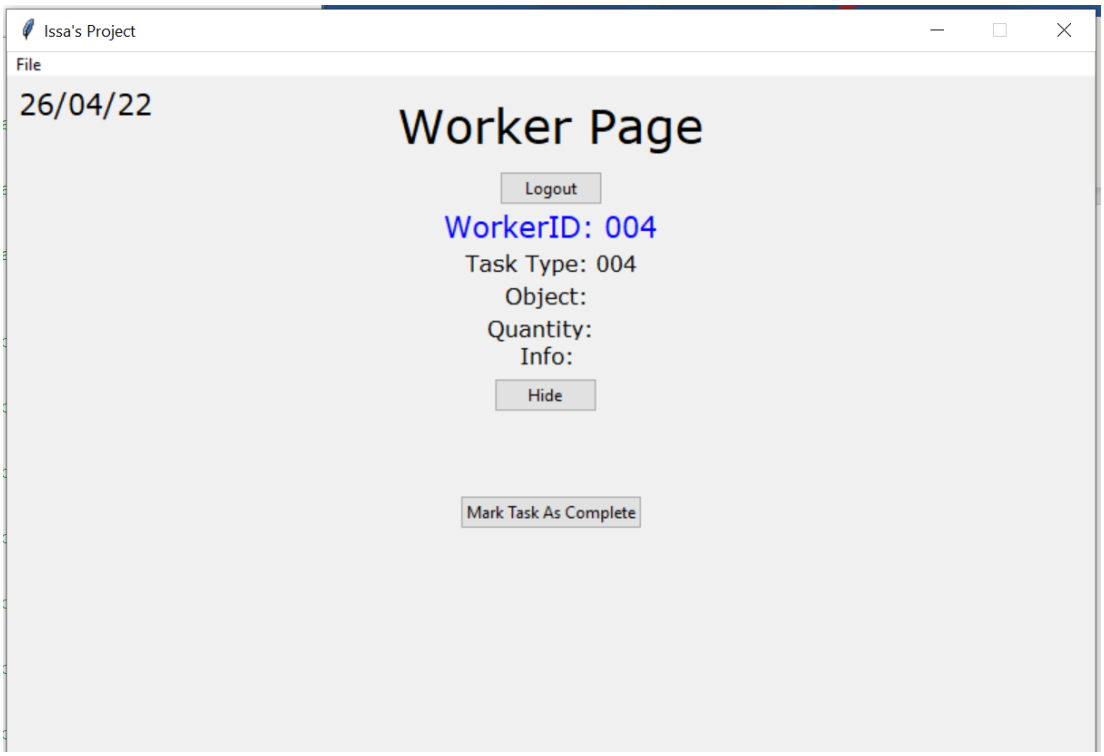

Issa's Project
File
26/04/22
Workers Login Page
Back To Home Page
Enter login details below:
Username
Worker_2
Password
{mz&4=P&
Login

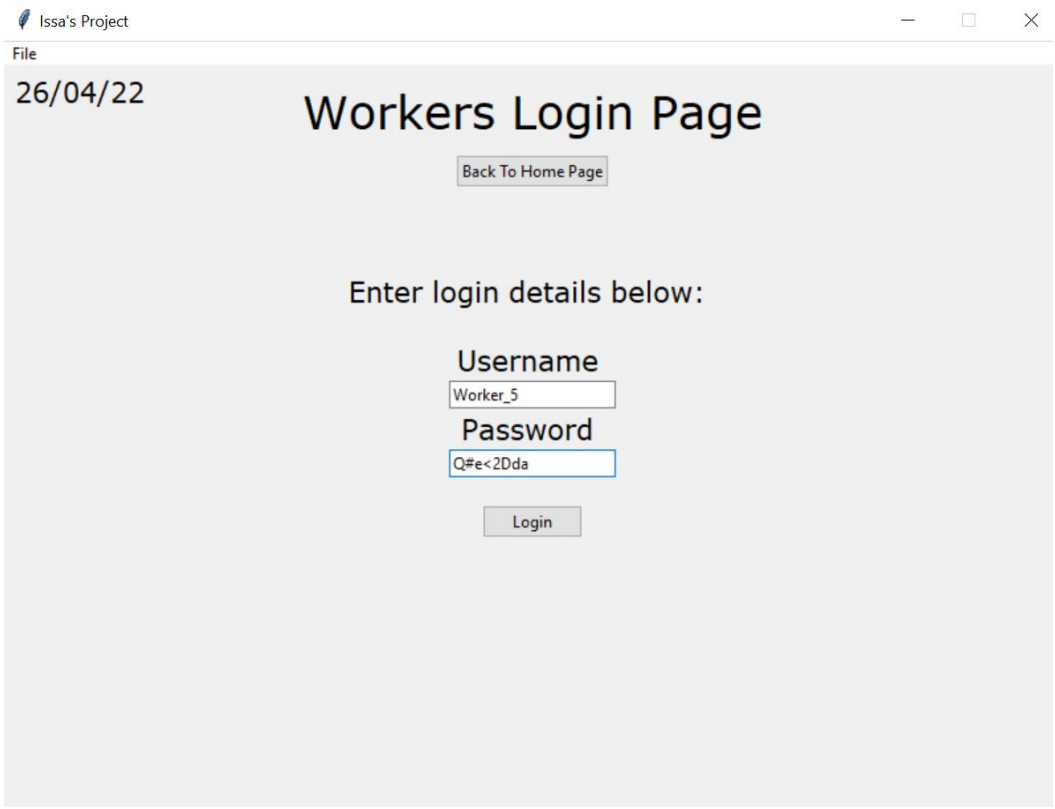

Issa's Project
File
26/04/22
Worker Page
Logout
WorkerID: 002
Task Type: 002
Object:
Quantity:
Info:
Hide
Mark Task As Complete

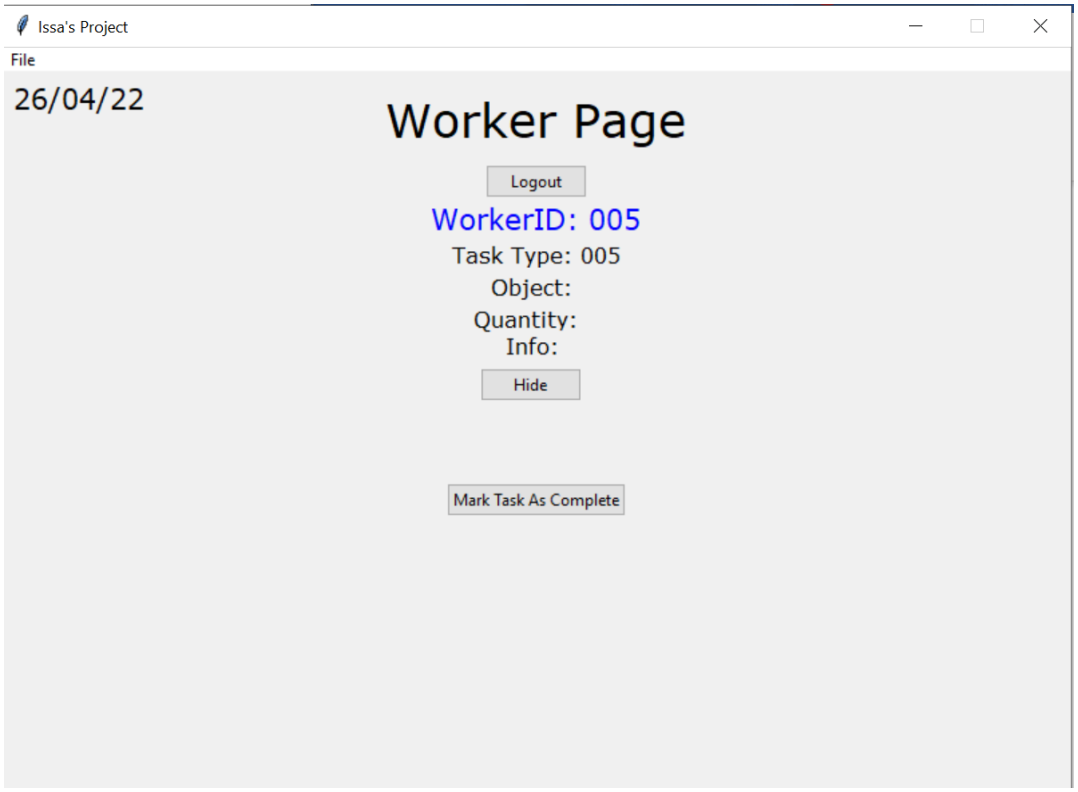

Issa's Project
File
26/04/22
Workers Login Page
Back To Home Page
Enter login details below:
Username
Worker_3
Password
6^Te{`7D
Login

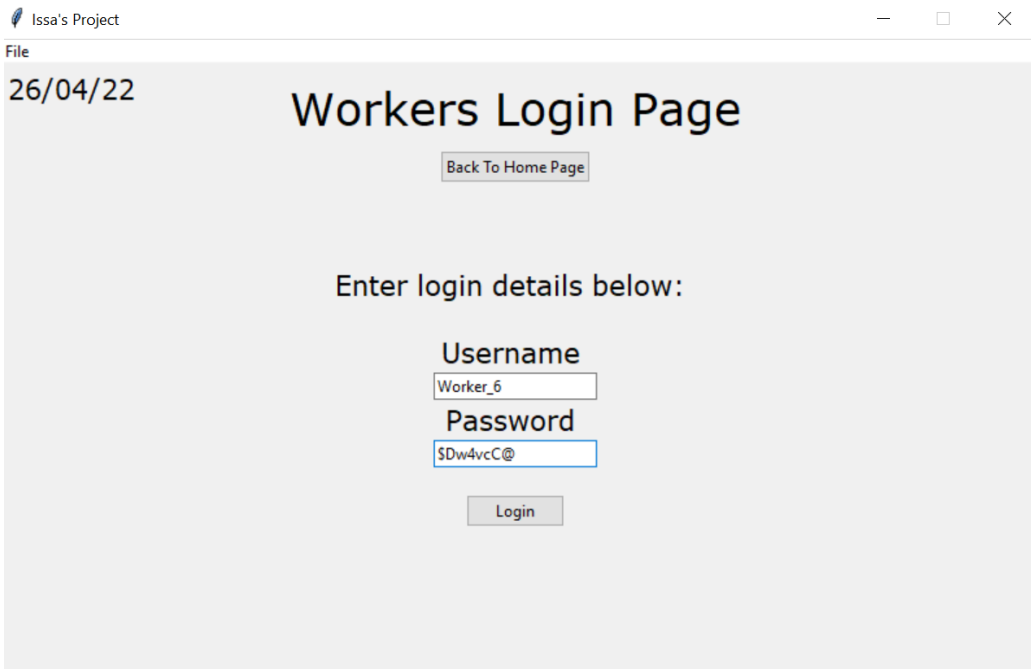

Issa's Project
File
26/04/22
Worker Page
Logout
WorkerID: 003
Task Type: 003
Object:
Quantity:
Info:
Hide
Mark Task As Complete

