SELF CHECKOUT BILLING SYSTEM

PROJECT REPORT

Submitted by

Cavin Shree Ramesh Kumar [RA2211003010740]

Anu Shree V S [RA2211003010739]

Under the Guidance of

Dr.M.KANDAN

Assistant Professor, Department of Computing Technologies

***In partial satisfaction of the requirements for** the degree of*

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

MAY 2023

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this Project Report titled **"Self-Checkout Billing System"** is the bonafide work done by **Cavin Shree Ramesh Kumar [RA2211003010740], Anu Shree V S [RA2211003010739]** who completed the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

Department of Computing Technologies Kattankulathur 603 203 SRMIST

SIGNATURE
Dr.M.Kandan
OODP – Course Faculty
Assistant Professor
Department of Computing Technologies
SRMIST

M. Pushpalatha

SIGNATURE
Dr.M.Pushpalatha
Professor & Head
Department of Computing Technologies
School of Computing
SRMIST

TABLE OF CONTENTS

S.No	CONTENTS	PAGE NO
1.	Problem Statement	
2.	Modules of Project	
3.	UML Diagrams	
	a. Use case Diagram	
	b. Class Diagram	
	c. Sequence Diagram	
	d. Collaboration Diagram	
	e. State Chart Diagram	
	f. Activity Diagram	
	g. Package Diagram	
	h. Component Diagram	
	i. Deployment Diagram	
4.	Code/Output Screenshots	
5.	Results and Conclusion	
6.	References	

1. PROBLEM STATEMENTS

Traditional billing methods need a lot of space, which could be better utilised for the display of products. Staff wastes time at the checkout counters during idle periods when they could be replenishing shelves or organising inventory, among other things.

1.1.1 Existing System

Current systems use the traditional method of having multiple checkout counters with staff at each counter, checking out customers' orders. This results in long queues which decrease customer satisfaction and store space on busy days. On slow days, due to staff waiting at empty checkout lines, in-store productivity is reduced.

1.2 OBJECTIVE OF THE PROJECT

Self-checkout billing systems not only reduce costs and have benefits for retailers but also save time for the customers. Customers will have a more satisfactory and enjoyable experience. The goal of our project is to enable the customer to have a smooth checkout experience and to save their precious time by cutting down the actual time spent waiting for cashiers and by allowing them to multitask and continue their previous tasks like being on the phone or listening to music, etc, that they were doing while shopping.

2 MODULES OF PROJECT

1. Class Bill:

   1.1. input()

   1.2. output(map<string, int>)

   1.3. loop_input(map<string, int>)

2. Class Database:

   2.1. loop(map<string, int>)

   2.2. authority_check(map<string, int>)

   2.3. add_item(map<string, int>)

   2.4. delete_item(map<string, int>)

2.5. show_item(map<string, int>)

2.6. show_database(map<string, int>)

2.7. edit_item(map<string, int>)

2.1.1 input()

The method adds items to be billed to the map container- items (private data member of class Bill). The user is first prompted to enter the item code and quantity in the following format- <item_code> <quantity>- and this pair of elements in inserted using the predefined method from map header file into items.

2.1.2 output()

This method calculates the total amount to be paid based on the elements inputted to the data member items. It iterates through items (map container, data member of class Bill) and multiplies the second element of each pair (quantity) into the second element of the map container given in the parameters whose first element matches with it's own. This value is then added to the total_price (data member of class Bill) and after the iteration is complete, the final value of total_price is displayed as the total amount to be paid.

2.1.3 loop_input(map<string, int>)

This method allows the user to keep adding elements to the items container. The method first calls the method input(), then gives the user a choice- add items or calculate amount. If the user chooses to add items, the methods calls itself. If the user chooses to calculate the amount, the control goes back to the main program.

2.2.1 loop(map<string, int>)

This method loops through the various database manipulation options till the user is satisfied with the changes. The options of manipulation are displayed to the user. Once the user makes the choice, the corresponding data manipulation function is called and the item database is altered. Once the alteration is done, the method calls itself to continue the loop. If the user chooses to exit, instead of calling itself, the program goes back to its previous state of offering the options- bill order and edit item database.

2.2.2 authority_check(map<string, int>)

This method verifies the user and ensures that only admins access the map container in the parameters. It prompts the user to enter the password and using strcmp from string.h header file, compares the inputted string and the data member password. If the strings

match, the method calls the function loop. If the strings do not match, access is not granted and the program goes back to showing the initial options of bill order and edit item database.

2.2.3 add_item(map<string, int>)

This method adds elements to the map container in the parameters. It takes input in the form of- <item_code> <item_price>- and using insert function from map header file, adds it to the map container in the parameters and returns the altered container.

2.2.4 delete_item(map<string, int>)

This method deletes elements from the map container in the parameters. It takes input in the form of-<item_code> <item_price>- and using erase function from map header file, deletes it from the map container in the parameters and returns the altered container.

2.2.5 show_item(map<string, int>)

This method displays the second element of a particular pair of the map container in the parameters. It takes the first element as the input from the user and using find function from map header to find its corresponding pair and displays it.

2.2.6 show_database(map<string, int>)

This method displays the map container given in the parameters. It iterates through the container given in the parameters and displays each of the pairs.

2.2.7 edit_item(map<string, int>)

This method edits pre-existing elements of the map container given in parameters. It first uses the erase function to delete the user inputted element and inserts the element again with the new value. It then returns the altered container.

