Jesus Nunez
Andy Arce
Austin Harmon
Dalila Sanchez
Ebsa Tufa
CS 3321
Dr. Emre Yilmaz
4/22/2022

Parts n' Tools Inventory Management System

I. Introduction

Utilizing software is now a requirement to operate a store. Due to the high demand of products by consumers around the globe, efficiency has become one of the top priorities for stores (businesses). At this rate of consumption, the demand for products will continue to increase. Every store needs to keep maintaining an inventory system in order to boost their business. Some businesses are still operating with outdated methods, our team believes that creating a software that improves the upkeep with inventory is highly important for businesses to maximize their resources. As a result, we chose to create a software system that will allow stores to manage their inventory and make simple transactions.

There are still some small businesses that operate with outdated methods. For instance, we are familiar with a small plumbing business that manually records inventory on paper. If this plumbing business integrated our software in their inventory maintenance, it would make their work less time consuming and it would produce more time to contract more clients. In addition, the users of this software will also be able to store customer's information securely without the exhaustion of looking for them on paper. As a business perk, in case a customer requests a receipt, we made the option to print receipts available!

An advantage of this software is that it can be utilized in businesses that require customer's information. We will use a store-model that we made up known as "Parts n' Tools". Let's assume that this company will provide business owners with exclusive tools. If a customer wants to buy an item from this store, then they will have to create an account in order to make a purchase. "Parts n' tools" are used in a variety of applications, such as home remodeling, construction, and maintenance. The "Parts n' Tools" hardware store will constantly be selling

and restocking various parts and tools, thus requiring a system that will efficiently and securely store inventory.

The interface will contain an employee (user) login and a manager (admin) login. When an employee logs in, they will only have access to the check-out tab. The employee is responsible for the customer's bills and transactions. When the employee enters the customer's information for purchases, the software will display information of the items in stock. The employee will have to click on the items available and add them to a bill. Once added, the employee must click on "print" to print a receipt.

When the manager (administrator) logs in, they will first encounter the employee's tab so they can edit the employee's information. The manager will have to click on the "Dashboard" tab to access a menu as it will display more information about the store. The dashboard will display the best employee, best customer, the number of sales, sales by employee (which they can choose any employee's name) and a summary of sales for inventory, number of customers, and the sales amount from employees. From the dashboard the administrator can access other tabs such as inventory, customers, employees, and manufacturers. Pressing any of the tabs will allow the administrator to view and edit information. The administrator is only able to edit information and an employee is only able to sell. The manager is the only one to create an employee's account for this software. The role of the manager is to edit any information that needs to be modified.

Our goal for this software is to provide inventory and transaction efficiency for businesses to maximize their products and resources. We want to simplify work for users to increase their sales and securely store customer information. This software will help decrease human error when recording inventory. Moreover, this system will be uncomplicated to navigate which many stores will find convenient, especially stores with outdated inventory methods. Stores will have the privilege of more time due to this new system, creating more potential opportunities for their business. Finally, we want users to be able to have a stronger summarization of their store products and customers.

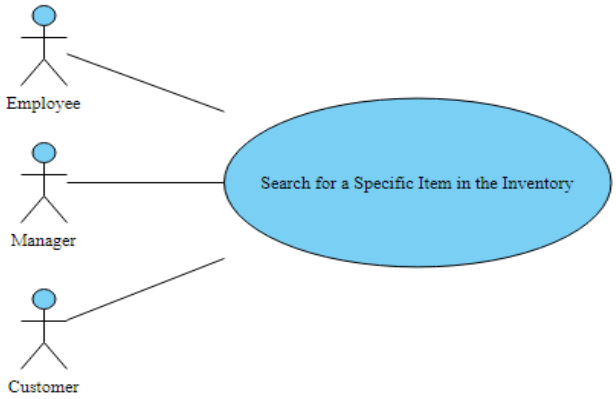
II. Functional Requirements

1. Software should allow a user (employee) login page.
2. Software should allow a manager (administrator) login page.
3. Software should allow a manager to edit information.
    a. Inventory information.
    b. Customer's information.
    c. Manufacturer information.
    d. Employee Information.
4. Software should allow a manager to update information.
    a. Inventory information.
    b. Customer's information.
    c. Manufacturer information.
    d. Employee Information.
5. Software should allow a manager to delete information.
    a. Inventory information.
    b. Customer's information.
    c. Manufacturer information.
    d. Employee Information.
6. Software should allow a manager to access the dashboard containing store information.
    a. Best employee.
    b. Best Customer.
    c. Amount of sales.
    d. Sales by employee.
    e. Store summary.
        i. Number of items sold.
        ii. Number of customers.
        iii. Number of employees.
7. Software should allow an employee to choose items to make transactions.
    a. Items available to choose.
    b. Customer Id.
    c. Quantity,

8. Software should allow the employee to add items selected to bill.
9. Software should allow the employee to print out receipts for customers.
10. Software should allow the employee to view all past transactions.

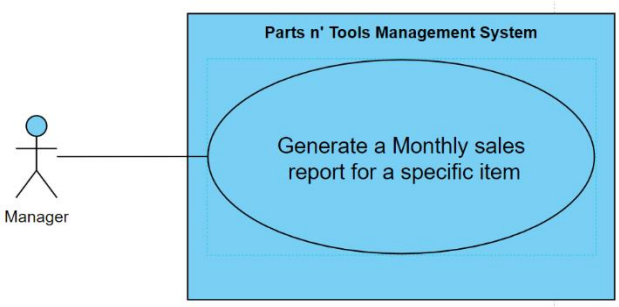
III. Use Cases:

Search for a Specific Item



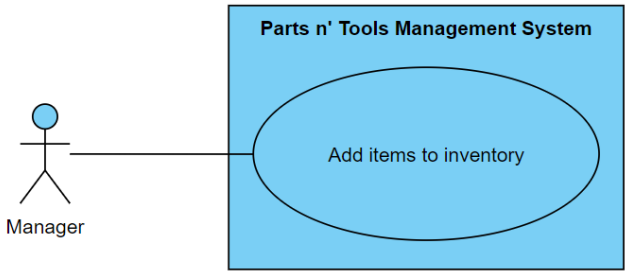
Brief Description: If a customer asks if a product is in stock, a manager or an employee could search for that product. A customer could also search for a product on their own. The information displayed includes product name, type, quantity, price, and manufacturer.

Generate a Monthly Sales Report for a Specific Item



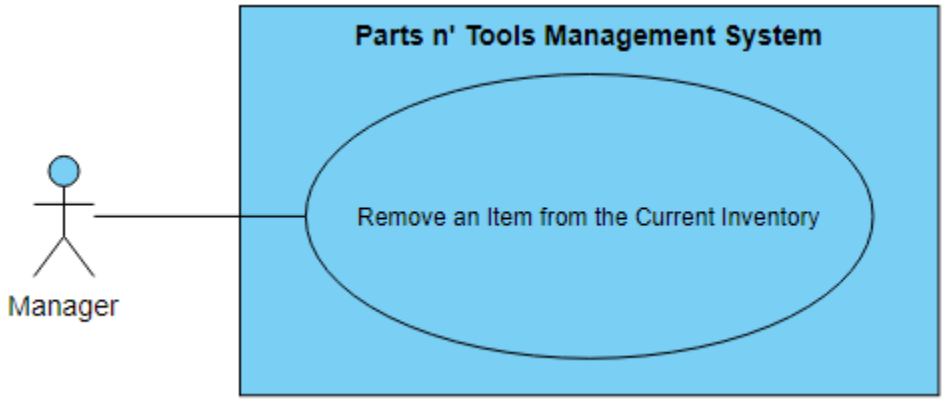
Brief Description: A manager can view the quantity of a specific product that has been sold in a chosen month. The manager first selects the product and then selects the appropriate month to view sales data.

Add items to inventory



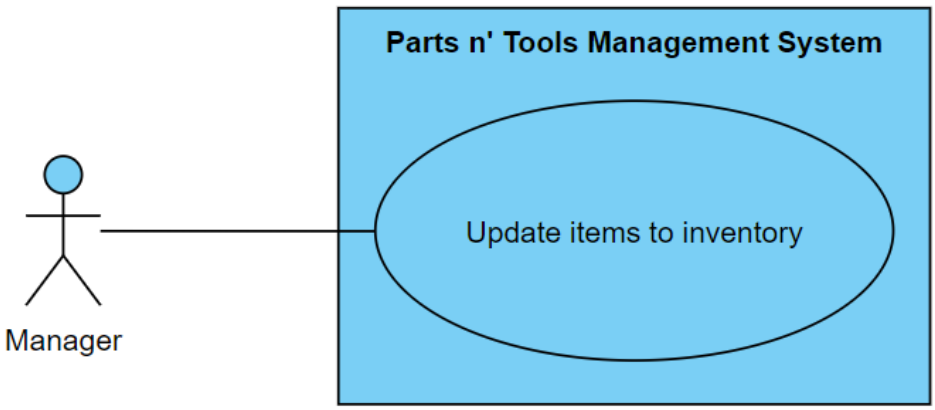
Brief Description: When a new truck arrives with loaded pallets of new inventory, an employee or a manager can add every item to the system. The added information includes the product name, product manufacturer, quantity, and price.

Remove an Item from the Current Inventory



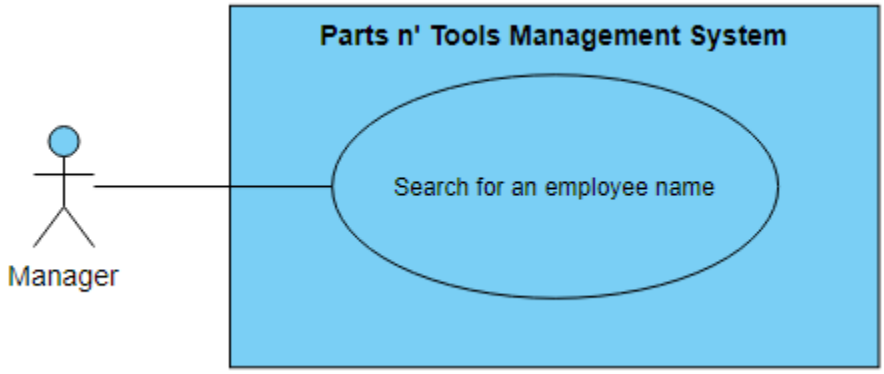
Brief Description: If a manager decides that a product is not selling, the manager has the authority to remove the entire product from the inventory system.

Update items to inventory



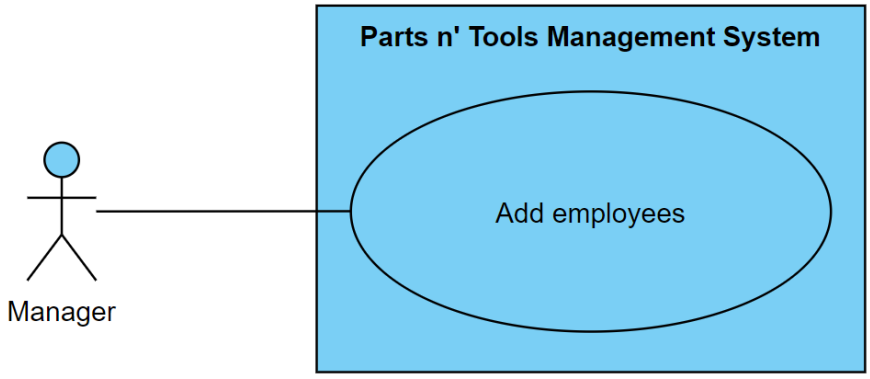
Brief Description: A manager can update items in the inventory. These changes include item name, type, quantity, price, and manufacturer.

Search for an Employee Name



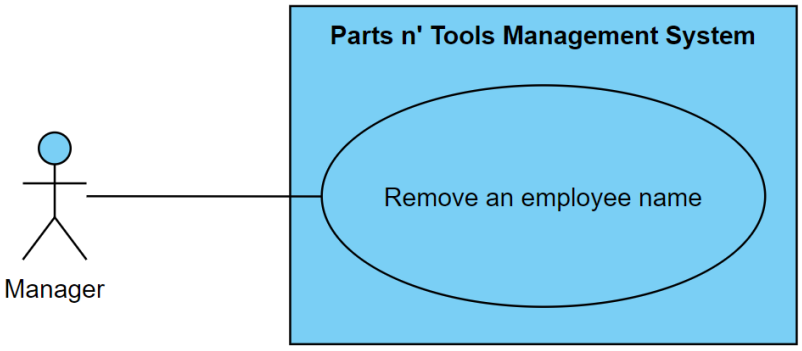
Brief Description: A manager can search for all the employees that are currently hired at the store. A manager can find an employee by using the list of names on a grid.