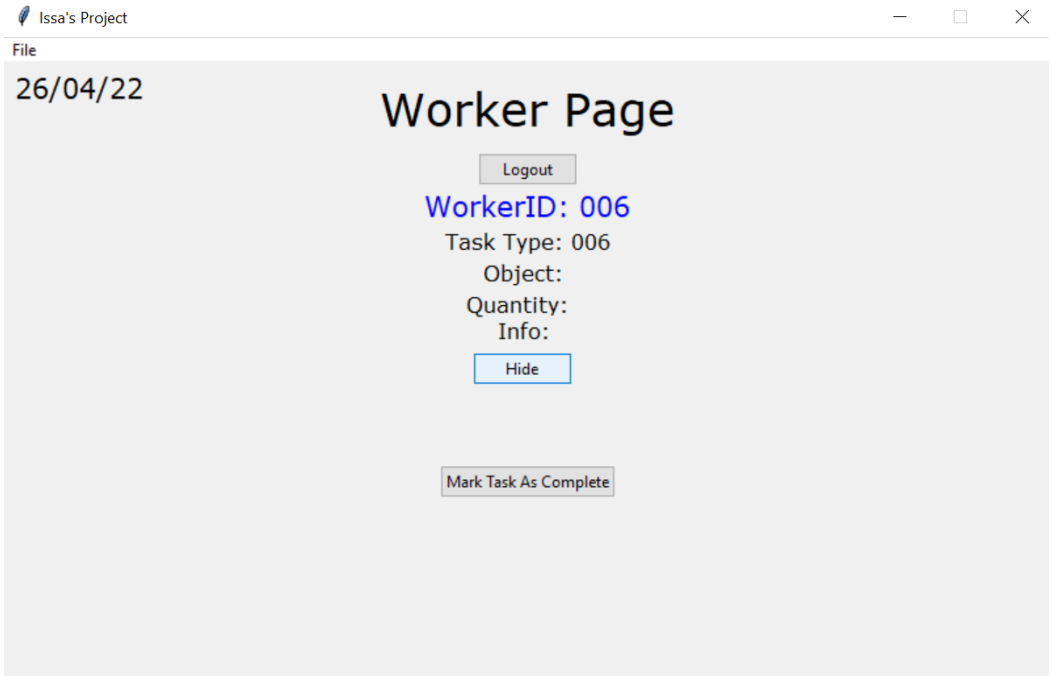

Issa's Project
File
26/04/22
Workers Login Page
Back To Home Page
Enter login details below:
Username
Worker_4
Password
J>w$5Gjy
Login

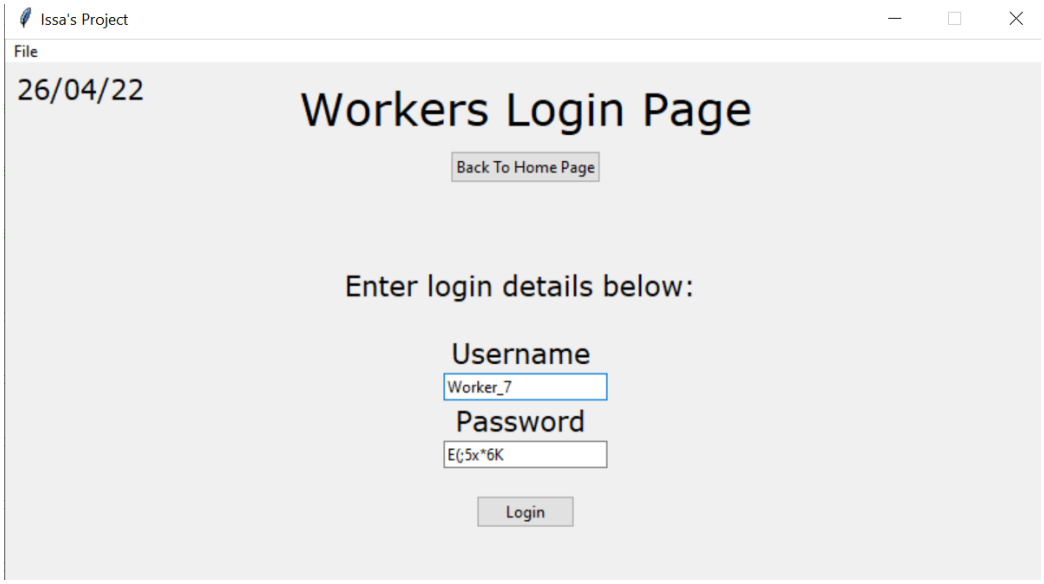

Issa's Project
File
26/04/22
Worker Page
Logout
WorkerID: 004
Task Type: 004
Object:
Quantity:
Info:
Hide
Mark Task As Complete

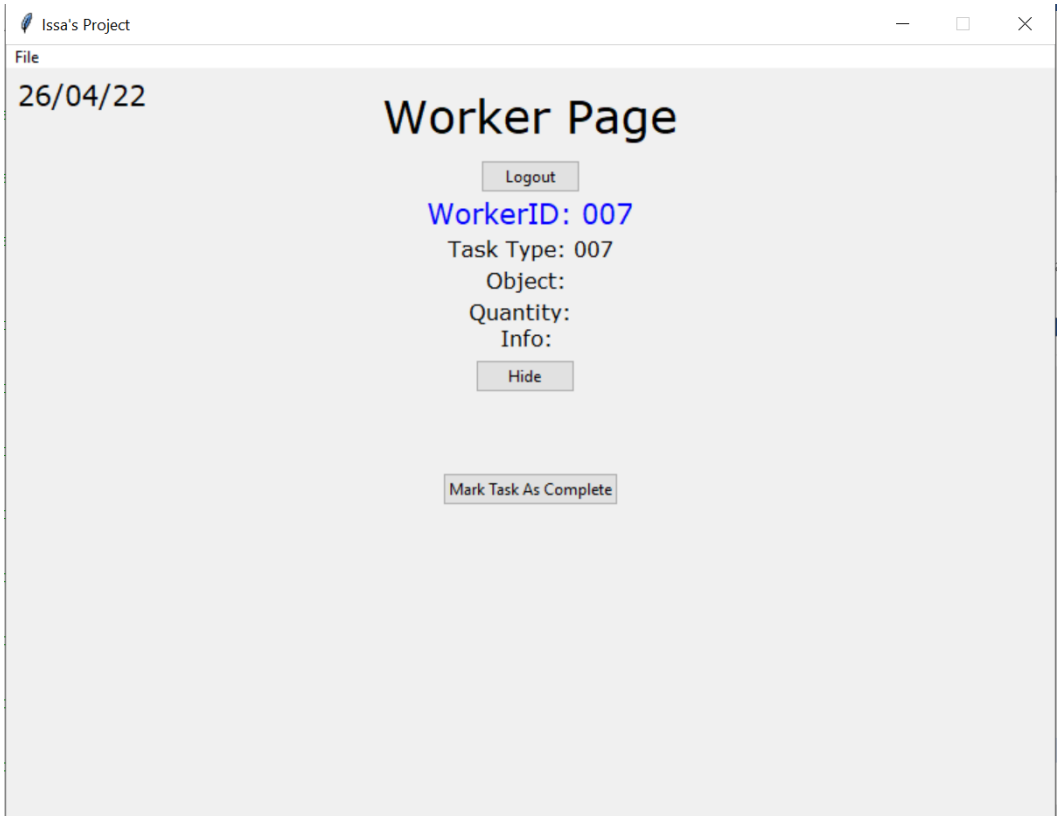

Issa's Project
File
26/04/22
Workers Login Page
Back To Home Page
Enter login details below:
Username
Worker_5
Password
Q#e<2Dda
Login

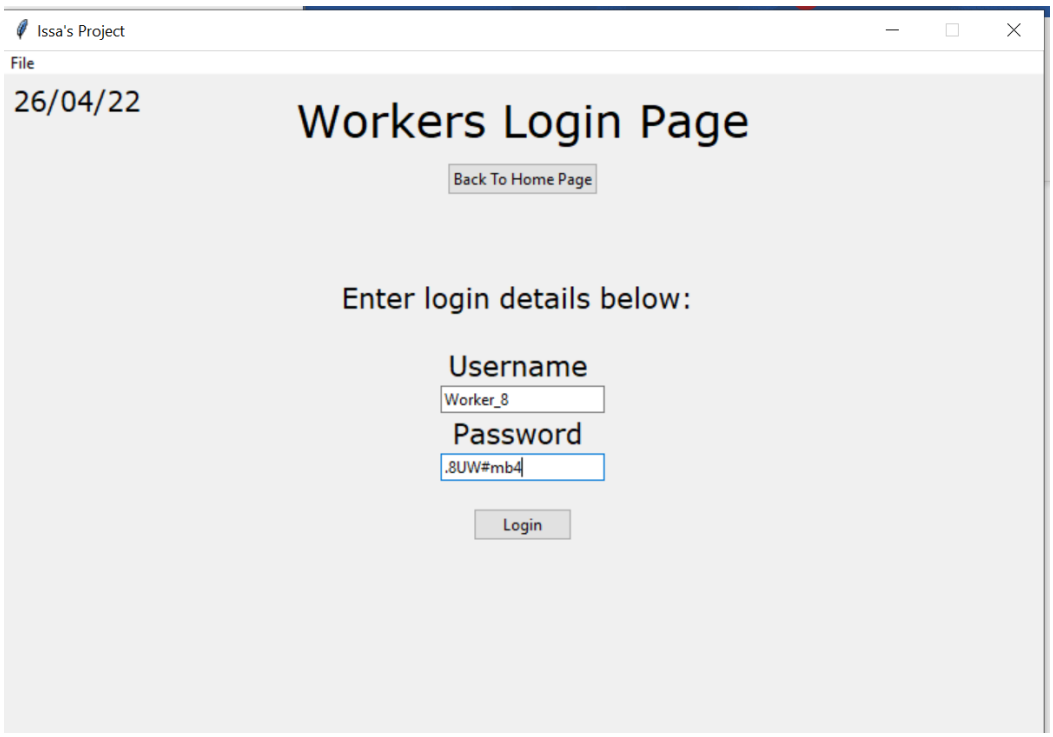

Issa's Project
File
26/04/22
Worker Page
Logout
WorkerID: 005
Task Type: 005
Object:
Quantity:
Info:
Hide
Mark Task As Complete

