A-Level Course Work

Contents

**List of Figures .................................................................................................3**

**Analysis..........................................................................................................4**  
Problem Identification...................................................................................................4  
Stakeholders ..................................................................................................................4  
Existing Solutions...........................................................................................................6  
Requirements ................................................................................................................8  
Success Criteria............................................................................................................10

**Design ..........................................................................................................11**  
Problem Decomposition ..............................................................................................11  
User Interface Design...................................................................................................11  
Usability Features ........................................................................................................15

List of Figures

Figure 1 Unleashed’s Sales Orders Interface.................................................................8

Figure 2 Unleashed’s Add Product Interface.................................................................8

Figure 3 Fishbowl’s Sales Orders Interface....................................................................9

Figure 4. Decomposition of the main window ........................................................................12

Figure 5. UI mock-up for the Stock Tab ...................................................................................13

Figure 6. UI mock-up for the Customers Tab ..........................................................................13

Figure 7. UI mock-up for the Orders Tab.................................................................................14

Figure 8. UI mock-up for the Help button ...............................................................................14

Figure 9. Decomposition of the add dialog .............................................................................15

Figure 10. UI mock-up for add customer dialog ......................................................................15

Figure 11. UI mock-up for add stock dialog .............................................................................15

Figure 12. UI mock-up for add order header dialog ................................................................16

Figure 13. UI mock-up for add order line dialog .....................................................................16

Analysis

# Problem Identification

A company is struggling to keep track of its current stock and the orders that need to be processed. Being able to store all the stock, customers and orders electronically would help the organisation of the business . It could help identify what orders need to be processed, provide contact information if there is any need to contact the customers and warn the company when stock is getting low.

Storing the information on a computer is vastly better than trying to keep track of physical copies of the data. This is because it would be extremely hard to search, copy or analyse the records if they are not stored electronically. It also increases chances of data being lost if the only copy of the data is a physical copy.

## Computational Approach

The problem lends itself to various methods such as problem decomposition when designing this solution. This allows the problem to be broken down into smaller tasks such as calculating order prices, adding or updating records, searching for information and notifying the business when the stock is low. The solution could use object-oriented programming so windows can inherit features and functions and use classes control behaviour of dialog. It could also use N-Tier architecture with a front-end GUI and an abstracted interaction with a database using a data layer.

## Main Solution Features

The database should have a customer, stock and orders table to keep track of the most essential information. The customer table will allow the company to record customer information and apply discounts to certain customers. Keeping track of stock, without having to do stock takes, helps notify the company when they need to order new stock and keep track of which items are the most popular. The orders table will help show all orders that need to be or have been processed and calculate the amount that would need to be paid. It must have an interface for the users to interact with it.

# Stakeholders

The clients for this project will be a small bookstore that sells books to distributers. The users will not necessarily have a in depth knowledge about computers and the use of databases so it must be very user friendly. This will allow the customers to use the database with little to no training making it very accessible. The company would also like to keep track of how many orders have been placed by specific customers so they can use that information when deciding if they should give that company a discount on their prices.

## Main Users

The main users of the system would be warehouse pickers and the sales team. This is because in the warehouse they need to check what orders they need to pick and what orders have already been processed. The sales team will need to use the database to add new orders and customers to the database and check or update discounts.

I decided to ask 5 of the main users questions the following questions to find out how they want the interface to look like and what they want to be able to view. The 2 warehouse pickers I asked where John and Samantha and the 3 members of sales team I asked were Thomas, Katy and Matt.

1. Do you have any experience using databases, if so what?
2. How do you want to access or input data into the database?
3. What layout do you want the interface to have?
4. Is there anything else you want to add?

Question 1 will allow me to find out how confident the people using the database are so I can ensure all they will be able to utilize the final solution.

For question 2, it will inform me how to show the information stored in the database and incorporate the add feature into my interface.

The third question will inform me on how to design the interface and how the main users would like to navigate it.

In the last question, I will check if there is any additional input the users want to give at this stage.

## Main Users Interview

Q1: “Do you have any experience using databases, if so what?”  
John: “Yes, I used one in my last job but I am not very confident with using databases”  
Samantha: “No, I have never used a database before”  
Thomas: “No, I don’t have any experience with using them”  
Katy: “Yes, but I have used one before but only to enter new orders”  
Matt: “Yes, I am fairly confident with using databases as long as they have a clear interface”

Therefore, the interface must be simple to interpret and easy use so it can assemble to all the main users regardless of previous experience.

Q2: “How do you want to access or input data into the database?”  
John: “I would prefer if the data is shown in tables and uses buttons to add data”  
Samantha: “I think the interface should include textboxes to type in the information”  
Thomas: “I would like there to be a separate window pop up to enter the information”  
Katy: “Using textboxes and drop downs will be the easiest way to input data”  
Matt: “The interface should have dialogs to enter data and tables to show the current information”

In my interface I will display the data in tables with buttons to create a dialog to enter new data.

Q3: “What layout do you want the interface to have?”  
John: “I would like the stock and orders to be displayed on different pages”  
Samantha: “The interface should allow you show different tables on different tabs”  
Thomas: “The data should be displayed in separate tables”  
Katy: “I want the interface to have separate labelled tabs for each of the section”  
Matt: “I would like there to be one table visible at any time”

I will use different tabs or pages to separate stock, customer and order tables. This will also help keep the interface not look too cluttered.

Q4: “Is there anything else you want to add?  
John: “No”  
Samantha: “I can’t think of anything else to add”  
Thomas: “No”  
Katy: “Try to make the interface very spacious”  
Matt: “No I don’t have anything else”

I will make sure my interface design is very spacious.

## Senior Users

I also decided to ask some questions to 3 senior users so that I could get a better idea of what features the stakeholders wanted out of my solution. I will ask the purchasing manager: Bethany, operations manager: Anna, and sales director: Mark.

1. Would you like the system to calculate order prices?
2. Do certain customers have discounts that need to be applied?
3. How would you want to sort the data?
4. Are there any other features you want the solution to have?

Question 1 will check if I should include this feature in my design.

In question 2 I need to find out if discounts need to be applied when calculating order prices and if they need to be stored in the customer table of the database.

This will help me find out what ways you must be able to sort the tables by in my final solution and me a better idea of what are the most import sort features to the stakeholders.

Question 4 will allow the senior stakeholders to suggest any other feature my design should include that they might find helpful and improve my solution.

## Senior Users Interview

Q1: “Would you like the system to calculate order prices?”  
Bethany: “Yes I think it would be helpful”  
Anna: “Yes, it could help keep track of finances”  
Mark: “Yes that would be a very useful feature and could track customer spending”

Since the response was very positive, I will make sure this featured is included in my final solution.

Q2: “Do certain customers have discounts that need to be applied?”  
Bethany: “Yes I think there are company discounts”  
Anna: “Yes different customers have different discounts”  
Mark: “Yes some customers have a certain percentage discount that applies to their orders”

As the customers have different discounts, I need to incorporate this into my customer table so when calculating order prices, the price will be correct.

Q3: “How would you want to sort the data?”  
Bethany: “I would like the stock quantity to be sorted in descending order so I can find out what stock is the lowest and needs to be purchased”  
Anna: “There should be a feature to sort orders by order date so I know how urgent the orders are”  
Mark: “I want to be able to sort customers based on their discount percentage, but being able to sort based of other variables could also be useful”

I will therefore unsure you can at least sort the data in the ways specified but will try to include ascending and descending sort for all the categories.

Q4: “Are there any other features you want the solution to have?”  
Bethany: “There needs to be a login to keep the data secure”  
Anna: “When adding new orders can the default address be the customer’s address”  
Mark: “I would appreciate if it kept track of total company spending in the customer table”

I will include user logins and ensure users have entered correct details before they can access the information stored in the database. I will also add a total spent column into the customer table and add to it when even a new purchase is made by that company.

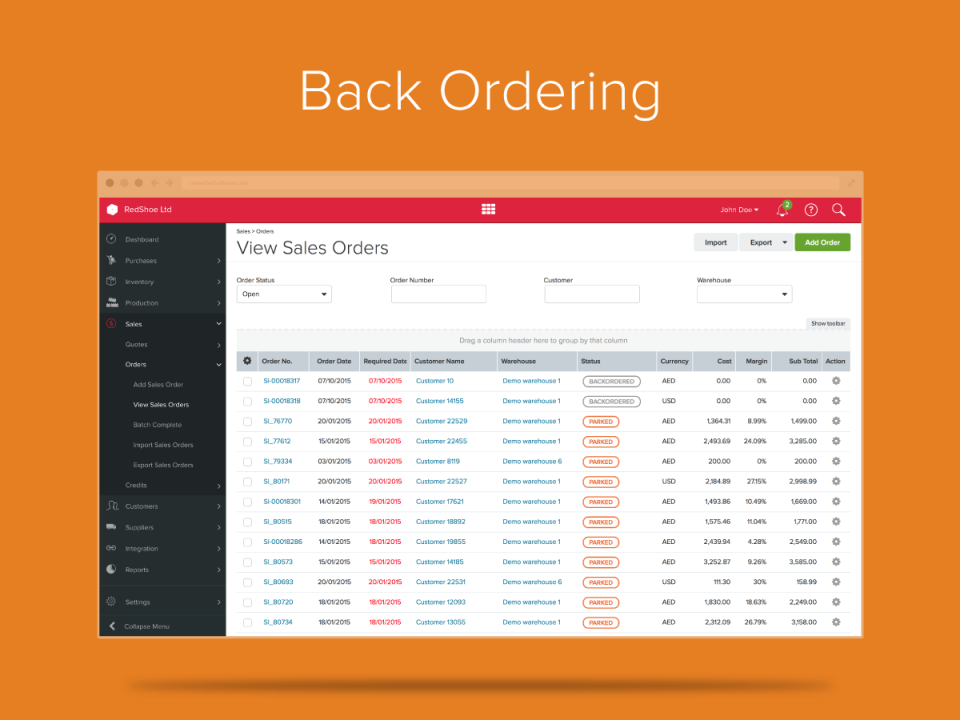
# Existing Solutions

A lot of small businesses like the bookstore, use Microsoft Excel and Google spreadsheets but they are not nearly as powerful or useful as databases. This is because with large volumes of information, it is hard to transfer data in or out of spreadsheets and sometimes require you to manually import and export the data into different programs. For example, keeping purchases and prices on a database instead of a spreadsheet allows the company to transfer the information directly to the accounting system without having to find the data and input it manually.

## Order Management Systems

I decided to look at existing order management systems and their interface so I could decide what features I would need to include in my solution. Both solutions I looked at used SQL for interacting with their database so I will use a type of SQL for add, reading and updating my database.

The first order management system I looked at was called Unleashed. This system uses the cloud to store the data and can be accessed on the web and on mobile devices. Unleashed does inventory management, billing and invoicing, order management, distribution management and production tracking. As it is a high cost solution and has a monthly cost of $135.00 the product has extra features such as sales forecasting and returns management. This solution also contains warehouse management which is not necessary for my solution as the company only operates out of one location.



Clear title so user is aware what table they are looking at.

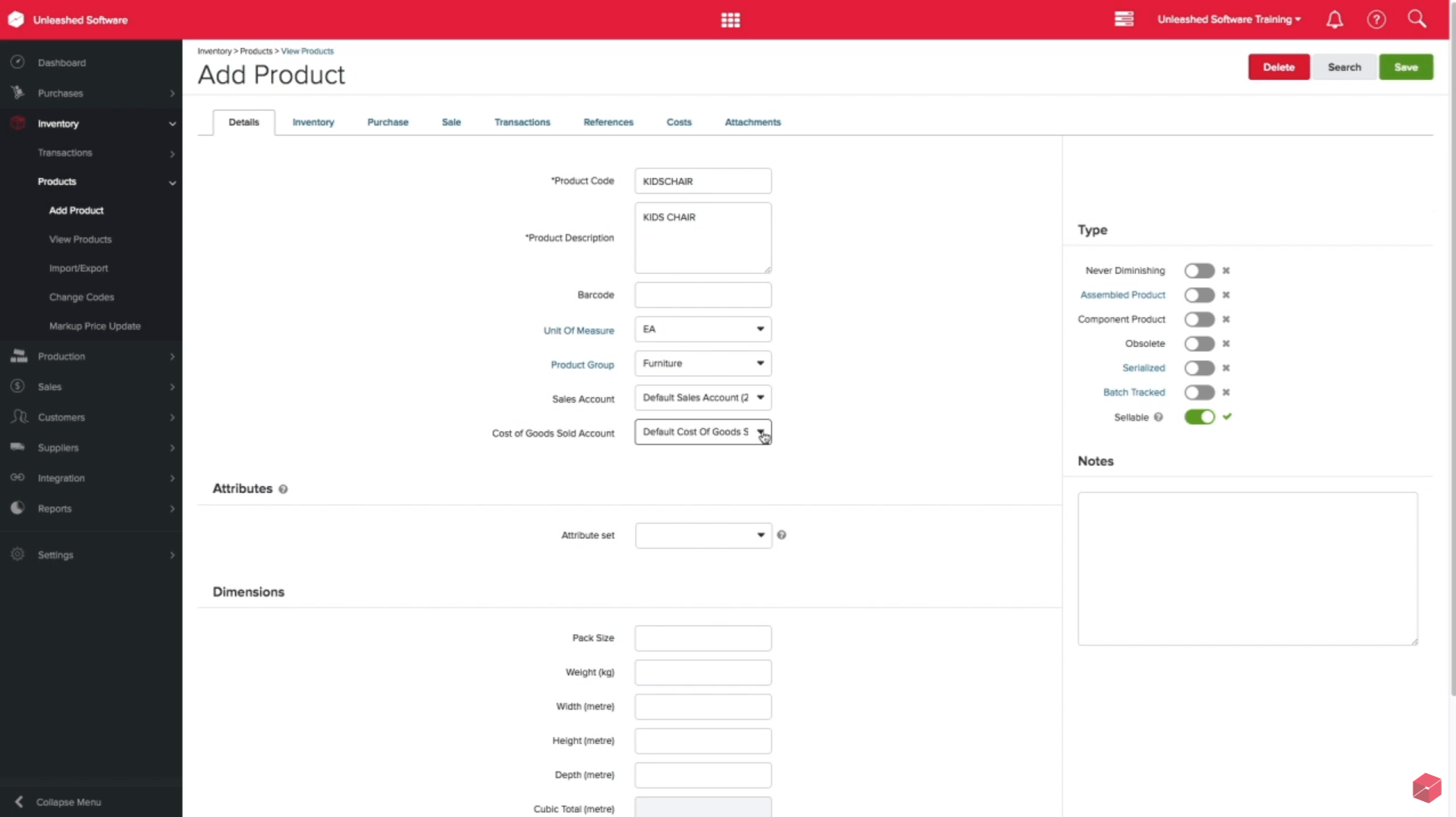
Easy options to search data using 4 pieces of data.

Bold colours make the design aesthetically pleasing.

Menu bar so the database is easy to navigate.

Colour Codes to make the interface easier to interpret.

Figure 1. Unleashed’s Sales Orders Interface



Uses clearly named text boxes to input unique information and drop-down boxes for when there is a set number of options so easy to input.

Help option to make the product more accessible with no training.

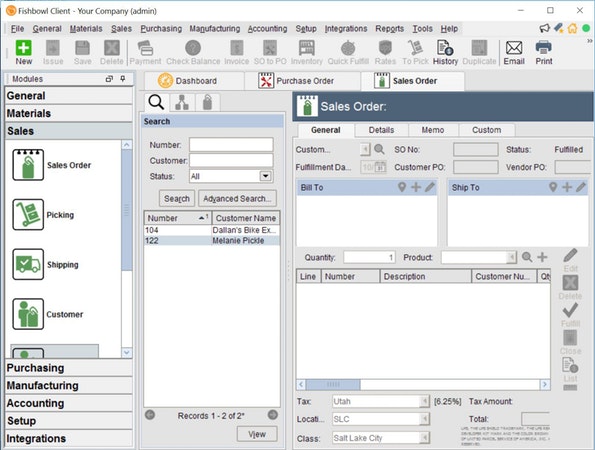
Switches so it is easy to input binary information.

Symbols used on the menu, so it is clearer to navigate.

Figure 2. Unleashed’s Add Product Interface

Unleashed has a great interface as the use of colours codes, clear options and titles, symbols and help button makes it easy to understand how to use it. The software is also accessible through both computers and mobile devices making it easier for the stockholders to have access. These key features will improve my solution so I will try to incorporate them into my final design.

Another existing solution is Fishbowl which has one upfront cost. It works with QuickBooks so small businesses can stay on the platform for accounting and incorporate inventory features into their database. Like Unleashed, this solution contains features such as inventory management, billing and invoicing, order management, distribution management and production tracking. It also contains a feature that allows back orders. Fishbowl is still expensive as it costs $4,395.00 which allows it to have additional features such as barcode generation and scanning.



Symbols used with subheadings to make them easily identifiable.

Help option to make it easier for inexperienced users.

The delete and accept options for adding order lines have symbols to make it more user friendly.

Uses order headers to group orders by one customer and reduce repeated information.

After looking at existing solutions of order management systems, I have found out that using bold colours and having the interface be very spacious are very useful as the Unleashed interface looks a lot more user friendly due to this. The interface designs also include symbols and a help option to make the interface easier to navigate. Therefore, I will try to include these features in my interface. I will also include a help feature like both of the existing solutions include to make the solution more approachable.

