1 Introduction

1.1 Aim

“Top Shelf” is a desktop-based, stand-alone application designed for whisky collectors and aficionados. It enables users to record and track their whisky collection.

1.2 Instructions for Use

Upon opening the application, the user is greeted with a welcome window, which informs them how many bottles are currently in their collection. From here, the main system actions can be accessed by clicking on one of the following buttons: “Show Collection”, “Add a Bottle”, “Edit a Bottle”, “Remove a Bottle”, “Find a Bottle”.

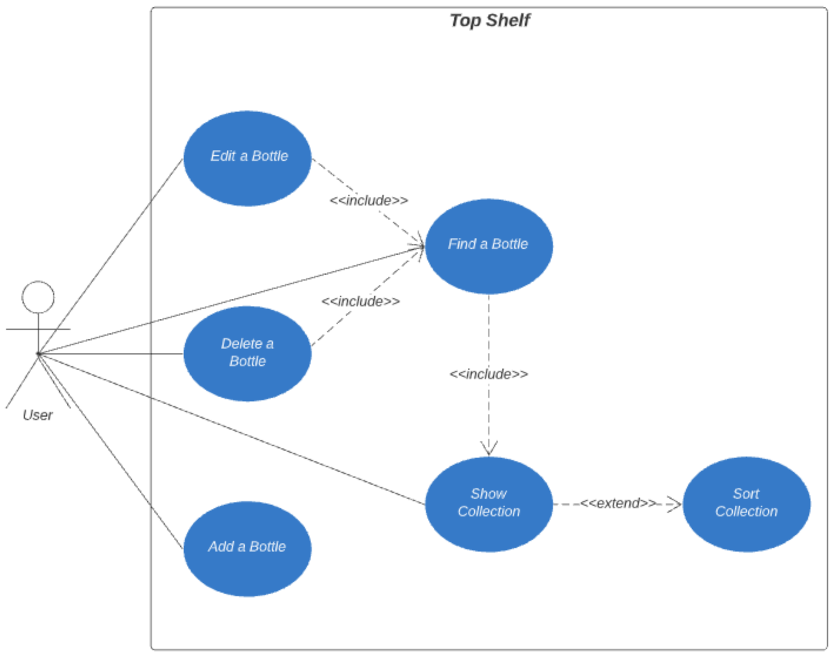
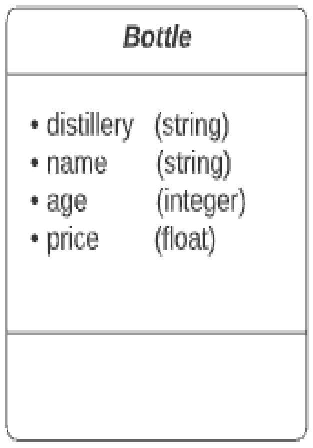


Figure 1: UML Use Case Diagram for Top Shelf Application

A UML use case diagram which provides an overview of the expected system behaviour of the Top Shelf application from a user's perspective is shown in Figure 1. As illustrated, both "Edit a Bottle" and "Remove a Bottle" make use of the "Find a Bottle" functionality to identify the bottle the user wishes to edit/remove from their collection. "Show Collection" displays the entire collection in a table, which is sorted by clicking the header of any of the four columns: distillery, name, age or price.

2 Data storage

The user's collection is stored in a csv file while the application is not running. Upon opening "Top Shelf", the contents of the csv file are fetched from the hard drive of the host machine. Each line of the csv contains the details of a bottle: its distillery, name, age, and price. The application reads each line and stores the data in a Bottle object, the UML diagram for which is included in figure 2.



distillery - the name of the distillery

name - the name of the individual bottling

age - the age statement of the bottle, which can be a whole number of years (e.g. 10).

price - the price of the bottle in pounds sterling stored as a float in the format ##.##

The Bottle objects are stored in a list data structure while the application is running. The list data structure was chosen as it is part of the standard Python library and includes ready-made functions for list manipulation, adding, removing and sorting items.

3 Functionality

“Top Shelf” allows the user to manipulate their collection by adding new bottles (Section 4.1), removing bottles (Section 4.2), editing existing bottles (Section 4.3), and searching their collection for particular bottles (Section 4.4). This section contains a detailed description of the steps the user must take in order to achieve each action. Each action is accompanied by a flowchart illustrating the user journey and a series of wireframes showing the design of the user interface.

3.1 Insertion

The user can add a new bottle to their collection. The flowchart in Figure 3 details how this action is executed.