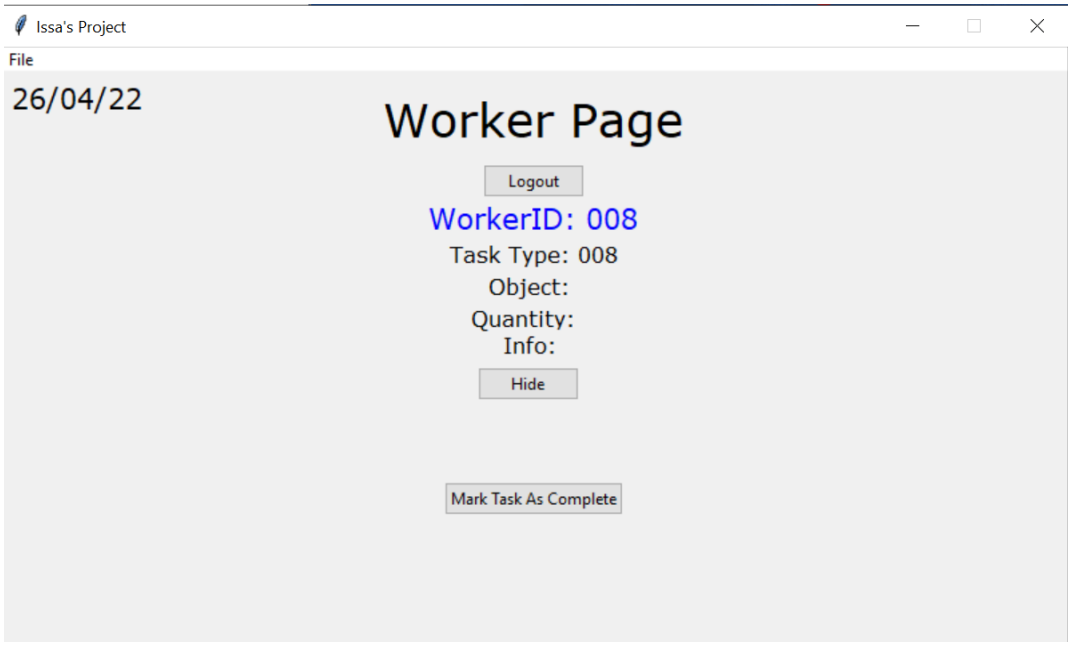

Issa's Project
File
26/04/22
Workers Login Page
Back To Home Page
Enter login details below:
Username
Worker_6
Password
$Dw4vcC@
Login


Issa's Project
File
26/04/22
Worker Page
Logout
WorkerID: 006
Task Type: 006
Object:
Quantity:
Info:
Hide
Mark Task As Complete


Issa's Project
File
26/04/22
Workers Login Page
Back To Home Page
Enter login details below:
Username
Worker_7
Password
E(;5x*6K
Login


Issa's Project
File
26/04/22
Worker Page
Logout
WorkerID: 007
Task Type: 007
Object:
Quantity:
Info:
Hide
Mark Task As Complete


Issa's Project
File
26/04/22
Workers Login Page
Back To Home Page
Enter login details below:
Username
Worker_8
Password
.8UW#mb4
Login


Issa's Project
File
26/04/22
Worker Page
Logout
WorkerID: 008
Task Type: 008
Object:
Quantity:
Info:
Hide
Mark Task As Complete

Test 3

```
    def create_database(self):
            new_path = "Project Database.db"
            connection = sqlite3.connect(new_path)
            data = connection.cursor()
            data.execute("""CREATE TABLE IF NOT EXISTS "Managers"(
            "ManagerID" TEXT PRIMARY KEY NOT NULL,
            "Username" TEXT,
            "Password" TEXT
);""")
            data.execute("""CREATE TABLE IF NOT EXISTS "Workers"(
            "WorkerID" TEXT PRIMARY KEY NOT NULL,
            "Username" TEXT,
            "Password" TEXT
);""")
```

The create_database method contains the SQL statements to create the two tables “Managers” and “Workers” and inserts the login details into both of them aswell as using primary keys. This is being tested.

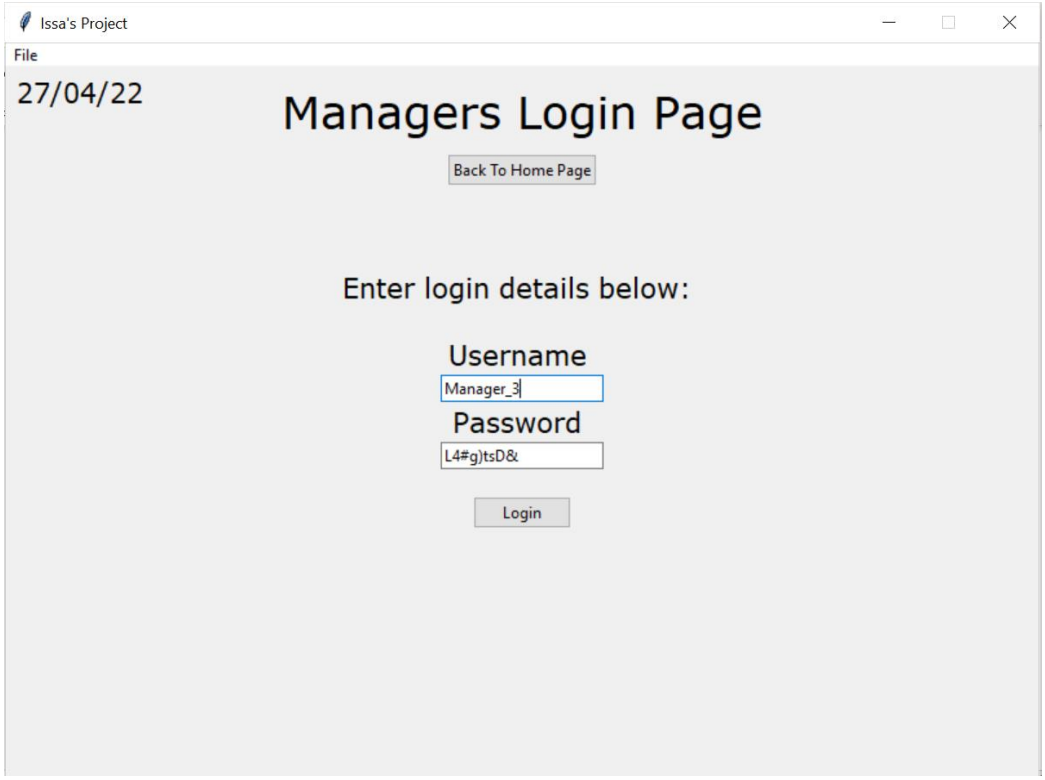
```
            data.execute("""INSERT INTO Managers(ManagerID,Username,Password)
Values("001","Manager_1","8_x4zAFBg")
""")
            data.execute("""INSERT INTO Managers(ManagerID,Username,Password)
Values("002","Manager_2","{Cy:2?MC>")
""")
            data.execute("""INSERT INTO Managers(ManagerID,Username,Password)
Values("003","Manager_3","L4#g)tsD&")
""")

            data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("001","Worker_1","md7P@F)7")""")

            data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("002","Worker_2","{mz&4=P&")""")

            data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("003","Worker_3","6^Te{`7D")""")

            data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("004","Worker_4","J>w$5Gjy")""")

            data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("005","Worker_5","Q#e<2Dda")""")

            data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("006","Worker_6","$Dw4vcC@")""")

            data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("007","Worker_7","E(;5x*6K")""")

            data.execute("""INSERT INTO Workers(WorkerID,Username,Password)
Values("008","Worker_8",".8UW#mb4")""")
```

	WorkerID	Username	Password
	Filter	Filter	Filter
1	001	Worker_1	md7P@F)7
2	002	Worker_2	{mz&4=P&
3	003	Worker_3	6^Te{`7D
4	004	Worker_4	J>w$5Gjy
5	005	Worker_5	Q#e<2Dda
6	006	Worker_6	$Dw4vcC@
7	007	Worker_7	E(;5x*6K
8	008	Worker_8	.8UW#mb4

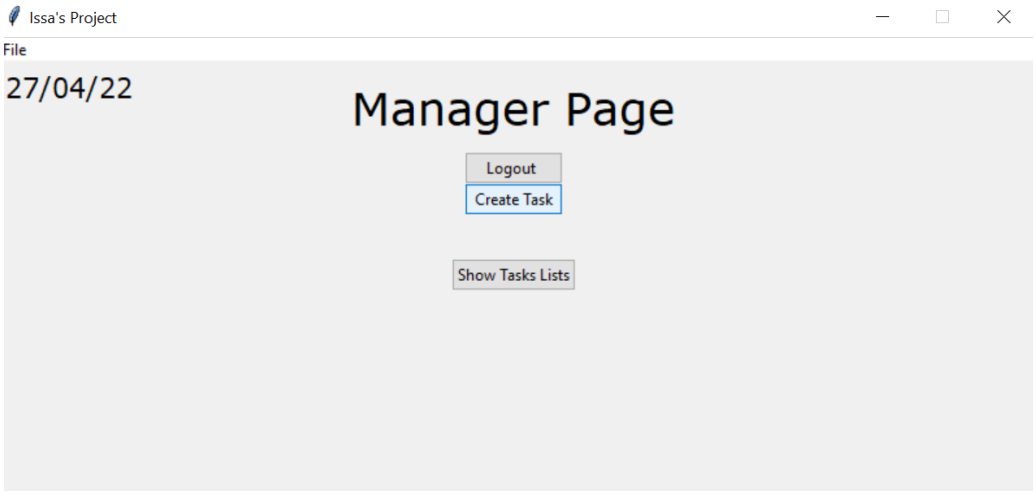
Using DB Browser, the two tables were checked and were found to have all the correct login details in.