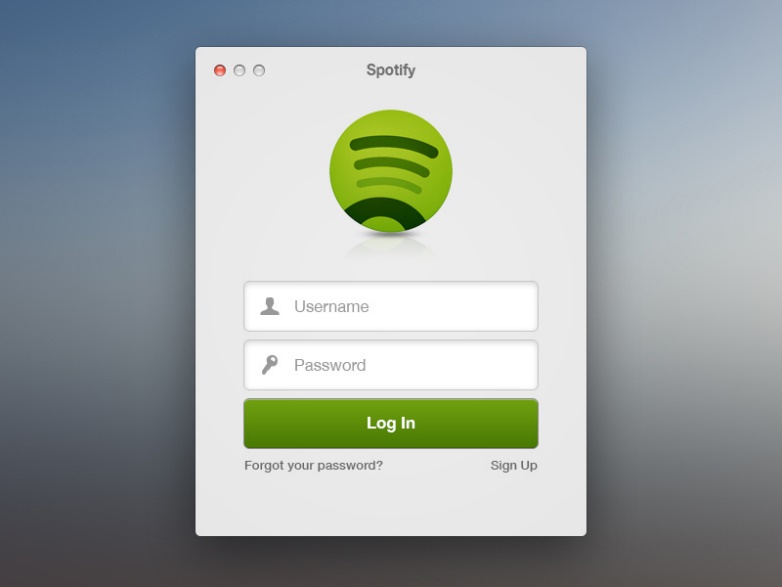
Figure 3. Fishbowl’s Sales Orders Interface

Both designs split orders into order headers and order lines so information such as customer name, delivery address and order date don’t have to be repeated for stock item they have ordered hence I will make the database have separate tables for order headers and order lines. To do this I will need the order number from the order headers to be a foreign key in my order lines.

These designs also have a forecasting feature but due to the time constraints and restrictions on my project I won’t be able to incorporate that into my design. Also, my solution will not be accessible on mobile devices as they would not be able to meet my software requirements.

## Login Window

I also decided to look at existing login windows to find out what features I should include in mine. I will look at both algorithms used and graphical elements of existing solutions.



Password hidden to protect security.

Logo clearly seen at top of login page.

Clearly labelled textbox to easily find it.

Bold coloured buttons to highlight submit details.

Symbols added to textboxes to identify them.

Bold logo makes the window look more aesthetically pleasing.

Both of the existing logins uses textboxes to input the data and only displayed dots for the password to protect the password from being seen by someone else. Spotify and Amazon also both use a bold coloured button underneath the login details to submit the details. This makes it very clear for the user to spot and since it is often located there, the user will aspect it at the bottom of the page so I will also locate my submit button there. I also found online that passwords normally have to be at least 8 characters so I will add this restriction to my passwords.

I also looked at what algorithms existing solutions used. I found out that the password has to be hashed to be stored securely as then no one can see the password even if the database is compromised. Hash algorithms map the password into a hash that can be stored. The hash algorithm should avoid hash collisions by being producing a unique hash for each password. I very well-known hash algorithm is MD5 but one of the main problem with this algorithm is due to its popularity, there are lots of existing tools for cracking MD5 hashes. This led to the CMU Software Engineering Institute to consider MD5 essentially “cryptographically broken and unsuitable for further use”. Therefore, in my project I will aim to use a more secure hash algorithm. Another group of hashing algorithm is the SHA-family. SHA-0 was released in 1993 and has since been compromised along with SHA-1. A safer SHA hash is SHA-2 which is a family that contains six hash functions and will provide the level of security needed for my project.

When hashing my password, I can also use a salt to make the hashed password more secure. Salting is where you add a unique value at the end of the password to create a different hash. This adds an additional layer of security to the hashing process, especially against brute force attack. You can also use multiple iterations of the hash algorithm to make the password more secure.

Overall, I will use a SHA-2 hash algorithm as it is secure enough for my project and will potentially add a salt or use multiple iterations to make to the password even more secure.

# Requirements

My solution must meet the following requirements:

|  |  |
| --- | --- |
| Requirement | Justification |
| The solution must be able to store customer, stock and order information in a database. | This is necessary to keep track of all the data the business needs so they can make more informed decisions. |
| There must be a feature to add new information into all the tables. | This allows to company to process new orders, grow when they receive new customers and adapt when they expand their product range. |
| The solution must include a user interface that is clear and easy to interpret. | The database must be accessible so the employees can easily use it as and from my interview I know some users have no experience with databases. |
| There must be a way to change some of the existing data in the tables. | This is necessary as the company will buy more stock of some products so should be able to increase stock levels. |
| The system must be able to automatically reduce stock quantity for each item. | From my interview the stakeholders said this would be very useful as tracking stock quantity allows the company to see when they need to buy more stock. |
| The interface must have a simple design and not be cluttered. | This makes it quicker for the users to become familiar with the system and was requested by the employees during the interview. |
| The GUI should have separate tabs or pages for each table and use buttons textboxes to add data. | During the interview they said this would be the easiest way to navigate the system. |
| The solution should calculate price orders. | In the interview with sales director they said this would be a useful feature so the cost don’t have to be manually imputed and can be calculated by the system. |
| It must allow you to apply discounts to certain customers’ orders. | This allows the sales team to negotiate and is essential if I am calculating prices as some customers get discounts of the products. |
| The solution must be able to sort the data based off stock quantity, order date and discount rates. | The stock quantity lets the company see which stock is running low, order dates show most pressing orders and discount rates can be used to compare companies. |
| It should calculate the total amount of money a company has spent. | During interview this was asked to be incorporated so they can analyse the data and use it when deciding companies’ discounts. |
| The user must login on using their username and password before they can access the main window containing the data. | The sales director said that he would like the user to be required to login to improve security of the solution as only approved users should be able to see the information stored. |
| Administrators must be able to add new users. | This is necessary as when new employees join the company, the managers must be able to create a new account for them. |
| The orders will be split into order headers and order lines, using the foreign key order number from order headers in the order lines table to connect them. | I looked at two existing solutions and both systems used this so there is no need to repeat information such a customer and delivery address that are stored in the header and each type of stock ordered is stored in the lines making it more efficient. |
| The interface should use bold colours or colour codes. | When looking at Unleashed I found this makes the design look more aesthetically pleasing. |
| There should be a menu bar with symbols. | Both existing solutions I looked at had this feature to make it easier to navigate. |
| The password must be hidden when entering it in. | The existing solutions did this to improve security by hiding the password from onlookers. |
| Users’ passwords must be hashed before they are stored in the database. | This makes the database more secure as the user’s password is not stored alone so the hash key would still be required. |
| Passwords must be 8 characters or longer. | Existing solutions use this requirement to make the passwords more secure. |

The solution also has the following software requirements:

* PyQt5 module to create the user interface
* SQLite module to connect to the SQLite database
* uic module to load the user interface

# Success Criteria

|  |  |
| --- | --- |
| Requirements | Testing |
| The solution must be able to store customer, stock and order information in a database. | I will use test data to check if the database stores the information. |
| There must be a feature to add new information into all the tables. | I will try to add test data into the each of the tables in the database. |
| You must be able to search the data stored in the database. | I will check if the search feature works correctly using test data. |
| The solution must include a user interface that is clear and easy to interpret. | I will do a survey to investigate if people find the interface easy to use and clear. |
| There must be a way to change existing data in the tables. | I will try to change some of the test data and check if it stores the new data correctly. |
| It must be able to sort the data based on certain categories. | I will check if you can change information stored using test data. |
| The system must be able to automatically reduce stock quantity for each item. | I will find out if my stock automatically decreases when orders are made. |
| The interface must have a simple design and not be cluttered. | I will do a survey to investigate if people think the interface is simple or too cluttered. |
| The GUI should have separate tabs or pages for each table and use buttons textboxes to add data. | I will check if my interface uses tabs or pages and how you add data using the interface. |
| The solution should calculate price orders and track customer spending. | I will use test data to see if it calculates the order prices correctly. |
| It must allow you to apply discounts to certain customers’ orders. | I will investigate if different order costs apply discounts appropriately to find prices. |
| The solution must be able to sort the data based off stock quantity, order date and discount rates. | I will try to sort the test data on stock quantity, order date and discount rates and will check if they are sorted correctly. |
| It should calculate the total amount of money a company has spent. | I will check if the system records the total amount a company had spent using test data. |
| The user must login on using their username and password before they can access the main window containing the data. | I will use test login to see if it only allows users with correct login details to access the main window. |
| Administrators must be able to add new users. | When logged on as an administrator I will try to add a new account then test if I can log in using it. |
| The orders will be split into order headers and order lines, using the foreign key order number from order headers in the order lines table to connect them. | I will look at the database to see if uses a foreign key from the order headers table in the order lines table. |
| The interface should use bold colours or colour codes. | I will ask my stakeholders if they think the interface makes good use of colours. |
| There should be a menu bar with symbols. | I will check if my interface uses symbols to help navigate the menu. |
| The password must be hidden when entering it in. | I will type in a test password and see if it is visible on the interface. |
| Users’ passwords must be hashed before they are stored in the database. | I will check how the passwords are stored in my database. |
| Passwords must be 8 characters or longer. | I will try and add a user with a password less than 7 characters. |

Design

# Problem Decomposition

Since the problem is made up of smaller tasks, we can break down the problem to make solving each smaller problem easier and quicker. For this problem we can break down the databases into separate tables: stock, customer and orders. The orders table will be further broken down into order headers and order lines. The separate tables can use the same or similar methods for retrieving, adding and searching data.

# User Interface Design

The interface will use a variety of windows, so each window is clear and easy to navigate and avoid the interface looking cluttered.

## Main Window

The main window will be split into 3 tabs, so the interface is easy to navigate and will also include a help button to explain how to use all the features. Each tab will contain the necessary tables as well as buttons to refresh, add and search stock. Having the separate tabs also allows the interface to look less cluttered and was requested when I interview the stakeholders.

Figure 4. Decomposition of the main window

Add User

Customer

**?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Title | Author | List Price | Quantity |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |
| 7 |  |  |  |  |
| 8 |  |  |  |  |
| 9 |  |  |  |  |
| 10 |  |  |  |  |
| 11 |  |  |  |  |

Figure 5. UI mock-up for the Stock Tab

Orders

Stock

Add

Refresh

Update Stock

X

Cus kjkn

Order Management System

Add User

Customer

**?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Name | Email | Tel | Address |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |
| 7 |  |  |  |  |
| 8 |  |  |  |  |
| 9 |  |  |  |  |
| 10 |  |  |  |  |
| 11 |  |  |  |  |

Figure 6. UI mock-up for the Customer Tab

Orders

Stock

Add

Refresh

Update Customer

X

Cus kjkn

Order Management System

**?**

Add User

Customer

Order Headers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Customer | Delivery Address | Delivery Charge | Order Date |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |

Order Lines:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Book | Quantity | Line Price |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |

Figure 7. UI mock-up for the Orders Tab

Orders

Stock

Add Header

Refresh

Add Line

X

Cus kjkn

Order Management System

Add User

**X**

Help Information

Customer

**?**

Figure 8. UI mock-up for the Help button

Orders

Stock

**?**

X

Cus kjkn

Order Management System

## Login

When the interface starts up the first window to pop up will be the login window. The purpose of this page is to ensure the user has correct login details to access the information stored in the system.

Invalid username or password error message

Username:

Password:

Login Button

Textbox for Username

Textbox for Password

Logo

X

Login

Figure 9. UI mock-up for the login window

## Adding Data

When you push the Add button a dialog will appear to add the data. The dialog will be specific to the current tab as each table requires different information. It will include textboxes which are needed for the user to type in the information and buttons to add the data to the database or close the dialog so it is easy for the user to submit the changes or return to the main window.

Figure 10. Decomposition of the add dialog

Error Message

X

Add Stock

Cancel

Text box for book’s author

Author:

Quantity:

Textbox for book title

Textbox for standard price

Title:

List Price:

Textbox for quantity in stock

OK

Figure 11. UI mock-up for add stock dialog

X

Update Stock

Error Message

Textbox for quantity in stock

List Price:

Title:

OK

Textbox for standard price

Dropdown combo box for book title

Quantity:

Author:

Text box for book’s author

Cancel

Figure 12. UI mock-up for update stock dialog

Error Message

X

Add Customer

Discount:

Textbox for name

Name:

Textbox for email

Email:

Tel:

Textbox for address

Textbox for discount

OK

Cancel

Address:

Textbox for telephone

Figure 13. UI mock-up for add customer dialog

X

Update Customer

Cancel

OK

Textbox for discount

Textbox for address

Tel:

Email:

Textbox for email

Name:

Textbox for telephone

Textbox for name

Discount:

Address:

Error Message

Figure 14. UI mock-up for update customer dialog

X

Add Order Header

Error Message

Delivery Charge:

Customer:

Cancel

OK

Textbox for cost

Drop down for customer name

Delivery Address:

DD/MM/YYYY

Textbox for delivery address

Order Date:

Figure 15. UI mock-up for add order header dialog

X

Add Order Line

Error Message

OK

Quantity:

Title:

Textbox for cost

Drop down for customer name

Cancel

Figure 16. UI mock-up for add order line dialog

Add User

X

Error Message

Password:

OK

Username:

Administrator

Textbox for cost

Textbox for username

Cancel

Figure 16. UI mock-up for add order line dialog

# Usability Features

The interface includes a variety of features to make to solution more accessible to a variety of users. I have decided to use a blue colour scheme in my design as it has connotations of calmness and stability as well as making the design look more aesthetically pleasing. I will use a bold darker blue to highlight the buttons on each tab making the options clearer to the user compared to a pale washed out blue for the background in order to not distract from the main features. I will also use the font Calibri in my deign as it is a clear and easy to interpret font. Using the same font and colour throughout my design will also make the separate tabs and windows look like they are all part of one system.

## Main Window

Each tab only contains three or four buttons and a table for the data. This is because it makes the interface very spacious and easy to navigate. The table takes up most of the tab as showing the data is the main purpose of each of the tabs. For the orders tab I split it into two tables so the information could clearly be displayed without out having to repeat all the main information like customer and delivery address on each type of item they purchased. Therefore, I split it into two tables where the first shows the overhand information and when you click on a certain row the second table will show the separate items that have been purchased in the order. All the buttons on each tab are located in the same order on the top right of every tab so it will be quicker for new users to become familiar with the layout of the design. Every table can be sorted by clicking on the column header. The data will then be sort in ascending order based of the category of the header selected. If they were to select that header again the system will sort the data in descending order. I will include a small arrow when a column is sorted to indicate if the data is sorted in ascending or descending order. This will make it easier for the user to see how they have sorted the data making it simpler to interpret. I decided to not include a button where you select a category header as adding more buttons could make the deign look more cluttered.

The add user button will only be seen on administrators accounts and will open a pop up to create a new user. I put it in the top corner in order to not distract from the main features of the system as it will only been needed when new employees are joining the company.

I chose to use a symbol for the help button as it is easily recognised and doesn’t draw away from the main features of the design. It also is outside of the tabs so it will always be in the same place making it easy for the user to find. I chose for the pop up to be in the centre of the window, so it was clear to see. I also decided to make the close a button red, so it clearly stood out from the rest of my design. I will put it in the top right-hand corner as most applications include the exit button there so that is where most users would look for it. Hence including it there will make it easier for the users to find.

## Login

The login uses a similar design to the login windows I looked at during the analysis stage of my project. This means the user will be very familiar with the layout as they would have used it before. One of the features I included is the bold coloured button at the bottom of the window, so it is easy to spot once the user wants to submit their login details. Part of the reason this is especially easy to navigate is because the user travels down the page entering the information so after entering the password the next step is where you would below like the previous steps. The bold colour also makes it clear to see. Another feature the interface uses that was in the existing solutions I looked at was the hidden password using the small back dots to stop ‘shoulder surfing’ making the passwords more secure.

I also decided to use the same colour scheme as the add dialogs and main window to make the solution look more cohesive and not like separate programs. The error message will be red to draw attention to it and alert the user to the problem. It will also originally be hidden and only show when the error occurs.

## Add and Update Data Dialogs

The dialogs used to add, or update data have a confirmation button with the text ‘OK’ and a cancellation button with the text ‘Cancel’. I chose to use ‘OK’ as the colloquial language makes it more accessible to a wider range of people. Both words are written in bold in the darker coloured buttons to make them clear to the users. The buttons are also the largest features on the window making them the first things the user sees. As these are the main features the user has to navigate on these windows it is useful to make them clear. All error messages will also be initially hidden and only pop up when they try to submit the data when it isn’t valid. The message will be bright red to draw attention to it.

For update dialogs the name will be on a drop-down combo box and the rest of the inputs will automatically fill in with the current information so the users can see what the current data is. This also is use useful if correcting a small mistake like a spelling mistake as you don’t have to renter all the information.

Each piece of information the user has to input is clearly labelled so they know what information they need and which textbox they need to type it in to. I decided to mostly use textboxes as it is the easiest way to input the new data into the system. The textboxes are a range of sizes depending on what information neds to be typed in. This allows most of the text the needs to be typed in to be clearly shown which out making the dialog windows bigger. For instances address has a bigger text box than discount as the data the user will probably have input is longer. However, the add dialogs don’t just use textboxes. The Add order Header dialog uses a drop-down list of all the customer names so whilst the user can still type the name in the dialog will only accept existing customers. It also allows the user to look at the customer names if they are unsure of the spelling or the full name of the company. To make it easier for the customer to look at the existing customers I have decided that the drop-down list will be in alphabetical order. Additionally, I have included a date edit box in my design of the Add Order Header dialog. This will restrict the user to only type in the date in one format so system can easily interpret the input and will validate what they have entered, stopping the user for making a mistake like typing in a date or month that is not on the calendar. The design will allow the user to type the date but will also have buttons so the user has multiple ways they can enter the date making easier to use.

# Stakeholder Input

Before making my interface, I decided to check what the stakeholders thought of my design so I could make any necessary adjustments before I began my project.

## Main Users

To start with I sent a copy of the designs to the main users to find out what they thought of them. John was the first to respond and said, “The system looks very bold and clear to navigate”. Thomas commented “The design looks good and user friendly” and Samantha said, “I love the design; it looks very clear and easy to use”. Thomas commented “I like colour scheme idea and the use of bold colours” and Katy said, “Overall it looks very simple and as long as the help button clearly explains how to use all the features, I would feel very comfortable using this”. Matt also said, “I am very happy with the design and don’t have any improvements”.