3. UML DIAGRAMS

3.1 SOFTWARE REQUIREMENTS/ HARDWARE REQUIREMENTS

3.1.1 Software Requirements

Operating System : windows 11

Frontend Languages : C++

Backend Languages : -

3.1.2 Hardware Requirements

Processor : i7 and 2300 Mhz speed

RAM : 16 GB

Hard Disk : 512 MB

3.2 UML DIAGRAMS

a. Use case Diagram

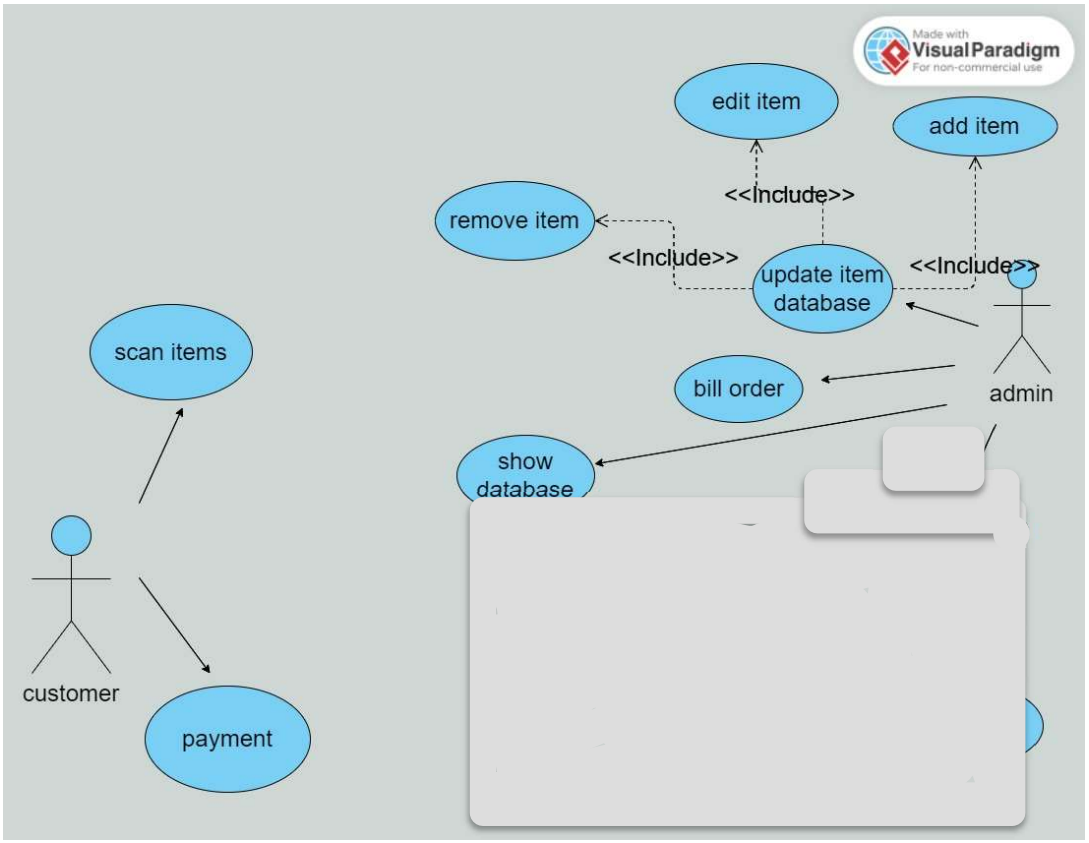

Fig. 3.2.a Use Case Diagram

There are two main entities under the billing system which are the customer entity -The use case of cashier manages scanning items and payment- and the admin entity- the use case of the manager entity takes care of updating items, adding items, editing items, removing items, editing customer, adding customer and removing customer.

b. Class Diagram

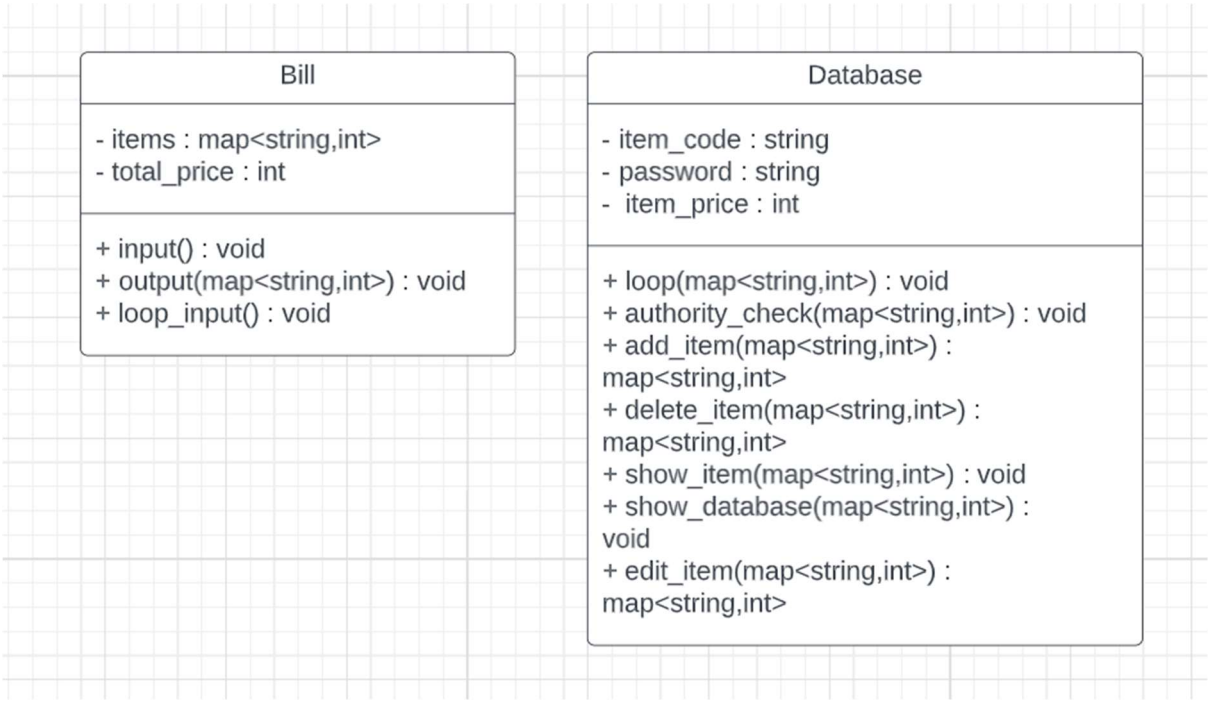

Fig. 3.2.b. Class Diagram

There are two classes- Bill and Database. Bill stores the data members items and total_price privately and has the member functions- input, output, loop_input that access and work on the data members. Database stores the data members item_code, password, item_price privately and has the member functions- loop, authority_check, add_item, delete_item, show_item, show_database, edit_items- that alter the item database. The data member password stores the password required to access the member functions and authority_check uses it to ensure it.