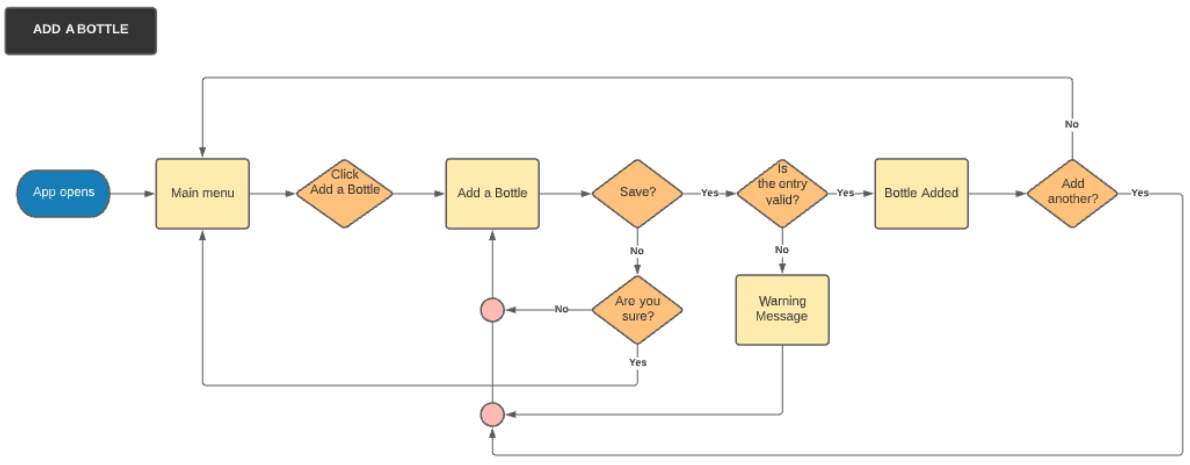
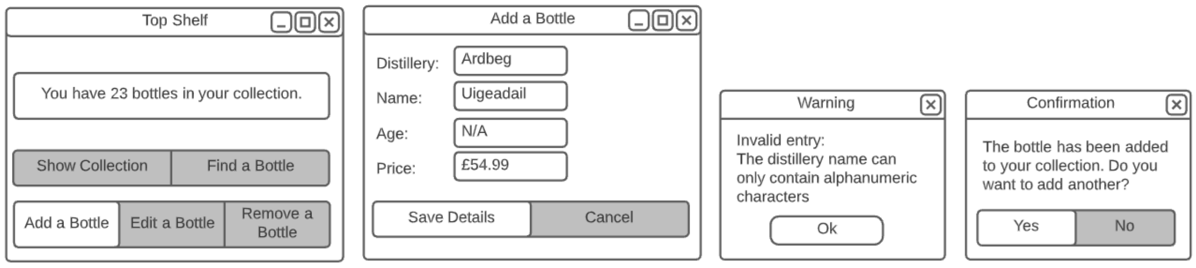


Figure 3: Flowchart depicting the Add a Bottle process

From the homepage, the user clicks the **“Add a Bottle”** button from the main menu. This brings them to the **“Add a Bottle”** page. After entering the details, the user clicks **“Save Details”**. The software checks the validity of the entry, ensuring that the distillery is a string containing only letters, the name has not been left blank, the age is an integer or “N/A”, and the price is an integer or a number with two decimal places. If any of these conditions are not met, an **“Invalid Entry”** warning message is displayed and the user is prompted to re-enter the details in the correct format. If the details entered are valid, the software adds the bottle to the list containing the collection and the display prompts the user that the entry was successfully recorded. The user is asked whether they want to add another bottle. Clicking **“Yes”** repeats the process from the empty **“Add a Bottle”** page, while **“No”** returns the user to the homepage and main menu. Wireframe designs for the pages are included in Figure 4.



You have 23 bottles in your collection.

Distillery: Ardbeg
Name: Uigeadail
Age: N/A
Price: £54.99

Invalid entry:
The distillery name can only contain alphanumeric characters

The bottle has been added to your collection. Do you want to add another?

Figure 4: Wireframes from screens in the Add a Bottle process

3.2 Deletion

The user can remove a bottle from their collection, as illustrated in the flowchart in Figure 5.

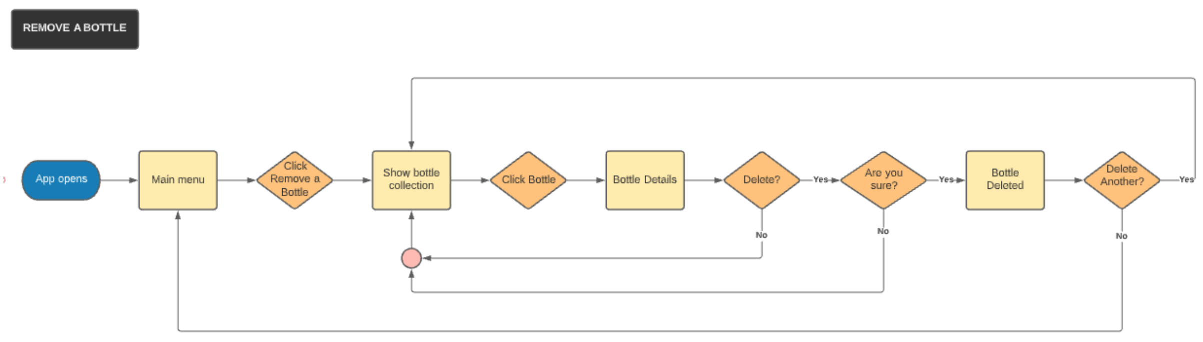
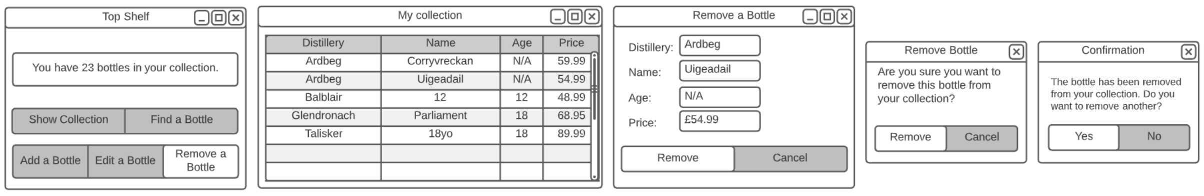


Figure 5: Flowchart depicting Remove a Bottle process

From the homepage, the user clicks the **“Remove a Bottle”** button from the main menu. This brings them to the **“Show Collection”** page. The user can sort the collection by any of the four attributes by clicking on the appropriate column header. The display prompts the user to double-click a bottle to delete it.