## Senior Users

I also sent a copy of the designs to the senior users I had previously contacted to interview during the Analysis stage. Bethany (purchasing manager) said, “I think the design looks good and people will like it”. Anna (operations manager) responded saying “I think the error message will be very useful to users” and Mark (sales director) said “it looks easy to add users and navigate”

Overall, all the users I interviewed where happy with design so I will replicate it in my final project.

# Key algorithms

Described the solution fully using appropriate and accurate algorithms justifying how these algorithms form a complete solution to the problem.

# Main data structures

For all add procedures it will open the relevant pop up and then refresh the table to update it with the new information.

# Pseudocode

Add after development is done

## ER diagrams and table relations

## Data Tables

The relational notation for my tables is:

Stock(StockID,Title,Author,ListPrice,Quantity)  
Customer(CustomerID,Name,Email,Tel,Address,Discount,TotalSpent)  
OrderHeader(OrderNumber,CustomerID,DeliveryAddress,DeliveryCharge,OrderDate,TotalCost)  
OrderLines(OrderLineID,OrderNumber,StockCode,Quantity,LinePrice)

The attributes of the stock table are listed in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Attribute | Data type | Constraints |
| Stock | StockID | Integer | Primary key, Autoincrement |
| Title | Text | NOT NULL |
| Author | Text | NOT NULL |
| ListPrice | Numeric | NOT NULL |
| Quantity | Integer | NOT NULL |
| Customer | CustomerID | Integer | Primary key, Autoincrement |
| Name | Text | NOT NULL |
| Email | Text |  |
| Tel | Text |  |
| Address | Text |  |
| Discount | Numeric |  |
| TotalSpent | Numeric |  |
| OrderHeader | OrderNumber | Integer | Primary key, Autoincrement |
| CustomerID | Integer | Foreign Key “Customer”(“CustomerID”) |
| DeliveryAddress | Text | NOT NULL |
| DeliveryCharge | Numeric |  |
| OrderDate | Text |  |
| TotalCost | Numeric |  |
| OrderLines | OrderLineID | Integer | Primary key, Autoincrement |
| OrderNumber | Integer | Foreign Key “OrderHeader”(“OrderNumber”) |
| StockCode | Integer | Foreign Key “Stock”(“StockID”) |
| Quantity | Integer | NOT NULL |
| LinePrice | Numeric |  |

|  |  |
| --- | --- |
| Customer | |
| CustomerID | Integer, Primary Key |
| Name | Text |
| Email | Text |
| Tel | Text |
| Address | Text |
| Discount | Numeric |
| TotalSpent | Numeric |

|  |  |
| --- | --- |
| OrderHeader | |
| OrderNumber | Integer, Primary Key |
| CustomerID | Integer, Foreign Key |
| DeliveryAddress | Text |
| DeliveryCharge | Numeric |
| OrderDate | Text |
| TotalCost | Numeric |

|  |  |
| --- | --- |
| Stock | |
| StockID | Integer, Primary Key |
| Title | Text |
| Author | Text |
| ListPrice | Numeric |
| Quantity | Integer |

|  |  |
| --- | --- |
| OrderLines | |
| OrderLineID | Integer, Primary Key |
| OrderNumber | Integer, Foreign Key |
| StockCode | Integer, Foreign Key |
| Quantity | Integer |
| LinePrice | Numeric |

## UML class diagrams

Add after development is done

UML class diagrams were generated using the pyreverse tool which is part of the pylint python module

# Testing

## Functionality and robustness testing during development

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## Post-development usability testing

Identified and justified any further data to be used in post development usability testing

Development and Testing

# Setting up the database tables

All the database tables where created in DB Browser for SQLite as it is a free and accessible platform.

## Stock Table

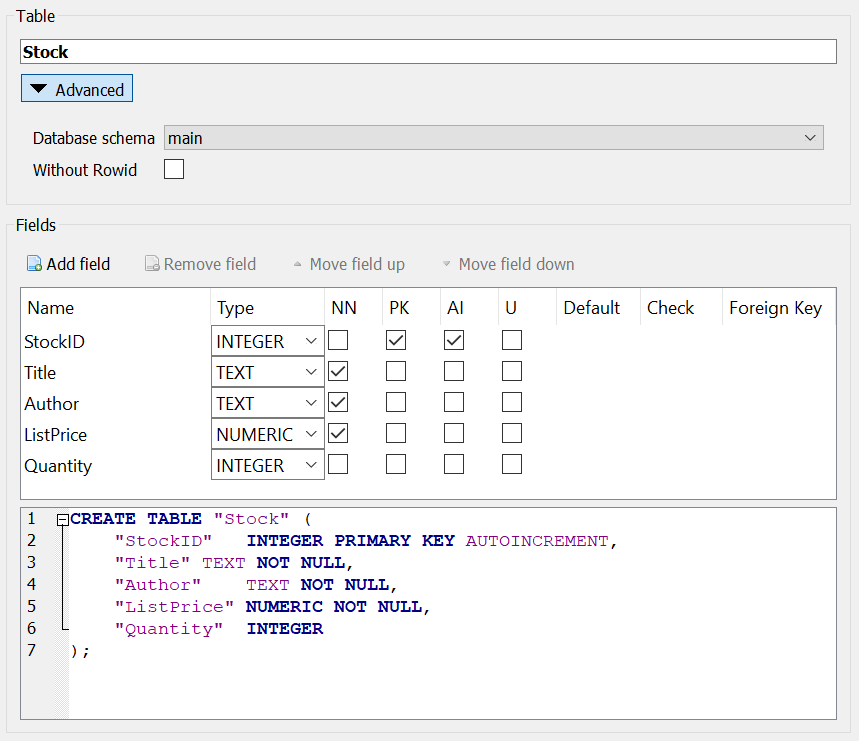


Figure 15. Creating Stock Table

The table has StockID field which is the primary key that is autoincremented, so the user doesn’t have to add it into the database. Title, Author and List Price must not be Null as the books must have a name and author to be referenced by the users of the interface and must have a list price as the books will always have a price and will be needed for order cost calculations. The Title and Author are texts as they are names. The list price is numeric as it must be a number but can have 2 decimal places. Quantity will only be an integer as you can only have whole numbers of stock.

## Customer Table

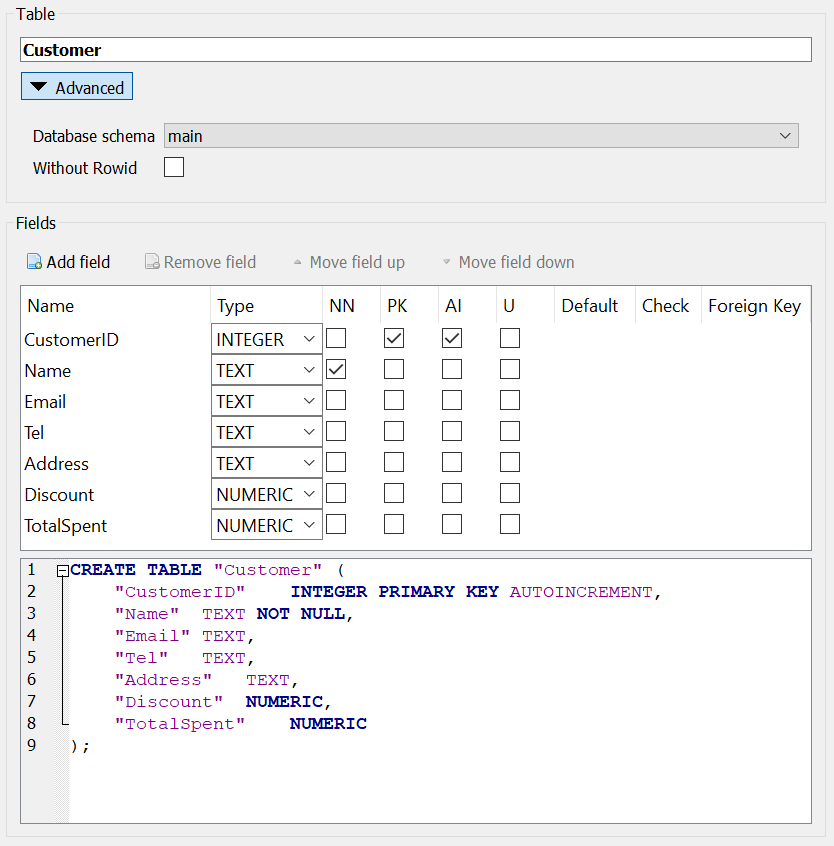


Figure 16. Creating Customer Table

The table has CustomerID field as the primary key and is autoincremented, hence the user doesn’t need to enter a value for this. The Name field cannot be null as it will be used to reference the customer. Email, Tel and Address are all text fields that are not necessary so can be null. The discount percentage and total spent by the customer will be numbers but could be decimals so are numeric.

## Order Header Table

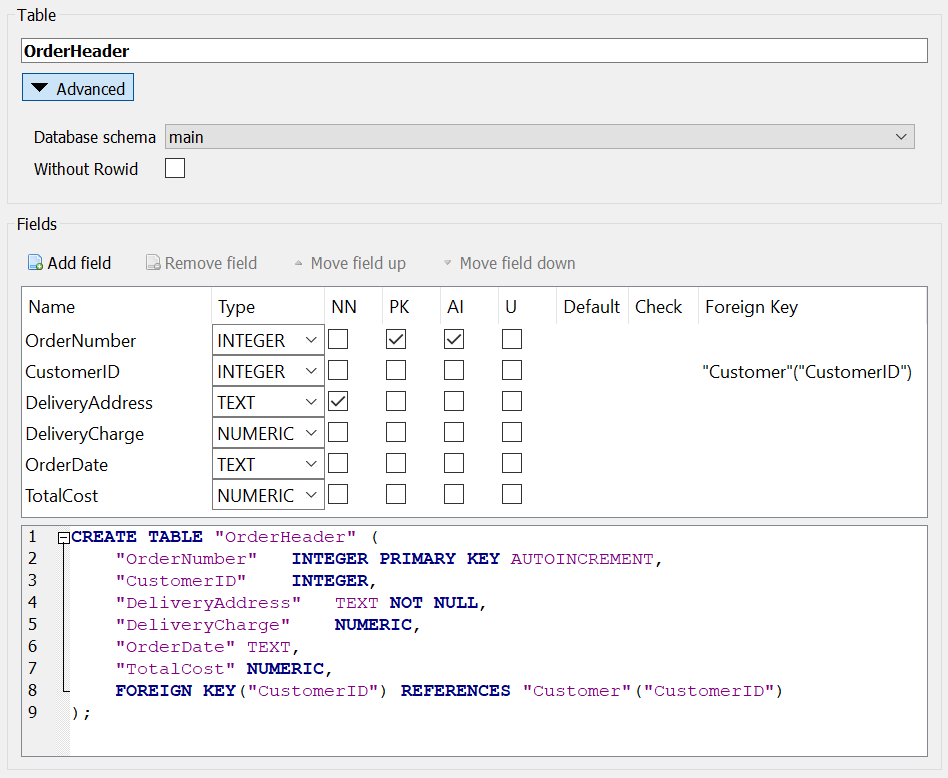


Figure 17. Creating Order Header Table

The table has OrderNumber field which is the primary key and is autoincremented like the other primary keys for the same reason. The Customer Id field is a foreign key that references the customer id in the customer table. This is to link the order to the customer so the table can access information such as the customer discount to work out total cost. The delivery address must not be null, so the company knows where to send the delivery. The delivery charge and total cost are both numbers that can be decimal so are numeric. Since SQLite has no date type, the date will be stored as text.

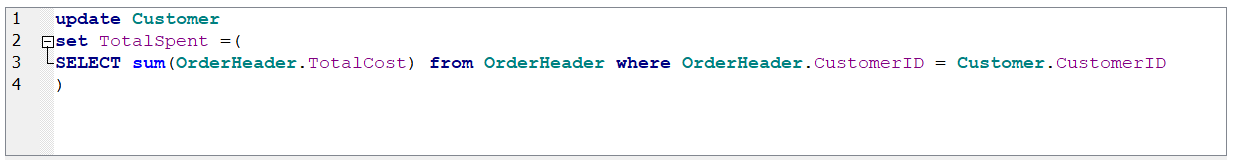
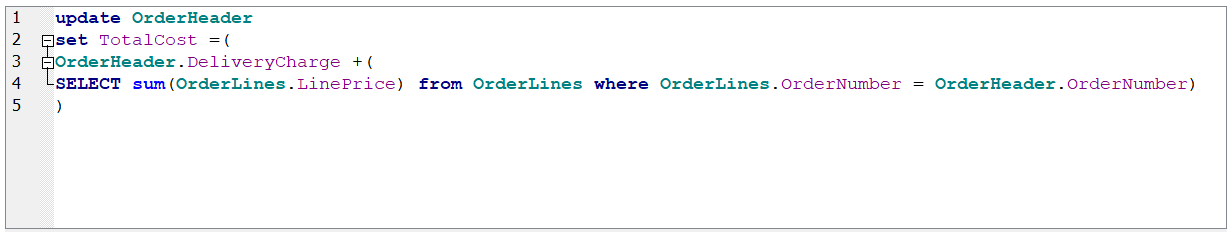
## Order Lines Table

Figure 18. Creating Order Lines Table

The table uses OrderLineID as its primary key and is autoincremented, so the user doesn’t have to type in a primary key. Order number references the order number from the order header information such as customer and delivery address are linked to the separate items ordered. The StockCode is a foreign key from the Stock table. This links the tables so the only items you can order are part of the company’s stock. It also will be used so the stock can automatically be reduced when new orders are placed. The quantity cannot be null as there then the item should not have been added to the order. The line price must also be a decimal number so will be stored as numeric.

SQL Queries

I then updated the total cost column in my order header so the test data would have the correct prices. To do this I executed the following SQL query on the database.

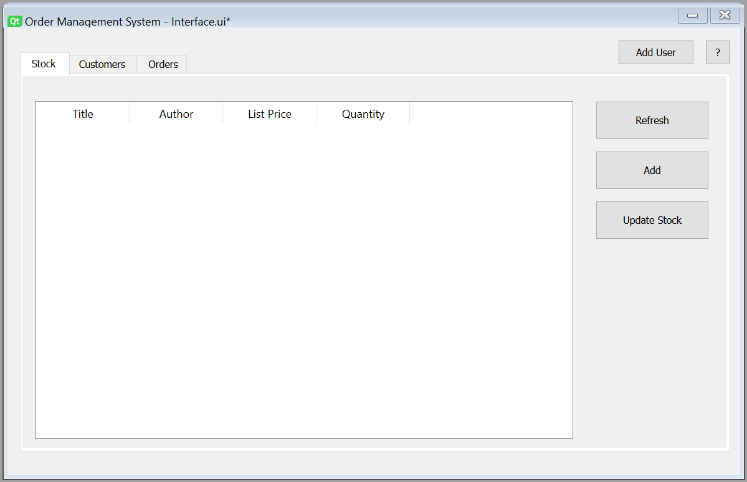
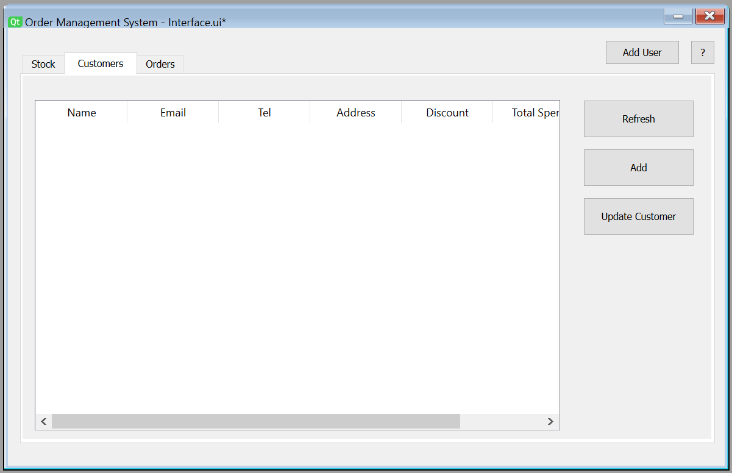
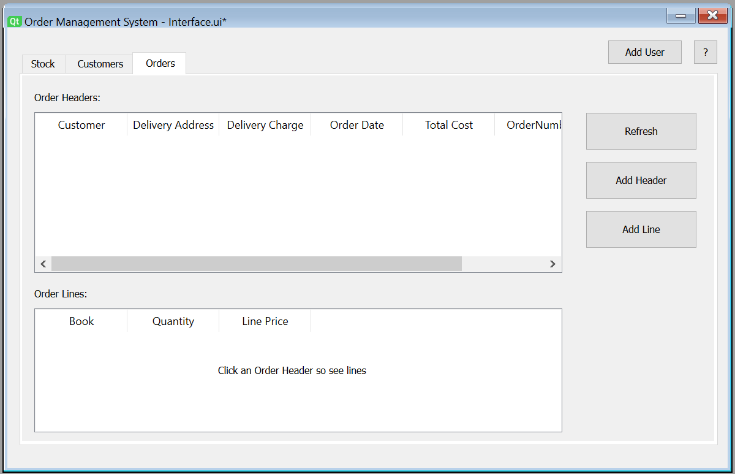
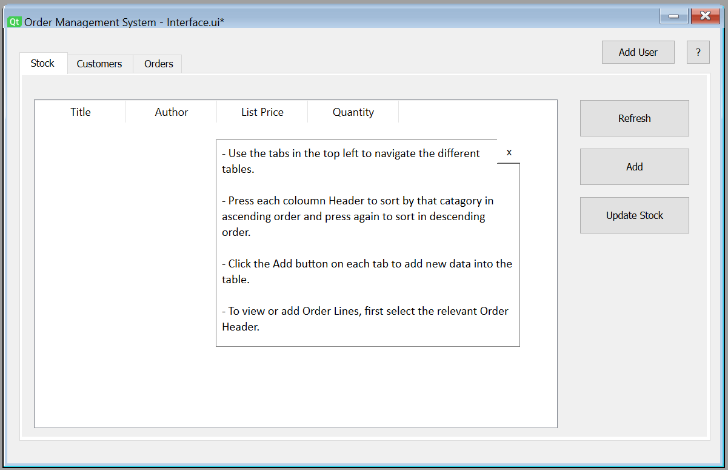
User Interface  
I then designed the user interface using QtDesigner. The user interface was made up of the main window a, login page as well as add stock, customers and order lines and order header dialogs. The first interface I main was the main window.