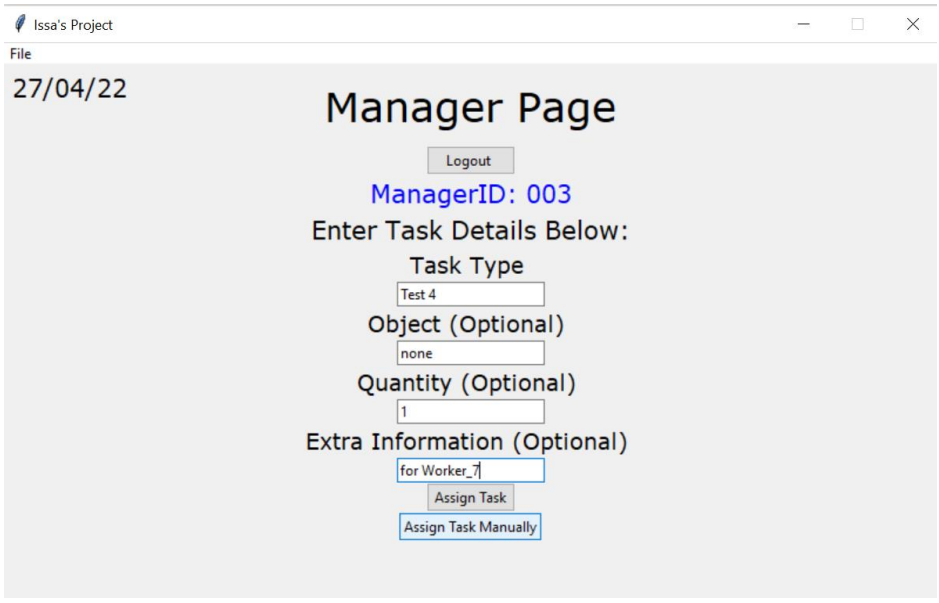
	ManagerID	Username	Password
	Filter	Filter	Filter
1	001	Manager_1	8_x4zAFBg
2	002	Manager_2	{Cy:2?MC>
3	003	Manager_3	L4#g)tsD&

Test 4

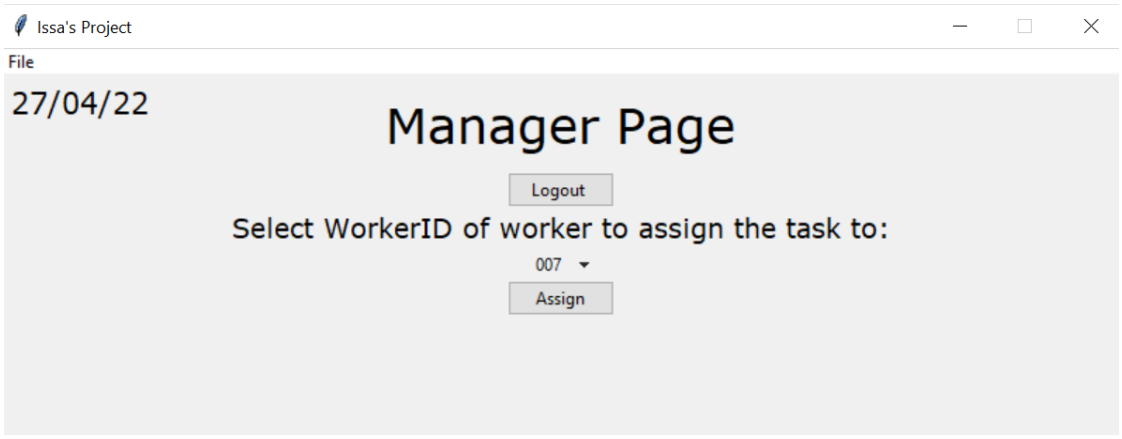


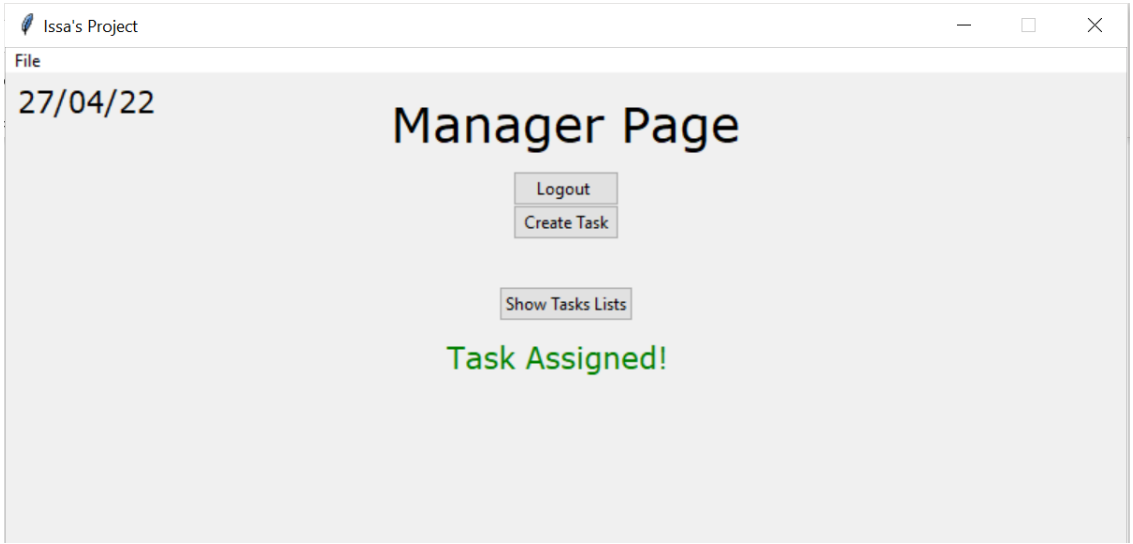
Manager_3 logged in

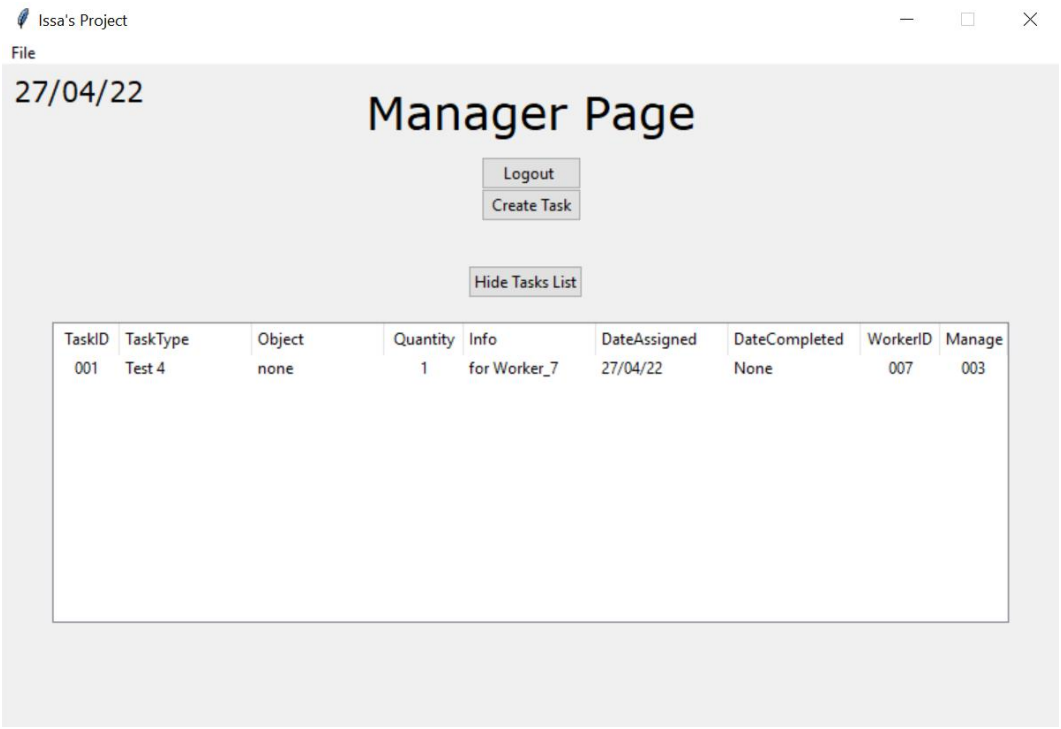




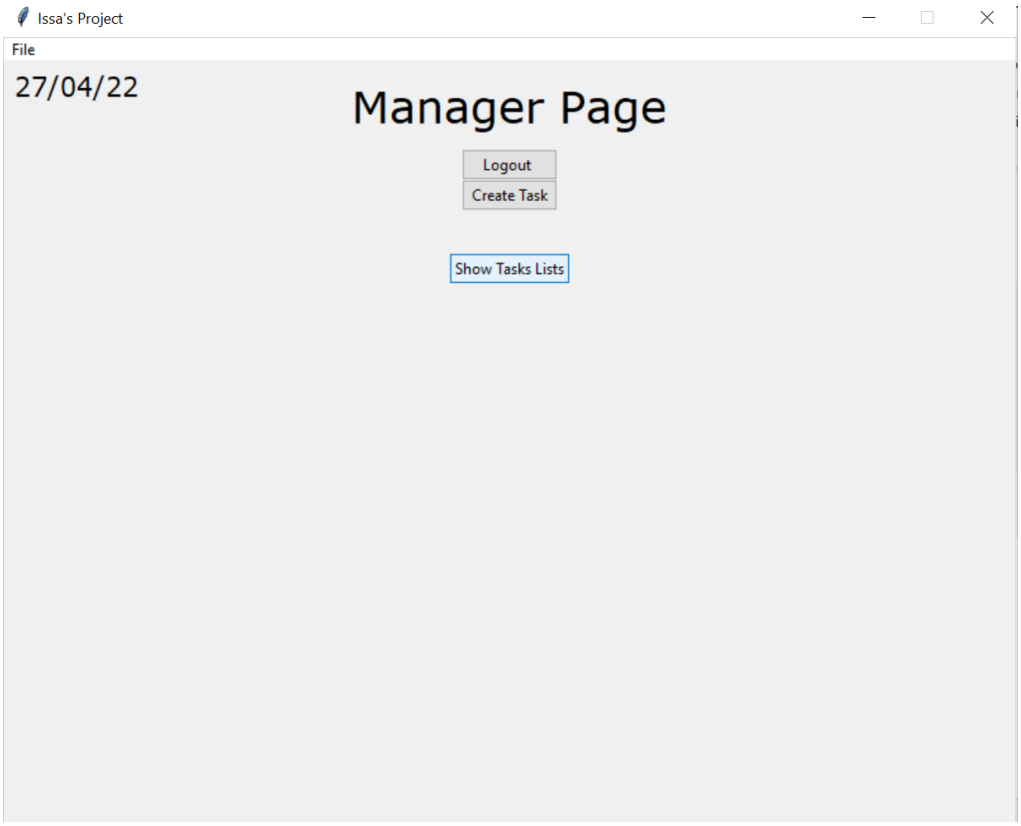
The manager created this task and assigned it manually to worker 7.