When a bottle in the table is double-clicked, the **“Bottle Details”** page is displayed. To remove the bottle from the collection, the user clicks **“Remove”**, after which a message is displayed asking them to confirm that they wish to permanently remove the bottle from their collection. Clicking **“Cancel”** interrupts the deletion and returns to the main menu on the homepage. After deleting the bottle, the user is asked whether they want to remove another. Clicking **“Yes”** repeats the process from the empty **“Show Collection”** page, while **“No”** returns the user to the homepage and main menu. Wireframe designs of the screens encountered in this process are included in Figure 6.



You have 23 bottles in your collection.

Distillery	Name	Age	Price
Ardbeg	Corryvreckan	N/A	59.99
Ardbeg	Uigeadail	N/A	54.99
Balblair	12	12	48.99
Glendronach	Parliament	18	68.95
Talisker	18yo	18	89.99

Distillery:	Ardbeg
Name:	Uigeadail
Age:	N/A
Price:	£54.99

Are you sure you want to remove this bottle from your collection?

The bottle has been removed from your collection. Do you want to remove another?

Figure 6: Wireframe designs of the screens encountered in the Remove a Bottle process

3.3 Manipulation

The user can edit any of the bottles in their collection, the steps of which are illustrated in the flowchart in Figure 9.

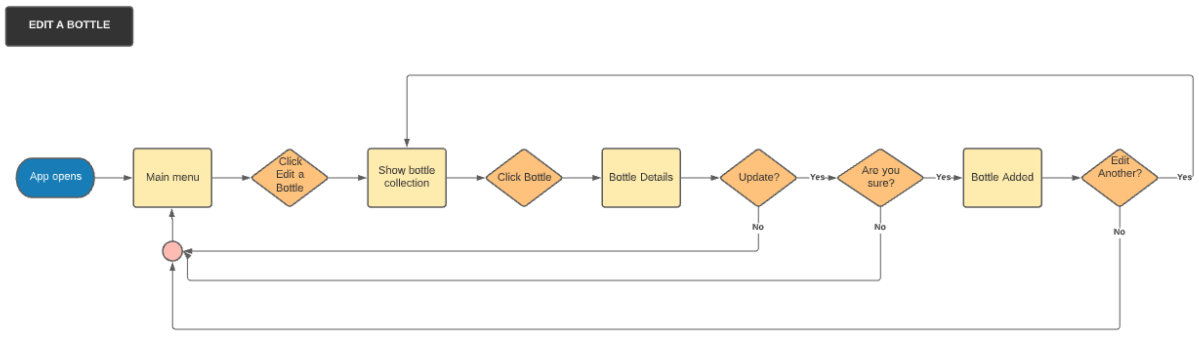
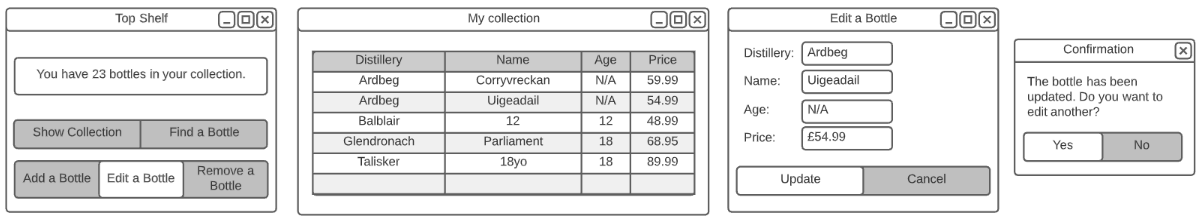


Figure 7: Flowchart depicting the Edit a Bottle process

From the homepage, the user clicks the **“Edit a Bottle”** button from the main menu. This brings them to the **“Show Collection”** page. The user can sort the collection by any of the four attributes by clicking on the appropriate column header. The display prompts the user to double-click a bottle to edit it.

When a bottle in the table is double-clicked, the **“Bottle Details”** page is displayed. To edit the details, the user enters the new information in the appropriate fields. To save the new information, the user clicks **“Update”**. After confirming their action a message notifies the user that the details were updated successfully. Clicking

“Cancel” allows the original entry to persist and returns to the main menu on the homepage. After deleting the bottle, the user is asked whether they want to remove another. Clicking **“Yes”** repeats the process from the empty **“Show Collection”** page, while **“No”** returns the user to the homepage and main menu. Wireframe designs of the screens encountered in this process are included in Figure 8.



You have 23 bottles in your collection.

Distillery	Name	Age	Price
Ardbeg	Corryvreckan	N/A	59.99
Ardbeg	Uigeadail	N/A	54.99
Balblair	12	12	48.99
Glendronach	Parliament	18	68.95
Talisker	18yo	18	89.99

Distillery:	Ardbeg
Name:	Uigeadail
Age:	N/A
Price:	£54.99

The bottle has been updated. Do you want to edit another?

Figure 8: Wireframe designs for screens encountered in the Edit a Bottle process

3.4 Searching

The user can search their collection for a specific bottle; the process for doing this is illustrated in the flowchart in Figure 7.