Figure 22. Original Help Pop Up

Figure 21. Original Order Tab

Figure 20. Original Customer Tab

Figure 19. Original Stock Tab

The widgets on are the Main Window are:

1. addUserButton - QPushButton
2. helpButton – QPushButton
3. helpWidget – QWidget
   1. exitHelpButton - QPushButton
   2. helpText – QTextBrowser
4. tabWidget – QTabWidget
   1. stockTab – QWidget
      1. addStockButton – QPushButton
      2. refresh\_stockButton – QPushButton
      3. updateStockButton – QPushButton
      4. stockTable – QtableWidget
   2. customerTab – QWidget
      1. addCustomerButton – QPushButton
      2. refresh\_customerButton – QPushButton
      3. updateCustomerButton – QPushButton
      4. customerTable - QTableWidget
   3. ordersTab – QWidget
      1. addHeaderButton – QPushButton
      2. addLinesButton – QPushButton
      3. clickHeaderLabel - QLabel
      4. headersLabel – QLabel
      5. headersTable – QTableWidget
      6. linesLabel – QLabel
      7. linesTable – QTableWidget
      8. refreshOrdersButton – QPushButton

Since the help button is always visible and doesn’t change when the code is running, I decided to not put it in any widgets. The help text and exit button go inside the help widget so can be shown or hidden with one command instead of having them be controlled separately. I then used the tab widget to group all the information for each category and only show the widgets on the current selected tab. This will make my code more efficient as I don’t have to program show and hide function for the separate tabs. The stock tab groups the stock refresh, add and save buttons as well as the stock table. The customer tab groups the order buttons and customer table. The order tab groups the order refresh and save buttons, add header and add lines buttons, order header label and table as well as the order lines label and table. Overall using the tab widget helped my group my widgets into each section.

The next thing I did add colours to match the design I wanted for the interface. To do this I edited the style sheets for some of the widgets.

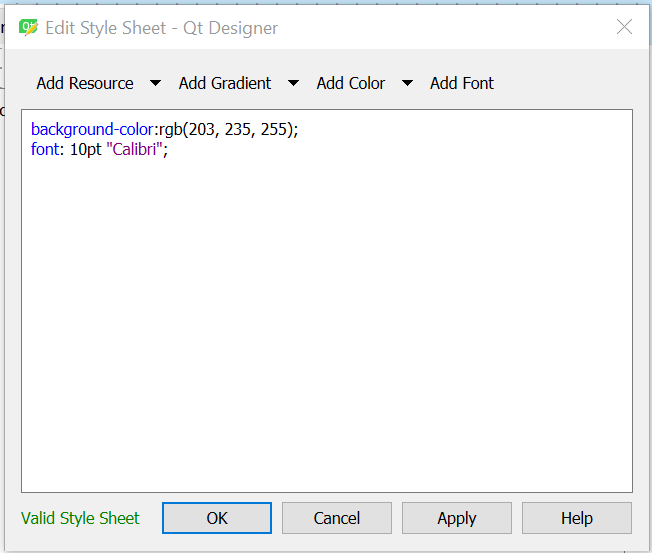
For the main window I set the background colour to blue to add some colour onto the interface. I also set the font style to Calibri and default font size to 10pt it the font would be clear and easy to read. I added the font style into the main window so I would not have to repeat this code for each of the widgets as they will all inherit from the main window.

Figure 23. Main Window Style Sheet

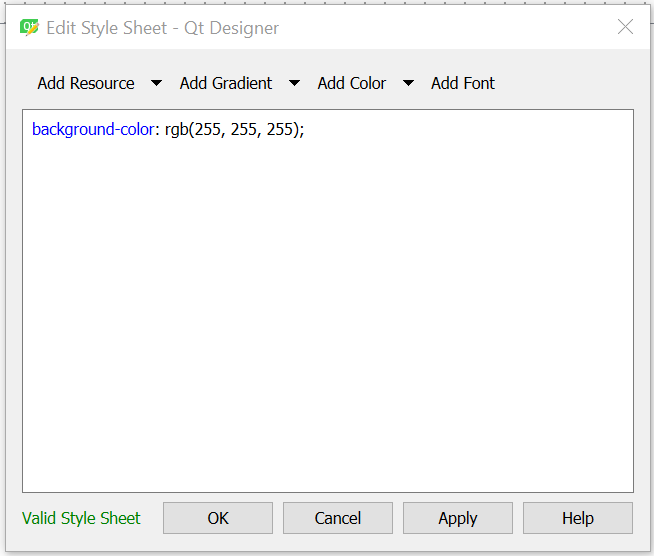
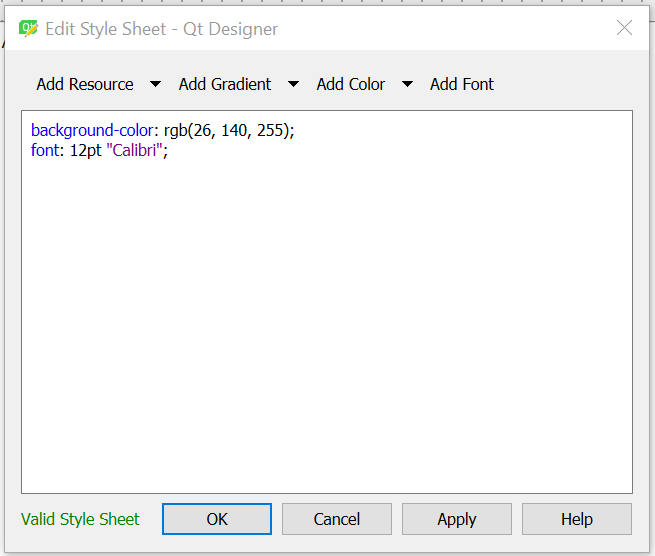
I then edited the tab widgets style sheet so the background of the tab will be white making the text clear to read. I did this in the tab widget so I would not have to change the style sheets on all the tabs and the tables inside the tab widget.  
  
I then changed the style sheets for the refresh, add and save buttons’ style sheets so the text would be larger and the colour bolder making them stand out more. This also matches the design for my buttons.

Figure 25. Button Widgets Style Sheet

Figure 24. Tab Widget Style Sheet

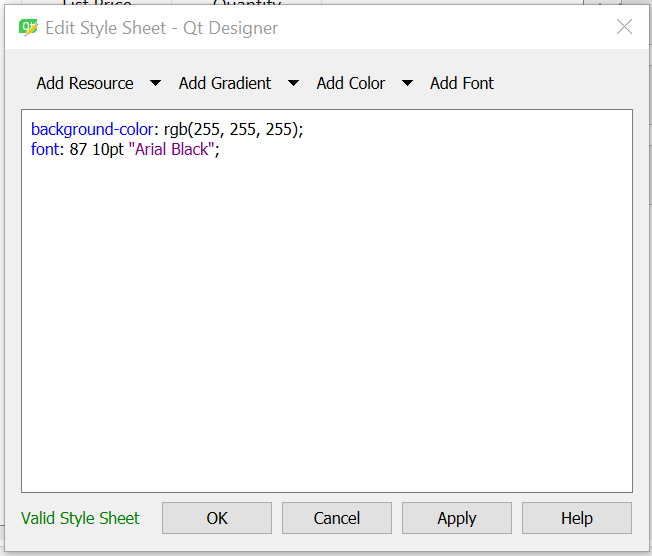
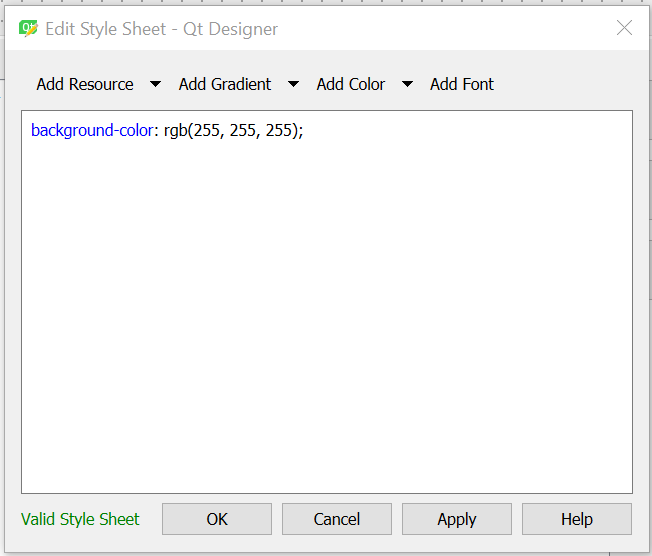
The next style sheet I changed was the help button. For this button I changed the font to make the question mark symbol to look bolder and more recognisable. I also had to set the background to white because it was not in the tab widget so had a blue background from the main window style sheet.  
  
I also had to set the background to white on the help widget so that it will also not be blue like the main windows background. I did not have to set the font as I had already done this in the main window style sheet.

Figure 27. Help Widget Style Sheet

Figure 26. Help Button Style Sheet

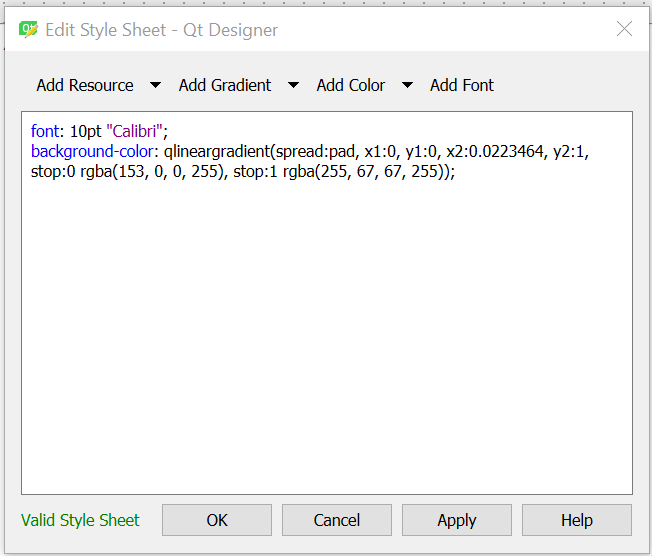
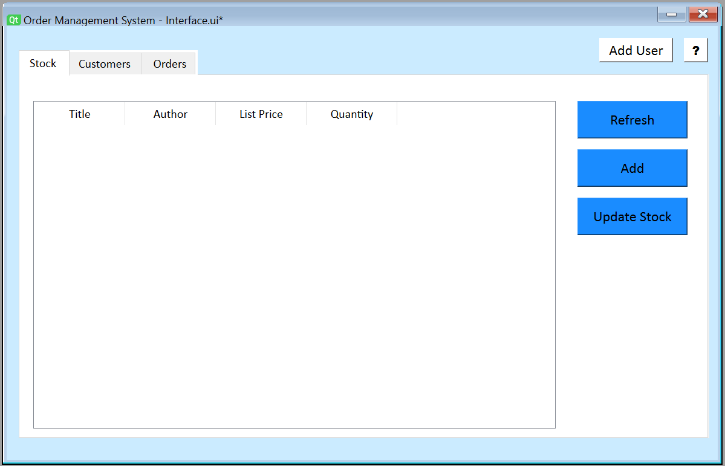
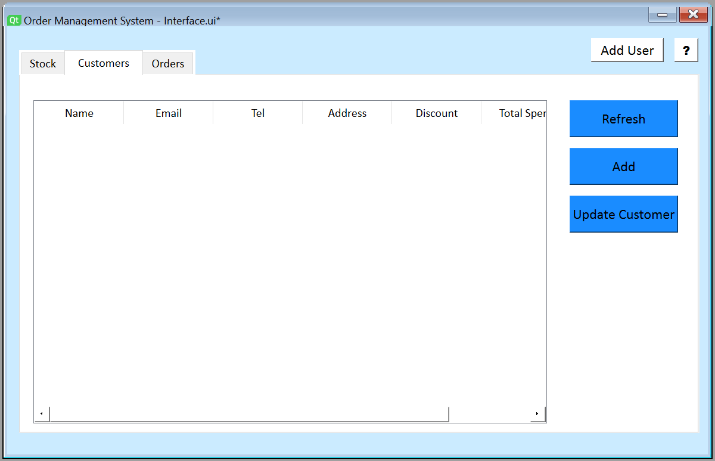
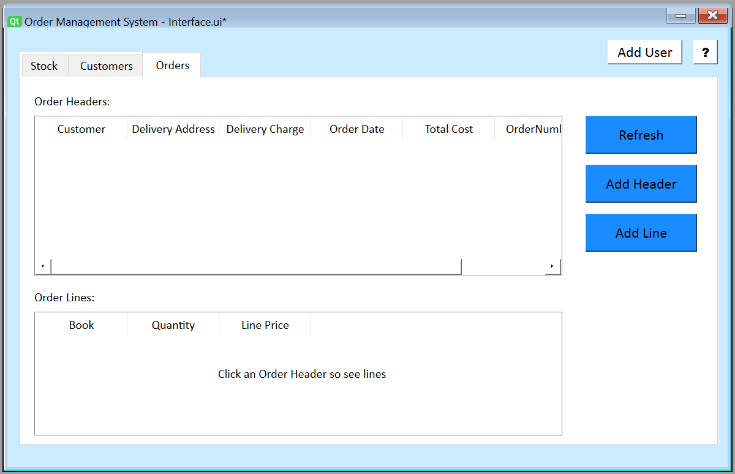
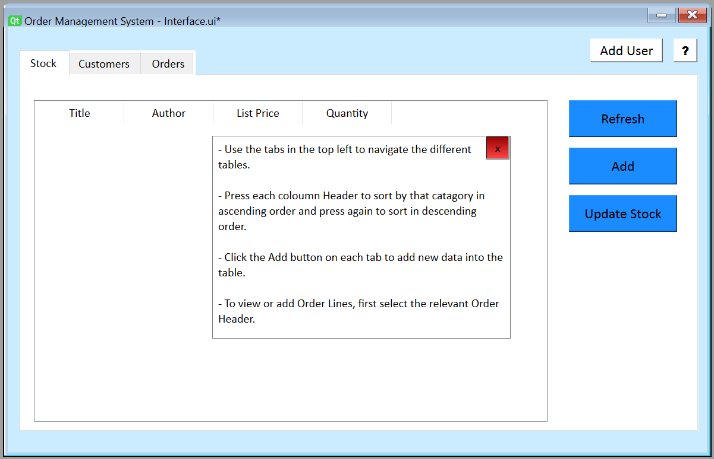
The last style sheet I changed in the main window was the exit button to the help widget. I used a larger font to make the exit symbol bigger. I also decided to make the buttons background to be a gradient to make it look more aesthetically pleasing.

Figure 30. Final Customer Tab

Figure 29. Final Stock Tab

Figure 31. Final Order Tab

Figure 32. Final Help Pop Up

Figure 28. Exit Help Widget Style Sheet

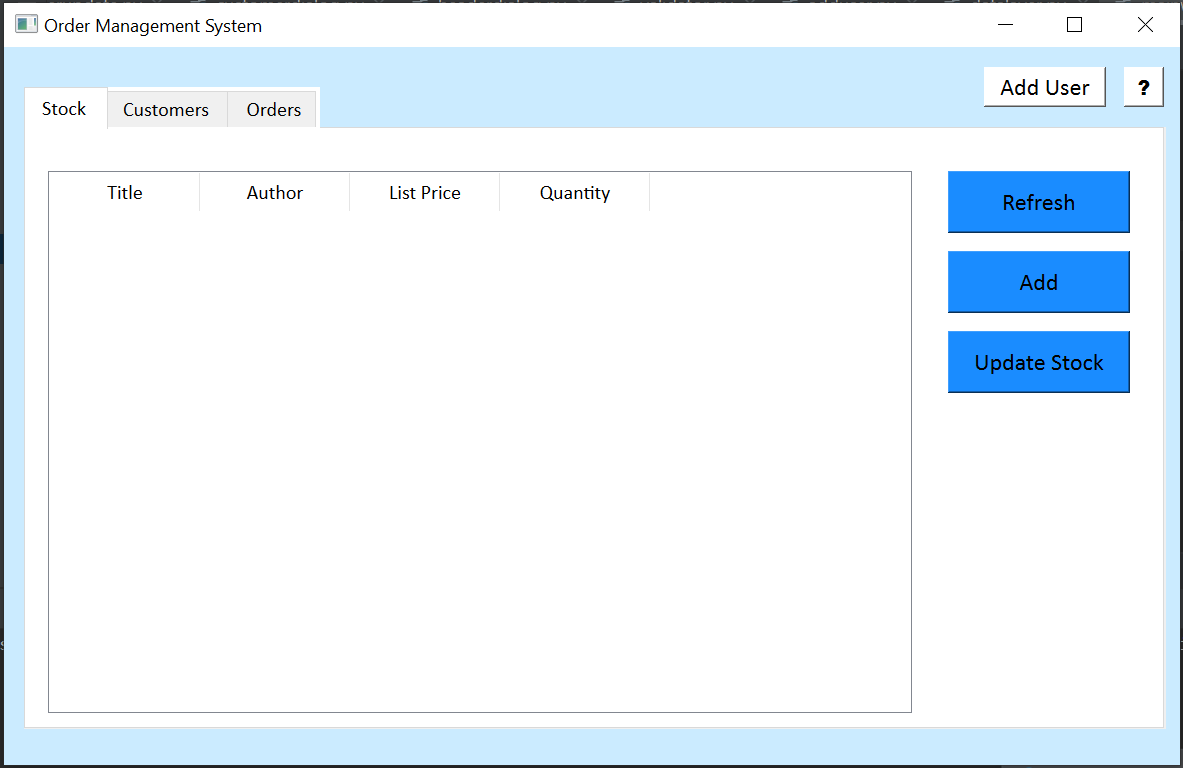
Main Window Code  
The first thing the main window code needs to do is open the main window which uses this code:

However, after implementing this code I found out since my main window had a high resolution, the program was having difficulty scaling the interface causing it to look slightly distorted. I therefore added some code to scale my interface correctly. I also included the database so it could be used throughout the class.

import sys  
from PyQt5 import QtWidgets, uic   
  
win1 = uic.loadUiType("Interface.ui")[0]  
  
  
class FirstWindow(QtWidgets.QMainWindow, win1):  
  
 def \_\_init\_\_(self, parent=None):  
 QtWidgets.QMainWindow.\_\_init\_\_(self, parent)  
 self.setupUi(self)  
  
def main():  
 app = QtWidgets.QApplication(sys.argv)  
 window1 = FirstWindow(None)  
 window1.show()  
 app.exec\_()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

import sys  
from PyQt5 import QtWidgets, uic, QtCore  
  
win1 = uic.loadUiType("Interface.ui")[0]  
  
  
class FirstWindow(QtWidgets.QMainWindow, win1):  
 db = 'Book Selling Database.db'

def \_\_init\_\_(self, parent=None):  
 QtWidgets.QMainWindow.\_\_init\_\_(self, parent)  
 self.setupUi(self)  
  
def main():  
 # Handle high resolution displays:  
 if hasattr(QtCore.Qt, 'AA\_EnableHighDpiScaling'):  
 QtWidgets.QApplication.setAttribute(QtCore.Qt.AA\_EnableHighDpiScaling, True)  
 if hasattr(QtCore.Qt, 'AA\_UseHighDpiPixmaps'):  
 QtWidgets.QApplication.setAttribute(QtCore.Qt.AA\_UseHighDpiPixmaps, True)  
 app = QtWidgets.QApplication(sys.argv)  
 window1 = FirstWindow(None)  
 window1.show()  
 app.exec\_()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Which successfully opened my interface.