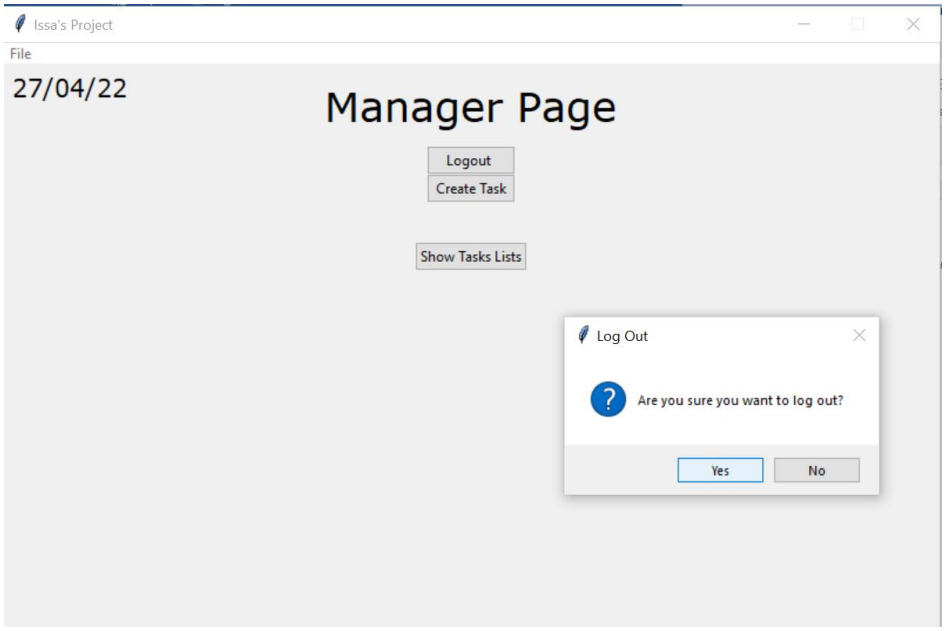




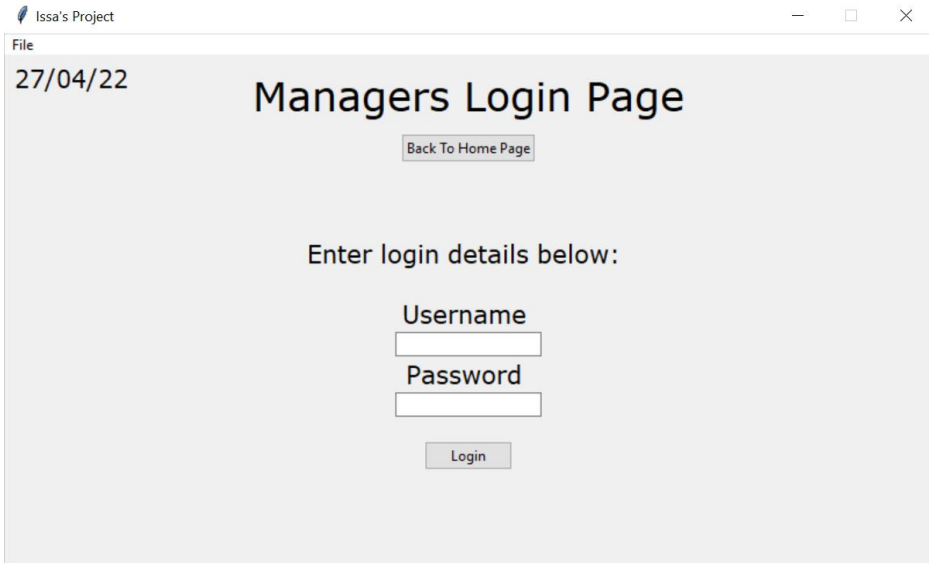
The manager used the Show Tasks List button to view the task that was just assigned and check it was correct.

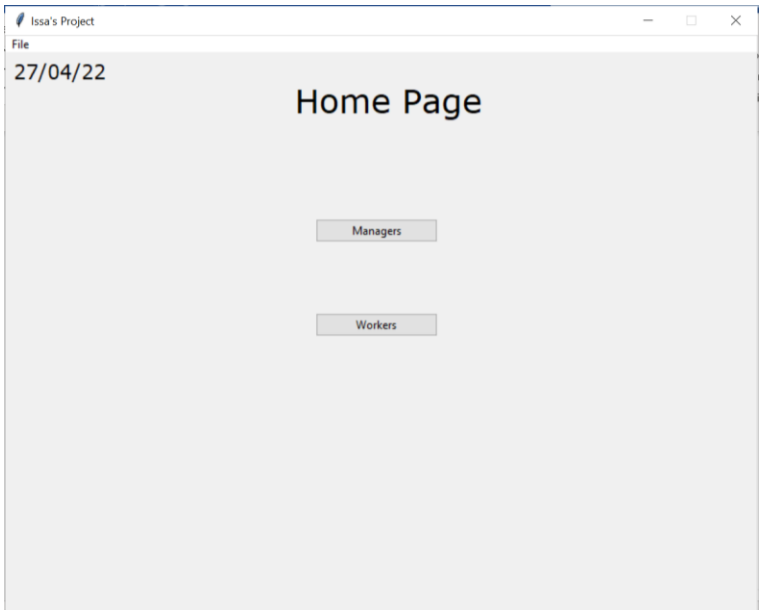


The manager used the Hide Task button to hide the task list

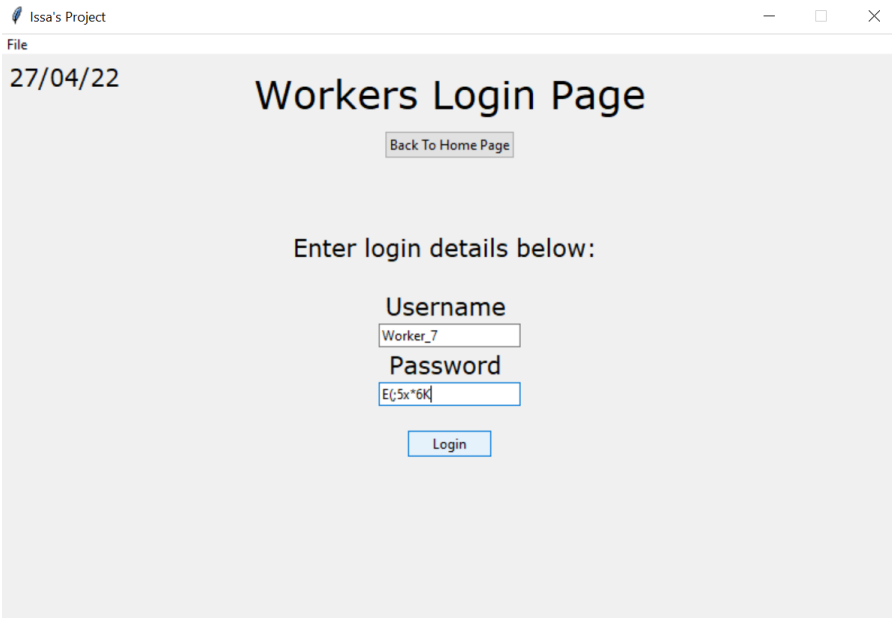


The manager used the Logout button to log out of the page.

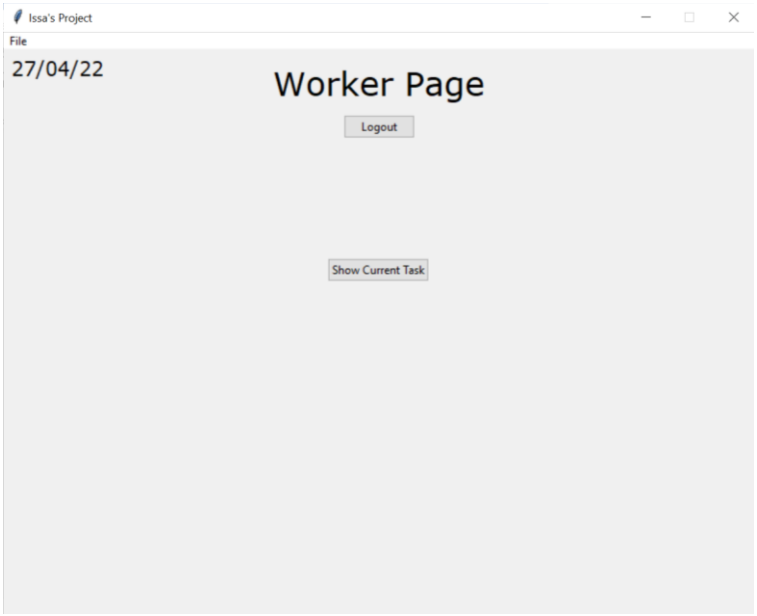


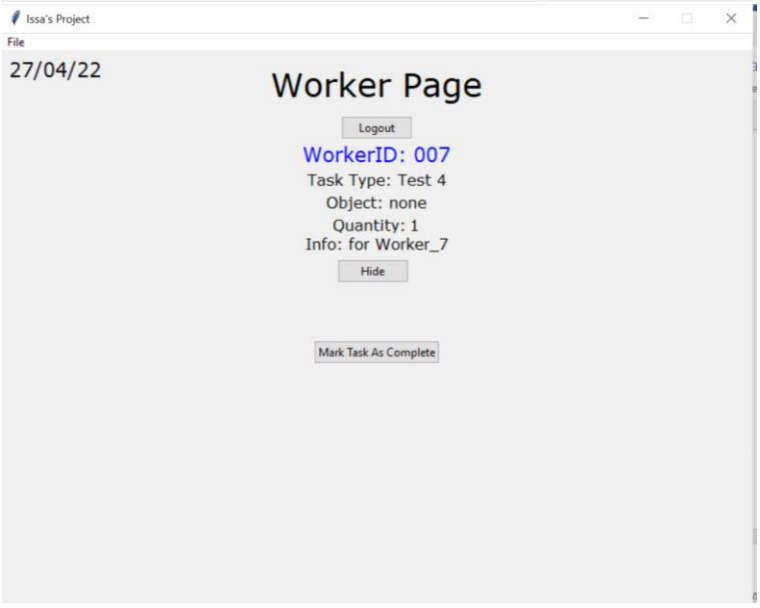


Then used the Back To Home button to go back to the Home Page.

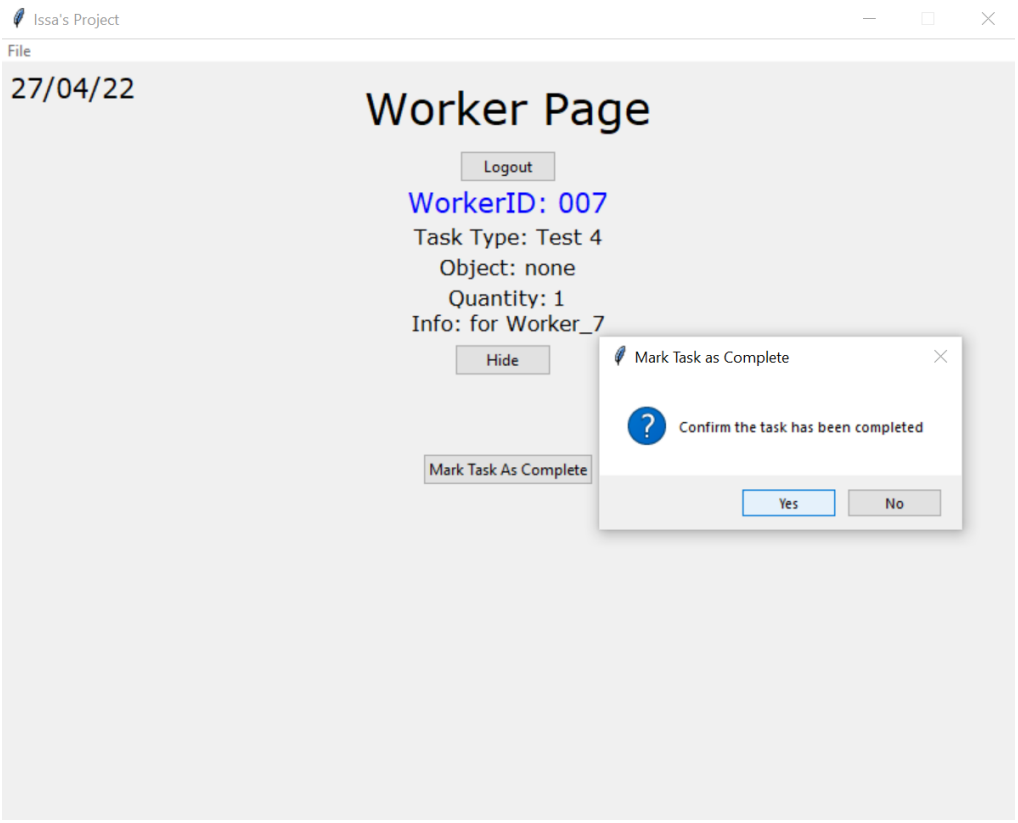


The worker used the Workers button to access the Worker Login Page and then logged in.