Add Employees



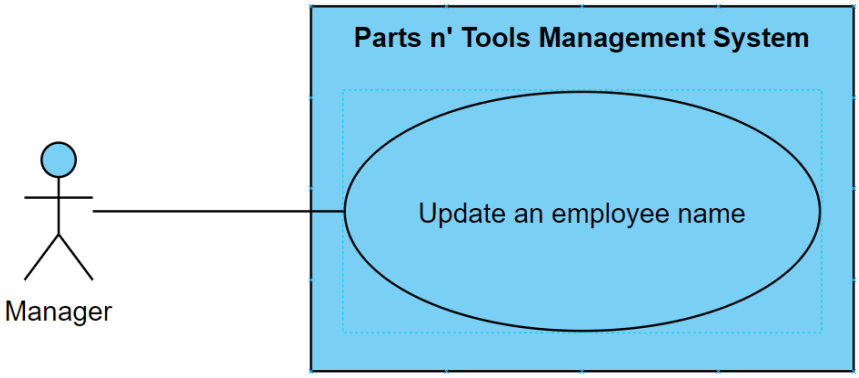
Brief Description: A manager can add a new employee to the system. When an employee is added to the system, the manager enters the first name, last name, and employee ID.

Remove an Employee Name



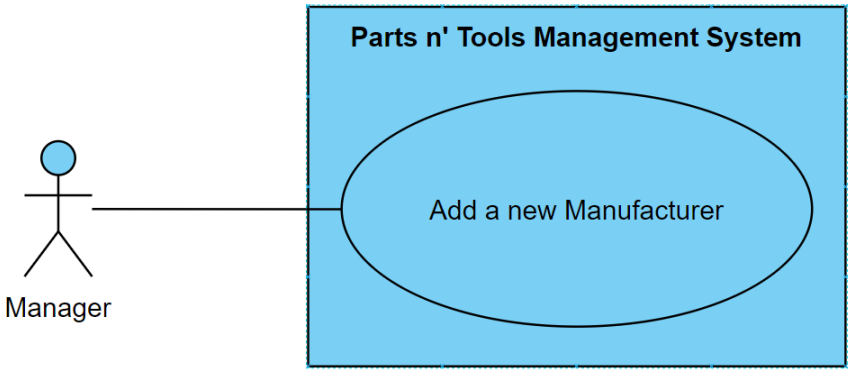
Brief Description: A manager can remove an employee when that employee is not employed by the store. When a manager removes an employee, the first name, last name, and employee ID are removed.

Update an employee name



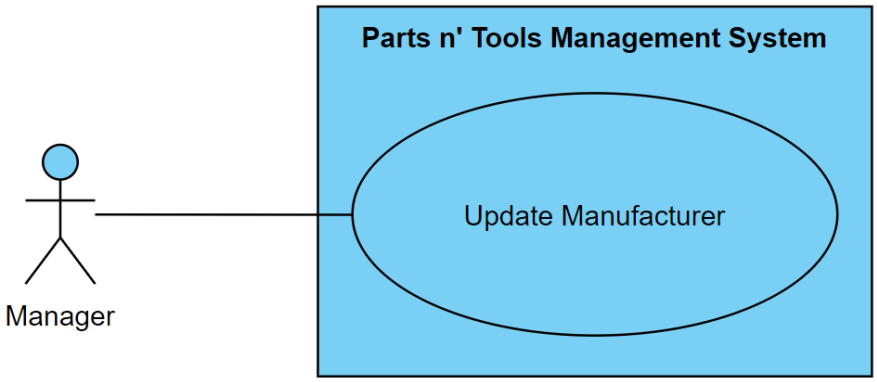
Brief Description: A manager can update an employee name for corrections or name changes.

Add a new manufacturer



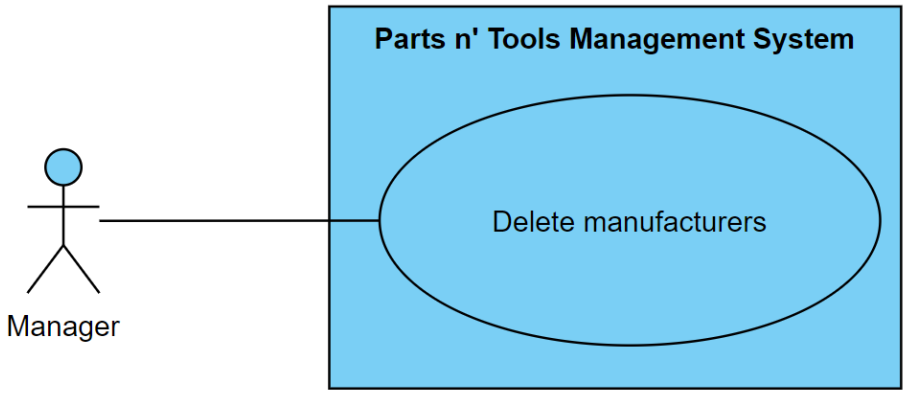
Brief Description: A manager can add new manufacturers to the software. The information entered by the manager includes manufacturer name and manufacturer ID.

Update manufacturer



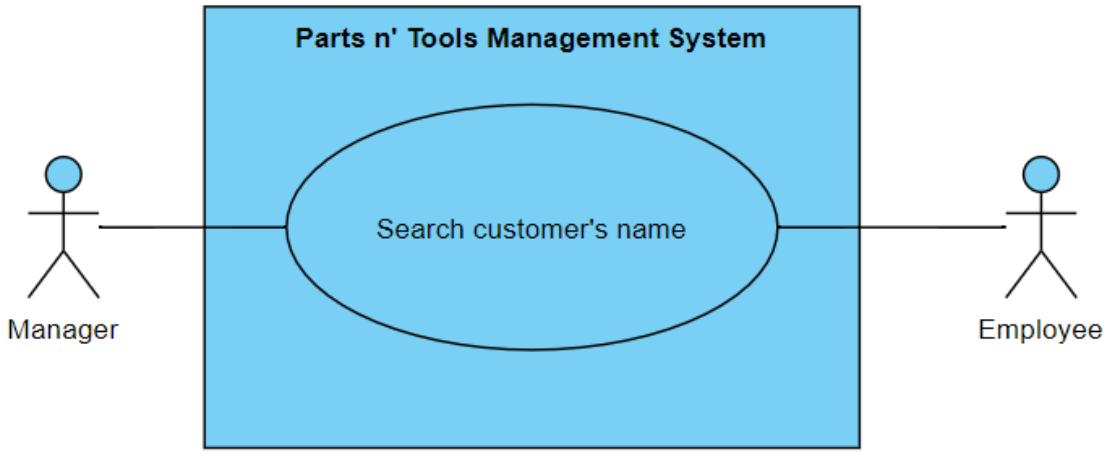
Brief Description: A manager can update manufacturers name in case of name changes or corrections.

Delete manufacturers



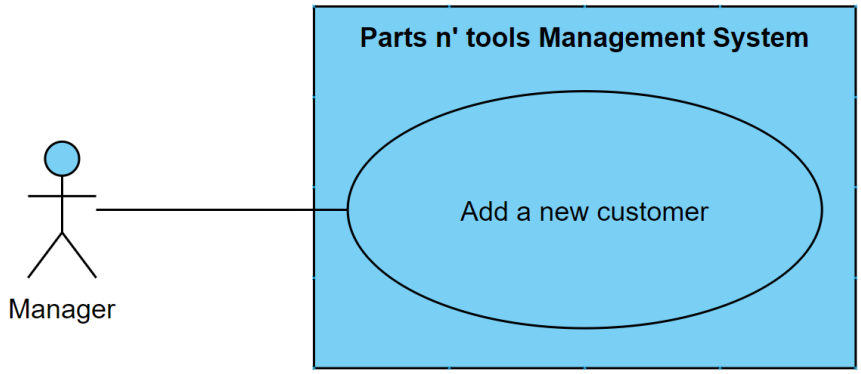
Brief Description: A manager can delete manufacturers for any reason. The manufacturer name and manufacturer ID are removed when this change occurs.

Search customer's name



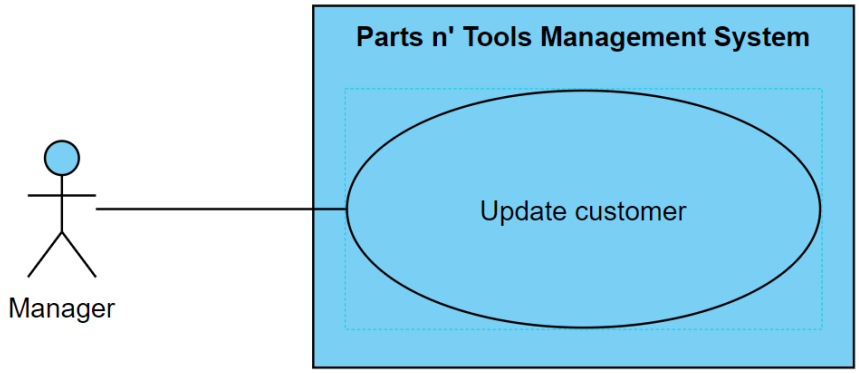
Brief Description: Both managers and employees can search for a customer's name. The manager can edit customer's info, while the employee searches customers for transactions.

Add a new customer



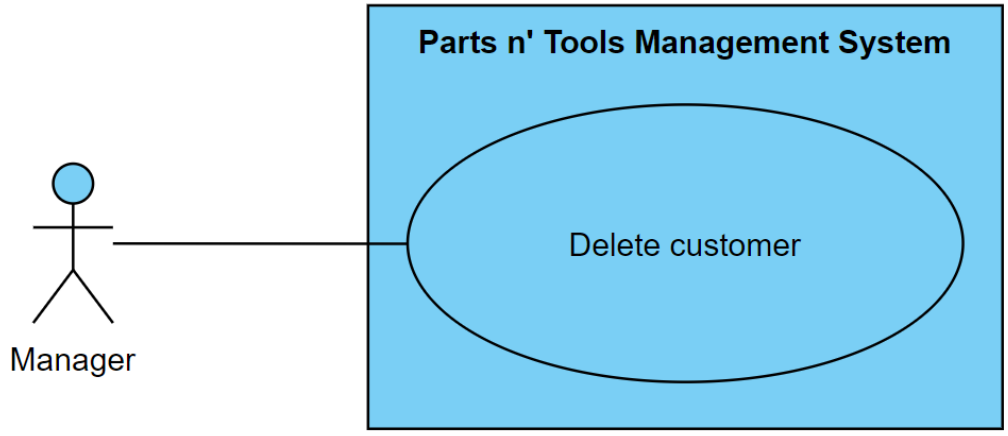
Brief Description: The manager is able to add a customer in the software to make purchases. The manager enters the first name and last name of the customer. A customer ID is also added to the system.

Update customer



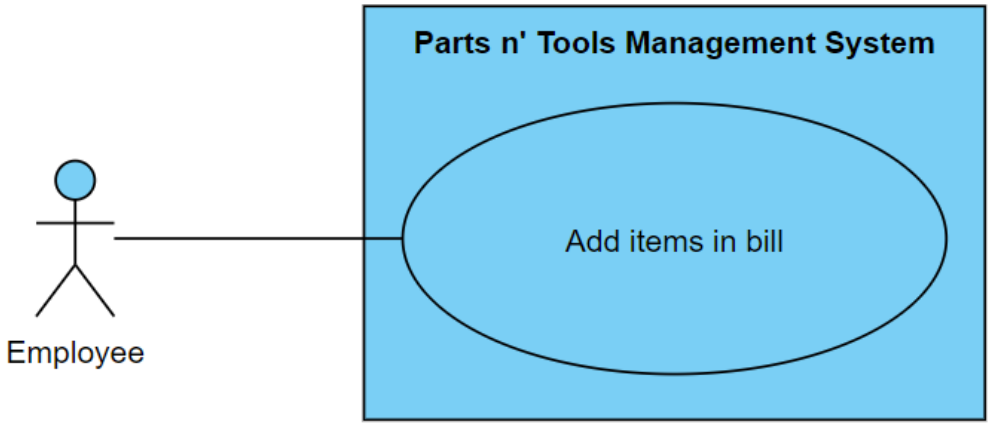
Brief Description: The manager is able to update a customer's information in case of name changes or corrections.

