**Assignment 2 – Supermarket Checkout – Admin**

The ITWorks Project Python Team Leader responsible for the overall project, has requested an additional set of requirements relating to the Supermarket Self-Service Checkout application previously completed to include an Admin section relating to the same application, based on the following requirements:

**Part A – Modifications to existing Supermarket Checkout System**

After evaluating several other software development methodologies such as Rapid Application Development, Clean House Development and Agile Development, the project lead at IT Works decided on adopting an Agile development strategy here. Several Sprint sessions were planned for implementation in the coming weeks that would result in the finalization of the project requirements specification and the resulting project deliverables.

The project team leader has requested you make the following changes to the existing Supermarket checkout system, and document resulting changes to the existing application code using the Python code comment features to explain the changes that will be based on the following implementation requirements:

1. Use a SQLite database to store the product details, instead of a text file Products.txt, you used in Assignment 1
2. Use a SQLite database to store the checkout transactions instead of a text file Transactions.txt, you used in Assignment 1
3. To complete the above new requirements, you will need to create the following components:

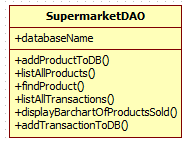
3.1. SQLite database with a **Products** and **Transactions** tables with columns that map to the corresponding class fields for **Product** and **Transaction** found in **Figure 1** and **Figure 2** respectively.

Diagram

Description automatically generated

**Note:** You are allowed to include the description in the name field instead of adding a separate description field (i.e., Milk – 2 Litres)

1. Add at least five dummy products to the **Products** table in the SQLite database for testing purposes.
2. In keeping with an **OO** focus, create a **Transaction** class with fields and method (see Figure 2.0)
3. Create a Python Class called **SupermarketDAO** based on the UML model (Figure 3.0) - that would be used to manage all Product/Transaction data



**(Figure 3.0)**

1. Modify the **save\_transction** (date, Barcode, amount) method in **CheckoutRegister** to create a **Transaction** instance that is saved in the database, by passing it as parameter to the **addTransactionToDB(transaction\_object)**  method of the **SupermarketDAO** object

**Part B –** **Administrative (Admin) System / Implementation Requirements**

The Project leader has directed that the new **Admin** system will be implemented as a prototyped **Console** application and has mandated the following requirements:

1. Create a module to manage the entire application which will serve as the start-up module,
2. When module is started the user must be authenticated using credentials in a **binary file** (not text file). The credentials should store a user’s **Username** and **Password.**
3. On successful authentication of a user, the system should display a menu with the following options:
   1. Add Products to Database
   2. List all Products in Database (Ascending order of Product bar-code)
   3. Find a Product in the Database, based on Product Bar-Code
   4. List All Transactions (Ascending order of date of transaction)
   5. Display a Bar chart of Products sold by quantity
   6. Display an Excel report of all transactions.
   7. Exit

Use the Use Case Report (Appendix A) to help implement the above menu options.

All code must adhere to the ITWorks coding standards and use the Python Comments feature to appropriately comment and document sections of your code for readability . **NOTE:** You must remember to close all database connection after you have run your queries. This will ensure that the connection will be destroyed and valuable memory resources will be recovered

1. The systems analysis /designers have recommended the following specifications for each Use Case to be implemented after evaluating other options against the above criteria
2. **Add Products to Database**

Selecting this option will call a method (see below) using the product details entered from the keyboard by the user, to, a **Product** instance which is passed as an input parameter to the following method of the **Supermarket\_DAO** instance:

**addProductToDB(product\_object)**

**### will save the product\_object details to the database and will return True or False to indicate if the product was inserted or not- Display an appropriate message to indicate the success/or not of the call ###**

1. **List all Products in Database (Ascending order of Product bar-code)**

Selecting this option will call a method (see below) on **SuperMarket\_DAO** instance that will read all products from the database and return a list of all Product objects sorted in ascending order of product Barcode.

It is recommended that you use a separate method as below:

**listAllProducts( )**

**### will get all products from the database and put them in a List, sort the list in ascending order of product id. If no products in database, display appropriate message ###**

1. **Find a Product in the Database, based on Product Bar-Code**

Selecting this option will use the Barcode entered from the keyboard by a user, which is passed as a parameter to the following method of the Supermarket\_DAO instance:

**findProduct(with\_this\_barcode)**

**### Use the Barcode to get the product from the database, create and return the Product instance or None if product is not found. ###**

1. **List All Transactions (Ascending order of date of transaction)**

Selecting this option will call the following method on Supermarket\_DAO instance-that will read all transactions from the database and return a list of all Transaction objects sorted in ascending order of product date.

It is recommended that you use a separate method as below:

**listAllTransactions( )**

**### will get all transactions from the database and put them in a List, sort the list in ascending order of date If no products in database, display appropriate message ###**

1. **Display a Bar chart of Products sold by quantity**

Selecting this option will call the following method on Supermarket\_DAO instance-that will read all transactions from the database into a list and create a Bar chart using the x-axis as the product name and y-axis as the sum of the product quantities sold. You must also use a Dictionary in your logic to implement this method (see details below)

**displayBarchartOfProductsSold( )**

**###You will require two lists of data to implement this method. A list of Product names (x-axis values) and a list of Integers - the sum of each product sale (y-axis values). Use a Dictionary, with the product Barcode as the Key and a product count as the Value. Populate the Dictionary by reading each transaction from the Transactions table in the database and work out the counts.**

**Using each key in the Dictionary, get each product name from the Products table in the database and populate the List of Product names (x-axis values). Use the count values in the Dictionary to populate the List of Integers (y-axis value) ###**

1. **Display an Excel report of all transactions**.