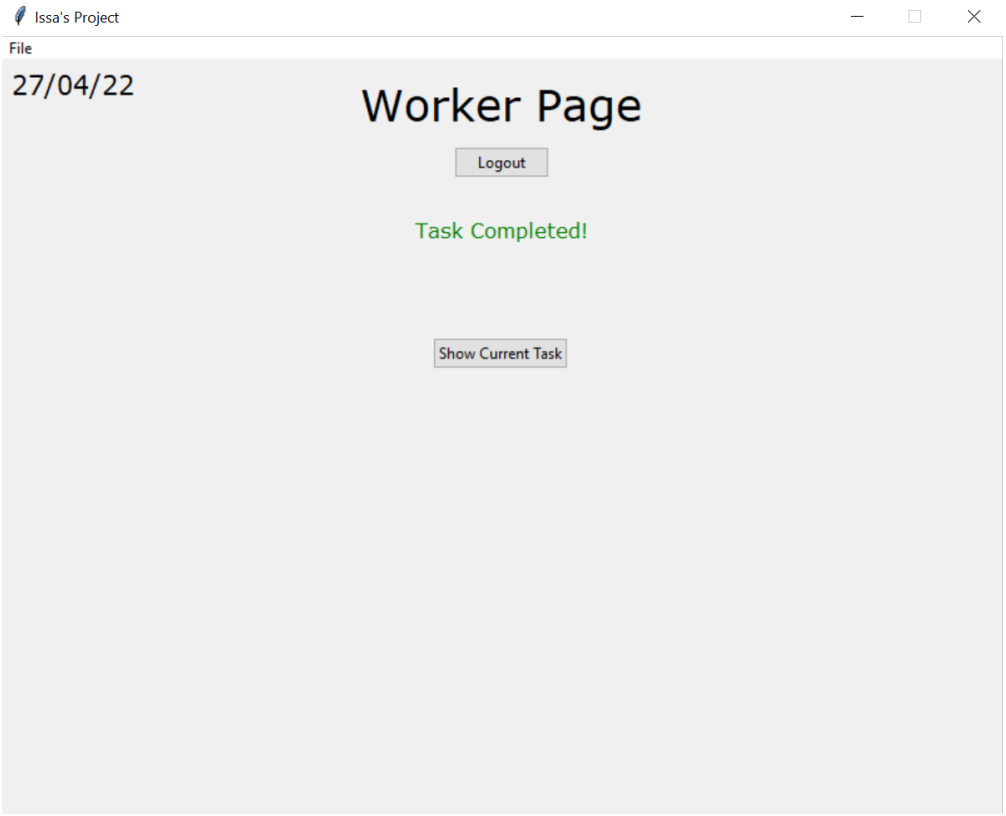


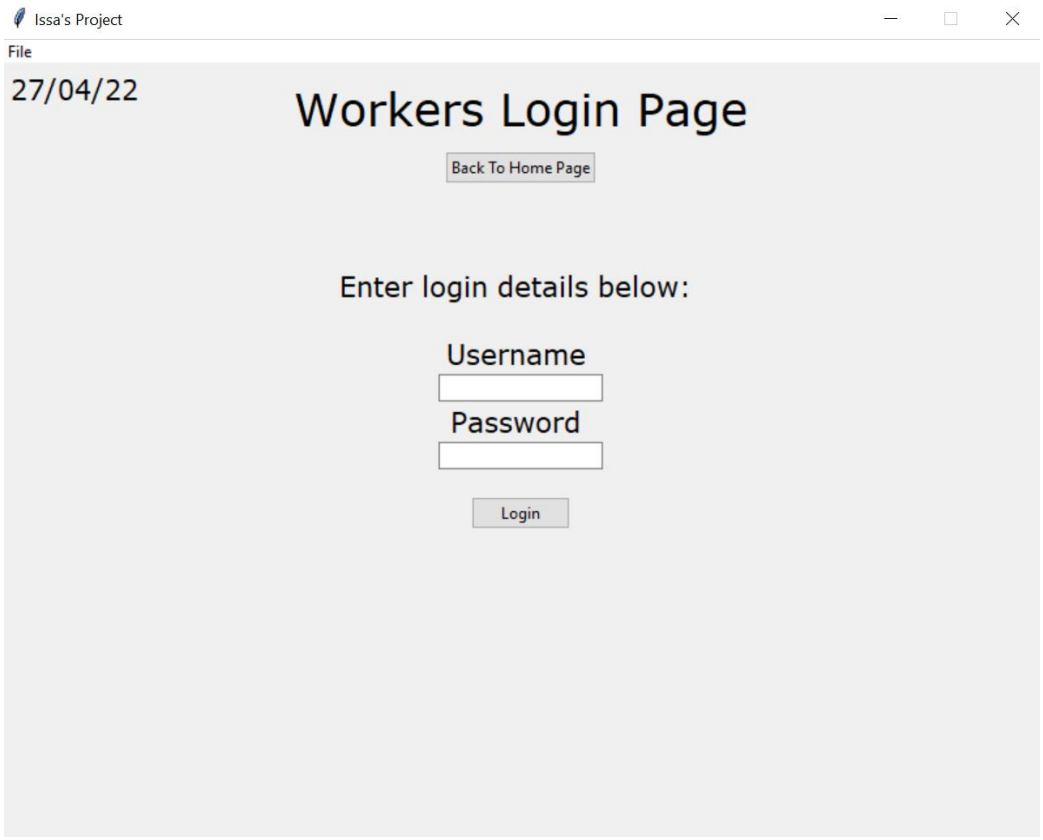


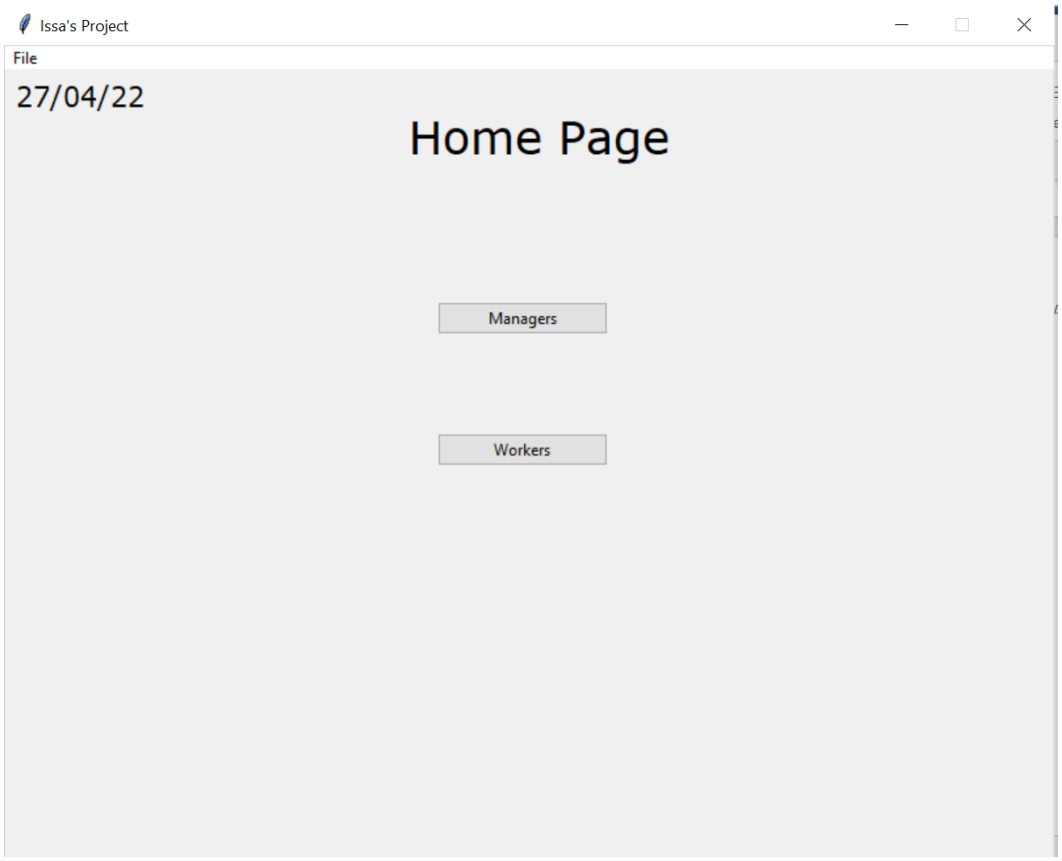
The worker used the Show Current Task button to display the current task assigned to them