BEHAVIOUR DIAGRAM

Interaction Diagram (Sequence Diagram and Collaboration Diagram)

c. Sequence Diagram

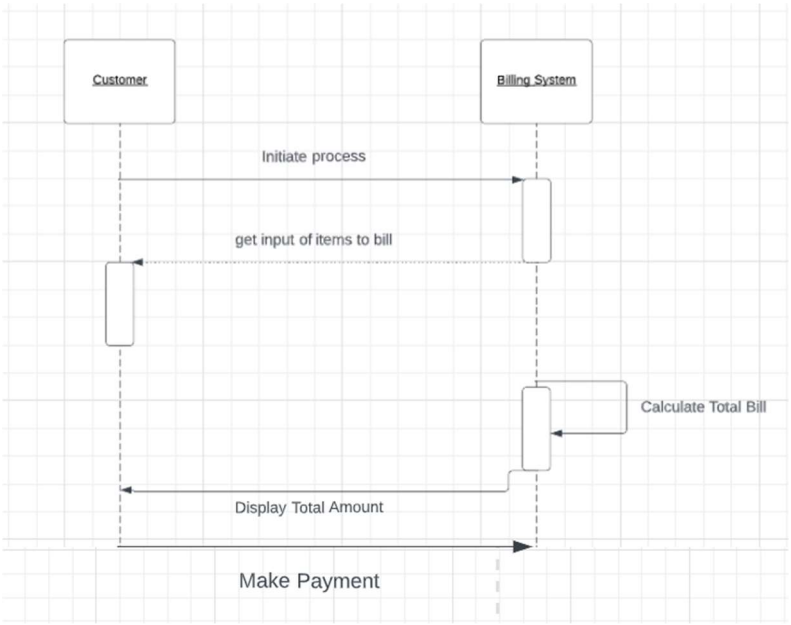

Fig. 3.2.c.1. Sequence Diagram 2

The customer initiates the process by choosing to bill their order. The Billing system gets the items to bill with their quantity from the user. After getting the entire list, the system calculates the bill and displays it. The customer can then make the payment.

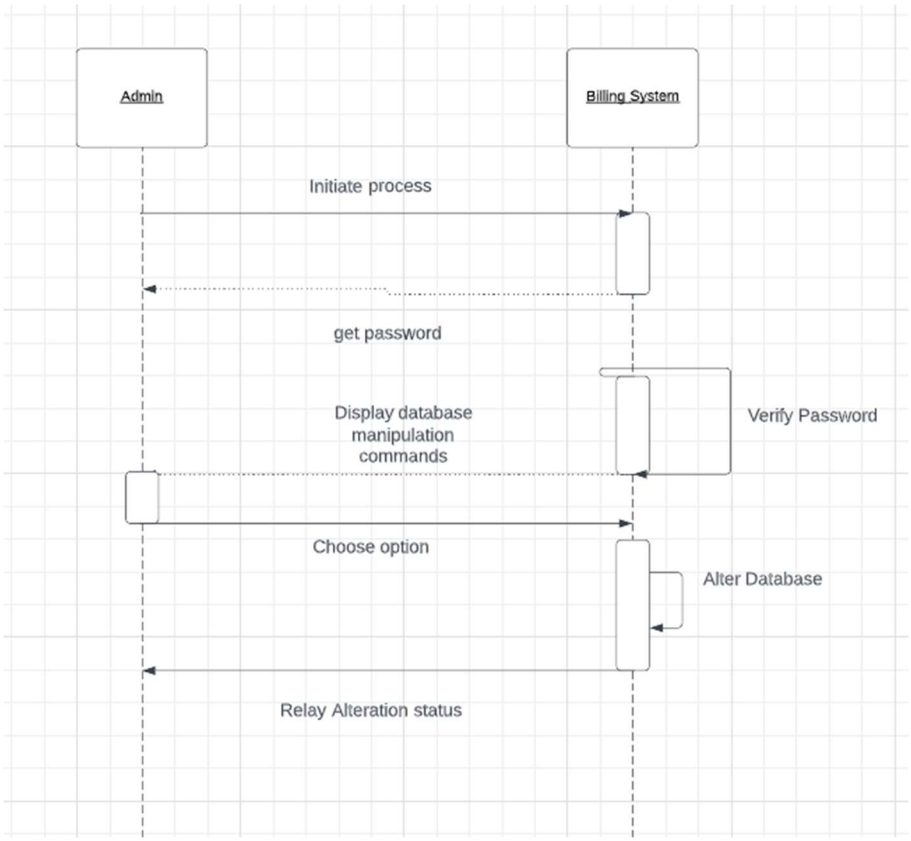

Fig. 3.2.c.2. Sequence Diagram 2

The admin initiates the process by choosing to edit the item database. The billing system then gets the password from the admin. After the verifying the password, the system displays the manipulation commands to the admin. The admin choose from the list. Based on their option chosen, the database is altered and the system relays the alteration status to the admin.