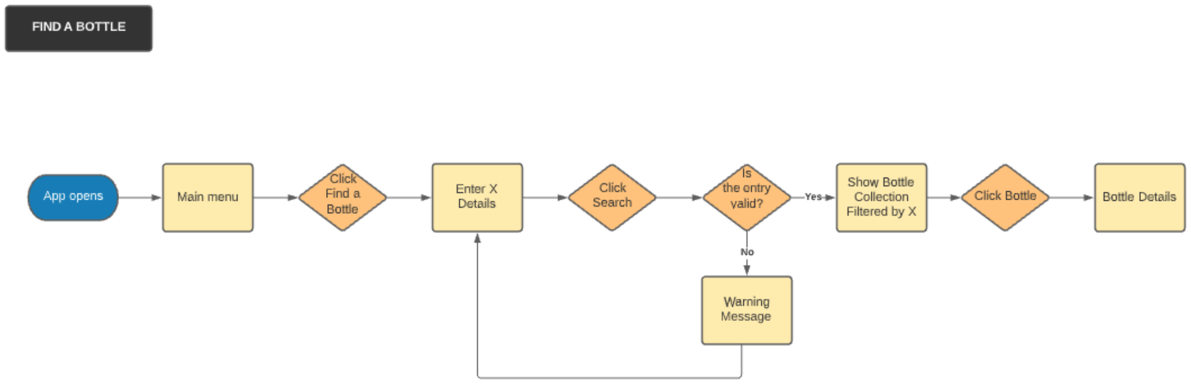
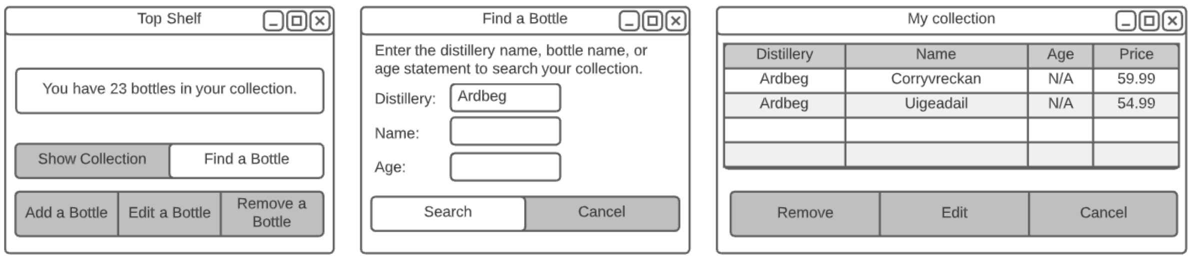


Figure 9: Flowchart depicting the Find a Bottle process

From the homepage, the user clicks the **“Find a Bottle”** button from the main menu. This brings them to the **“Bottle Details”** page. The user can enter details in any of the four attributes: distillery, name, age, price, and click **“Find”**. The details they have entered are verified to make sure that they are valid. Then, they are redirected to the **“Show Collection”** page, which has a table containing details of all bottles in the collection that match the terms of the search. The table can be sorted by clicking on the appropriate column header.

When a bottle in the table is double-clicked, the **“Bottle Details”** page is displayed with three action buttons, **“Edit”**, **“Remove”**, **“Cancel”**. Clicking either of the first two will lead to one of the actions detailed above, while clicking **“Cancel”** will return the user to the homepage and main menu. Wireframe designs of each of the screens visited in this process are shown in Figure 8.



You have 23 bottles in your collection.

Enter the distillery name, bottle name, or age statement to search your collection.

Distillery: Ardbeg

Name:

Age:

Distillery	Name	Age	Price
Ardbeg	Corryvreckan	N/A	59.99
Ardbeg	Uigeadail	N/A	54.99

Figure 10: Wireframe designs for screens encountered in the Find a Bottle process

4 Test Plan

Two testing strategies will be implemented on the “Top Shelf” application: unit tests and functional tests. Unit tests will be automated and run on the Bottle object to ensure that it stores only the intended data types and that other non-valid data will raise an appropriate error handling message.

4.1 Unit Tests

The tables in Figures 11-14 contain the individual unit tests for each attribute of the Bottle object. Each test will attempt to instantiate a Bottle object from the Bottle class. The first column shows the individual test case, the second column contains the input, and the third column contains the expected output.

4.1.1 Testing the Distillery Attribute

Figure 11 contains the four test cases being run against the distillery name. The entry for a distillery should be a string beginning with a capital letter, containing only letters from the alphabet, and should never be left blank.

Test Case Name	Input	Expected Result
test_distillery_Ardbeg_expect_Ardbeg	“Ardbeg”	“Ardbeg”
test_distillery_lowercase_expect_Titlecase	“ardbeg”	“Ardbeg”
test_distillery_empty_string_expect_inputerror	“”	Inputerror - “distillery
test_distillery_including_numbers_expect_inputerror	“Ardbeg10”	Inputerror - “distillery”

Figure 11: Unit tests for the distillery attribute of the Bottle object

4.1.2 Testing the Name Attribute

The entry for a bottling name should be a string containing any combination of alphanumeric characters which should never be left blank. The table in Figure 12 contains the four test cases being run against the bottling name.

Test Case Name	Input	Expected Result
test_name_Uigeadail_expect_Uigeadail	“Uigeadail”	“Uigeadail”
test_name_lowercase_expect_Titlecase	“uigeadail”	“Uigeadail”
test_name_with_integer_expect_string	10	“10”
test_name_empty_string_expect_inputerror	“”	Inputerror - “name”

Figure 12: Unit tests for the name attribute of the Bottle object

4.1.3 Testing the Age Attribute

Figure 13 contains five test cases being run against the age attribute. Valid entries are integers representing the age of the bottling in whole years, or N/A which represents no age statement and will be stored as 0 in the Bottle object. This will ensure all bottles have an integer for the age making it easy to sort the collection by age if required. Should the age entry box be left blank when creating a new bottle, the bottle will be given a default value of “N/A”.

Test Case Name	Input	Expected Result
test_age_NA_expect_0	“N/A”	0
test_age_15_expect_15	15	15
test_age_15.5_expect_inputerror	15.5	Inputerror - “age”
test_age_with_string_expect_inputerror	“Ten”	Inputerror - “age”
test_age_with_empty_string_expect_0	“”	0

Figure 13: Unit tests for the age attribute of the Bottle object

4.1.4 Testing the Price Attribute

The price of a bottle should be stored as a float - a number with two decimal places. Figure 14 contains five test cases being run against the price attribute. Entries other than an integer or a floating point number would be met with an Inputerror.