Delete customer



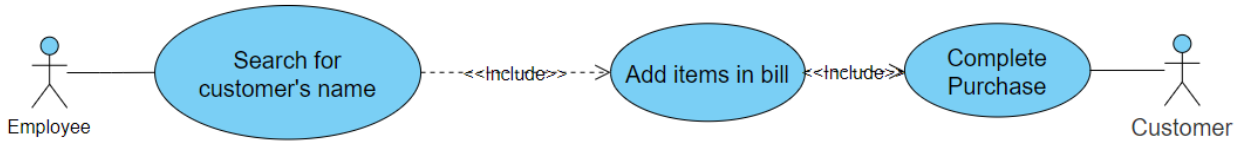
Brief Description: The manager is able to delete a customer's information of the system upon the customer's request. The first name, last name, and customer ID are removed when this change occurs.

Add items in bill



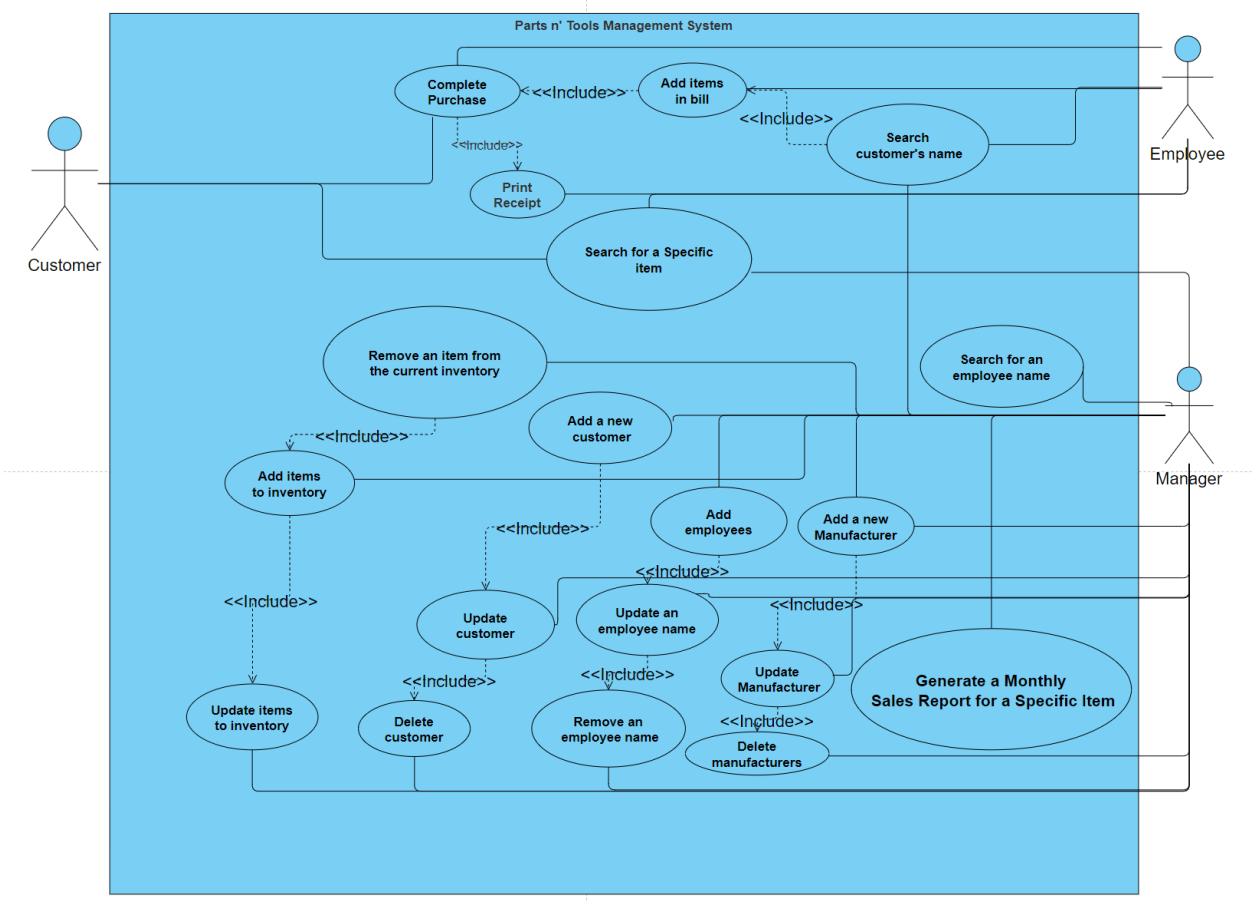
Brief Description: An employee must add items to add items to the bill for customers to purchase. The total price is updated as the items are added to the bill.

Complete purchase

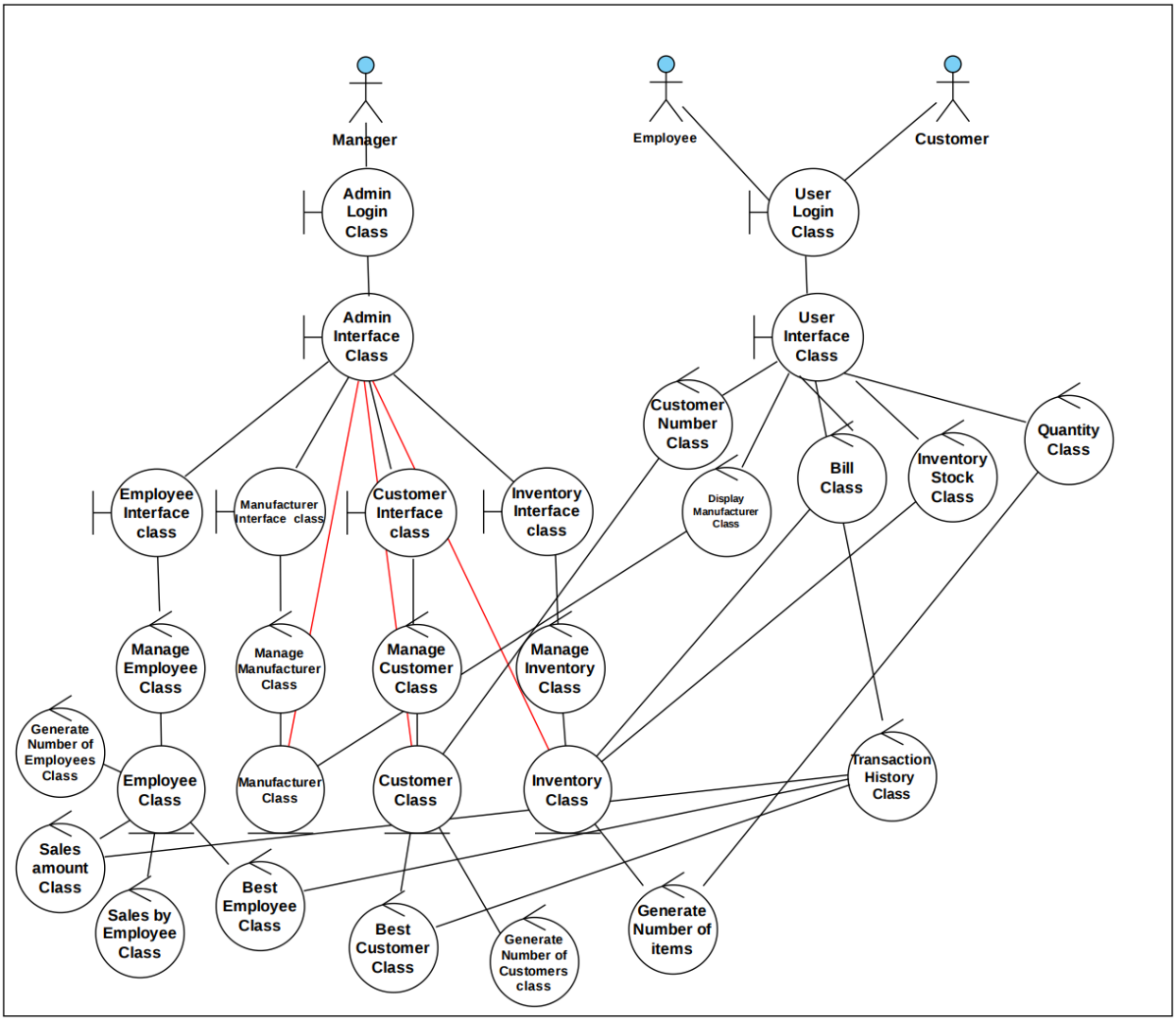


Brief Description: An employee is able to complete transactions for the customer. The employee must search for a customer number to complete transactions. The total price of all items is displayed to the customer. The customer must confirm the purchase to officially complete the purchase.

Full Use Case diagram

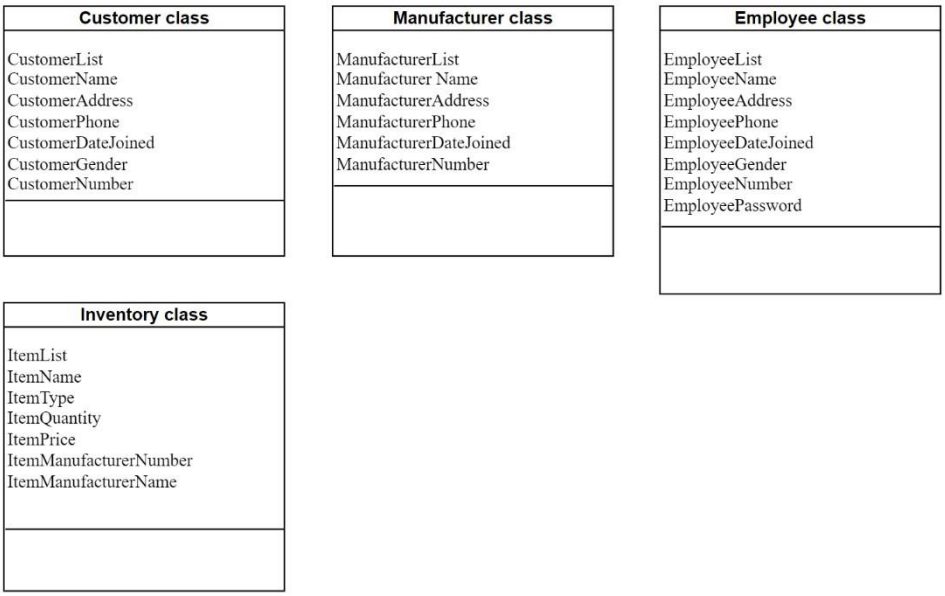


IV. Class Diagram

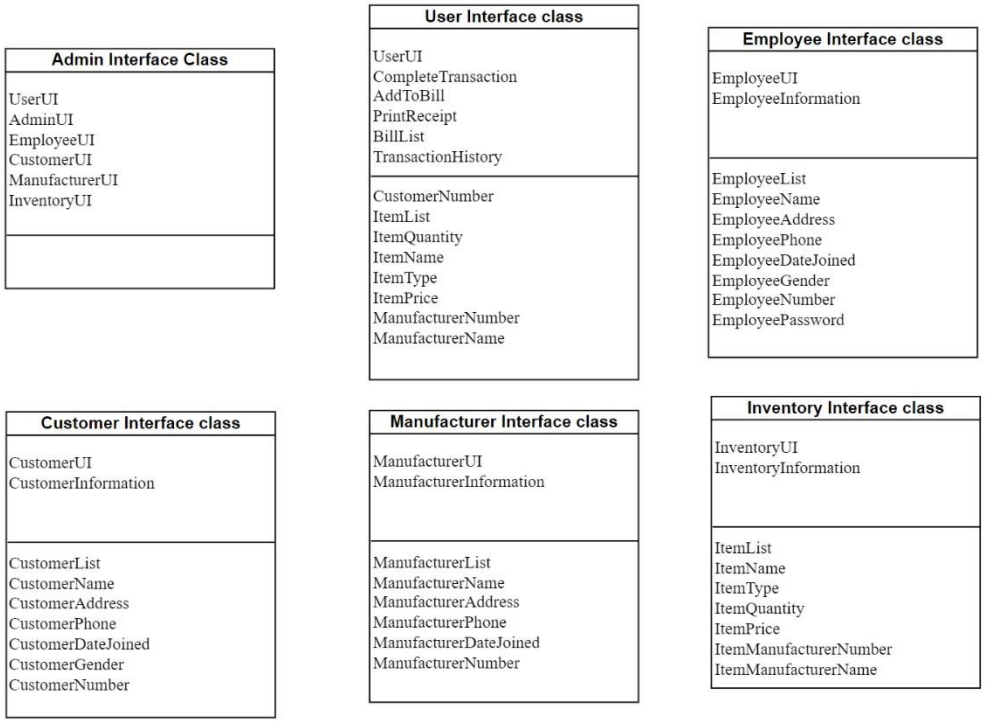


Classes with Attributes:

Entity classes:



Boundary Classes:



Control Classes:

Admin

Manage Customer Class
AddCustomer UpdateCustomer RemoveCustomer

Manage Employee Class
AddEmployee UpdateEmployee RemoveEmployee

Manage Manufacturer Class
AddManufacturer UpdateManufacturer RemoveManufacturer

Manage Inventory Class
AddInventroy UpdateInventory RemoveInventory

Admin Login
AdminUI
AdminLogin AdminUI

Sales Amount class
AddSalesAmount
AdminUI UserUI BillList

Estimate Item Price Class
NumberOfItems
AdminUI InventoryList

Estimate Number Of Customers Class
NumberOfCustomers
AdminUI CustomerList

Sales By Employee class
SalesByEmployee
AdminUI EmployeeList BillList

Best Employee Class
AddBestEmployee
AdminUI BillList EmployeeList

Estimate Number of Employee Class
NumberOfEmployees
AdminUI EmployeeList

Best Customer Class
AddBestCustomer
AdminUI CustomerList

User

Transaction History Class
AddProcessedTransaction

Customer Number Class
ChooseCustomerNumber
CustomerList

Inventory Stock Class
GetInventoryStockInfo
InventoryList ItemQuantity

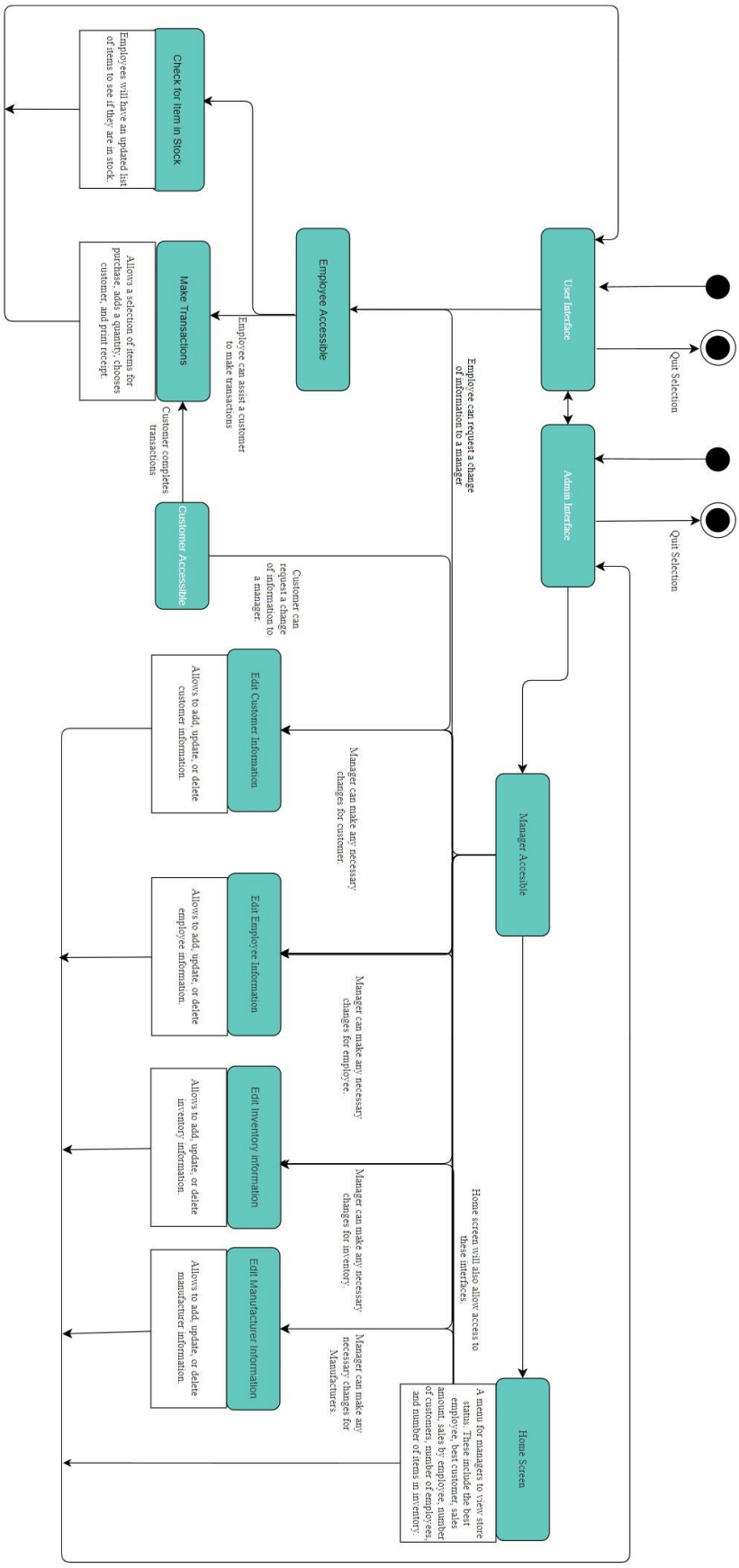
Bill Class
AddNewBill PrintReceipt

Quantity Class
AddItemQuantity
AddNewBill

Display Manufacturer
GetManufacturerInfo
InventoryList

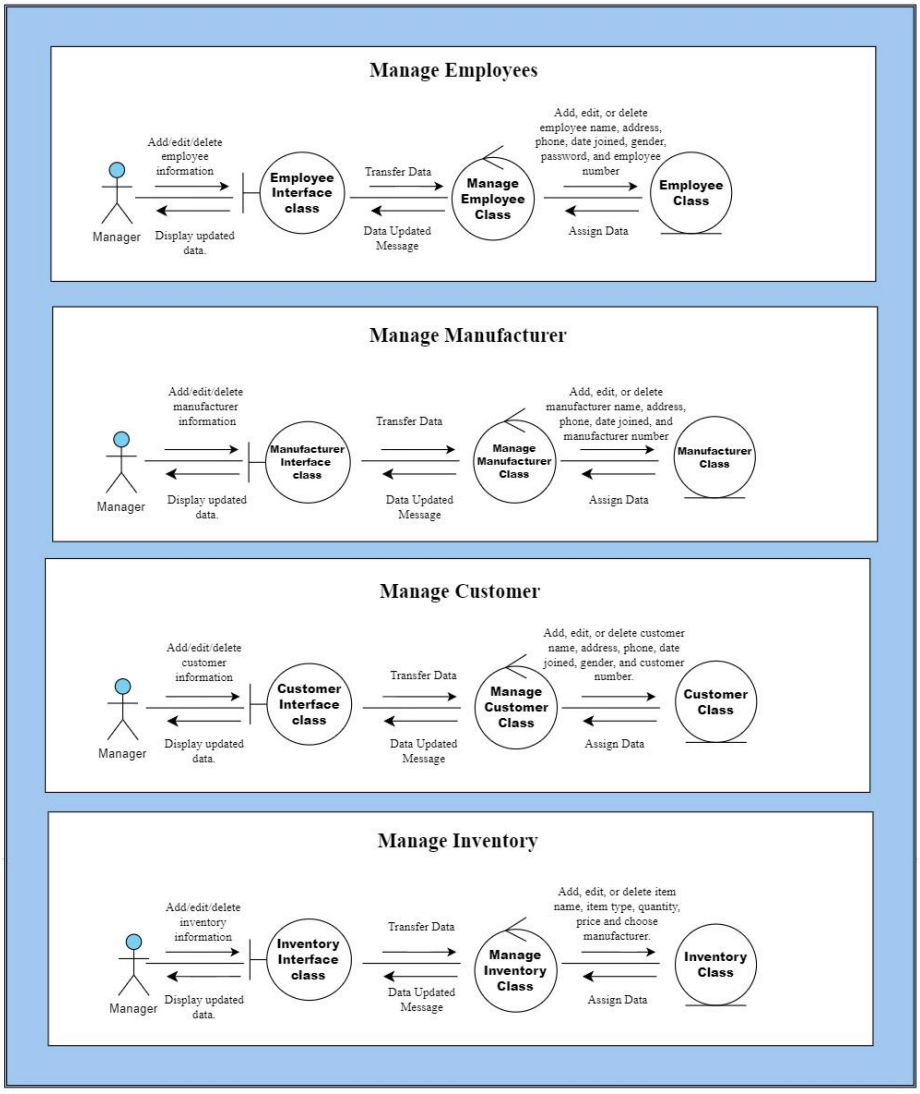
User Login
UserUI
AdminLogin UserUI TransactionsUI

V.State Chart:



VI. Use-Case Realization:

Admin



Manage Employees

A manager wants to manage (add, edit, or delete) employee information.

1. The manager clicks into the employee interface menu.
2. The manager will be transferred to the employee's menu displaying a list of employees.
3. The manager can add, edit, or delete employee information such as name, address, phone, date joined, gender, employee number, and password.
4. Data will be assigned in a grid after updating employee information.
5. The manager will be notified that employee information has been updated.

6. The updated information will be displayed in a grid in the employee interface.

Manage Manufacturer

A manager wants to manage (add, edit, or delete) manufacturer information.

1. The manager clicks into the manufacturer interface menu.
2. The manager will be transferred to the list of manufacturers
3. The manager can add, edit, or delete manufacturers. The manager can edit/add manufacturer name, address, phone, date joined, and manufacturer number.
4. Data will be assigned in a grid after updating manufacturer information.
5. The manager will be notified that manufacturer information has been updated.
6. The updated information will be displayed in a grid in the manufacturer interface.

Manage Customer

A manager wants to manage (add, edit, or delete) customer information.