Selecting this option will call the following method on Supermarket\_DAO instance that will get all the transactions from the Transaction table in the database (you can call the **listAllTransactions**( ) method here) and use this data to generate an Excel workbook, to list all transactions according to date order.

**displayExcelReportOfTransactions( ) # Implements the above**

1. **Exit**

Selecting this option will call the following method that will close all running applications

**exit()**

**Part C – Exploring optional Data Structures (Proof of Concept)***.*

ITWorks is keen on investigating the use of various data structures that are supported by the language with the aim of improving the efficiency relating to the searching and sorting of data. In this section you have been tasked to evaluate and weigh-up other options re Data Structure and Sorting Algorithms Efficiencies - and document the results in a Word document titled ‘Exploring Data Structures – Proof of Concept’.

1. **Data Structure Efficiency**

One of the ideas proposed is to read all the data once from the products.txt file into a List of Product objects (You will be given the code to do this) and use this List to populate any three of the following data structures) shown below:

1. List
2. Stack
3. Queue
4. Linked List
5. Binary Trees
6. Hash Tables
7. Hash Maps
8. Graphs
9. Sets

Create a separate Python module titled ‘Data\_Structure\_Efficieny\_ PofC.py, and implement 3 different methods (one for each data structure), using the pseudocode below as a guide. The aim here is to measure the performance (in milliseconds) of each data structure to add the Product objects from the List of Product instances

START CLOCK

FOR EACH PRODCUT IN THE LIST

ADD PRODUCT TO DATA STRUCTURE

END\_CLOCK

PRINT TIME ELAPSED

Run each of the 3 methods and record the execution time in milliseconds for each data structure

Record in your documentation which data structure was the most efficient?

**Documentation**

Document your findings in the ‘Exploring Data Structures – Proof of Concept’ technical document

1. **Sorting Algorithms Efficiency**

The team is also keen on exploring other sorting algorithms to find out the most performant one in terms of performance in milli seconds. From the following sorting algorithm options which one would be the most efficient choice for the Supermarket Admin program by comparing their performance executions times through your research. You are not expected to implement any of the chosen Sorting algorithms. Document your reasons as instructed below.

1. Insertion sort

2. Merge Sort

3. Bubble Sort

4. Selection Sort

**Documentation**

This is a research topic and your expected to explore at least two relevant online resources and document and reference your findings in the ‘Exploring Data Structures – Proof of Concept’ technical document in about 200 words

1. **Searching Algorithms Efficiency**

In this project when we wanted to find a Product, we used a method to query the database using the product Barcode as the primary key value.

It has been suggested that in order to reduce the number of Product search queries – we plan to initially read all the products from the database into an appropriate list structure (as a cache of data) and search for a Product from the list rather than using a database query. The two most popular search options (there are others)

1. Sequential search

2. Binary search

Using Python module titled ‘Data\_Structure\_Efficiency\_ PofC.py - code two separate search functions to read all Product records from the List you created in (2) and use this list to perform a Binary Search (separate function here) and a Sequential Sort (separate function here) and determine which Sorting strategy was quicker as measured in milliseconds.

**Documentation**

Document your findings in the ‘Exploring Data Structures – Proof of Concept’ technical document

**Submission Details**

Create a folder**<ID\_NUM><NAME><Assignment2>** and add the following artifacts:

1. The modified Assignment 1 Python project
2. The Supermarket\_Admin Python project
3. The Part C Python code with implemented functions documenting findings from Part C sections in Word document titled ‘Exploring Data Structures – Proof of Concept’ document

Please zip the above folder as below and submit using the appropriate upload links on the subject Learn page

Submit 1 and 2 as **<ID\_NUM><NAME><Assignment2> - Part A and B.**

Submit 3 as **<ID\_NUM><NAME><Assignment2> - Part C**

**Appendix A – Use Case Report for Part B**

Diagram

Description automatically generated

1. **Add Products to Database**
   1. User selects menu option **(a)**
   2. The user will be prompted to enter the Product attributes (Barcode, Name … etc)
   3. Validate inputs
   4. If all valid, save product to Product Table
   5. If not valid, display an appropriate Error Message and go to step (2)
2. **List all Products in Database (Ascending order of Product bar-code)**
   1. User selects menu option **(b)**
   2. All Products and all their attributes are displayed in ascending order of bar-code.
3. **Find a Product in the Database, based on Product Bar-Code**
   1. User selects menu option **(c)**
   2. User enters the Barcode of the Product to find
   3. Validate Barcode
   4. If Barcode is valid – display the details of the Product
   5. Else display a suitable Error Message and go to step 2
4. **List All Transactions (Ascending order of date of transaction)**
   1. User selects menu option **(d)**
   2. All Transactions and all their attributes are displayed in ascending order of date of Transaction
5. **Display a Bar chart of Products sold by name/quantity.**
   1. User selects menu option **(e)**
   2. The system will display a bar chart with the name of the product on the x axis and the quantity of the product sold on the y axis (the height of the bar)
6. **Display an Excel report of all transactions.**
   1. User selects menu option **(f)**
   2. The systems will create an Excel spreadsheet report and display all checked out transactions by product date of transaction, barcode, name, and quantity, displayed by ascending order of date of transaction.
7. **Exit**
   1. User selects menu option **(g)**
   2. The system will close all open applications