Test Case Name	Input	Expected Result
test_price_with_55point95_expect_55point95	55.95	55.95
test_price_with_integer_expect_two_dps_added	55	55.00
test_price_with_three_dps_expect_two_dps_added	55.555	55.56
test_price_with_string_expect_inputerror	“Expensive”	Inputerror - “price”
test_price_with_empty_string_expect_inputerror	“”	Inputerror - “price”

Figure 14: Unit tests for the price attribute of the Bottle object

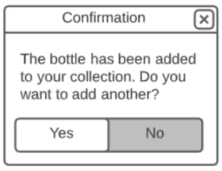
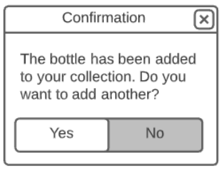
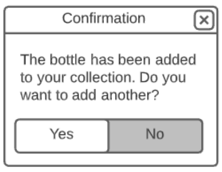
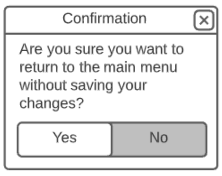
4.2 Functional Tests

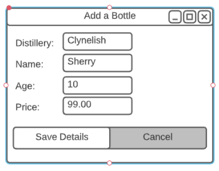
Functional tests have two main purposes, to check the basic usability of an application, i.e. does it do what it’s supposed to do?, and to check the error handling of the application, i.e. when things go wrong, does the software deal with the errors appropriately? These tests can be manual and/or automated. The flowcharts demonstrating the functionality of the application are very useful when designing functional tests as they describe the “happy path”, what the application should do provided everything goes as expected.

4.2.1 Basic Usability Tests

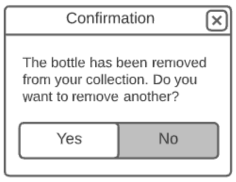
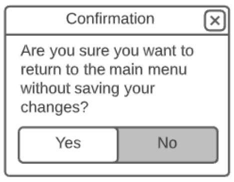
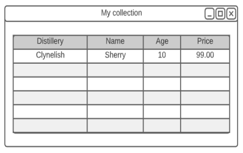
This section lists the basic usability tests which check each of the four main functions of the app: insertion tests are listed in Section 5.2.1, deletion tests in Section 5.2.2, data manipulation tests in Section 5.2.3, and sorting tests in Section 5.2.4. The tables included in each section have five columns. Column 1 contains a short summary of what is being tested, Column 2 shows the initial dataset contained in the list before the test begins, Column 3 lists the steps to be taken by the tester, Column 4 shows the expected screen after the tester's input, and Column 5 shows the expected resulting dataset.

4.2.1.1 Insertion - basic usability tests

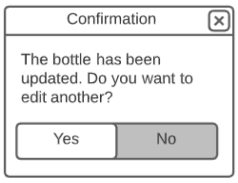
Test Name	Initial dataset	User actions	Resulting Screen	Expected Dataset
Enter a bottle with valid attributes - integers for age and price	[empty list]	● Open Application ● Click **"Add a Bottle"** ● Enter the following values: Distillery: Clynelish Name: Sherry Age: 10 Price: 99 ● Click **"Save Details"**	 The bottle has been added to your collection. Do you want to add another?	[["Clynelish", "Sherry", 10, 99.00]]
Enter a bottle with valid attributes - N/A for age and float for price	[empty list]	● Open Application ● Click **"Add a Bottle"** ● Enter the following values: Distillery: Clynelish Name: Sherry Age: N/A Price: 99.00 ● Click **"Save Details"**	 The bottle has been added to your collection. Do you want to add another?	[["Clynelish", "Sherry", 0, 99.00]]
Enter a bottle with valid attributes - [blank entry] for age and float for price	[empty list]	● Open Application ● Click **"Add a Bottle"** ● Enter the following values: Distillery: Clynelish Name: Sherry Age: Price: 99.00 ● Click **"Save Details"**	 The bottle has been added to your collection. Do you want to add another?	[["Clynelish", "Sherry", 0, 99.00]]
Enter bottle details but click "Cancel"	[empty list]	● Open Application ● Click **"Add a Bottle"** ● Enter the following values: Distillery: Clynelish Name: Sherry Age: N/A Price: 99.00 ● Click **"Cancel"**	 Are you sure you want to return to the main menu without saving your changes?	[empty list]

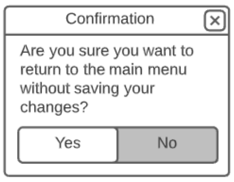
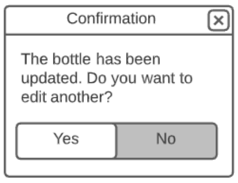
Enter bottle details, press Cancel, but reject confirmation	[empty list]	● Open Application ● Click **"Add a Bottle"** ● Enter the following values: Distillery: Clynelish Name: Sherry Age: N/A Price: 99.00 ● Click **"Cancel"** ● Click **"No"**	 Add a Bottle Distillery: Clynelish Name: Sherry Age: 10 Price: 99.00 Save Details Cancel	[empty list]