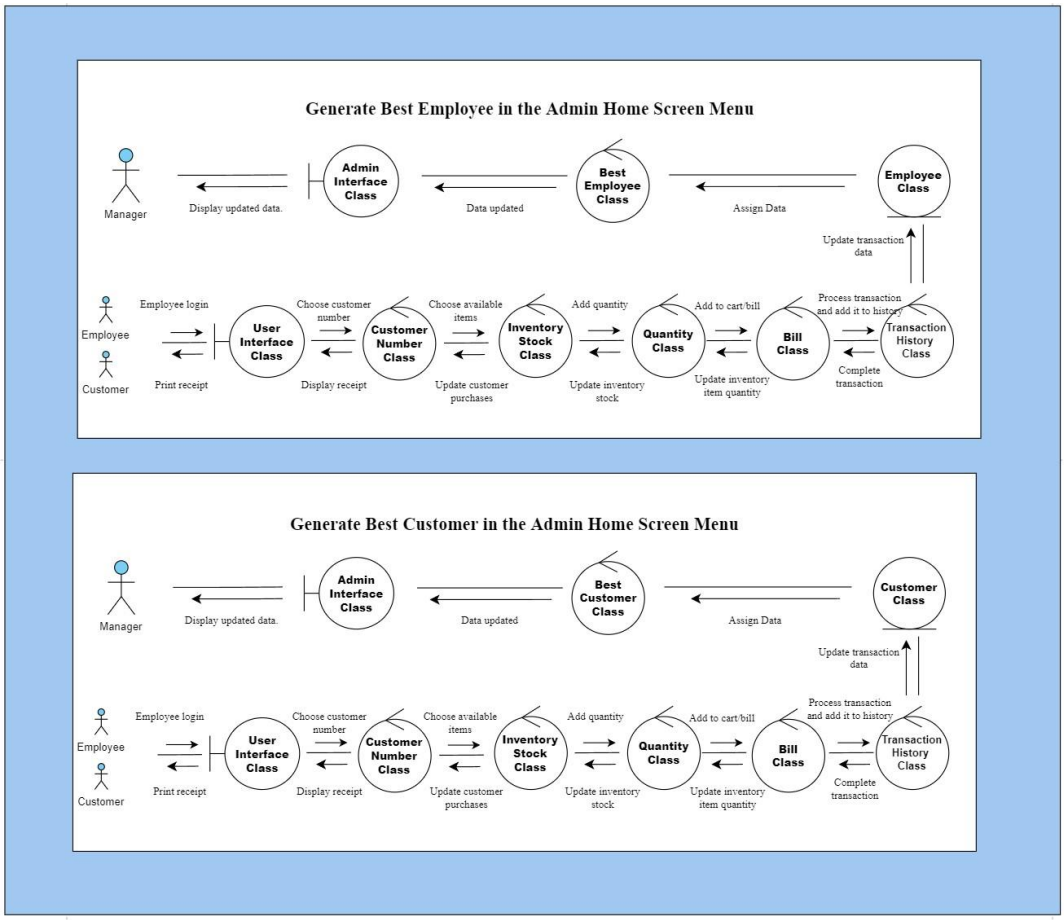
1. The manager clicks into the customer interface menu.
2. The manager will be transferred to the customer's menu displaying a list of customers.
3. The manager can add, edit, or delete customers. Managers can edit/add customer information such as name, address, phone, date joined, gender, and customer number.
4. Data will be assigned in a grid after updating customer information.
5. The manager will be notified that customer information has been updated.
6. The updated information will be displayed in a grid in the customer interface.

Manage Inventory

A manager wants to manage (add, edit, or delete) inventory information.

1. The manager clicks into the Inventory Interface menu.
2. The manager will be transferred to the manage inventory menu
3. The manager can add, edit, or delete items. The manager can edit/add item name, item type, quantity, price, and choose manufacturer.
4. Data will be assigned in a grid after updating inventory information.
5. The manager will be notified that the inventory information has been updated.

6. The updated information will be displayed in a grid in the inventory interface.



Generate Best Employee in the Admin Home Screen Menu

How a manager generates the Best Employee.

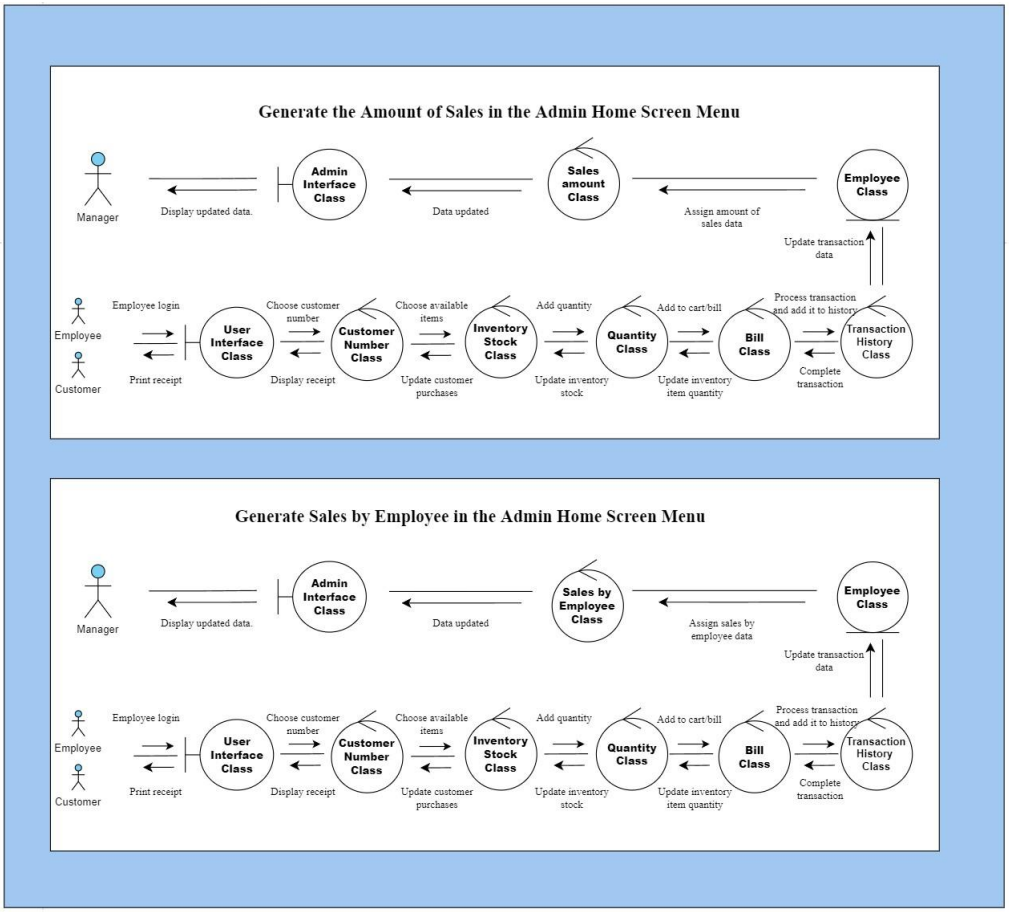
1. An employee must log in to the user interface.
2. Employees must choose a customer number.
3. Employees must choose an available item(s) before purchase.
4. Employees must add a quantity of the chosen item(s) that a customer wants to purchase.
5. Employees must add items to cart to proceed purchase.
6. Employees must process transactions and it will be added to a grid of transactions history.
7. The data will be saved to a transaction's history grid.
8. The software will generate which employee has the most sales.
9. Data will be updated in the admin interface.

10. When the manager logs into the home screen menu, the manager will see the best employee being updated.

Generate Best Customer in the Admin Home Screen Menu

A manager generates the Best Customer.

1. An employee must log in to the user interface.
2. Employees must choose a customer number.
3. Employees must choose an available item(s) before purchase.
4. Employees must add a quantity for the chosen item(s) that a customer want to purchase.
5. Employees must add item(s) to the cart and proceed to purchase.
6. Employees must process transactions and it will be added to a grid of transactions history.
7. The data will be saved to a transaction history grid.
8. The software will generate which customer has the most purchases.
9. Data will be updated in the admin interface.
10. When the manager logs into the home screen menu, the manager will see the best customer being updated.



Generate the Amount of Sales in the Admin Home Screen Menu

A manager can view the generated number of sales.

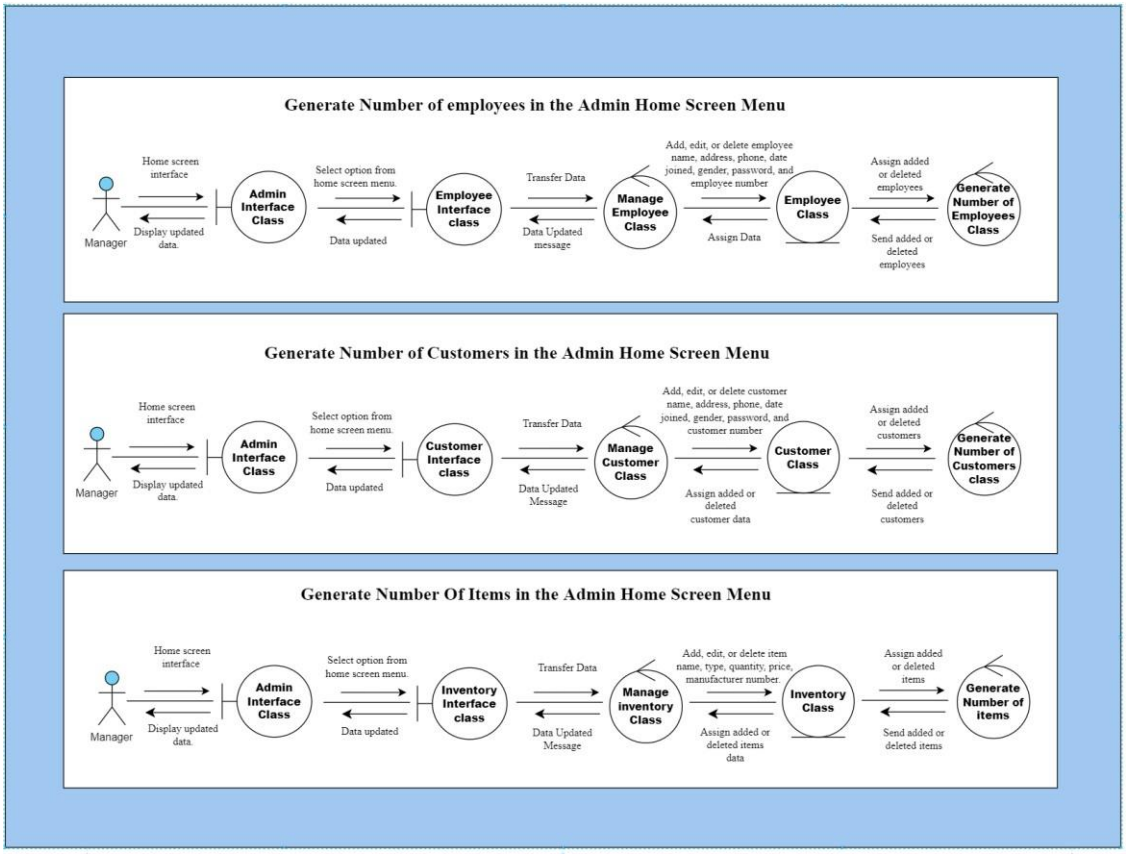
1. An employee must log in to the user interface.
2. Employees must choose a customer number.
3. Employees must choose available item
4. Employees must add a quantity for the item chosen.
5. Employees must add item(s) to the cart and proceed to purchase.
6. Employees must process the transaction and it will be added to a grid of transactions history.
7. The data will be saved to the transactions history grid.
8. The software will add the total amount of sales from every customer purchase.
9. Data will be updated in the admin interface.

10. When the manager logs into the home screen menu, the manager will see the information being updated.

Generate Sales by Employee in the Admin Home Screen Menu

A manager can view the generated sales from a specific employee.

1. An employee must log in to the user interface.
2. Employees must choose a customer number.
3. Employees must choose available item
4. Employees must add a quantity for the item chosen.
5. Employees must add item(s) to the cart and proceed to purchase.
6. Employees must process the transaction and it will automatically be added to a grid of transaction history.
7. The data will be saved to the transaction history.
8. The software will add the total amount of sales from an employee's account.
9. Data will be updated in the admin interface.
10. When the manager logs into the home screen menu, the manager can select an employee and view the number of sales the chosen employee has made.



Generate Number of employees in the Admin Home Screen Menu

A manager can view the total number of employees based on accounts created.

1. The manager must log in to access the admin home screen menu.
2. The manager must select the employee interface.
3. The manager will be transferred to the employee's interface displaying a list of employees.
4. The manager can add, edit, or delete employees. The manager can edit/add employee's information such as name, address, phone, date joined, gender, password, and employee number.
5. The software will count the total number of employees when they are added or deleted by saving a value.
6. The saved value will be sent to the employee's class.
7. Data will be assigned in the grid list of employees.
8. Manager will be notified that employee information has been updated.

9. Data will be updated in the admin interface.
10. When the manager accesses the home screen menu, the manager will see the total number of employees.

Generate Number of Customers in the Admin Home Screen Menu

A manager can view the total number of customers based on accounts created.

1. The manager must log in to access the admin home screen menu.
2. The manager must select the customer interface.
3. The manager will be transferred to the customer's menu displaying a list of customers.
4. The manager can add, edit, or delete customer information. The manager can edit /add customer information such as name, address, phone, date joined, gender, and customer number.
5. The software will count the total number of customers when they are added or deleted by saving a value.
6. The saved value will be sent to the customer's class.
7. Data will be assigned in the grid list of customers.
8. Manager will be notified that customer information has been updated.
9. Data will be updated in the admin interface.
10. When the manager accesses the home screen menu, they will see the total number of customers.