I then created refresh functions for the 3 main tabs to load the data into the relevant table.

I first made the refresh customer function to show the customer names. To start I got the customer names from the data layer and then found the customer table using the find child function. I then refreshed the column, inputting the data into the table.

def refresh\_customers(self):  
 results = datalayer.customer\_name(self.db)  
  
 # Find the customer table widget  
 tab: QTabWidget = self.findChild(QTabWidget, "tabWidget")  
 first\_tab: QWidget = tab.findChild(QWidget, "customerTab")  
 table: QTableWidget = first\_tab.findChild(QTableWidget, "customerTable")  
  
 # Clear the table  
 table.clearContents()  
  
 # Set the table row count to be the number of results, excluding the heading row  
 row\_count: int = len(results) - 1  
 table.setRowCount(row\_count)  
  
 row\_index: int = 0  
 heading\_row = True  
 for row in results:  
 if heading\_row:  
 # Skip the heading by not adding it  
 heading\_row = False  
 else:  
 # Set the item in the table to being the customer name  
 name: str = [str(x) for x in row][0]  
 table.setItem(row\_index, 0, QTableWidgetItem(name))  
 row\_index = row\_index + 1

I realised that the code to refresh column will be the same for all columns just with different tables and different data. I therefore decided to make a refresh column procedure that all my tables will use.

I made my refresh stock, customer, order header and lines that get the data from the data layer and then refresh each column with that data.

def refresh\_stock(self):  
  
 # Find the stock table widget  
 table: QTableWidget = self.find\_stock\_table()  
 header: QHeaderView = table.horizontalHeader()  
  
 # Read the data from the database and enter it into the relevant column  
 title = datalayer.stock\_title(db)  
 self.refresh\_column(title, 0, table)  
 author = datalayer.stock\_author(db)  
 self.refresh\_column(author, 1, table)  
 price = datalayer.stock\_list\_price(db)  
 self.refresh\_column(price, 2, table)  
 quantity = datalayer.stock\_quantity(db)  
 self.refresh\_column(quantity, 3, table)

def refresh\_column(self, results, column, table):  
  
 table.setColumnWidth(column, 100)  
 row\_count: int = len(results) - 1  
 table.setRowCount(row\_count)  
  
 row\_index: int = 0  
 heading\_row = True  
 for row in results:  
 if heading\_row:  
 # Skip the heading by not adding it  
 heading\_row = False  
 else:  
 # Set the item in the table to being the customer name  
 name: str = [str(x) for x in row][0]  
 item: QTableWidgetItem = QTableWidgetItem(name)  
 table.setItem(row\_index, column, QTableWidgetItem(name))  
 row\_index = row\_index + 1

def refresh\_customers(self):  
 # Find the customer table widget  
 table: QTableWidget = self.find\_customer\_table()  
 header: QHeaderView = table.horizontalHeader()  
  
 # Read the data from the database and enter it into the relevant column  
 names = datalayer.customer\_name(db)  
 self.refresh\_column(names, 0, table)  
 email = datalayer.customer\_email(db)  
 self.refresh\_column(email, 1, table)  
 tel = datalayer.customer\_tel(db)  
 self.refresh\_column(tel, 2, table)  
 address = datalayer.customer\_address(db)  
 self.refresh\_column(address, 3, table)  
 discount = datalayer.customer\_discount(db)  
 self.refresh\_column(discount, 4, table)  
 totalSpent = datalayer.customer\_total\_spent(db)  
 self.refresh\_column(totalSpent, 5, table)

For refreshing order headers, I had to add an order number column so when an order header is clicked the primary key can be found to find the relevant order lines.

def refresh\_header(self):  
 # Find the stock table widget  
 table: QTableWidget = self.find\_headers\_table()  
 header: QHeaderView = table.horizontalHeader()  
  
 # Read the data from the database and enter it into the relevant column  
 customerName = datalayer.order\_customer\_name(db)  
 self.refresh\_column(customerName, 0, table)  
 deliveryAddress = datalayer.order\_delivery\_address(db)  
 self.refresh\_column(deliveryAddress, 1, table)  
 deliveryCharge = datalayer.order\_delivery\_charge(db)  
 self.refresh\_column(deliveryCharge, 2, table)  
 date = datalayer.order\_date(db)  
 self.refresh\_column(date, 3, table)  
 totalCost = datalayer.order\_cost(db)  
 self.refresh\_column(totalCost, 4, table)  
 # add orderNumber column for order lines  
 orderNumber = datalayer.order\_number(db)  
 self.refresh\_column(orderNumber, 5, table)

I later revisited the refresh column procedure to improve it. I had found when sorting the data with the built-in sort function, the data gets sorted alphabetically not numerically so that 12.99 would be before 8.99 as 1 is less than 8. To correct this, I added a numeric parameter to the refresh column procedure that used a separate class that would store the numeric value or zero if it is None. I also changed the built in \_\_lt\_\_ function used for sorting so that it would instead compare the numeric values.

class NumericTableWidgetItem(QTableWidgetItem):  
 def \_\_init\_\_(self, number):  
  
 # For numbers store float so can be compared numerically for sorting algorithms  
 QTableWidgetItem.\_\_init\_\_(self, number, QTableWidgetItem.UserType)  
  
 # Convert None to the value 0 to group all None as smallest value  
 if number == "None":  
 number = "0"  
 self.\_\_number = float(number)  
  
 def \_\_lt\_\_(self, other):  
 return self.\_\_number < other.\_\_number

I also added a price parameter to the refresh column procedure so prices will be shown to 2 decimal places. I included a size parameter so instead of all my columns being the same width, columns with short pieces of data like quantity would have a narrower column than long pieces of data such as an address.

def refresh\_column(self, results, column, table, size, numeric=False, price=False):  
  
 # Set the table to column to the size passed into the function  
 table.setColumnWidth(column, size)  
  
 # Set the row to the length of results  
 row\_count: int = len(results)  
 table.setRowCount(row\_count)  
  
 # Set the starting index to 0  
 row\_index: int = 0  
  
 if not price:  
 for row in results:  
 # Set the item in the table to being the name  
 name: str = [str(x) for x in row][0]  
  
 if numeric:  
 # use our own class so that sorting of numeric values works correctly  
 table.setItem(row\_index, column,  
 NumericTableWidgetItem(name))  
 else:  
 table.setItem(row\_index, column, QTableWidgetItem(name))  
  
 # Increment the row pointer  
 row\_index = row\_index + 1  
 else:  
 for row in results:  
 # Set the item in the table to being the name  
 number = [self.get\_price(x) for x in row][0]  
  
 table.setItem(row\_index, column,  
 NumericTableWidgetItem(number))  
 row\_index = row\_index + 1  
  
def get\_price(self, number):  
 price = float("{0:.2f}".format(number))  
 price = str(price)  
 return price

I then updated my refresh stock, customer, order header and order lines. I added the size, numeric and price parameter for when I called the refresh column procedure. I also turned sort indictor off so when refreshing the table, it would no longer show where it was previously sorted.

def refresh\_stock(self):  
  
 # Find the stock table widget  
 table: QTableWidget = self.find\_stock\_table()  
 header: QHeaderView = table.horizontalHeader()  
  
 # Read the data from the database and enter it into the relevant column  
 title = datalayer.stock\_title(db)  
 self.refresh\_column(title, 0, table, 150)  
 author = datalayer.stock\_author(db)  
 self.refresh\_column(author, 1, table, 150)  
 # Set numeric to True for price and quantity  
 price = datalayer.stock\_list\_price(db)  
 self.refresh\_column(price, 2, table, 50, True, True)  
 quantity = datalayer.stock\_quantity(db)  
 self.refresh\_column(quantity, 3, table, 50, True)  
  
 header.setSortIndicatorShown(False)

linesTable.setEditTriggers(QtWidgets.QTableWidget.NoEditTriggers)

def refresh\_customers(self):  
 # Find the customer table widget  
 table: QTableWidget = self.find\_customer\_table()  
 header: QHeaderView = table.horizontalHeader()  
  
 # Read the data from the database and enter it into the relevant column  
 names = datalayer.customer\_name(db)  
 self.refresh\_column(names, 0, table, 100)  
 email = datalayer.customer\_email(db)  
 self.refresh\_column(email, 1, table, 100)  
 tel = datalayer.customer\_tel(db)  
 self.refresh\_column(tel, 2, table, 100)  
 address = datalayer.customer\_address(db)  
 self.refresh\_column(address, 3, table, 100)  
 # Set numeric to true for discount  
 discount = datalayer.customer\_discount(db)  
 self.refresh\_column(discount, 4, table, 50, True)  
 totalSpent = datalayer.customer\_total\_spent(db)  
 self.refresh\_column(totalSpent, 5, table, 50, True, True)  
  
 header.setSortIndicatorShown(False)

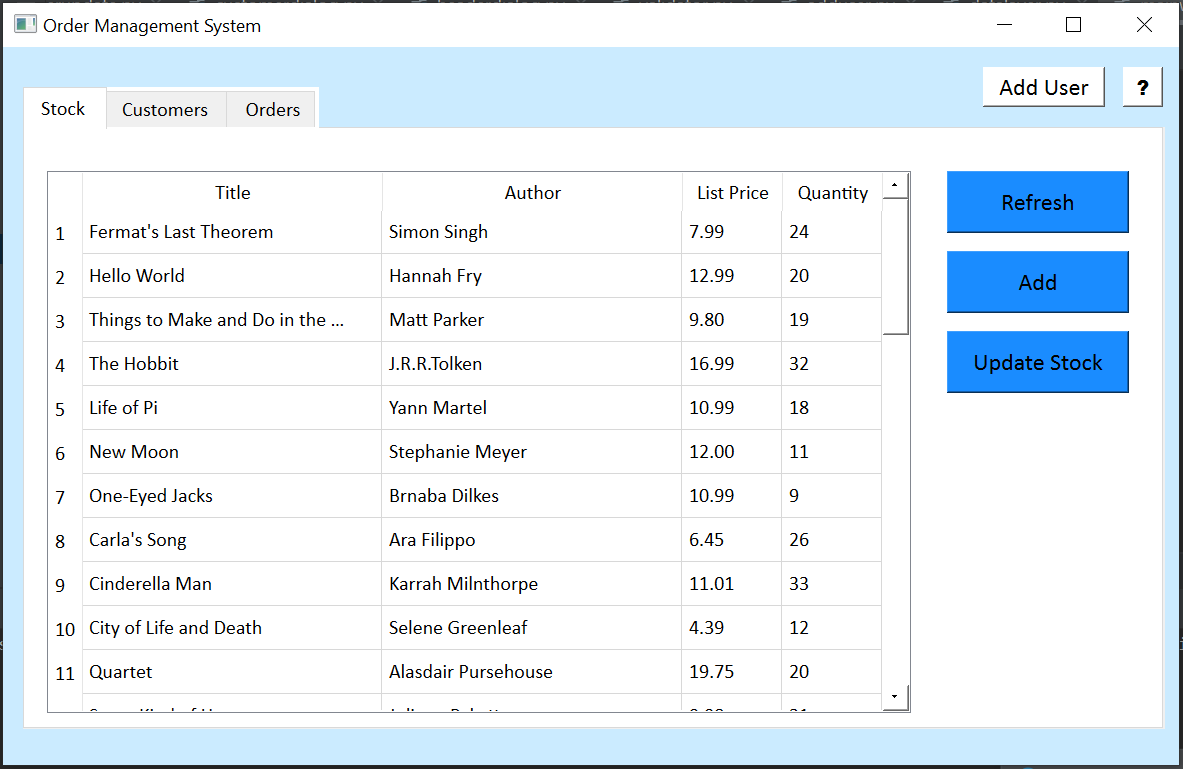
For the refresh order headers, I chose to make the order number have a column width of 0 so it won’t be visible to the user as it is not relevant to them.

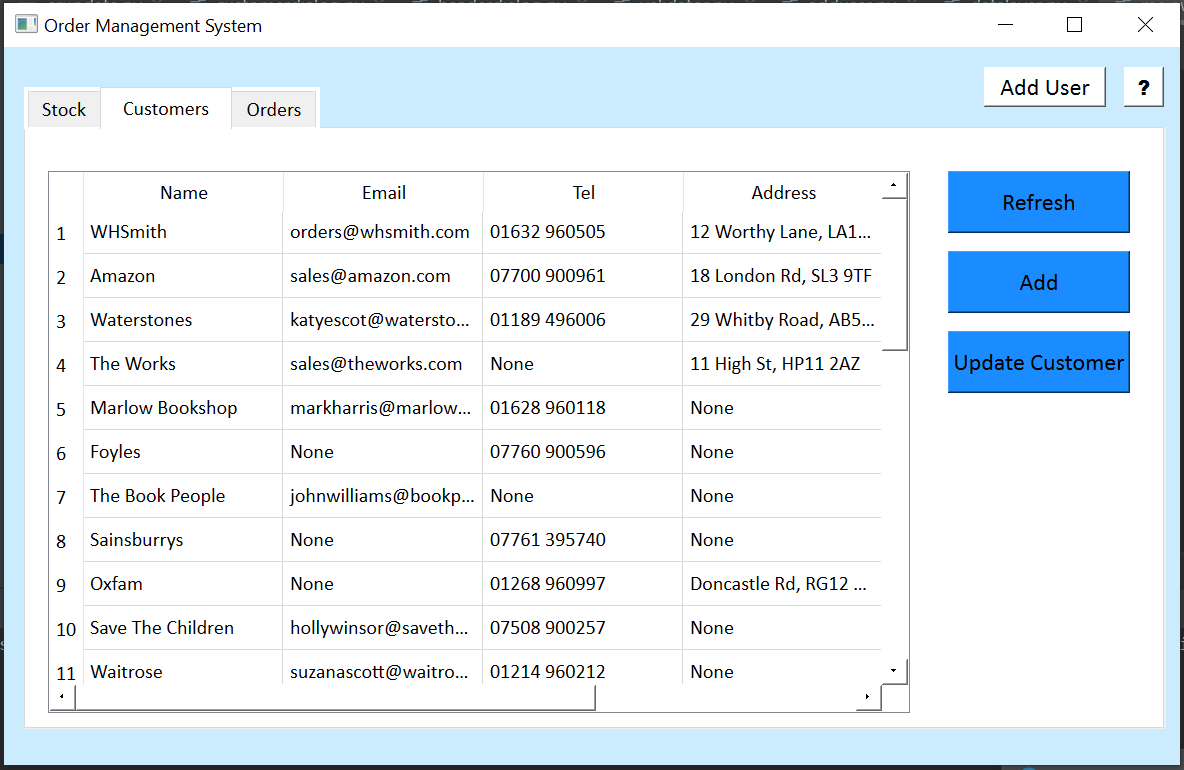
def refresh\_header(self):  
 # Find the stock table widget  
 headerTable: QTableWidget = self.find\_headers\_table()  
 orderTable: QTableWidget = self.find\_lines\_table()  
 header: QHeaderView = headerTable.horizontalHeader()  
  
 # Read the data from the database and enter it into the relevant column  
 customerName = datalayer.order\_customer\_name(db)  
 self.refresh\_column(customerName, 0, headerTable, 100)  
 deliveryAddress = datalayer.order\_delivery\_address(db)  
 self.refresh\_column(deliveryAddress, 1, headerTable, 100)  
 # Set numeric True for delivery charge  
 deliveryCharge = datalayer.order\_delivery\_charge(db)  
 self.refresh\_column(deliveryCharge, 2, headerTable, 75, True, True)  
 date = datalayer.order\_date(db)  
 self.refresh\_column(date, 3, headerTable, 75)  
 totalCost = datalayer.order\_cost(db)  
 self.refresh\_column(totalCost, 4, headerTable, 45, True, True)  
 # Set orderNumber column size to zero so it cant be seen  
 orderNumber = datalayer.order\_number(db)  
 self.refresh\_column(orderNumber, 5, headerTable, 0)  
  
 header.setSortIndicatorShown(False)  
  