d. Collaboration Diagram

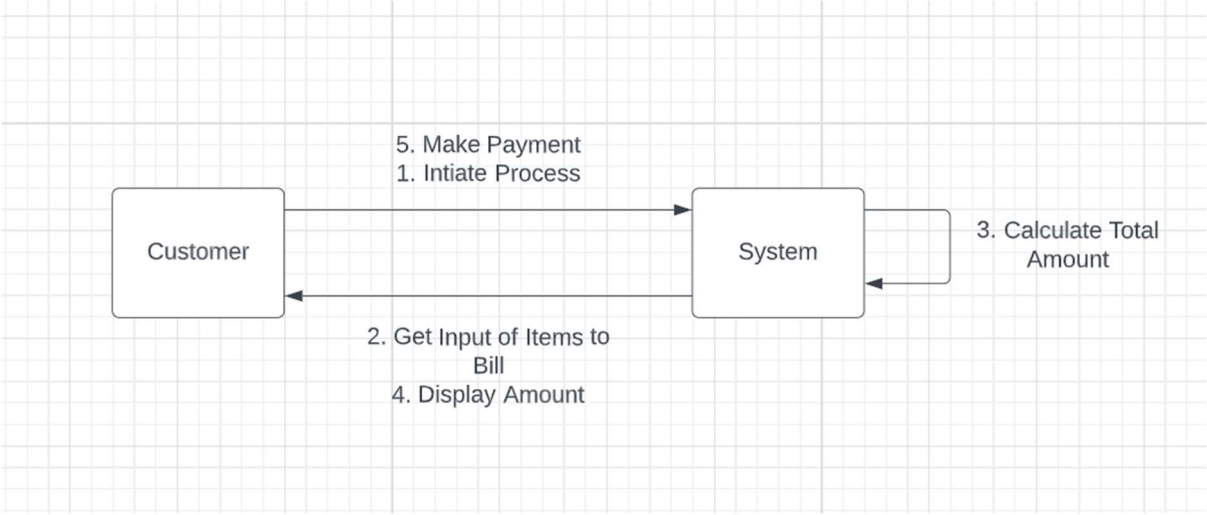

Fig 3.2.d. Collaboration Diagram 1

The customer initiates the process by choosing to bill their order. The Billing system gets the items to bill with their quantity from the user. After getting the entire list, the system calculates the bill and displays it. The customer can then make the payment.

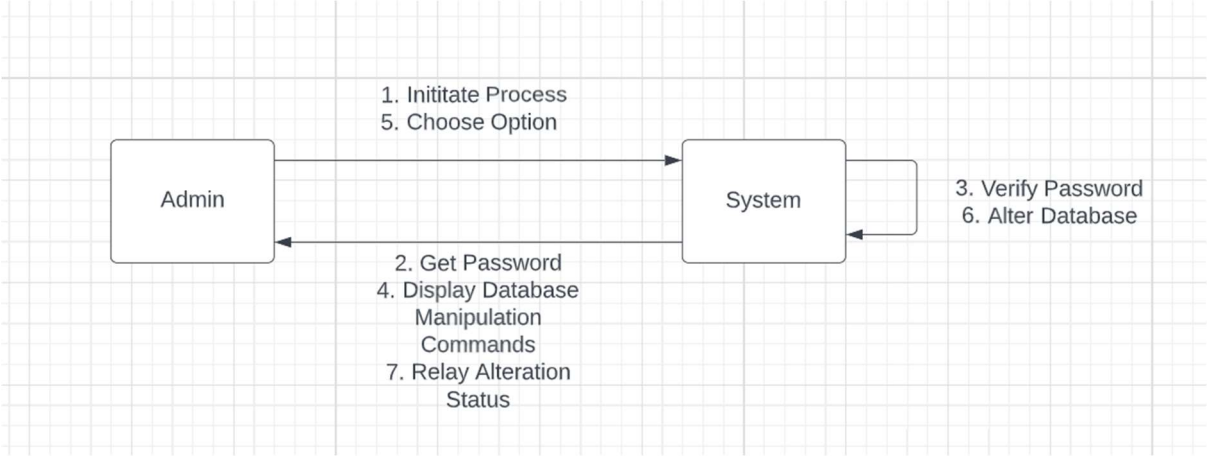

Fig. 3.2.d. Collaboration Diagram 2

The admin initiates the process by choosing to edit the item database. The billing system then gets the password from the admin. After the verifying the password, the system displays the manipulation commands to the admin. The admin choose from the list. Based on their option chosen, the database is altered and the system relays the alteration status to the admin.