The worker used the button to mar the task as complete

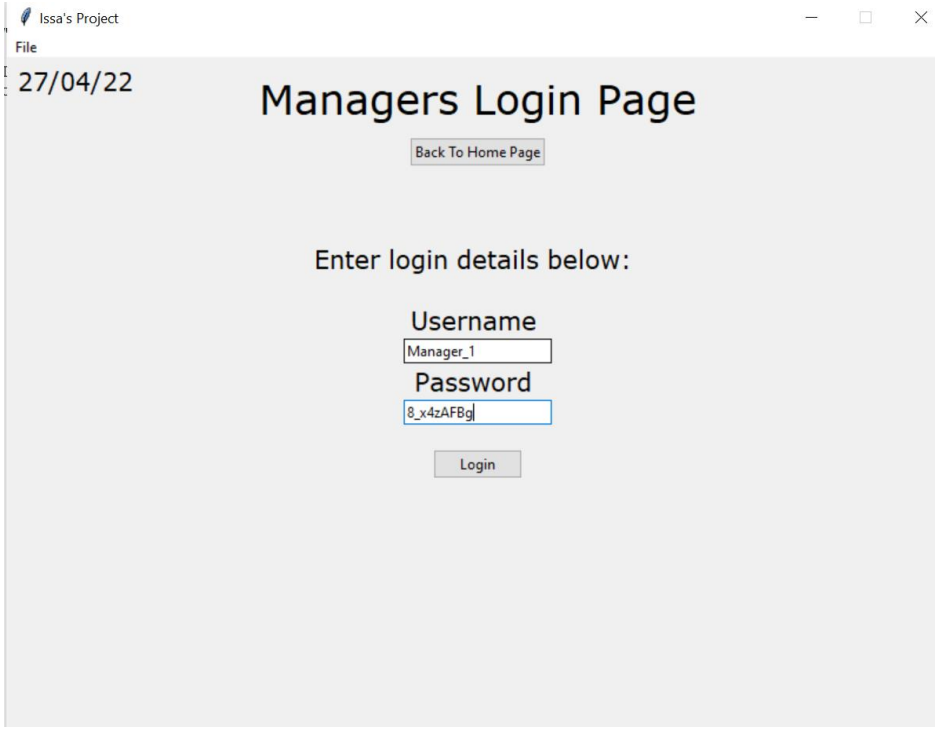




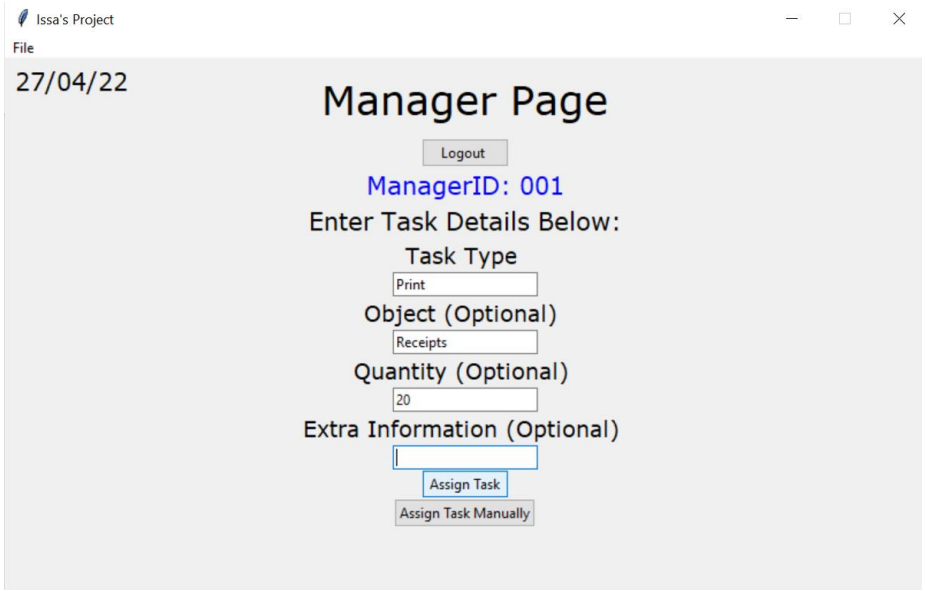


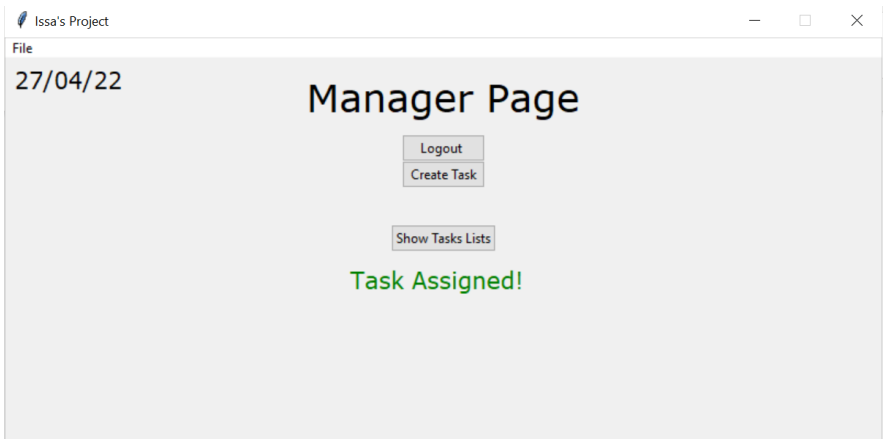
The worker logged out using the Logout button. Then went back to the Home Page using the Back To Home button.

Test 5

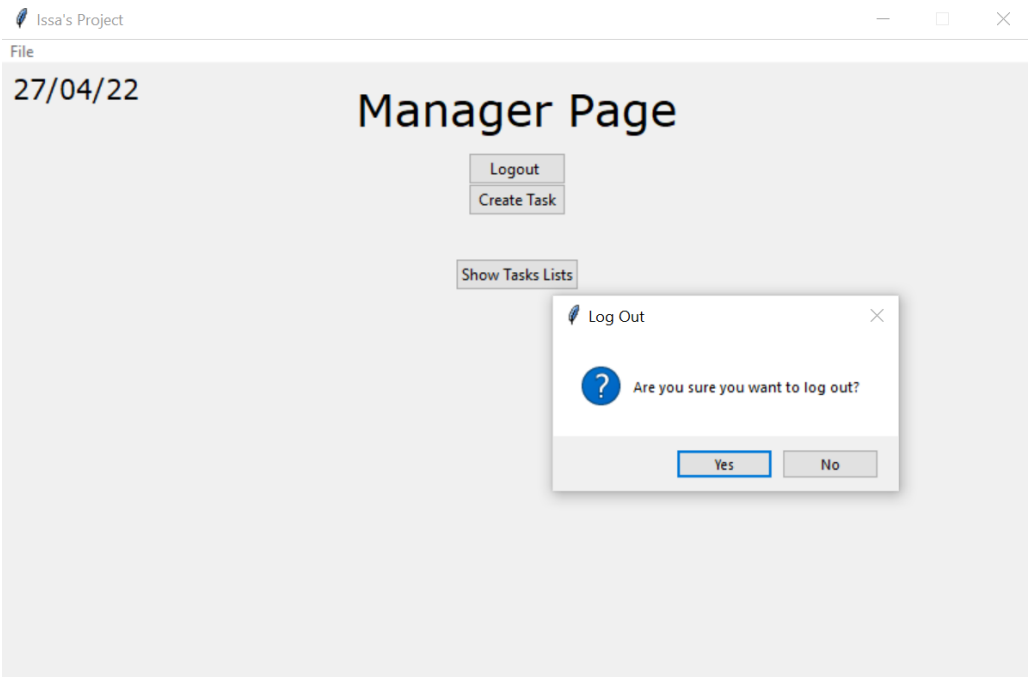


I logged into Manager_1's page.

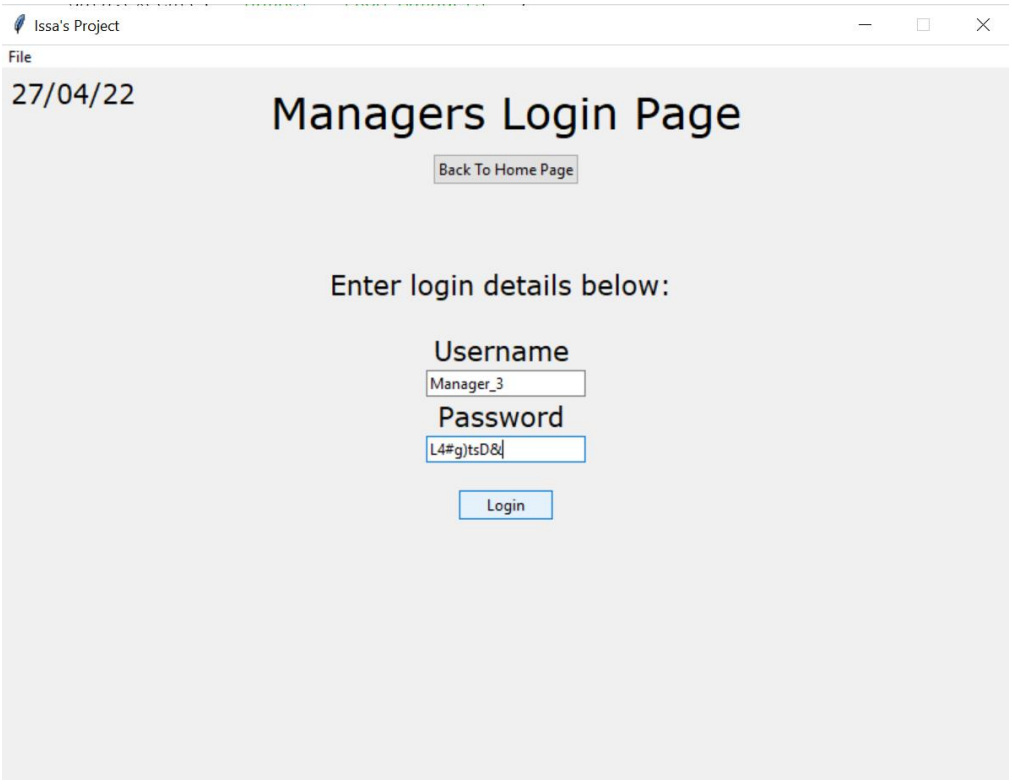




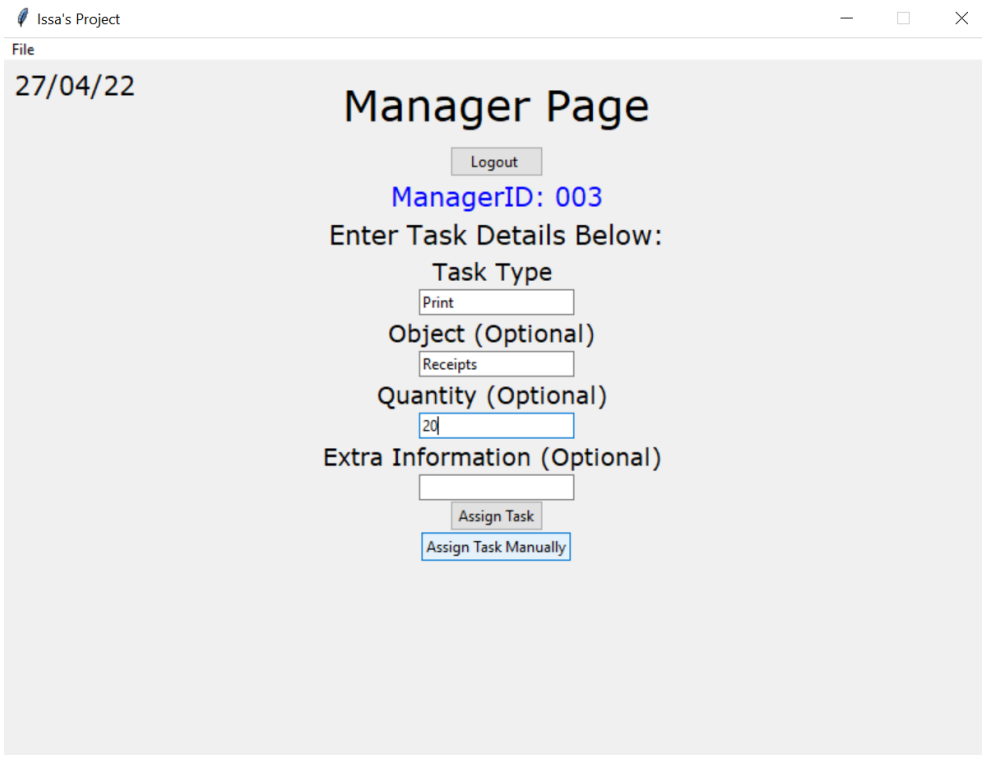
I created the task and assigned it automatically using the Assign Task button