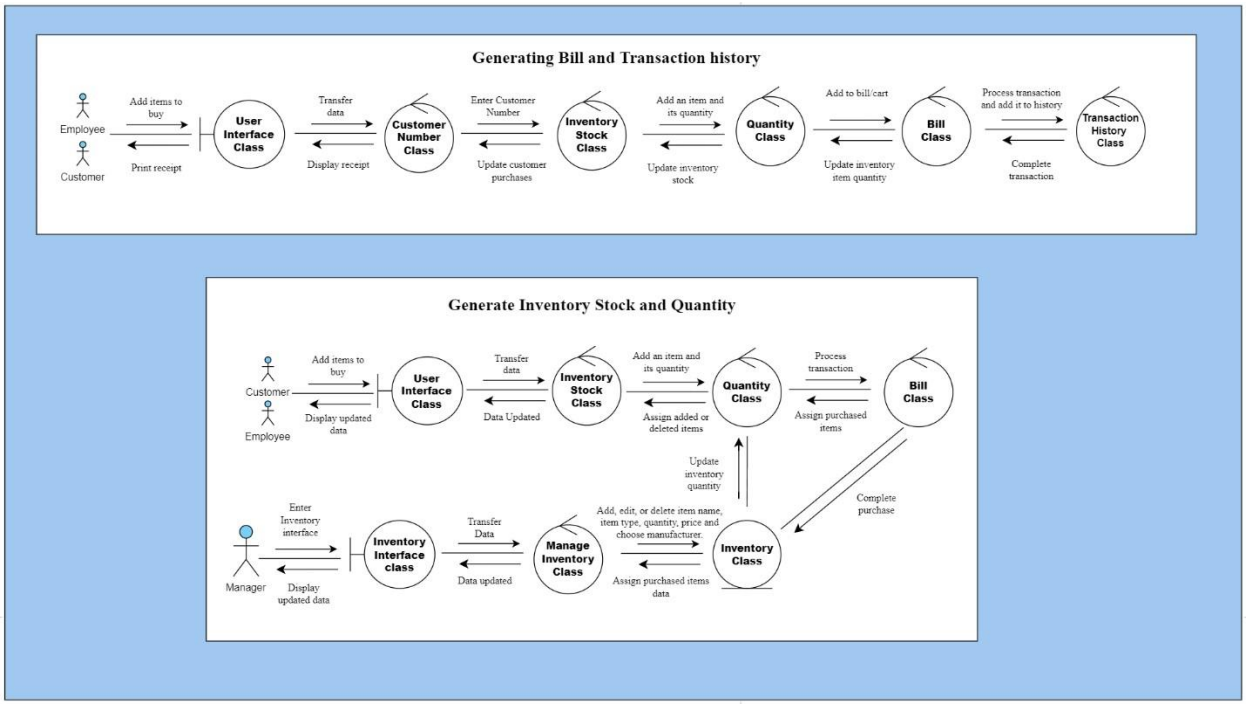
Generate Number of Items in the Admin Home Screen Menu

A manager can view the total number of items.

1. The manager must log in to access the admin home screen menu.
2. The manager must select the items interface.
3. The manager will be transferred to the item's menu displaying a list of items.
4. The manager can add, edit, or delete items. Managers can edit item information such as item name, item type, quantity, price, and manufacturer number.
5. The software will count the total number of items when they are added or deleted by saving a value.
6. The saved value will be sent to the items class.
7. Data will be assigned in the grid list of employees.
8. Manager will be notified that employee information has been updated.

9. Data will be updated in the admin interface.
10. When the manager accesses the home screen menu, they will see the total number of items.

User



Generating Bill and Transaction History

An employee can assist a customer in making purchases. When purchase is made, the employee has the option to print a receipt for customer.

1. An employee must log in to the user interface.
2. Employees will be transferred to the user interface displaying the inventory stock.
3. Employees must select customer number.
4. Employees must add an item and its quantity.
5. Employees must add item to bill/cart.

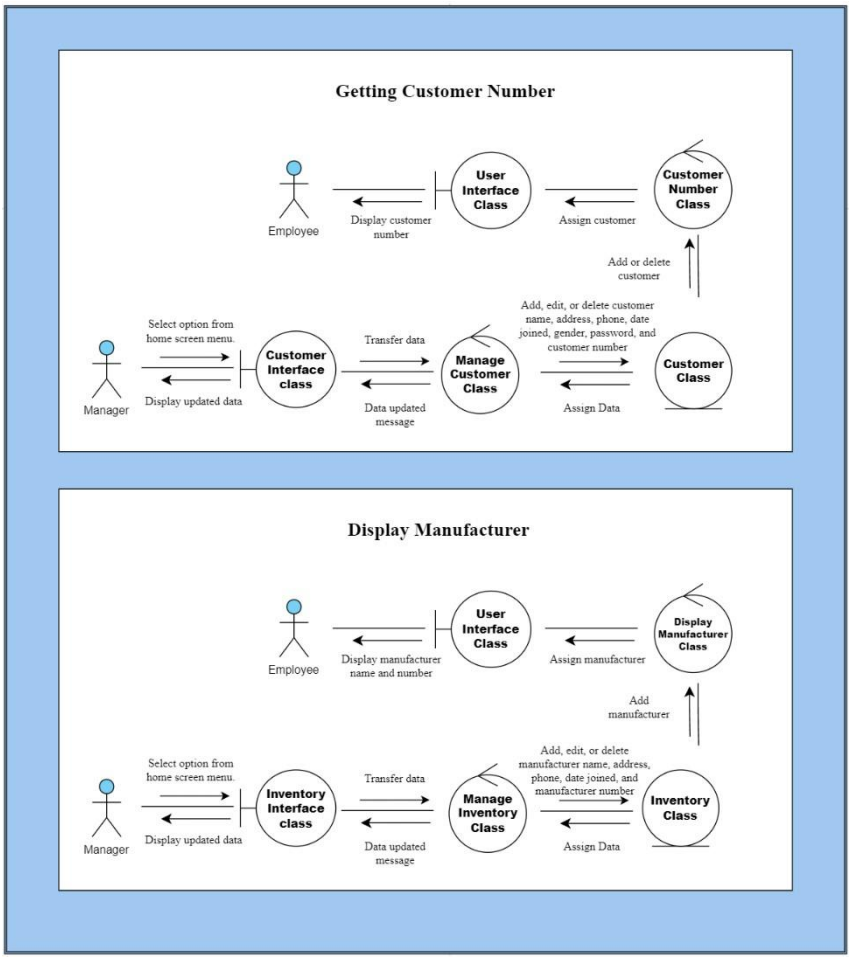
6. Customers must process transaction; the transaction will be saved in a grid that lists transaction history.
7. Customers must complete transaction.
8. When an item is purchased, the quantity of an item will update.
9. The updated quantity will also update inventory stock.
10. The amount of purchases a customer makes will be update to their number.
11. When employee clicks on prints receipt, it will display receipt first.
12. The employee can print the receipt for a customer.

Generating Inventory Stock and Quantity

When an employee logs in as the user, they will get to see what is in stock in the inventory. When a manager updates inventory stock, it will also display it in user's interface. When a customer makes a purchase, inventory stock will update for both employee and manager interfaces.

1. **For the manager:** The manager clicks into the Inventory Interface menu
2. **For the manager:** The manager will be transferred to the Manage Inventory menu
3. **For the manager:** The manager can add, edit, or delete items. The manager can edit item name, item type, quantity, price and choose manufacturer.
4. **For the manager:** Data will be assigned in a grid after updating inventory information.
5. **For the manager:** The manager will be notified that the inventory information has been updated.
6. **For the manager:** The updated information will be displayed in the inventory interface grid.
7. **From the manager:** Item quantity will be update in the user's interface grid that displays item quantity.
8. **For employee:** An employee must log in to the user interface.
9. **For employee:** The employee will be transferred to the user interface displaying the inventory stock.
10. **For employee:** The employee must add an item and its quantity.
11. **For employee:** The employee assists customer in processing transactions.

12. **From employee to manager diagram route:** When a customer completes a purchase, the item quantity will also be update. Afterwards starting from (4) to (6), data will be assigned in a grid after updating inventory information, then the manager will be notified that the inventory information has been updated, and the updated information will be displayed in the inventory interface grid.
13. **For employee diagram route:** Number of purchased items will be assigned to a grid after customer completes transaction.
14. **For employee:** Quantity will be updated from customer purchases. It will be assigned to a grid that displays inventory stock for employees.
15. **For employee:** Data will be updated in the user interface.
16. **For employee:** New updated data will be displayed for employee.



Getting Customer Number

An employee can select a customer based on their number. The software stores the number into the user interface when a manager adds or deletes a customer.

1. **For the manager**: The manager must select customer interface.
2. **For the manager**: The manager will be transferred to the customer interface menu displaying a list of customers.
3. **For the manager**: The manager can add, edit, or delete customer information such as name, address, phone, date joined, gender, and customer number.
4. **For the manager**: Data will be assigned in a grid after updating inventory information, the manager will be notified that the inventory information has been updated, and the update information will be displayed in a grid in the inventory interface.
5. **From the manager:** The added or deleted customer data will send a value to the user interface.
6. **For employee**: Data will be assigned to a list displaying customer numbers.
7. **For employee**: The employees will be able to see the updated customer number.

Display Manufacturer

An employee can view an item's manufacturer when selecting or searching an item for a customer.

1. **For the manager**: The manager must select inventory interface.
2. **For the manager**: The manager will be transferred to the inventory interface menu displaying a list of customers.
3. **For the manager**: The manager can add, edit, or delete inventory information. Managers can edit item name, item type, quantity, price, and choose manufacturer.
4. **For the manager**: Data will be assigned in a grid after updating inventory information, the manager will be notified that the inventory information has been updated, and the updated information will be displayed in a grid in the inventory interface.
5. **From the manager:** The added or deleted customer data will send a value to the user interface.

6. **For employee**: Data will be assigned to a grid with a list of items and displaying manufacturers.
7. **For employee**: The employees will be able to see the updated item's manufacturer.

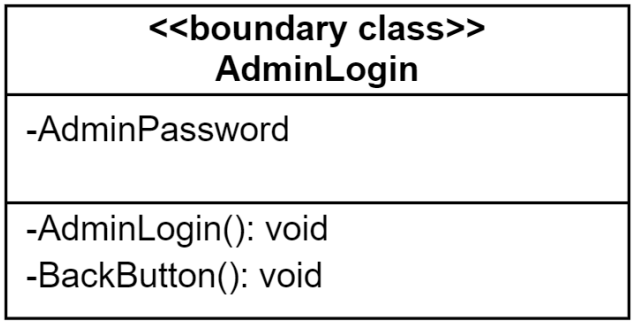
VII. Detailed design

Classes:

<<boundary class>> UserLogin
-EmployeeName:varchar(100) -EmployeePassword: varchar(100)
-UserLogin(): void -AdminButton(): void

```
void UserLogin()
{
    If(EnterUserNameTb.Text == " " || UserNamePasswordTb.Text == " ")
    {
        Display << "Please Enter Both Username and Password";
    }
    else
    {
        open EmployeeData_Connection();
        select Count(*) from EmployeeTableData;
        if(UserDataTable.Rows[0][0] == "1")
        {
            Application.Transactions();
        }
        else
        {
            display << "Wrong Username or Password";
        }
```

```
            close EmployeeData_Connection();
        }
}
void AdminButton()
{
        Application.AdminLogin();
}
```



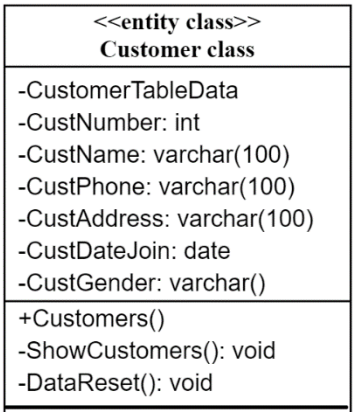

```
void AdminLogin()
{
        if(AdminPasswordTb.Text == " ")
        {
            display << "Please enter a password";
        }
        else if (AdminPassword.Text = "Admin")
        {
            Application.HomeScreenMenu();
        }
        else
        {
            display << "Wrong Admin Password";
        }
}
```

void **BackButton()**

{

Application.UserLogin();