e. State Diagram

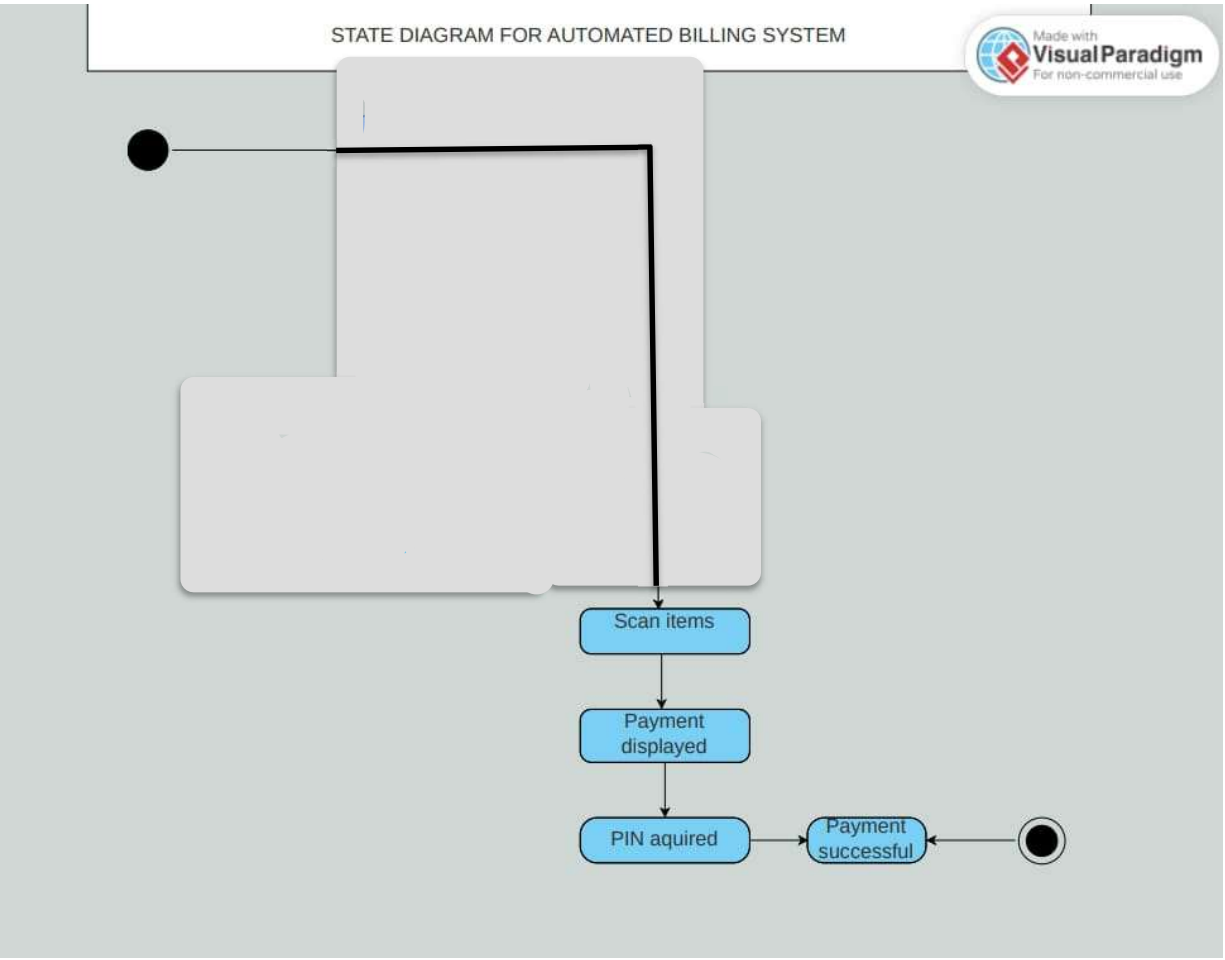

Fig. 3.2.e. State Diagram

State diagram also know as the state machine diagram or the state chart diagram consists of the states, transitions, events and activities. Enter the database, the database is searched, when the names of the customers are not found then , the customer name is added otherwise the processing takes place. Next the items are scanned and the payment is displayed, the PIN is accepted and the payment is successful.

f. Activity Diagram

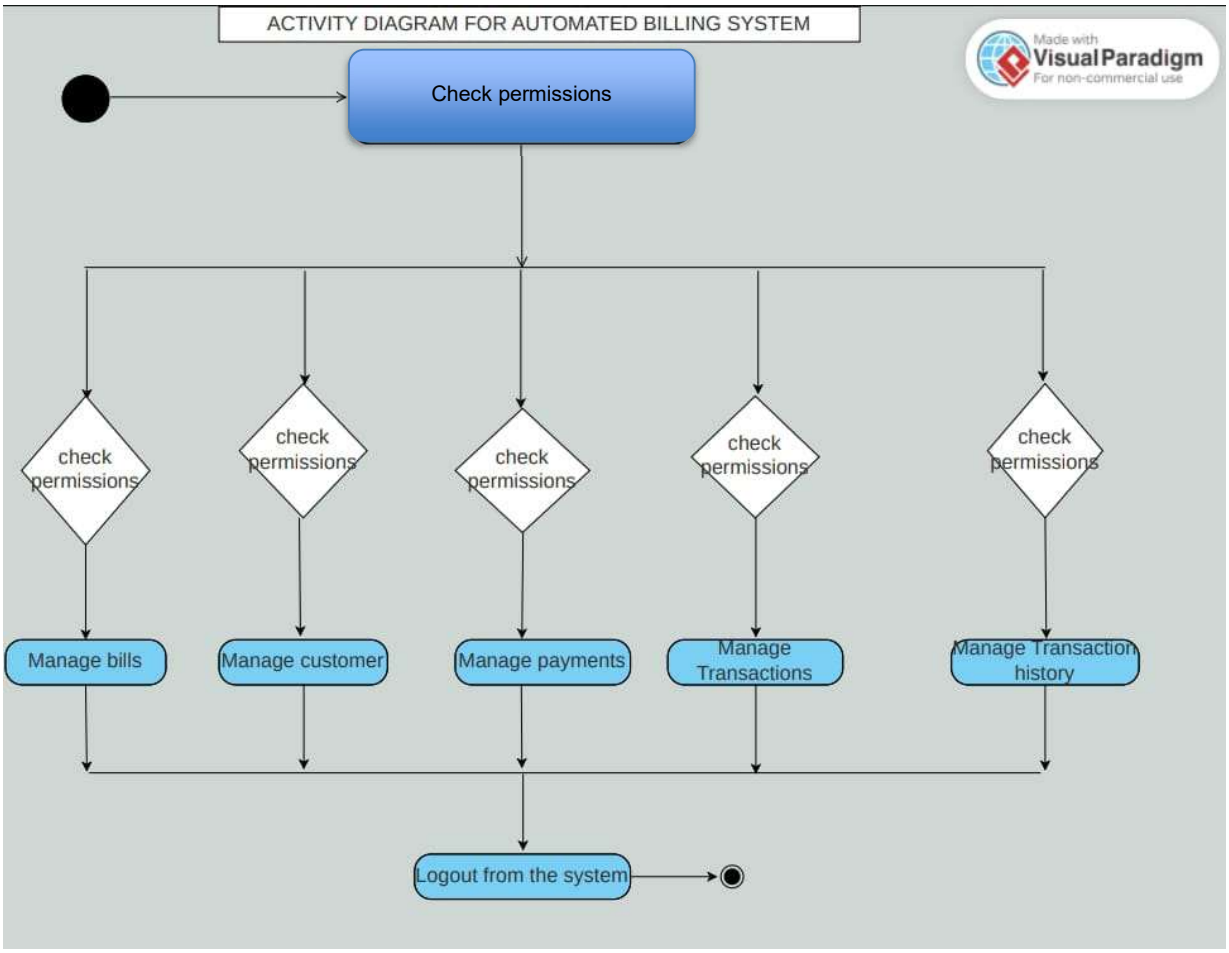

Fig. 3.2.f. Activity Diagram

In the activity diagram we first check if we have permission to access the database. Then from there we need to check if the permissions are granted to the system to perform the next step, if the permissions are granted then the user can manage bills, manage customers, manage payments, manage transactions, manage transaction history. Then after getting the required work done the user logs out of the system.