 # clear order table when the page is refreshed  
 while orderTable.rowCount() > 0:  
 orderTable.removeRow(0)  
  
 self.clickHeaderLabel.show()

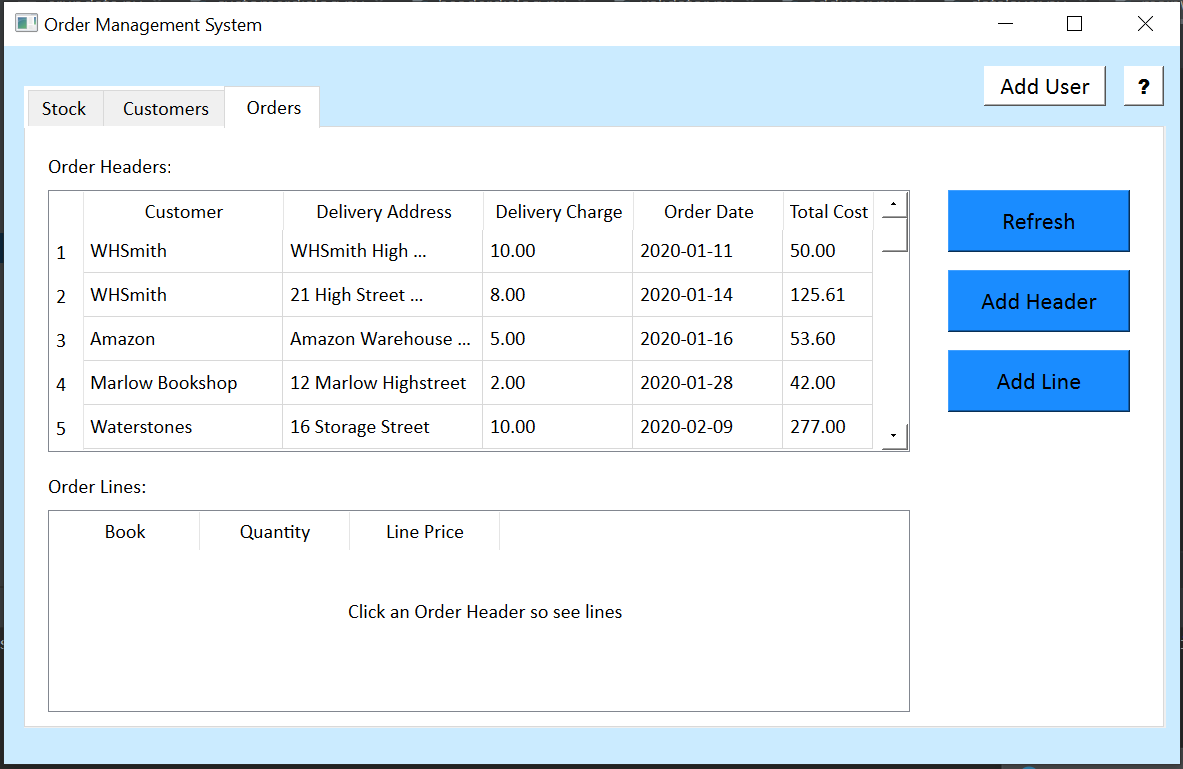
I originally called the refresh function for each of my tables when I loaded my interface in the main window class.

def \_\_init\_\_(self, parent=None):  
 QtWidgets.QMainWindow.\_\_init\_\_(self, parent)  
 self.setupUi(self)  
 self.refresh\_customers()  
 self.refresh\_stock()  
 self.refresh\_header()

This worked and successfully loaded all my data into the tables.







To reduce start up time I created a refresh tab function in my main window class so instead of loading all the data when you start the program it only loads the information on the table when you click onto that tab.

def refresh\_tab(self, tabnumber: int):  
  
 # load data for the relevant table  
 if tabnumber == 0:  
 self.refresh\_stock()  
 elif tabnumber == 1:  
 self.refresh\_customers()  
 elif tabnumber == 2:  
 self.refresh\_header()  
 else:  
 pass

I then realized refreshing the table when you click on the tab loses any changes or sorting you had done on that tab, so I added a variable to track if you have already loaded that tab. Then the program will only refresh the data if you have not previously loaded that tab.

loadedTab = [False]\*3

def refresh\_tab(self, tabnumber: int):  
  
 # check tab has not been loaded  
 if not self.loadedTab[tabnumber]:  
 # load data for the relevant table  
 if tabnumber == 0:  
 self.refresh\_stock()  
 elif tabnumber == 1:  
 self.refresh\_customers()  
 elif tabnumber == 2:  
 self.refresh\_header()  
 # set that tab to loaded  
 self.loadedTab[tabnumber] = True  
 else:  
 pass

I then decided to make the refresh order lines. Instead of refreshing the order lines when you clicked on the tab, I wanted to refresh them when you clicked a row on the order header. This is because I only wanted to show the order lines relevant to an order header. To do this I had to first call a function when the header was clicked. It would also need to return the index or the selected row. To do this I added some code in the \_\_init\_\_ function.

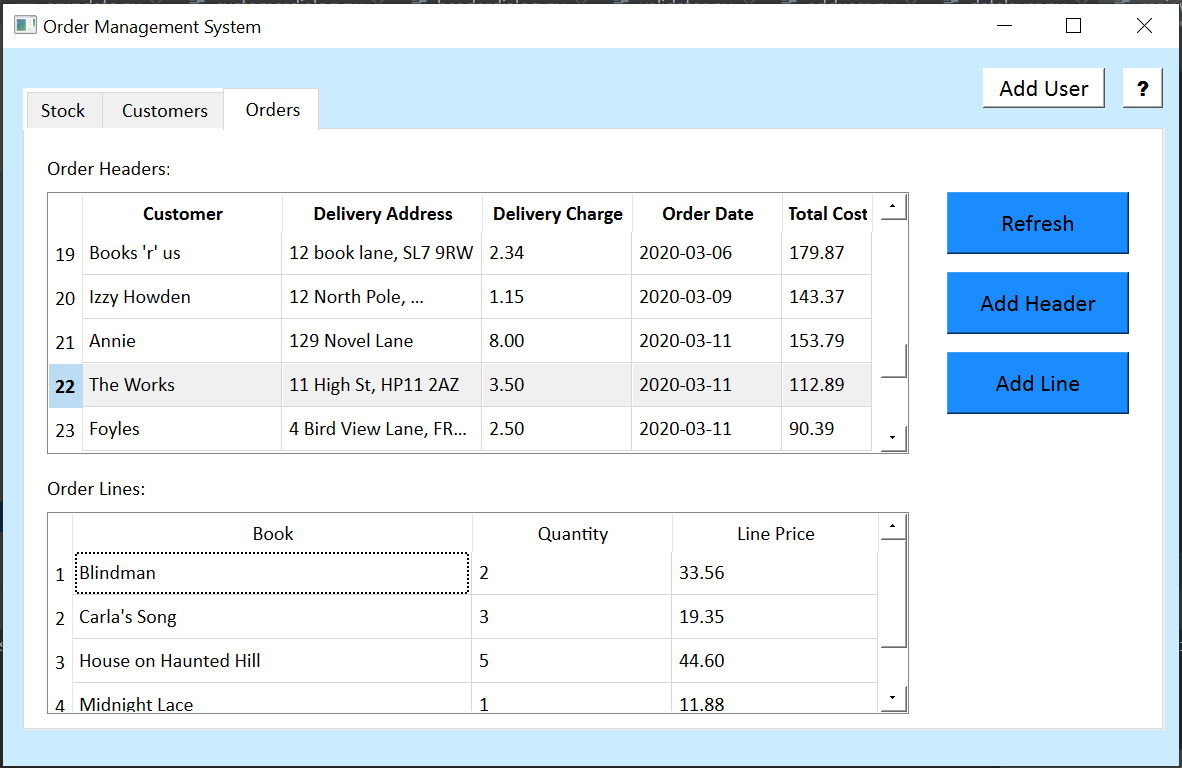
headers\_table: QTableWidget = self.find\_headers\_table()  
headers\_vertical\_header: QHeaderView = headers\_table.verticalHeader()  
headers\_vertical\_header.sectionClicked.connect(self.header\_row\_clicked)

This called a separate function where I found the Order Number of the selected row sine this was the primary key and will be needed to find what data in the order lines was relevant.

def header\_row\_clicked(self, logicalindex: int):  
 headersTable: QTableWidget = self.find\_headers\_table()  
  
 # find Order Number that has been selected  
 self.selectedOrderNumber = headersTable.item(logicalindex, 5).text()  
  
 self.clickHeaderLabel.hide()  
 # refresh the OrderLines for relevant order number  
 self.refresh\_order\_lines()

I then made my refresh lines function to add the data into the order lines table on my interface. I also had to get the information from my database by using functions in my datalayer.

def refresh\_order\_lines(self):  
 linesTable: QTableWidget = self.find\_lines\_table()  
  
 # repeat the function for each column  
 bookName = datalayer.order\_book\_name(db, self.selectedOrderNumber)  
 self.refresh\_column(bookName, 0, linesTable, 200)  
  
 # Set numeric as true so it can be sorted correctly  
 quantity = datalayer.order\_quantity(db, self.selectedOrderNumber)  
 self.refresh\_column(quantity, 1, linesTable, 100, True)  
 lineCost = datalayer.order\_line\_cost(db, self.selectedOrderNumber)  
 self.refresh\_column(lineCost, 2, linesTable, 103, True, True)

I then ran my code to find out if it worked correctly and loaded the right information into the table.

I then decided to update my sorting procedures that sort the table based of the column selected. This is because I now could track the current tab by adding code to the refresh tab procedure allowing me to make the code more efficient by having one procedure instead of a separate one for each tab. In the \_\_init\_\_ procedure I had used the built-in function of sectionClicked.connect() to call the relevant header\_section\_clicked procedure.

Previously my code for sorting the tables used the following init procedure:

# Get the header view from the stock table and connect it for when the section is clicked  
stock\_table: QTableWidget = self.find\_stock\_table()  
stock\_header: QHeaderView = stock\_table.horizontalHeader()  
stock\_header.sectionClicked.connect(self.stock\_header\_section\_clicked)  
  
# Get the header view from the customer table and connect it for when the section is clicked  
customer\_table: QTableWidget = self.find\_customer\_table()  
customer\_header: QHeaderView = customer\_table.horizontalHeader()  
customer\_header.sectionClicked.connect(self.customer\_header\_section\_clicked)  
  
# Get the header view from the headers table and connect it for when the section is clicked  
headers\_table: QTableWidget = self.find\_headers\_table()  
headers\_horizontal\_header: QHeaderView = headers\_table.horizontalHeader()  
headers\_horizontal\_header.sectionClicked.connect(self.headers\_header\_section\_clicked)  
headers\_vertical\_header: QHeaderView = headers\_table.verticalHeader()  
headers\_vertical\_header.sectionClicked.connect(self.header\_row\_clicked)

I then used the following functions to sort each table since the selection clicked function only returned an index for the header not which table it was that had been selected.

def stock\_header\_section\_clicked(self, logicalindex: int):  
 # Find the stock table widget and header row  
 table: QTableWidget = self.find\_stock\_table()  
 header: QHeaderView = table.horizontalHeader()  
  
 # Check if the table is currently sorted in ascending order and if so sort in descending order  
 if self.selectedStockHeader != logicalindex:  
 table.sortItems(logicalindex, QtCore.Qt.AscendingOrder)  
 # Change indicator on the header  
 header.setSortIndicator(logicalindex, QtCore.Qt.AscendingOrder)  
 # set selectedStockHeader to the new selected header index  
 self.selectedStockHeader = logicalindex  
 else:  
 table.sortItems(logicalindex, QtCore.Qt.DescendingOrder)  
 # Change indicator on the header  
 header.setSortIndicator(logicalindex, QtCore.Qt.DescendingOrder)  
 # reset selectedStockHeader as the column is currently sorted in descending order  
 self.selectedStockHeader = None  
  
 # Show the indicator  
 header.setSortIndicatorShown(True)

def headers\_header\_section\_clicked(self, logicalindex: int):  
 # Find the header table widget and header row  
 table: QTableWidget = self.find\_headers\_table()  
 header: QHeaderView = table.horizontalHeader()  
  
 # Check if the table is currently sorted in ascending order and if so sort in descending order  
 if self.selectedHeaderHeader != logicalindex:  
 table.sortItems(logicalindex, QtCore.Qt.AscendingOrder)  
 # Change indicator on the header  
 header.setSortIndicator(logicalindex, QtCore.Qt.AscendingOrder)  
 # Set selectedHeaderHeader to the new selected header index  
 self.selectedHeaderHeader = logicalindex  
 else:  
 table.sortItems(logicalindex, QtCore.Qt.DescendingOrder)  
 # Change indicator on the header  
 header.setSortIndicator(logicalindex, QtCore.Qt.DescendingOrder)  
 # Reset selectedHeaderHeader as the column is currently sorted in descending order  
 self.selectedHeaderHeader = None

# Show the indicator  
 header.setSortIndicatorShown(True)

def customer\_header\_section\_clicked(self, logicalindex: int):  
 # Find the customer table widget and header row  
 table: QTableWidget = self.find\_customer\_table()  
 header: QHeaderView = table.horizontalHeader()  
  
 # Check if the table is currently sorted in ascending order and if so sort in descending order  
 if self.selectedCustomerHeader != logicalindex:  
 table.sortItems(logicalindex, QtCore.Qt.AscendingOrder)  
 # Change indicator on the header  
 header.setSortIndicator(logicalindex, QtCore.Qt.AscendingOrder)  
 # Set selectedCustomerHeader to the new selected header index  
 self.selectedCustomerHeader = logicalindex  
 else:  
 table.sortItems(logicalindex, QtCore.Qt.DescendingOrder)  
 # Change an indicator on the header  
 header.setSortIndicator(logicalindex, QtCore.Qt.DescendingOrder)  
 # Reset selectedStockHeader as the column is currently sorted in descending order  
 self.selectedCustomerHeader = None  
  
 # Show the indicator  
 header.setSortIndicatorShown(True)

I then updated refresh\_tab() to store change the current tab that was set to zero at the start of the class.

currentTab = 0

def refresh\_tab(self, tabnumber: int):  
  
 # check tab has not been loaded  
 if not self.loadedTab[tabnumber]:  
 # load data for the relevant table  
 if tabnumber == 0:  
 self.refresh\_stock()  
 elif tabnumber == 1:  
 self.refresh\_customers()  
 elif tabnumber == 2:  
 self.refresh\_header()  
 # set that tab to loaded  
 self.loadedTab[tabnumber] = True  
 # update current tab number  
 self.currentTab = tabnumber  
 else:  
 self.currentTab = tabnumber

I then updated the \_\_init\_\_ procedure to call one procedure instead of a procedure specific to that table when the header section is clicked.

# Get the header view from the stock table and connect it for when the section is clicked  
stock\_table: QTableWidget = self.find\_stock\_table()  
stock\_header: QHeaderView = stock\_table.horizontalHeader()  
stock\_header.sectionClicked.connect(self.header\_section\_clicked)  
  
# Get the header view from the customer table and connect it for when the section is clicked  
customer\_table: QTableWidget = self.find\_customer\_table()  
customer\_header: QHeaderView = customer\_table.horizontalHeader()  
customer\_header.sectionClicked.connect(self.header\_section\_clicked)  
  
# Get the header view from the headers table and connect it for when the section is clicked  
headers\_table: QTableWidget = self.find\_headers\_table()  
headers\_horizontal\_header: QHeaderView = headers\_table.horizontalHeader()  
headers\_horizontal\_header.sectionClicked.connect(self.header\_section\_clicked)  
headers\_vertical\_header: QHeaderView = headers\_table.verticalHeader()  
headers\_vertical\_header.sectionClicked.connect(self.header\_row\_clicked)

I then coded a procedure that would work for any current tab. To save space I also replaced the separate variables recording which header was selected for each table with one array that could store the selected header for all the tables. This reduced the amount of code and made the program more efficient.