4.2.1.2 Deletion - basic usability tests

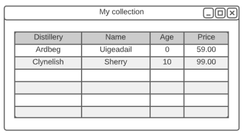
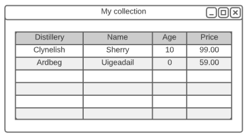
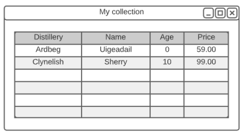
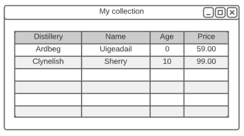
Test Name	Initial dataset	User actions	Resulting Screen	Expected Dataset
Remove bottle from collection	[["Clynelish", "Sherry", 0, 99.00]]	● Open Application ● Click **"Remove a Bottle"** ● Double click the row in the table with the following values: Distillery: Clynelish Name: Sherry Age: N/A Price: 99.00 ● Click **"Remove"**	Confirmation The bottle has been removed from your collection. Do you want to remove another? Yes No	[empty list]
Remove bottle from collection, but click "Cancel"	[["Clynelish", "Sherry", 0, 99.00]]	● Open Application ● Click **"Remove a Bottle"** ● Double click the row in the table with the following values: Distillery: Clynelish Name: Sherry Age: N/A Price: 99.00 ● Click **"Cancel"**	Confirmation Are you sure you want to return to the main menu without saving your changes? Yes No	[["Clynelish", "Sherry", 0, 99.00]]
Remove bottle from collection, but don't double-click bottle	[["Clynelish", "Sherry", 0, 99.00]]	● Open Application ● Click **"Remove a Bottle"** ● Single click the row in the table with the following values: Distillery: Clynelish Name: Sherry Age: N/A Price: 99.00	My collection Distillery | Name | Age | Price Clynelish | Sherry | 10 | 99.00	[["Clynelish", "Sherry", 0, 99.00]]

4.2.1.3 Manipulation - basic usability tests

Test Name	Initial dataset	User actions	Resulting Screen	Expected Dataset
Edit bottle in collection	[["Clynelish", "Sherry", 0, 99.00]]	● Open Application ● Click **"Edit a Bottle"** ● Double click the row in the table with the following values: Distillery: Clynelish Name: Sherry Age: N/A Price: 99.00 ● Enter 14 in the name entry box. ● Click **"Update"**	Confirmation The bottle has been updated. Do you want to edit another? Yes No	[["Clynelish", "14", 0, 99.00]]

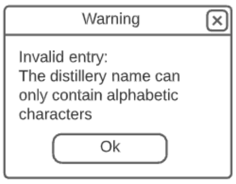
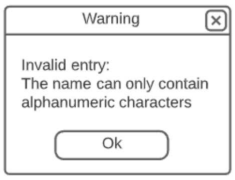
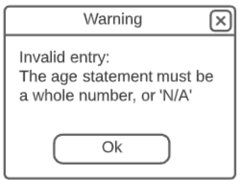
Edit bottle in collection, but click “Cancel”	[[“Clynelish”, “Sherry”, 0, 99.00]]	● Open Application ● Click **“Edit a Bottle”** ● Double click the row in the table with the following values: Distillery: Clynelish Name: Sherry Age: N/A Price: 99.00 ● Enter 14 in the name entry box. ● Click **“Cancel”**	 Confirmation Are you sure you want to return to the main menu without saving your changes? 	[[“Clynelish”, “Sherry”, 0, 99.00]]
Edit bottle in collection without changing any values	[[“Clynelish”, “Sherry”, 0, 99.00]]	● Open Application ● Click **“Edit a Bottle”** ● Double click the row in the table with the following values: Distillery: Clynelish Name: Sherry Age: N/A Price: 99.00 ● Do not change any of the entries ● Click **“Update”**	 Confirmation The bottle has been updated. Do you want to edit another? 	[[“Clynelish”, “Sherry”, 0, 99.00]]

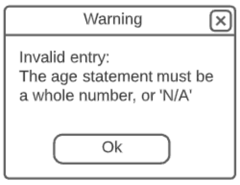
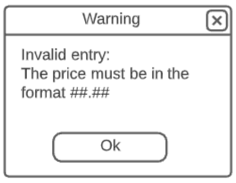
4.2.1.4 Sorting - basic usability tests

Test Name	Initial dataset	User actions	Resulting Screen	Expected Dataset
Sort collection by distillery name	[[“Clynelish”, “Sherry”, 10, 99.00], [“Ardbeg”, “Uigeadail”, 0, 59.00]]	● Open Application ● Click **“Show Collection”** ● Double click the header of the distillery column	 My collection Distillery | Name | Age | Price Ardbeg | Uigeadail | 0 | 59.00 Clynelish | Sherry | 10 | 99.00	[[“Ardbeg”, “Uigeadail”, 0, 57.00], [“Clynelish”, “Sherry”, 10, 99.00]]
Sort collection by name	[[“Clynelish”, “Sherry”, 10, 99.00], [“Ardbeg”, “Uigeadail”, 0, 59.00]]	● Open Application ● Click **“Show Collection”** ● Double click the header of the distillery column	 My collection Distillery | Name | Age | Price Clynelish | Sherry | 10 | 99.00 Ardbeg | Uigeadail | 0 | 59.00	[“Clynelish”, “Sherry”, 10, 99.00], [“Ardbeg”, “Uigeadail”, 0, 57.00]]
Sort collection by age	[[“Clynelish”, “Sherry”, 10, 99.00], [“Ardbeg”, “Uigeadail”, 0, 59.00]]	● Open Application ● Click **“Show Collection”** ● Double click the header of the distillery column	 My collection Distillery | Name | Age | Price Ardbeg | Uigeadail | 0 | 59.00 Clynelish | Sherry | 10 | 99.00	[[“Ardbeg”, “Uigeadail”, 0, 57.00], [“Clynelish”, “Sherry”, 10, 99.00]]
Sort collection by price	[[“Clynelish”, “Sherry”, 10, 99.00], [“Ardbeg”, “Uigeadail”, 0, 59.00]]	● Open Application ● Click **“Show Collection”** ● Double click the header of the distillery column	 My collection Distillery | Name | Age | Price Ardbeg | Uigeadail | 0 | 59.00 Clynelish | Sherry | 10 | 99.00	[[“Ardbeg”, “Uigeadail”, 0, 57.00], [“Clynelish”, “Sherry”, 10, 99.00]]

4.2.1 Error Handling Tests

This section lists the error handling functional tests which check how the application reacts when the user enters invalid data. It is important that the application displays appropriate error messages when the user enters corrupt/invalid data, and also important that accurate prompts are given on how to correct the entry, e.g. informing the user of the data type that is expected for a certain field. The tables included in each section have five columns. Column 1 contains a short summary of what is being tested, Column 2 shows the initial dataset contained in the list before the test begins, Column 3 lists the steps to be taken by the tester, Column 4 shows the expected screen after the tester's input, and Column 5 shows the expected resulting dataset.