g. Package Diagram

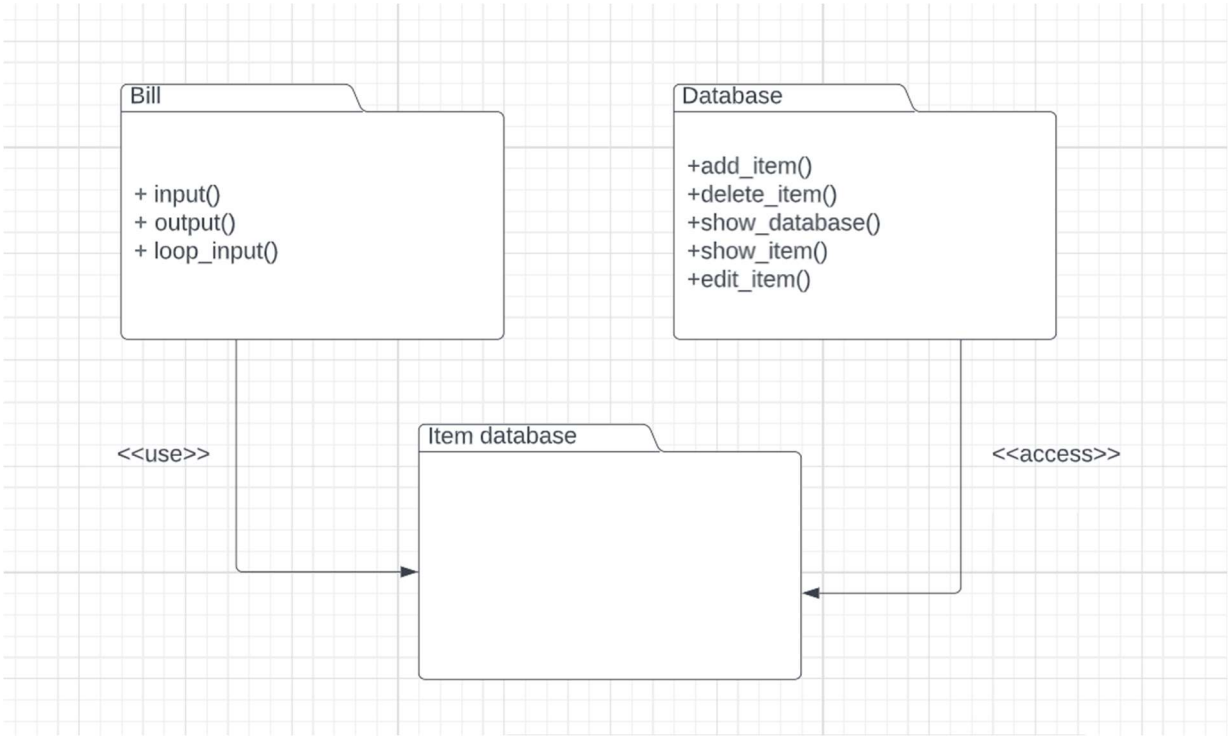

Fig. 3.2..g Package Diagram

The program has two packages- Bill and Database. Bill uses the item database, while Database can access the item database and alter its data.

IMPLEMENTATION DIAGRAM

h. Component Diagram

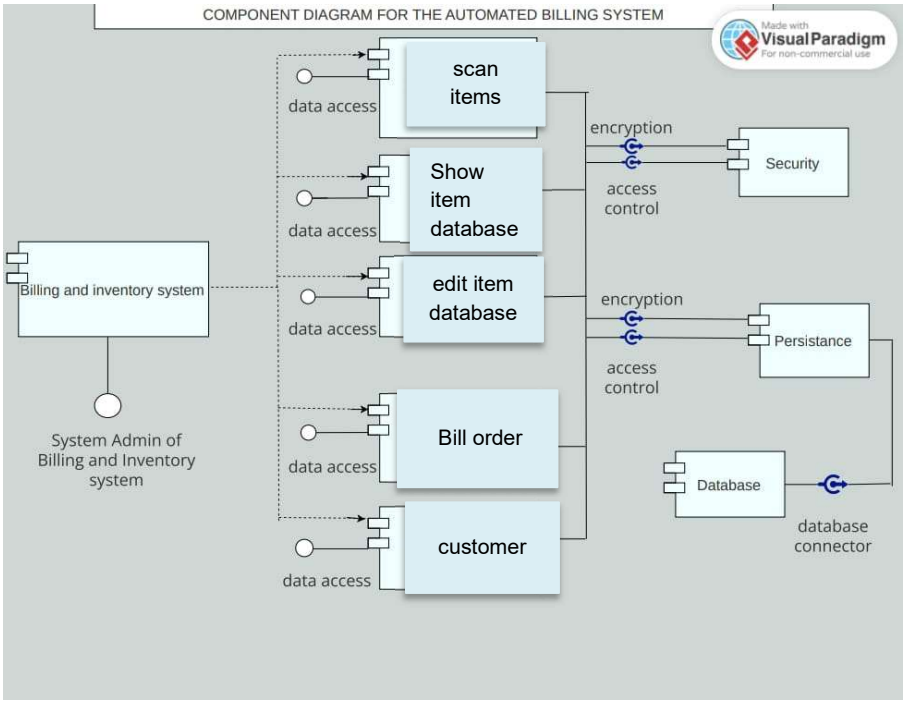

Fig. 3.2.h Component Diagram

This is a component diagram of a self checkout Billing System which shows components, provided and required interfaces, ports and relationships between the customer, payments, items. This diagram is used the describe the organization and wiring of the components of the system.