}



Customers()

{

ShowCustomers();

}

void **ShowCustomers()**

{

open CustomerData_Connection();

select * from CustomerTableData;

close CustomerData_Connection();

}

void **DataReset()**

{

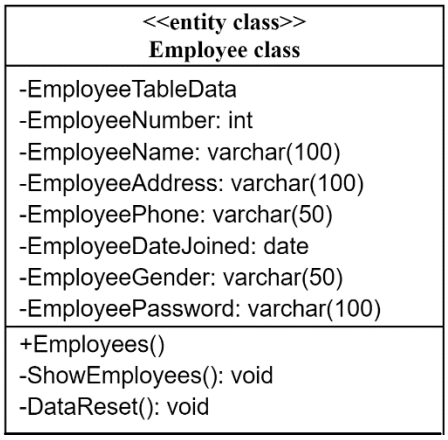
set EnterCustomerNameTb.Text to “ ”;

set EnterCustomerPhoneTb.Text to “ ”;

set SelectCustomerGenderCb.SelectedIndex to 0;

set EnterCustomerAddressTb.Text to “ ”;

}



Employees()

{

ShowEmployees();

}

void **ShowEmployees()**

{

open EmployeeData_Connection();

select * from EmployeeTableData;

close EmployeeData_Connection();

}

void **DataReset()**

{

set EnterEmployeeNameTb.Text to “ ”;

set EnterEmployeePhoneTb.Text to “ ”;

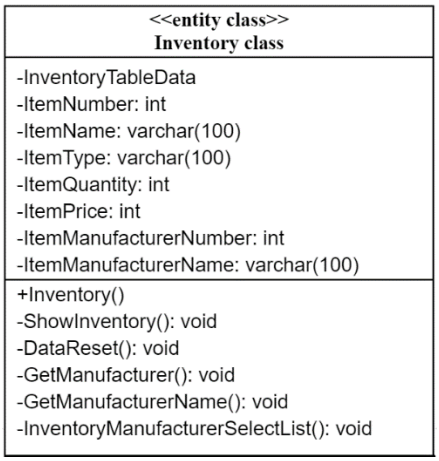
set SelecEmployeeGenderCb.SelectedIndex to 0;

set EnterEmployeeAddressTb.Text to “ ”;

```
        set EmterEmployeePasswordTb.Text to " ";
        set DataKey to 0;
}
```

<<entity class>> Manufacturer class
-ManufacturerTableData -ManufacturerNumber: int -ManufacturerName: varchar(100) -ManufacturerAddress: varchar(100) -ManufacturerPhone: varchar(50) -ManufacturerDateJoin: date
+Manufacturers() -ShowManufacturers(): void -DataReset(): void

```
Manufacturers()
{
        ShowManufacturers();
}
void ShowManufacturers()
{
        open ManufacturerData_Connection();
        select * from ManufacturerTableData;
        close ManufacturerData_Connection();
}
void DataReset()
{
        set EnterManufacturerNameTb.Text to " ";
        set EnterManufacturerPhoneTb.Text to " ";
        set EnterManufacturerAddressTb.Text to " ";
        set DataKey to 0;
}
```



Inventory()

{

ShowInventory();

}

void **ShowInventory()**

{

open InventoryData_Connection();

select * from InventoryTableData;

close InventoryData_Connection();

}

void **DataReset()**

{

set EnterItemNameTb.Text to “ ”;

set EnterItemTypeCb.SelectedIndex to 0;

set EnterItemPriceTb.Text to “ ”;

set EnterItemQuantity.Text to “ ”;

set ItemManufacturerNumber.Text to “ ”;

set DataKey to 0;

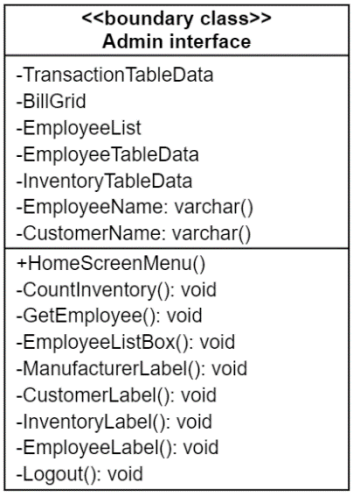
}

```
void GetManufacturer()
{
        open ManufacturerData_Connection();
        Select MaufacturerNumber from ManufacturerTable;
        Procedure GetManufacturer(ManufacturerNumber);
        Display << ManufacturerNumber;
        end procedure;
        close ManufacturerData_Connection();
}

void GetManufacturerName()
{
        open ManufacturerData_Connection();
        Select * from ManufacturerTable where ManufacturerNumber =’”;
        Procedure GetManufacturerName(ManufacturerNumber, ManufacturerName);
        display << ManufactuerNumber << ManufacturerName;
        end procedure;
        close ManufacturerData_Connection();
}


        set integer DataKey to 0;

void InventoryManufacturerSelectList()
{
        GetManufacturerName();
}
```



HomeScreenMenu()

{

CountInventory();

CountEmployees();

CountCustomers();

SalesTotalAmount();

SalesTotalAmountByEmployee();

BestEmployee();

BestCustomer();

}

void **CountInventory**()

{

open InventoryData_Connection();

Select Count(*) from InventoryTableData;

procedure Count(InventoryTableData);

display << InventoryTableData;

end procedure;

close InventoryData_Connection();

}

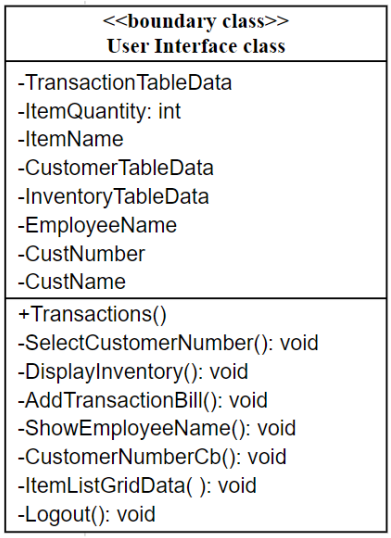
void **GetEmployee**()

{

```
        open Transaction_Connection();
        Select EmployeeName from EmployeeDataTable;
        procedure Select(EmployeeName);
        display << EmployeeName;
        end procedure;
        close Transaction _Connection();
}

void EmployeeListBox()
{
        display << Listbox;
        GetEmployee();
}
void ManufacturerLabel
{
        display << "Manufacturer";
        Application.Manufacturer();
}
void CustomerLabel
{
        Display << "Customers";
        Application.Customers();
}
void InventoryLabel()
{
        Display << "Inventory";
        Application.Inventory();
}
void EmployeeLabel()
{
        Display << "Employee";
```

```
        Application.Employee();
}
void Logout()
{
        Display << “Logout”;
        Application.User_Login();
}
```

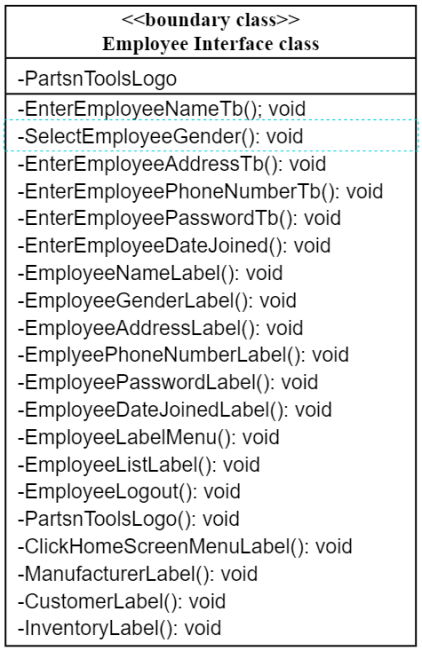



```
Transactions()
{
        ShowInventory();
        DisplayTransactions();
        SelectCustomerNumber()
}
void SelectCustomerNumber()
{
        open Customer_Connection();
        Select * from CustomerTableData;
        procedure get(CustomerTableData);
```

```
        display << get(CustomerTableData);
        end procedure;
        close Customer_Connection();
}
void DisplayInventory()
{
        open Transaction_Connection();
        Select * from InventoryTableData;
        display << InventoryTableData;
        close Transaction_Connection();
}
void AddTransactionBill()
{
        if(ItemQuantity == " ")
        {
                display << Enter Quantity Please";
        }
        else
        {
                display << TransactionTableData;
                UpdateQuantity();

        }
}
void ShowEmployeeName()
{
        open Transaction_Connection();
        Select * from TransactionTableData;
        Display << EmployeeName;
        Close Transaction_Connection();
```

```
}
void PrintReceipt();
{
	print << AddBill();
}
void ItemListGridData()
{
	set EnterItemNameTb.Text to InventoryGridData.rows[0].cells[1];
	set ItemQuantity to InventoryGridData.rows[0].cells[2];
	set EnterItemPriceTb.Text to InventoryGridData.rows[0].cells[3];
	if(EnterItemNameTb.Text == " ")
	{
			DataKey = 0;
	}
	Else
	{
		DataKey = InventoryGridData.rows[0].cells[0];
	}
}
void Logout()
{
	Application.User_Login();
}
```



void **EnterEmployeeNameTb**()

{

display << Textbox;

}

void **SelectEmployeeGender**()

{

display << Listbox;

}

void **EnterEmpoyeeAddressTb**()

{

display << Textbox;

}

void **EnterEmployeePhoneNumberTb**()

{

display << Textbox;

}

void **EnterEmployeePasswordTb()**

{

```
        display << Textbox;
}
void EnterEmployeeDateJoinedTb()
{
        display << Date;
}
void EmployeeNameLabel()
{
        display << “Employee Name”;
}
void EmployeeGenderLabel()
{
        display << “Gender”;
}
void EmployeeAddressLabel()
{
        display << “Address”;
}
void EmployeePhoneNumberLabel()
{
        display << “Phone Number”;
}
void EmployeePasswordLabel()
{
        display << “Password”;
}
void EmployeeDateJoinedLabel()
{
        display << “Date Joined”;
}
void EmployeeLabelMenu()
```

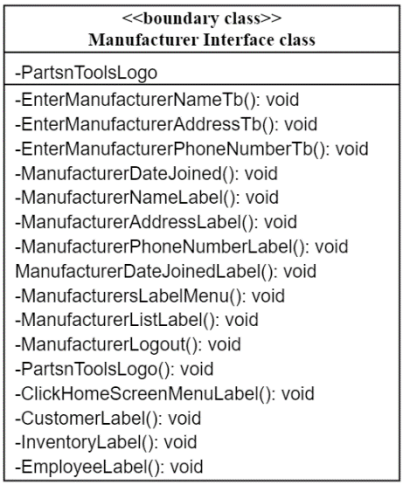
```
{
        display << “Manage Employees”;
}
void EmployeeListLabel()
{
        display << “Employee List”;
}
void EmployeeLogout()
{
        Application.User_Login();
}
void PartsnToolsLogo()
{
        display << PartsnToolsLogo;
}
void ClickHomeScreenMenuLabel()
{
        Display << “Home Screen Menu”;
        HomeScreenMenu();
}
void ManufacturerLabel
{
        display << “Manufacturer”;
        Application.Manufacturer();
}
void CustomerLabel
{
        Display << “Customers”;
        Application.Customers();
}
void InventoryLabel()
```

{

Display << “Inventory”;

Application.Inventory();

}



void **EnterManufacturerNameTb()**

{

display << Textbox;

}

void **EnterManufacturerAddressTb()**

{

display << Textbox;

}

void **EnterManufacturerPhoneNumberTb()**

{

display << Textbox;

}

void **ManufacturerDateJoined()**

{

display << Date;

}

void **ManufacturerNameLabel()**

```
{
        display << “Manufacturer Name”;
}
void ManufacturerAddressLabel()
{
        display << “Address”;
}
void ManufacturerPhoneNumberLabel()
{
        display << “Phone Number”;
}
void ManufacturerDateJoinedLabel()
{
        display << “Date Joined”;
}
void ManufacturersLabelMenu()
{
        display << “Manage Manufacturers”;
}
void ManufacturerListLabel()
{
        display << “List of Manufacturers”;
}
void ManufacturerLogout()
{
        Application.User_Login();
}
void PartsnToolsLogo()
{
        display << PartsnToolsLogo;
}
```

void **ClickHomeScreenMenuLabel()**

{

Display << “Home Screen Menu”;