Test Name	Initial dataset	User actions	Resulting Screen	Expected Dataset
Add new bottle with numbers in the distillery name	[empty list]	● Open Application ● Click **"Add a Bottle"** ● Enter the following values: Distillery: Clynelish14 Name: Sherry Age: N/A Price: 99.00 ● Click **"Save Details"**	 Invalid entry: The distillery name can only contain alphabetic characters	[empty list]
Add new bottle with no name	[empty list]	● Open Application ● Click **"Add a Bottle"** ● Enter the following values: Distillery: Clynelish Name: Age: N/A Price: 99.00 ● Click **"Save Details"**	 Invalid entry: The name can only contain alphanumeric characters	[empty list]
Add new bottle with age written in letters	[empty list]	● Open Application ● Click **"Add a Bottle"** ● Enter the following values: Distillery: Clynelish Name: Sherry Age: Ten Price: 99.00 ● Click **"Save Details"**	 Invalid entry: The age statement must be a whole number, or 'N/A'	[empty list]

Add new bottle with age including a decimal point	[empty list]	● Open Application ● Click **“Add a Bottle”** ● Enter the following values: Distillery: Clynelish Name: Sherry Age: 10.00 Price: 99.00 ● Click **“Save Details”**	 Warning Invalid entry: The age statement must be a whole number, or 'N/A' 	[empty list]
Add new bottle with no price	[empty list]	● Open Application ● Click **“Add a Bottle”** ● Enter the following values: Distillery: Clynelish Name: Sherry Age: 10 Price: ● Click **“Save Details”**	 Warning Invalid entry: The price must be in the format ##.## 	[empty list]

5 Conclusion

This application has been designed for whisky collectors. The functionality is very basic but includes the main functions needed to manage a list of data - insertion, deletion, manipulation, and searching. The simple, user-friendly GUI ensures that the application will be useful to a wider range of users than if it only had a command line interface. The comprehensive testing plan should ensure that the application works as expected, and should not suffer from unexpected, unhandled errors.

6 README

6.1 Overview

The TopShelf application allows a user to record, track, and update their whisky collection, allowing users to record four attributes for each bottle in their collection, distillery name, bottling name, age, and price. Standard functionality for data storage and manipulation is included: insertion, deletion, manipulation (editing and sorting), and searching. The application has been designed using object oriented principles and uses the Tkinter library for GUI implementation. A **bottles.csv** file has been provided with sample data so the application can be used as intended.

6.2 Classes

The **Bottle** class is used for storing each bottle in the collection as an object. The attributes are initiated as certain data types - integers, strings, etc. which makes it possible to ensure the data is valid and functions/methods can be written which treat each bottle as generic examples of the same 'thing'.

The **TopShelfApp** class is the main class for the application. This opens a window containing a Tkinter frame which is used as a container for all the other pages (which are actually frames layered on top of each other). This class contains most of the methods used throughout the program so that they can be accessed from anywhere within the code.

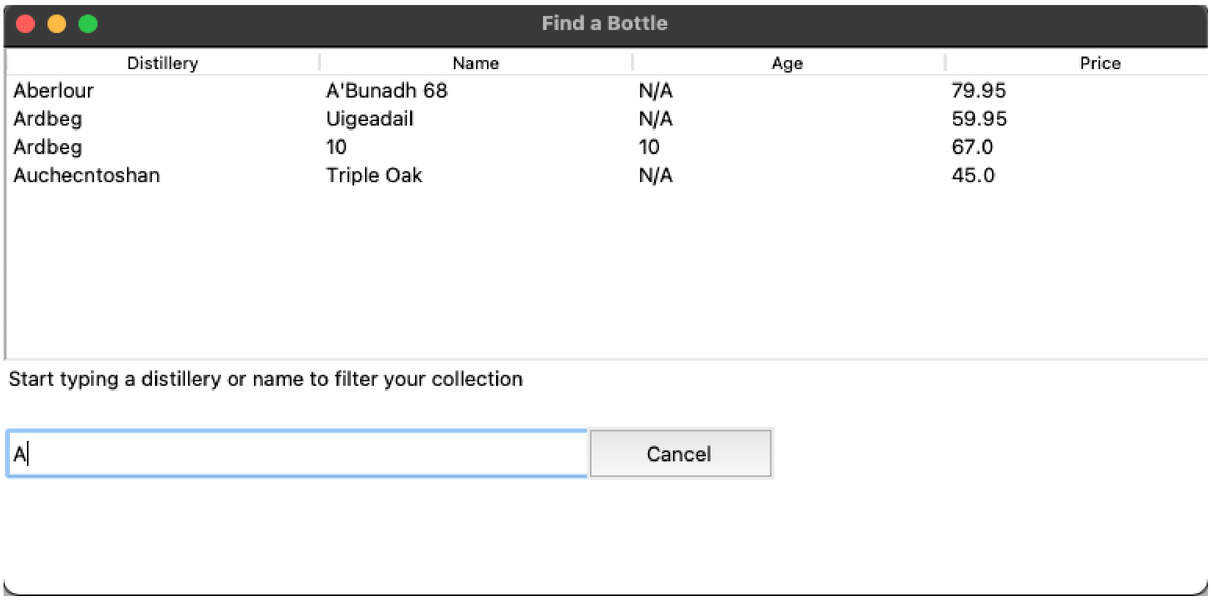
There are six classes which act as separate pages for the application: **Homepage**, **AddBottlePage**, **EditBottlePage**, **FindBottlePage**, **RemoveBottlePage**, **ShowCollectionPage**. Each of these is a Tkinter Frame object which has been initialised inside the TopShelfApp frame and can be 'displayed' by bringing it to the front of the pile calling the **show_frame** method.