i. Deployment Diagram

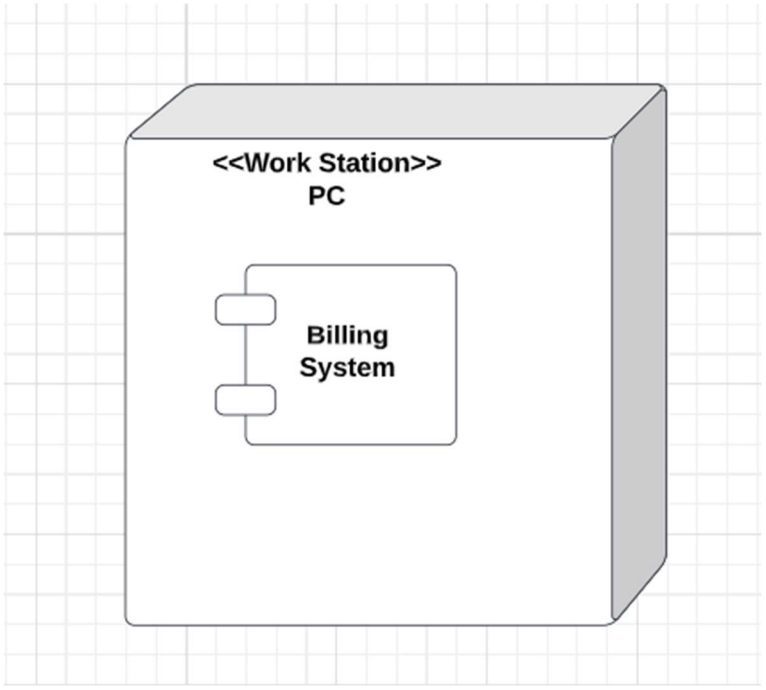

Fig. 3.2.i. Deployment Diagram

4. CODE/OUTPUT SCREENSHOTS

4.1 SOURCE CODE

```
#include <iostream>
#include <map>
#include <string.h>

using namespace std;

class Bill
{
  map < string, int >items;
  int total_price;
public:
  void input ()
  {
    char item_code_input[5];
    int item_quantity_input;
      cout << "enter the item code and quantity :";
      cin >> item_code_input >> item_quantity_input;
      items.insert (pair < string,
                          int >(item_code_input, item_quantity_input));
  }
  void output (map < string, int >database)
  {
    total_price = 0;
    map < string, int >::iterator it;
    for (it = items.begin (); it != items.end (); it++)
      {
           total_price += (it->second) * ((database.find (it->first))->second);
      }
    cout << "\nTotal amount to be paid : " << total_price;
  }
```

```
void loop_input ()
  {
   input ();
   cout << "[1] Add items\n[2] Calculate amount\n";
   int choice;
   cin >> choice;
   if (choice == 1)
     {
          loop_input ();
     }
  }
};

class Database
{
  char item_code[5], password[20] = "agBkW92Sh9*C6$nT";
  int item_price;
public:
  void loop (map < string, int >database);
  void authority_check (map < string, int >database)
  {
   cout << "\n\nEnter password : ";
   char temp_pass[20];
     cin >> temp_pass;
   if (strcmp (temp_pass, password) == 0)
     {
          loop (database);
     }
   else
     {
          cout << "Incorrect Password\nAccess not granted\n\n";
          cout << "\nWelcome User!\n[1] Bill order\n[2] Edit item database\n";
          int choice;
          cin >> choice;
          if (choice == 1)
```

```
{
            Bill b;
            b.loop_input ();
            b.output (database);
           }
         else if (choice == 2)
          {
           authority_check (database);
          }
    }
}
map<string,int> add_item (map < string, int >database)
{
  cout << "\nenter item code and price : ";
  cin >> item_code >> item_price;
  database.insert (pair < string, int >(item_code, item_price));
  cout<<item_code<<" added\n";
  return database;
}
map<string,int> delete_item (map < string, int >database)
{
  cout << "\nenter item code to delete : ";
  cin >> item_code;
  database.erase (item_code);
  cout << item_code << " deleted from database";
  return database;
}
void show_item (map < string, int >database)
{
  cout << "\nenter item code to view : ";
  cin >> item_code;
  cout << "Code : " << item_code << "\nPrice : " << (database.
                                                    find (item_code))->
    second;
}
void show_database (map < string, int >database)
```

```
{
   cout << "Code    Price" << endl;
   map < string, int >::iterator it;
   for (it = database.begin (); it != database.end (); it++)
     {
          cout << it->first << "    " << it->second << endl;
     }
 }
 map<string,int> edit_item (map < string, int >database)
 {
   cout << "enter item code and new price : ";
   cin >> item_code >> item_price;
   database.erase (item_code);
   database.insert (pair < string, int >(item_code, item_price));
   cout<<item_code<<" edited";
   return database;
 }
};

void
Database::loop (map < string, int >database)
{
 cout <<
   "\nEdit Database : \n[1] Add items\n[2] Delete items\n[3] Show item \n[4] Show database\n[5]
Edit Items\n[6] Exit"
   << endl;
 int choice;
 cin >> choice;
 if (choice == 1)
   {
     database=add_item (database);
     loop (database);
   }
 else if (choice == 2)
   {
     database=delete_item (database);
```

```
loop (database);
   }
  else if (choice == 3)
   {
    show_item (database);
    loop (database);
   }
  else if (choice == 4)
   {
    show_database (database);
    loop (database);
   }
  else if (choice == 5)
   {
    database=edit_item (database);
    loop (database);
   }
  else if (choice == 6)
   {
    cout << "\nWelcome USer!\n[1] Bill order\n[2] Edit item database\n";
    int temp;
    cin >> temp;
    if (temp == 1)
         {
          Bill b;
          b.loop_input ();
          b.output (database);
         }
    else
         {
          authority_check (database);
         }
   }
  else
   {
```

```
cout << "Invalid choice\nPress 0 to try again" << endl;
      loop (database);
    }
}

int
main ()
{
  map < string, int >item_database;
  item_database["C0001"] = 100;
  item_database["C0002"] = 200;
  item_database["C0003"] = 300;
  item_database["C0004"] = 400;
  item_database["C0005"] = 500;
  Database db;
  cout << "Item database initialised" << endl;
  cout << "\nWelcome User!\n[1] Bill order\n[2] Edit item database\n";
  int choice;
  cin >> choice;
  if (choice == 1)
    {
      Bill b;
      b.loop_input ();
      b.output (item_database);
    }
  else if (choice == 2)
    {
      db.authority_check (item_database);
    }
  else
    {
      cout << "Invalid input\nTerminating program";
    }
  return 0;
}
```