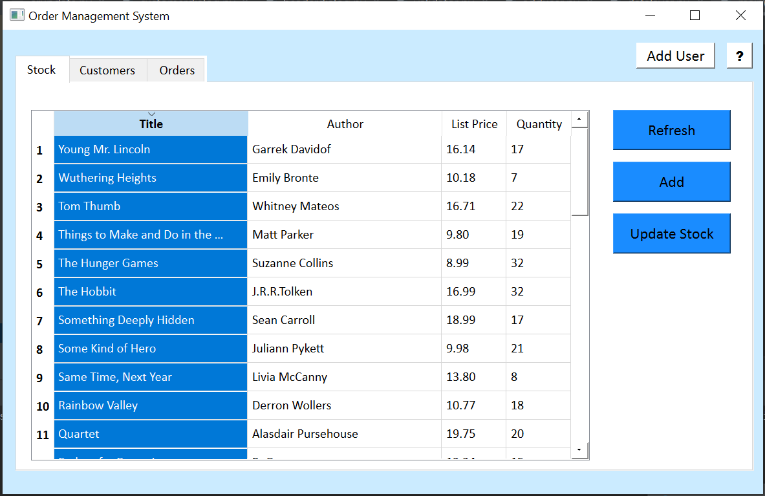
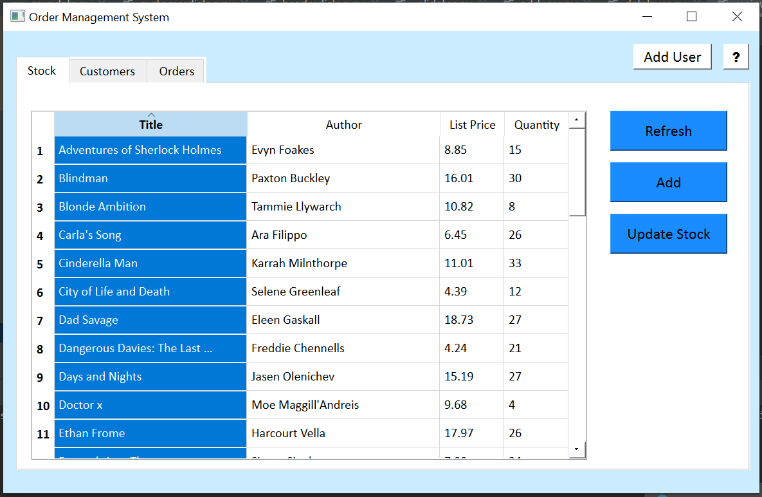
selectedHeader = [None]\*3

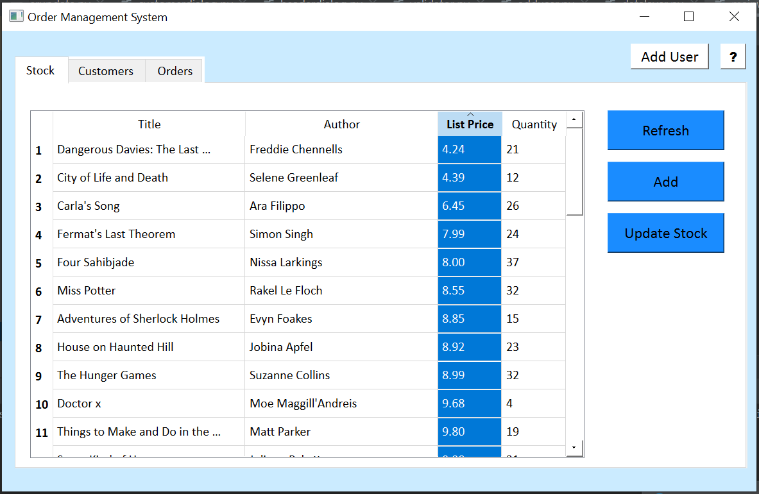
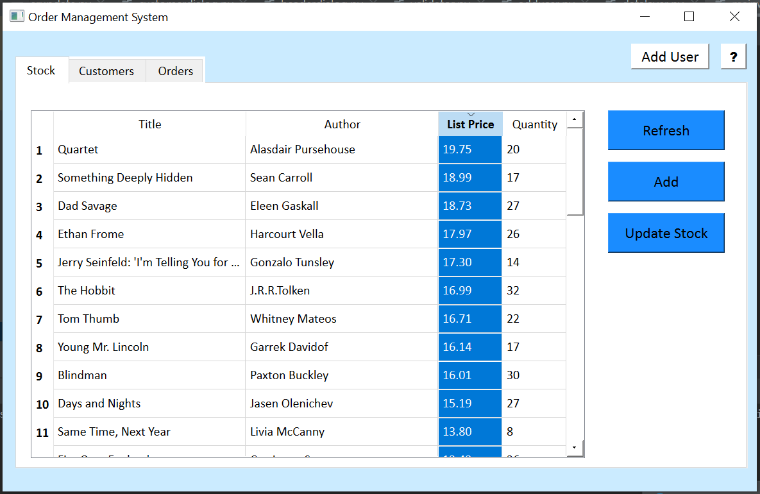
def HeaderSectionClicked(self, logicalindex: int):  
  
 # Find the relevant table and selected header  
 if self.currentTab == 0:  
 table: QTableWidget = self.find\_stock\_table()  
 selectedHeader = self.selectedHeader[0]  
 elif self.currentTab == 1:  
 table: QTableWidget = self.find\_customer\_table()  
 selectedHeader = self.selectedHeader[1]  
 elif self.currentTab == 2:  
 table: QTableWidget = self.find\_headers\_table()  
 selectedHeader = self.selectedHeader[2]  
  
 header: QHeaderView = table.horizontalHeader()  
  
 # Check if the table is currently sorted in ascending order and if so sort in descending order  
 if selectedHeader != logicalindex:  
 table.sortItems(logicalindex, QtCore.Qt.AscendingOrder)  
 # Change indicator on the header  
 header.setSortIndicator(logicalindex, QtCore.Qt.AscendingOrder)  
 # set selectedHeader to the new selected header index  
 selectedHeader = logicalindex  
 else:  
 table.sortItems(logicalindex, QtCore.Qt.DescendingOrder)  
 # Change indicator on the header  
 header.setSortIndicator(logicalindex, QtCore.Qt.DescendingOrder)  
 # reset selectedHeader as the column is currently sorted in descending order  
 selectedHeader = None  
  
 # show indicator so clear it is clear to the user how it is sorted  
 header.setSortIndicatorShown(True)  
  
 # return the update selected header  
 self.selectedHeader[self.currentTab] = selectedHeader

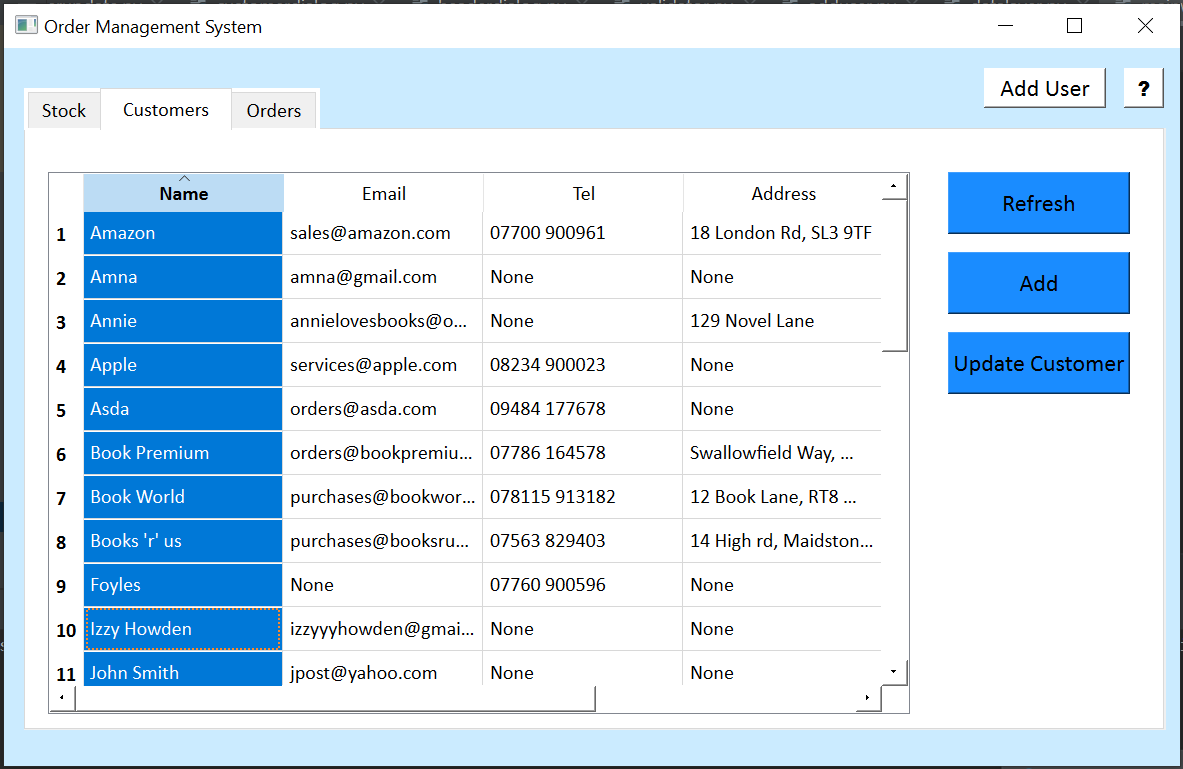
I also added the following code to the refresh functions to reset that tables selected header to None so it would not automatically sort in descending order when you re-click the last header it was sorted before the data was refreshed.

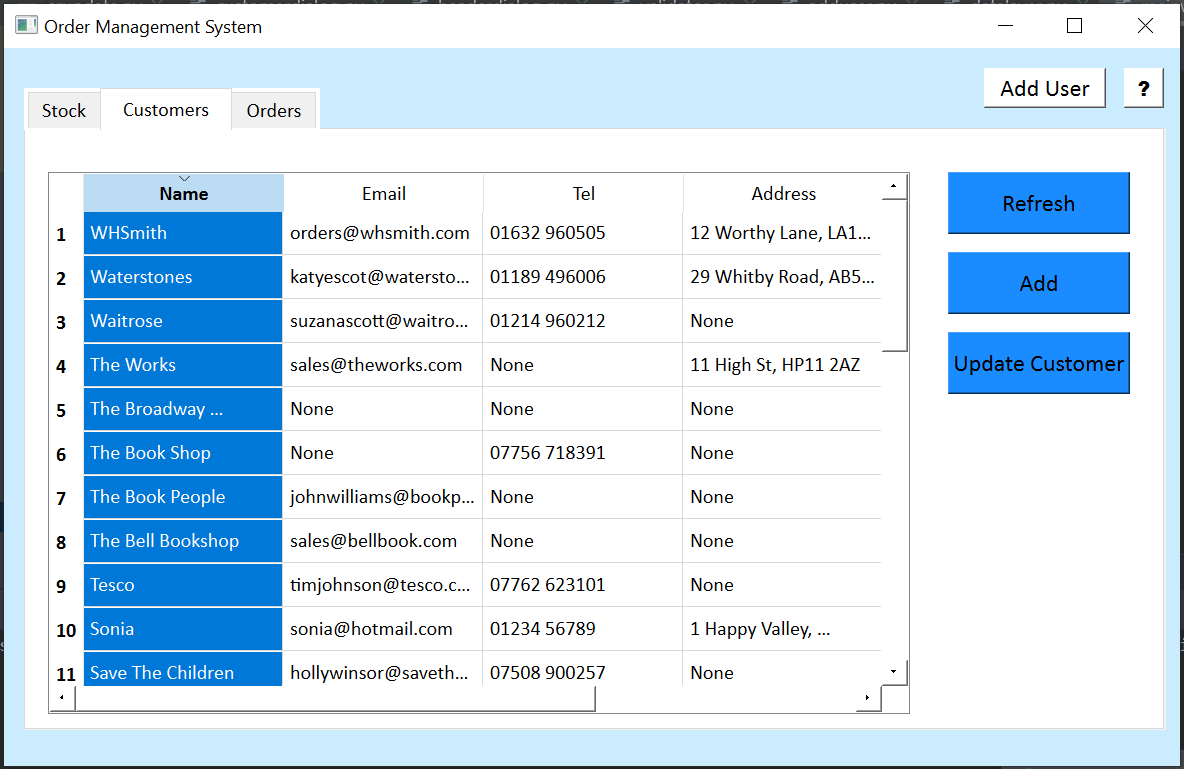
self.selectedHeader[self.currentTab] = None

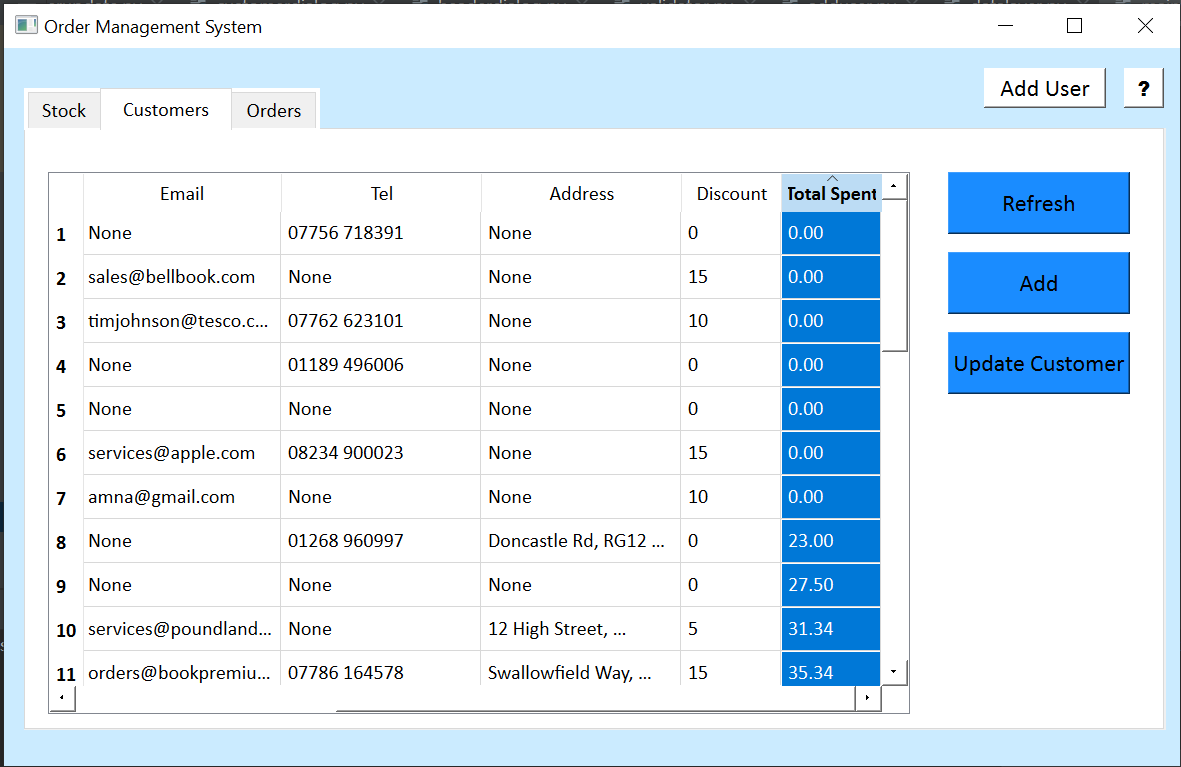
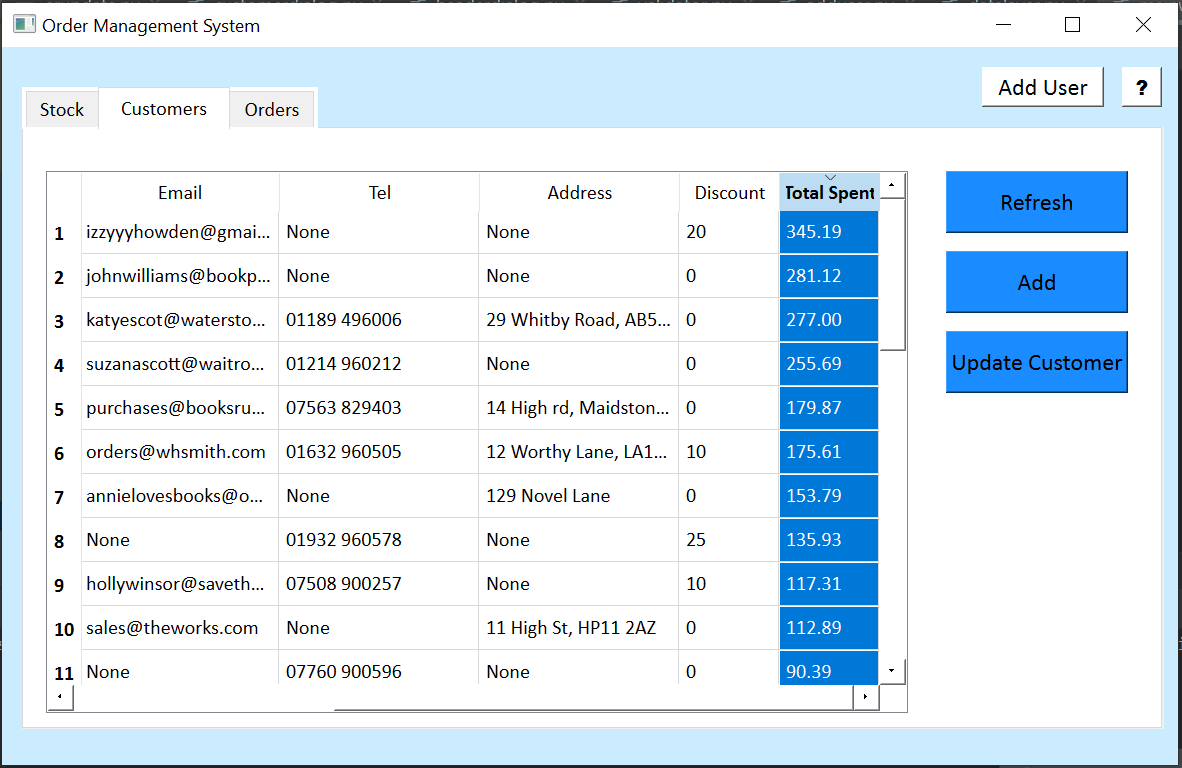
I then tested this to check if it sorted my data correctly for both numbers and text.