There are two classes to enable error handling: the **Error** class which provides a base class for exceptions, and the **InputError** class, a subclass of the Error class which displays a message to the user each time an InputError is raised.

6.3 Comparison with Design

The implementation closely follows the design specified in part 1. The user journey represented by the flow charts has been closely followed, while the user interface is an exact implementation of the wireframes with one exception - the searching functionality. It was originally envisaged that the user would be allowed to enter a distillery, name, and age on a details page and that this would be used to filter the collection, displaying the results in a table with functional buttons below it, as shown in Figure 10 on page 9.

Searching the collection using three different parameters left a lot of room for user error, and if any of the parameters were left empty, it was challenging to write a simple search function which could ignore blank parameters. It was also decided that this is not really a user-friendly way of searching a collection. In the end, it seemed more user-friendly to have a text entry box beneath the table of bottles which allowed the user to enter a distillery name or a bottle name and the table would be filtered on both fields. This was also made responsive by filtering the collection as the user is typing, e.g. you enter ‘a’ and all bottles with a distillery or bottle name beginning with ‘A’ are displayed in the table; you add an ‘r’ and then the table is further filtered to show only entries that begin with ‘Ar’. This seemed like a much more user-friendly way of searching. There is a screenshot of this process below.



Distillery	Name	Age	Price
Aberlour	A'Bunadh 68	N/A	79.95
Ardbeg	Uigeadail	N/A	59.95
Ardbeg	10	10	67.0
Auchecntoshan	Triple Oak	N/A	45.0

Start typing a distillery or name to filter your collection

A

6.4 Implementation

Insertion is achieved using the in-built append method which adds an item to the end of a list.

Deletion is achieved by allowing users to choose a bottle from the collection, the index of which is stored in the **current_bottle** attribute of the Main Application. This index is then used by the in-built **pop(index)** method to remove the bottle from the list.

Editing follows a similar pattern, combining the functionality above to delete the bottle with the **current_bottle** index and append a new bottle with the updated details to the end of the list.

Sorting is controlled by the **sort_bottles_by** method which takes an attribute and uses Python’s **sorted()** method to update the bottle list to a version of the list sorted by the specified attribute.

Searching is implemented in the **search_bottle_for_entry** method which takes a search term and uses Python list iteration and the in-built **startswith** string method to return a sublist of bottles where the **distillery** or **name** attributes start with the search term.

7 Testing Results

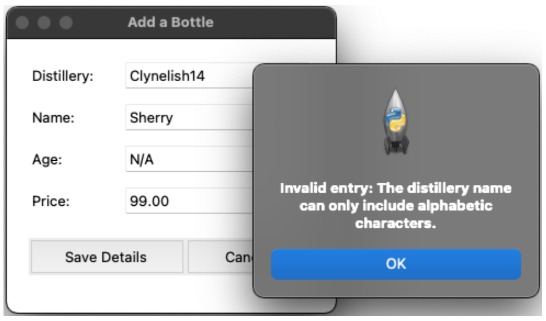
7.1 Unit Tests

The automated unit tests which test the Bottle object stores only the intended data types and that other non-valid data will raise an appropriate error handling messages were implemented exactly as planned in the design (Part 1). There are 17 individual tests which can be found in the **test_bottle.py** file.

7.2 Functional Tests

Functional tests were also completed exactly as specified in Part 1. The test expectations and results are listed in detail in section 4 above. Due to constraints on word limit, it is not possible to detail all of the tests again, but as an example a couple of tests with actual screenshots from the application are included below.

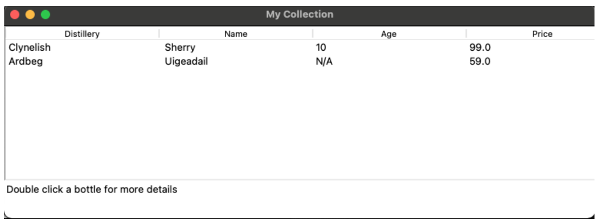
Add a bottle with numbers in the distillery name:



Distillery:	Clynelish14
Name:	Sherry
Age:	N/A
Price:	99.00

Invalid entry: The distillery name can only include alphabetic characters.

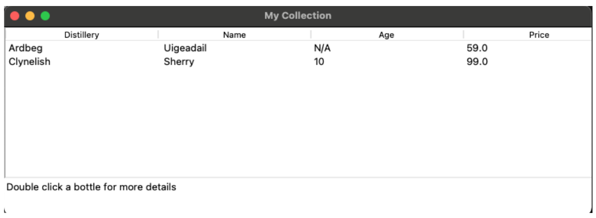
Sort collection by distillery name - double click ‘distillery’ header:



Distillery	Name	Age	Price
Clynelish	Sherry	10	99.0
Ardbeg	Uigeadail	N/A	59.0

Double click a bottle for more details

Before



Distillery	Name	Age	Price
Ardbeg	Uigeadail	N/A	59.0
Clynelish	Sherry	10	99.0

Double click a bottle for more details

After

All other tests that were specified in Part 1 of this assignment were also completed and passed.