4.2 SCREENSHOTS

Output:

```
Item database initialised

Welcome User!
[1] Bill order
[2] Edit item database
2


Enter password : agBkW92Sh9*C6$nT

Edit Database :
[1] Add items
[2] Delete items
[3] Show item
[4] Show database
[5] Edit Items
[6] Exit
1

enter item code and price : C0006 600
C0006 added

Edit Database :
[1] Add items
[2] Delete items
[3] Show item
[4] Show database
[5] Edit Items
[6] Exit
```

Fig. 4.2.1. Accessing item codes and editing it

```
2

enter item code to delete : C0003
C0003 deleted from database
Edit Database :
[1] Add items
[2] Delete items
[3] Show item
[4] Show database
[5] Edit Items
[6] Exit
3

enter item code to view : C0005
Code : C0005
Price : 500
Edit Database :
[1] Add items
[2] Delete items
[3] Show item
[4] Show database
[5] Edit Items
[6] Exit
4
Code    Price
C0001    100
C0002    200
C0004    400
C0005    500
```

Fig. 4.2.2. Editing item codes

```
C0006    600

Edit Database :
[1] Add items
[2] Delete items
[3] Show item
[4] Show database
[5] Edit Items
[6] Exit
5
enter item code and new price : C0001 150
C0001 edited
Edit Database :
[1] Add items
[2] Delete items
[3] Show item
[4] Show database
[5] Edit Items
[6] Exit
6

Welcome USer!
[1] Bill order
[2] Edit item database
2


Enter password : 78
Incorrect Password
```

Fig. 4.2.3. Editing item codes

```
Welcome USer!
[1] Bill order
[2] Edit item database
2


Enter password : 78
Incorrect Password
Access not granted


Welcome User!
[1] Bill order
[2] Edit item database
1
enter the item code and quantity :C0001 3
[1] Add items
[2] Calculate amount
1
enter the item code and quantity :C0004 5
[1] Add items
[2] Calculate amount
2

Total amount to be paid : 2450
```

Fig. 4.2.4. Billing order

5. RESULTS AND CONCLUSION

5.1. RESULTS

Our project allows customers to check in without the assistance of staff saving time, money and resources for the retailer and gives the customer a more satisfactory experience.

5.1.1 Results Comparison

Our project saves a lot of time compared to the traditional billing systems and saves a lot of resources as well for the retailers.

5.1.2 Application

Customers can finalise their purchases using self-checkout lanes without the help of a cashier. With this system, the conventional staffed checkout is no longer necessary. Customers gather the items at self-checkout lanes and scan the barcodes on them. Without any help, they finish their shopping. Self-checkout facilities are supervised by one or two staff members who are available to help consumers in need.

Self-checkout kiosks can save costs for companies while improving consumer satisfaction. Stores get bigger and the checkout lines lengthen as your business expands. Additional room needs to be allocated for checkout counters and hire more cashiers to deal with this. Retailers can shorten lines and minimise wait times if clients check out independently. It takes far less time to use a self-checkout billing system than it does to wait for a cashier to bag the items and verify the customer's account. The less lines there are, the less room you need to process orders. Real estate costs can be reduced for small retailers. Massive retail chains, on the other hand, have more chances to promote more of their merchandise.

Customers who are pressed for time might desire a quick checkout and little engagement with staff. Customers can use self-checkout machines to complete their transactions on average in slightly under 4 minutes. They can use self-service checkout to make purchases while enjoying a podcast or listening to music.

One staff can oversee several kiosks at once with self-checkout billing systems. Retailers can reduce the number of cashiers as a result, saving money when business is slow. Customer demand can be satisfied while letting the personnel concentrate on giving customers an enjoyable shopping experience.

Self-checkout systems can free up the staff's time so they can focus on other tasks rather than waiting in the cashier lane. They can replenish shelves, rearrange inventory, and better consult with customers. Now that the store is organised and has a great staff, customers will feel more welcome.

5.2 CONCLUSION

Our project allowed us to utilise the skills and knowledge obtained in class and understand them even better by giving us the opportunity to apply them in practice first hand.

5.2.1 FUTURE ENHANCEMENT

Our project can be further improved by implementing a login system for customers and admins alike so a points system could be implemented for the customers and an data alteration log can be created for the admins. Connecting it to an SQLite database will also further improve its usability in commercial spaces. Another aspect to implement would be to use barcode to keyboard emulation and add in a barcode scanner device to make it more user friendly.

6. REFERENCES

- www.posrg.ca
- www.margcompusoft.com
- www.research.aimultiple.com