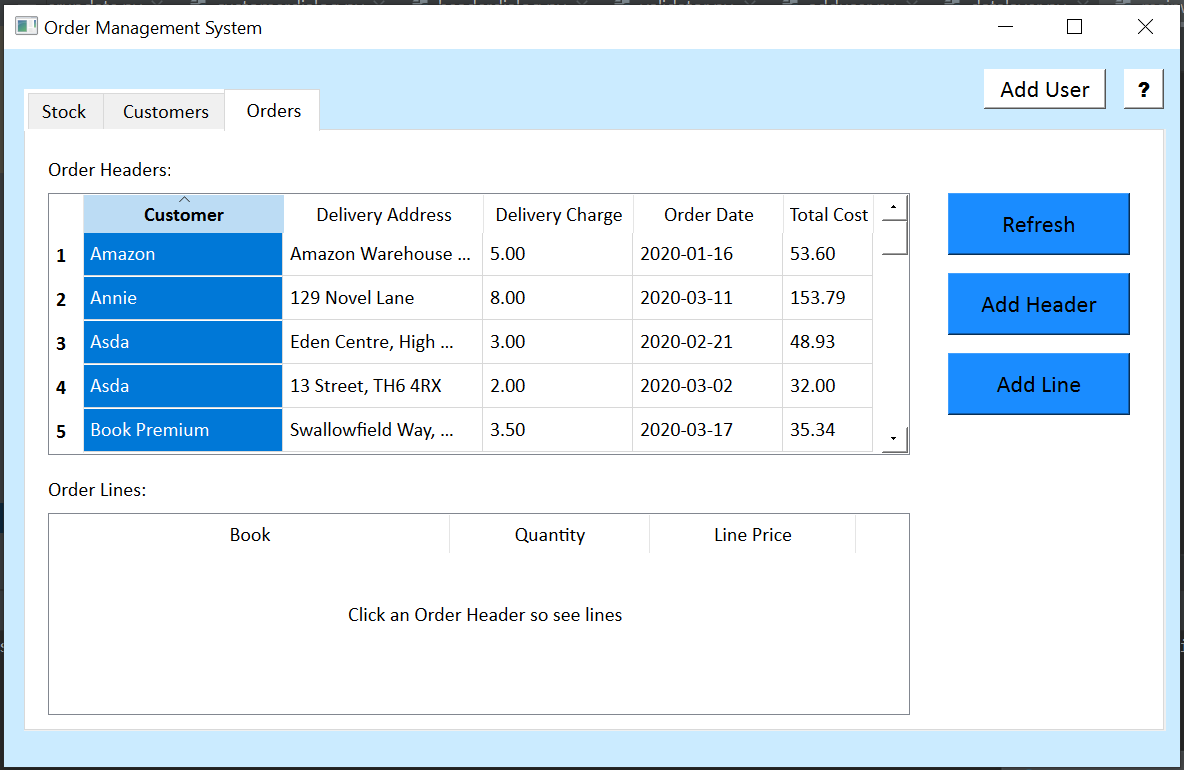


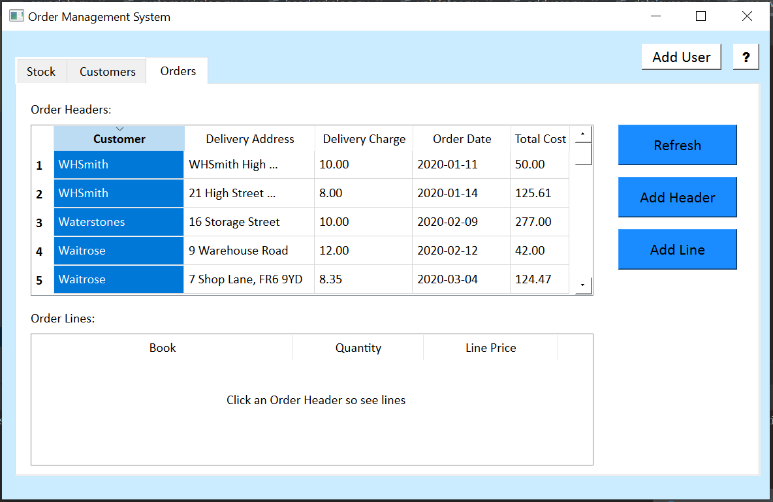


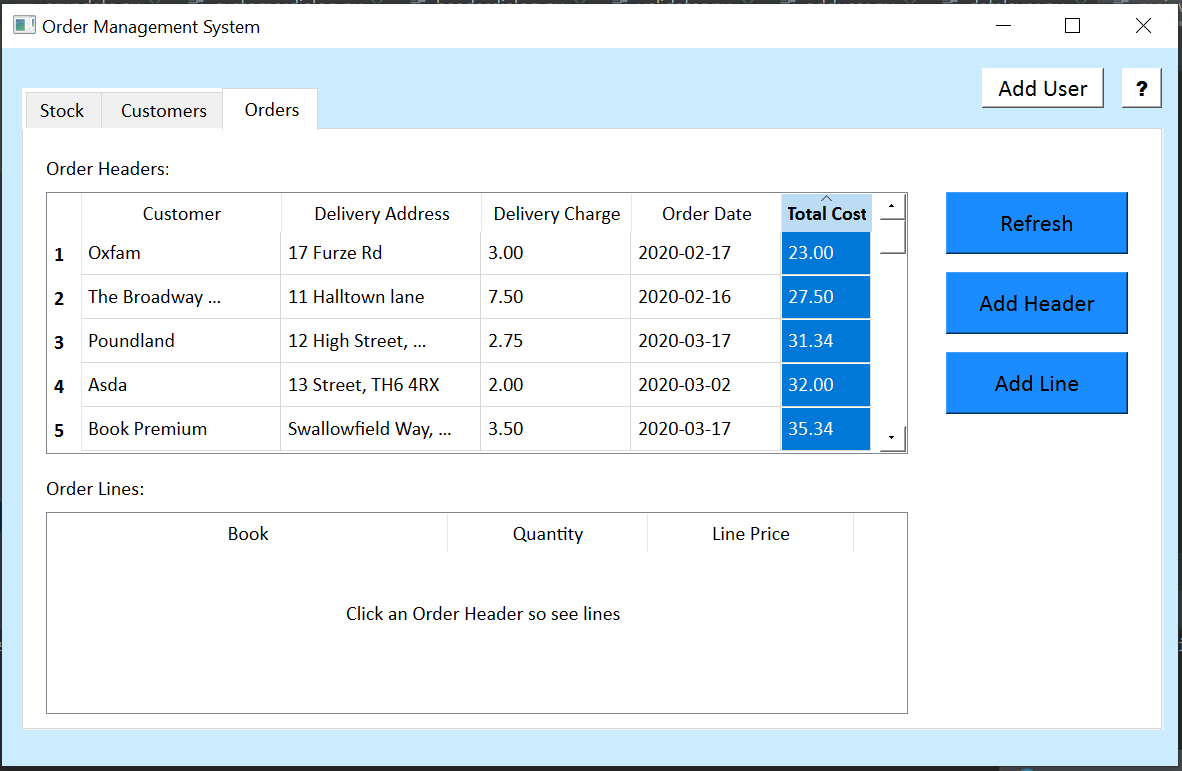
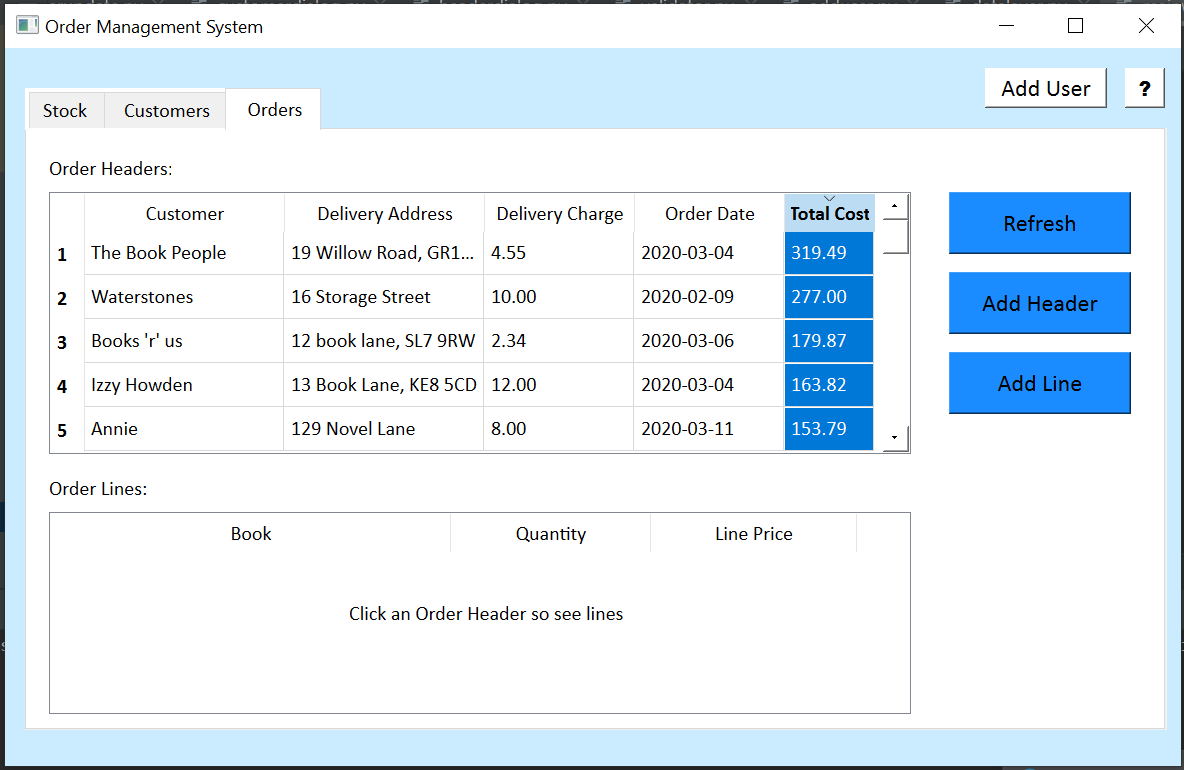






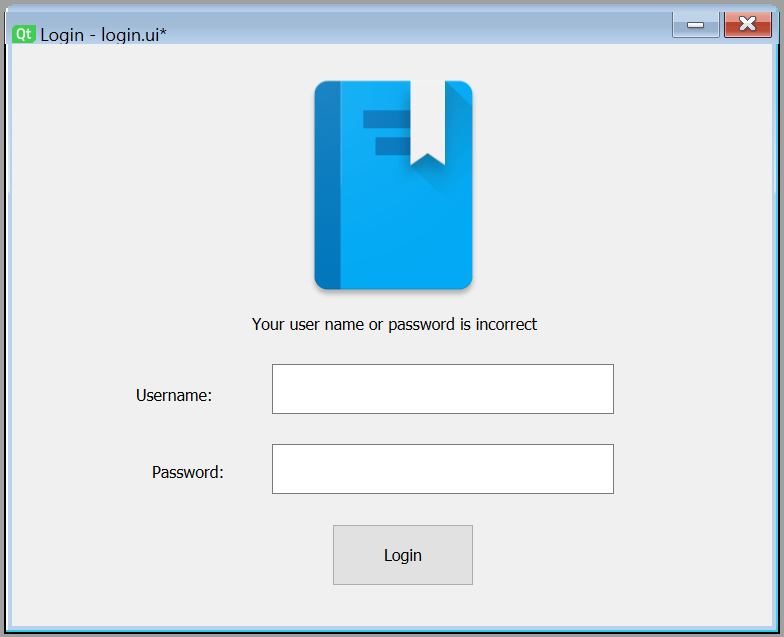






Since I had then completed the procedures that would run just on the main window, I decided to work on my other windows.

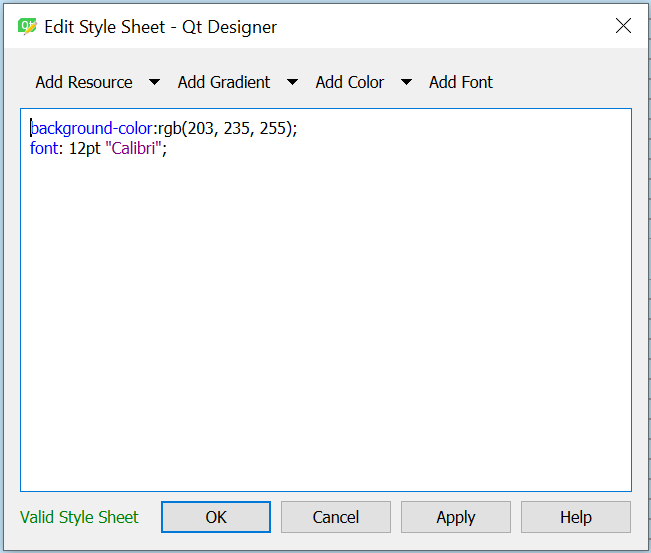
Login  
I decided to first focus on the login which I designed using QtDesigner.

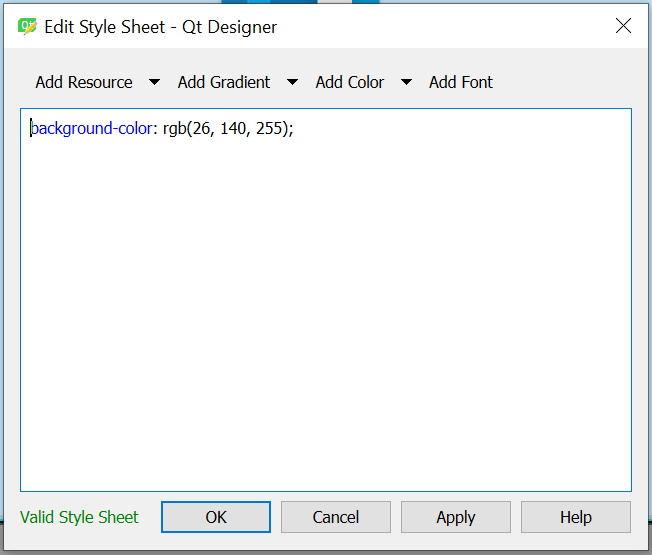


1. enterButton - QPushButton
2. incorrectDetails - QLabel
3. logo - QLabel
4. nameLabel – QLabel
5. nameLineEdit – QLineEdit
6. passwordLabel – QLabel
7. passwordLineEdit – QLineEdit

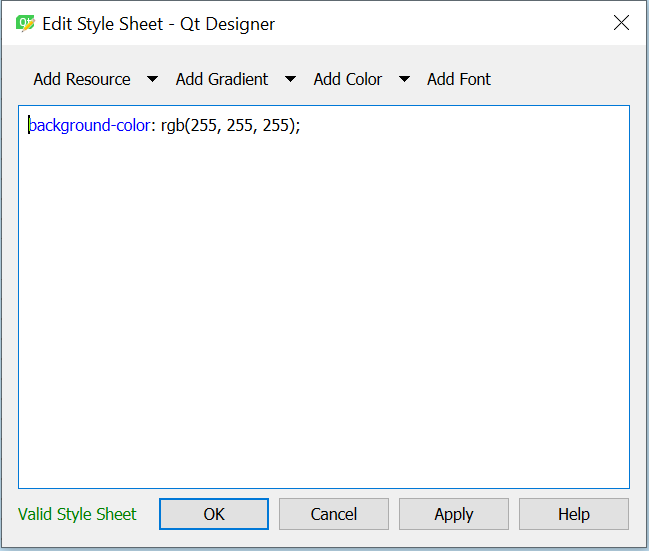
I added the logo by setting its pixmap to the image I wanted it to display. I also set the error label to align the text to the centre.

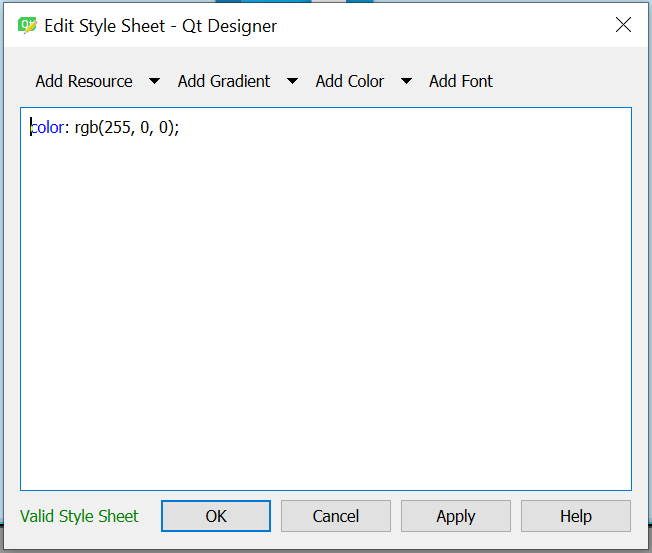
I then started changing the style sheets so I could make the design more aesthetically pleasing like my design. I first changed the style sheet of my background. This will change the colour of the background as well as set the font style and size for the rest of the widgets.



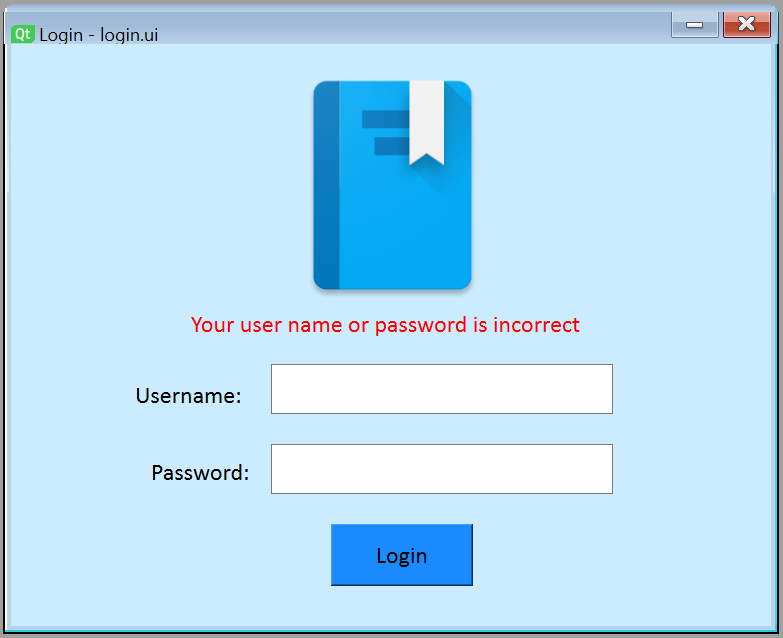
I also changed the style sheet of the button to make it the same dark blue as the buttons on the main window.

I also set the line edits to have a background of white by editing the style sheet and changed the echo mode to password to hide the information typed in, This was in my design because it stopped on lookers from seeing other peoples’ password.



The last style sheet I had to change was the error message as I wanted the text to be red so it would stand out to the user and tell them what the problem was.

This was my final edit I made to the graphics of my login page.



Add dialogs  
I then designed the add dialogs for each of the tables using QtDesigner.