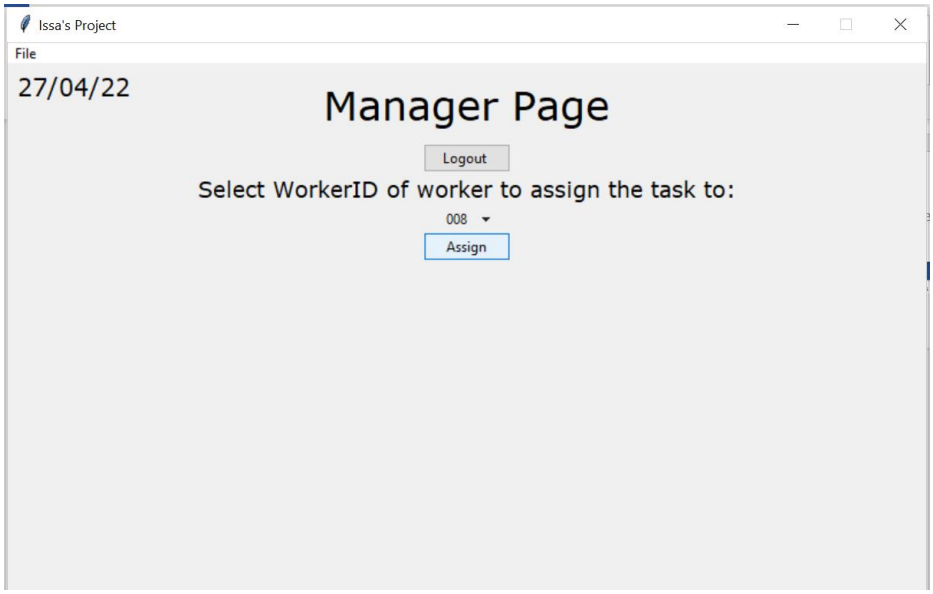


I logged out of the account

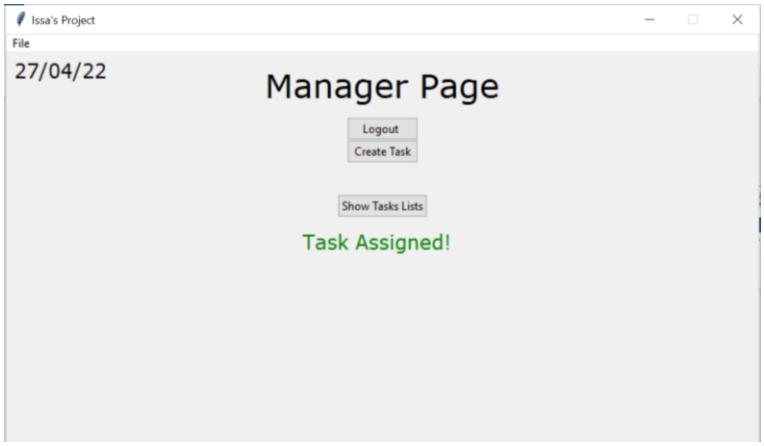


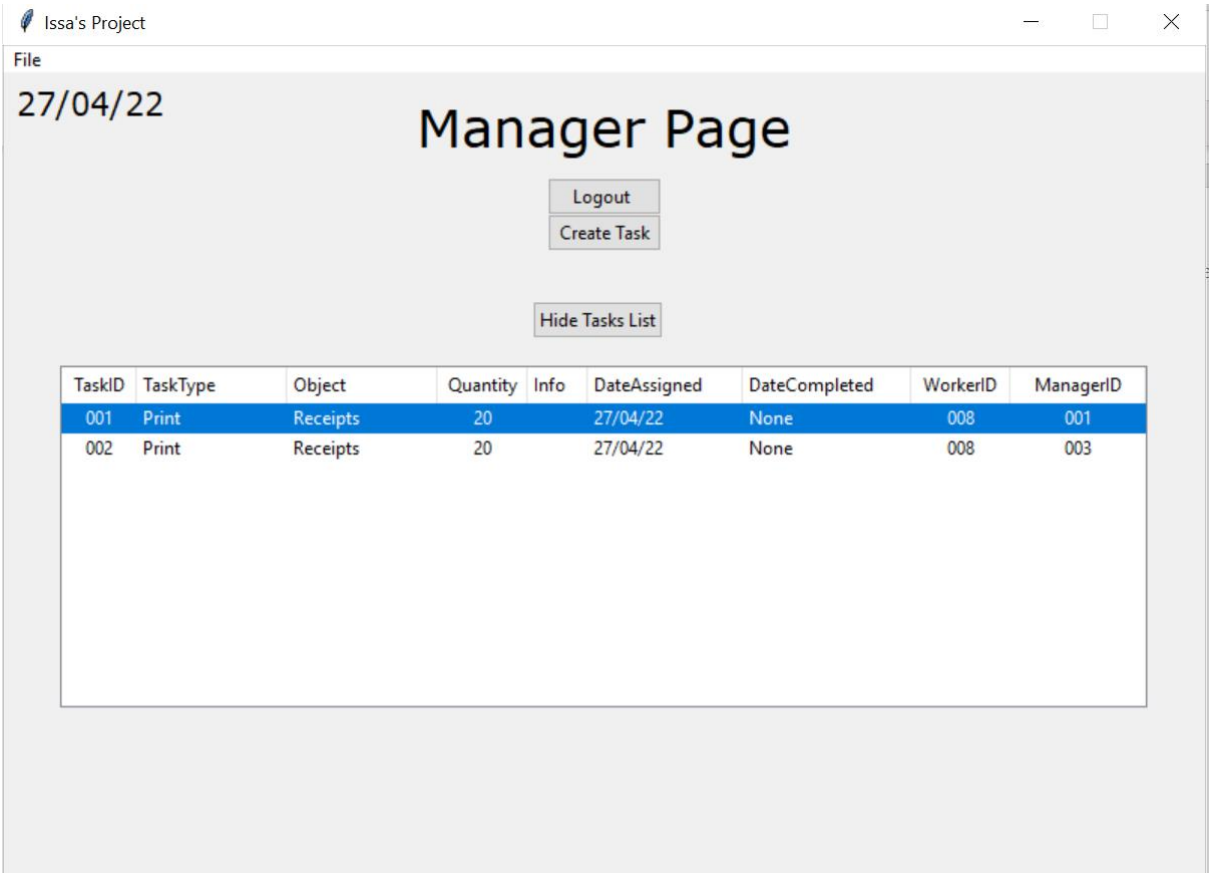
Manager_3 logged in





The manager created the same task and assigned it to whoever they thought was best suited





Using the Show Tasks List button we checked the tasks and both WorkerIDs were 008, meaning they had been assigned to the same worker by both the algorithm and the manager.

Evaluation

Objectives

Objective	Met?	Comments
The code utilises a proficient algorithm which analyses any relevant task and correctly assigns it to the most suitable person	Yes	From consulting the user after the end user testing, we decided that the objective was met but only to a certain extent. The central algorithm was accurate in picking out the correct person for the task but if it was to be used in practice, to assign a large volume of tasks it would lose accuracy so there is definitely lots to improve.
The Graphical User Interface is clear and easy to use whilst also being complex enough to be used professionally	Yes	This objective was met very well. The user found the Graphical User Interface very clear and understandable to use. The colour scheme helped maintain a suitable amount of professionalism.
Contains a user login system where each user has their own personal login details to access their own page which looks and acts uniquely for each user	Yes	The objective was definitely met at a high level. The user was very satisfied with the login system, the usernames were appropriate the passwords were randomly generated so more secure and the system dealt with incorrect entries very well.
Stores user login details within a table in SQL	Yes	The objective was met, the correct user login details were stored in tables using SQL which were created within the code.
Stores information about the employees within a table in SQL which is used by the algorithm	Yes	The objective was met. The code allows the manager to enter in the details of the workers which gets stored into the table. The details from this table are accessed by the algorithm and used to decide a worker for a task.

Overall, all of the objectives were met. 4/5 of the objectives were met to a good level with the first objective being met for now but would need improvement for the system to be used further.

Possible Improvements

Given more time and opportunity to improve the software I would firstly improve the central algorithm. I would take into account more factors from worker details such as Age and Job Role when coding the algorithm so it has as much information as possible about the workers when deciding which one to choose.

Another improvement that I would make is to implement a system where new user accounts can easily be created within the code. Manager accounts can be created by entering a new password and the Manager ID and username will automatically be incremented and then these will be stored in the Managers table. Worker accounts would work the same way but worker details would also need to be entered in. I have already coded a method to enter worker details in my code so this can be used

Finally, my last improvement would be to improve the appearance of the Graphical User Interface. Although it functions well, it still looks a bit plain and boring so I would want to improve the colour scheme and style of the pages.

However, although there are some possible extensions to add to the program if there was extra time, the user feedback shows that the program was a great success and met the SMART objectives.

User Feedback

After the completion of the program, I then asked to follow up questions to ensure the primary client was satisfied with the program. Below are the responses to the questions:

Was the program easy to set up and use?

I believe the program was easy to set up and use. I used the program on many devices such as windows 7 system and a windows 10 system. On both devices the program ran smoothly with no crashes or errors. It was easy to use as the interface was informative, clear and guided us on how to use the program. It was also easy to set up as I could enter the worker details in straightaway and the database was created automatically upon start up.

Are there any other improvements you think would be suitable to this program?

I believe one improvement that could be made is communication between the workers and managers. The manager could be alerted as soon as a worker that they assigned a task to completes it. Or the worker could respond to a task with a message in case there's anything that needs communicating back to the manager.
Apart from that the low-cost software satisfied all our needs.

Was there a time that you struggled to use the program?

On one occasion an worker was trying to log in and forgot their password. The passwords are random characters so are good for security but it makes them harder to remember. This was a problem as there was no way for the worker to check their password. An improvement I would recommend to overcome this would be a forgot username or password button to reset or change the password.

Analysis of User Feedback

Overall believe that the user feedback was generally positive, therefore I can make the ultimate judgement, that the program was a success. The user feedback did also highlight some areas that could be developed further such as improving customisation and adding communication sections. These areas can easily be added to the program with more time, and do not need many complex algorithms.