Application.HomeScreenMenu();

}

void **CustomerLabel**

{

Display << “Customers”;

Application.Customers();

}

void **InventoryLabel**()

{

Display << “Inventory”;

Application.Inventory();

}

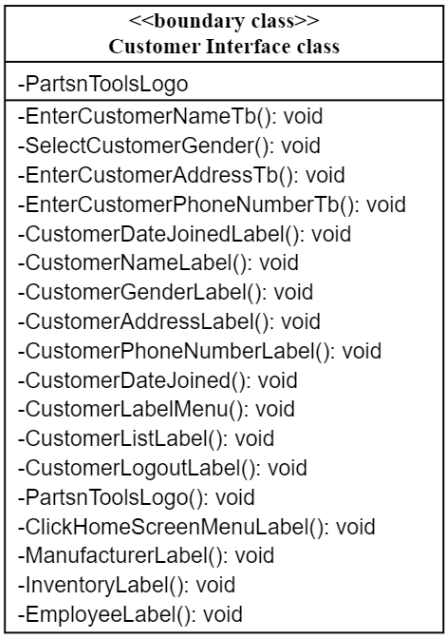
void **EmployeeLabel**()

{

Display << “Employee”;

Application.Employee();

}



```
void EnterCustomerNameTb()
{
	display << Textbox;
}
void SelectCustomerGender()
{
	display << Listbox;
}
void EnterCustomerAddressTb()
{
	display << Textbox;
}
void EnterCostumerPhoneNumberTb()
{
	display << Textbox;
}
void CustomerDateJoinedLabel()
{
	display << “Date Joined”;
}
void CustomerNameLabel()
{
	display << “Customer Name”;
}
void CustomerGenderLabel()
{
	display << “Gender”;
}
void CustomerAddressLabel()
{
	display << “Address”;
```

```
}
void CustomerPhoneNumberLabel()
{
        display << “Phone Number”;
}
void CustomerDateJoined()
{
        display << Date;
}
void CustomerLabelMenu()
{
        display << “Manage Customer”;
}
void CustomerListLabel()
{
        display << “Customer List”;
}
void CustomerLogoutLabel()
{
        Application.User_Login();
}
void PartsnToolsLogo()
{
        display << PartsnToolsLogo;
}
void ClickHomeScreenMenuLabel()
{
        display << “Home Screen Menu”;
        HomeScreenMenu();
}
void ManufacturerLabel
```

{

display << “Manufacturer”;

Application.Manufacturer();

}

void **InventoryLabel**()

{

Display << “Inventory”;

Application.Inventory();

}

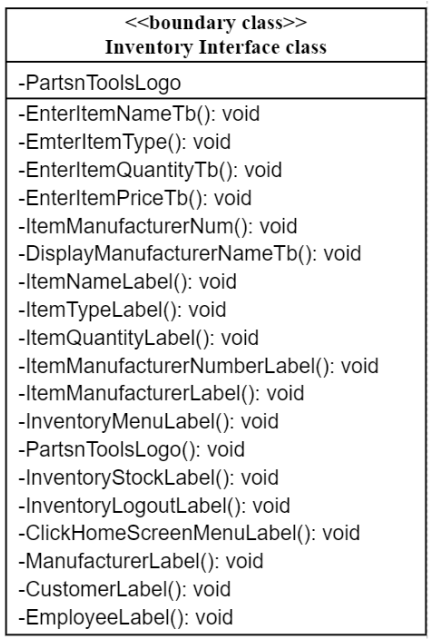
void **EmployeeLabel**()

{

Display << “Employee”;

Application.Employee();

}



void **EnterItemNameTb()**

{

display << Textbox;

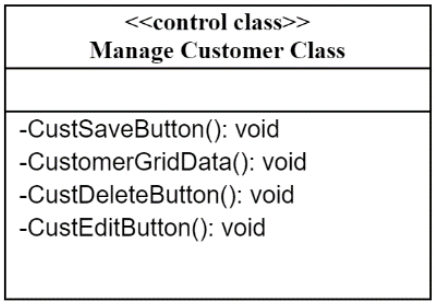
}

```
void EnterItemType()
{
	display << Listbox;
}
void EnterItemQuantityTb()
{
	display << Textbox;
}
void EnterItemPriceTb()
{
	display << Textbox;
}
void ItemManufacturerNum()
{
	display << ListBox;
}
void DisplayManufacturerNameTb()
{
	display << Textbox;
}
void ItemNameLabel()
{
	display << “Item Name”;
}
void ItemTypeLabel()
{
	display << “Item Type”;
}
void ItemQuantityLabel()
{
	display << “Item Quantity”;
```

```
}
void ItemManufacturerNumberLabel()
{
        display << “Item Manufacturer Number;”
}
void ItemManufacturerLabel()
{
        display << “Item Manufacturer Label”;
}
void InventoryMenuLabel()
{
        display << “Manage Inventory”;
}
void PartsnToolsLogo()
{
        display << PartsnToolsLogo;
}
void InventoryStockLabel()
{
        display << “Inventory Stock”;
}
void InventoryLogoutLabel()
{
        Application.User_Login();
}
void ClickHomeScreenMenuLabel()
{
        Display << “Home Screen Menu”;
        HomeScreenMenu();
}
void ManufacturerLabel
```

```
{
        display << “Manufacturer”;
        Application.Manufacturer();
}
void CustomerLabel
{
        Display << “Customers”;
        Application.Customers();
}
void EmployeeLabel()
{
        Display << “Employee”;
        Application.Employee();
}
```

Control Classes: Admin



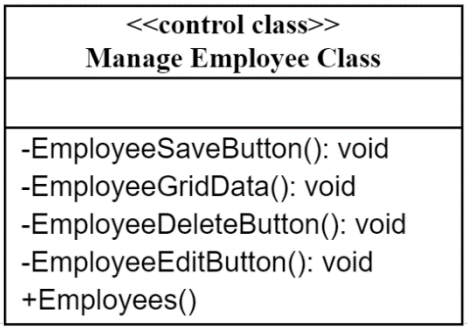

```
void CustSaveButton()
{
        if (EnterCustomerNameTb.Text == “” || EnterCustomerPhoneTb.Text == “ ” ||
        EnterCustomerAddressTb.Text == “ ” || SelectCustomerGenderCb.SelectedIndex == -1)
        {
                display << “Need missing info”;
        }
        else
```

```
        {
            Try
            {
                open Customer_Connection();
                insert into CustomerTableData;
                procedure Save(CustNumber, CustName, CustPhone, CustAddress,
                CustDateJoin,CustGender);
                display << "Customer saved";
                end procedure;
                close Custom_Connection();
                ShowCustomers();
                DataReset();
            }
            catch(exception)
            {
                display << exception;
            }
        }
}
        Set integer DataKey to 0;
Void CustomerGridData()
{
        set EnterCustomerNameTb.Text to CustomerGridData.Rows[0].Cells[1];
        set EnterCustomerPhoneNumberTb.Text to CustomerGridData.Rows[0].Cells[2];
        set EnterCustomerAddressTb.Text to CustomerGridData.Rows[0].Cells[3];
        set CustomerDateJoined.Text to CustomerGridData.Rows[0].Cells[4];
        set SelectCustomerGenderCb.SelectedItem to CustomerGridData.Rows[0].Cells[5];

        if(EnterCustomerNameTb.Text == " ")
        {
            set DataKey to 0
```

```
        }
        else
        {
                set DataKey to convert(CustomerGridData.rows[0].cells[0]);
        }
}
void CustDeleteButton();
{
        if(DataKey == 0)
        {
                display << “Choose Customer First”;
        }
        else
        {
                try
                {
                        open Customer_Connection();
                        Delete from CustomerTableData;
                        procedure Delete(CustNumber, CustName, CustPhone, CustAddress,
                        CustDateJoin,CustGender);
                        display << “Customer Deleted”;
                        end procedure;
                        Close Customer_Connection();
                        ShowCustomers();
                        DataReset();
                }
                Catch(exception)
                {
                        Display << execption;
                }
        }
```

```
}
void CustEditButton()
{
        if (EnterCustomerNameTb.Text == “” || EnterCustomerPhoneTb.Text == “ ” ||
        EnterCustomerAddressTb.Text == “ ” || SelectCustomerGenderCb.SelectedIndex == -1)
        {
                display << “Need missing info”;
        }
        else
        {
                try
                {
                        open Custom_Connection();
                        Update CustomerTableData;
                        procedure Update(CustNumber, CustName, CustPhone, CustAddress,
                        CustDateJoin,CustGender);
                        display << “Customer Updated”;
                        end procedure;
                        close Custom_Connection();
                        ShowCustomers();
                        DataReset();
                }
        }
}
```



void **EmployeeSaveButton**()

{

if (EnterEmployeeNameTb.Text == “” || EnterEmployeePhoneTb.Text == “ ” ||
EnterEmployeeAddressTb.Text == “ ”|| EmterEmployeePasswordTb.Text == “ ” ||
SelectEmployeeGenderCb.SelectedIndex == -1)

{

display << “Need missing info”;

}

else

{

try

{

open EmployeeData_Connection();

insert into EmployeeTableData;

procedure Save(EmployeeNumber, EmployeeName, EmployeePhone,
EmployeeAddress, EmployeeDateJoin, EmployeeGender,
EmployeePassword);

display << “Employee Saved”;

end procedure;

close EmployeeData_Connection();

ShowEmployees();

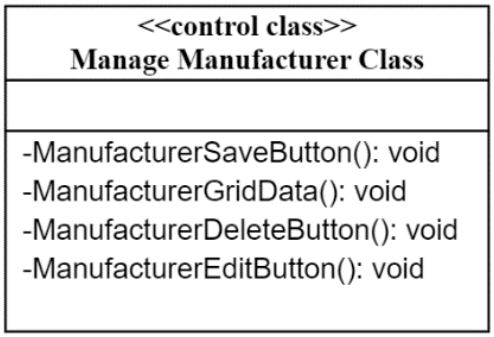
DataReset();

}

```
		catch(exception)
		{
			display << exception;
		}
}
set integer DataKey to 0;
void EmployeeGridData()
{
	set EnterEmployeeNameTb.Text to EmployeeGridData.Rows[0].Cells[1];
	set EnterEmployeePhoneNumberTb.Text to EmployeeGridData.Rows[0].Cells[2];
	set EnterEmployeeAddressTb.Text to EmployeeGridData.Rows[0].Cells[3];
	set EmployeeDateJoined.Text to EmployeeGridData.Rows[0].Cells[4];
	set SelectEmployeeGenderCb.SelectedItem to EmployeeGridData.Rows[0].Cells[5];
	set EnterEmployeePasswordTb.Text to EmployeeGridData
	if(EnterCustomerNameTb.Text == " ")
	{
		set DataKey to 0;
	}
	else
	{
		set DataKey to convert(EmployeeGridData.rows[0].cells[0]);
	}
}
void EmployeeDeleteButton()
{
	if(DataKey == 0)
	{
		display << "Choose Employee First";
	}
	else
	{
```

```
            try
            {
                open EmployeeData_Connection();
                Delete from EmployeeTableData;
                procedure Delete(EmployeeNumber, EmployeeName, EmployeePhone,
                EmployeeAddress, EmployeeDateJoin, EmployeeGender,
                EmployeePassword);
                display << “Employee Deleted”;
                end procedure;
                Close EmployeeData_Connection();
                ShowEmployees();
                DataReset();
            }
            catch(exception)
            {
                Display << execption;
            }
        }
}
void EmployeeEditButton()
{
        if (EnterEmployeeNameTb.Text == “” || EnterEmployeePhoneTb.Text == “ ” ||
        EnterEmployeeAddressTb.Text == “ ”|| EmterEmployeePasswordTb.Text == “ ” ||
        SelectEmployeeGenderCb.SelectedIndex == -1)
        {
            display << “Need missing info”;
        }
        else
        {
            try
            {
```

```
                open EmployeeData_Connection();
                Update EmployeeTableData;
                procedure Update(EmployeeNumber, EmployeeName, EmployeePhone,
                EmployeeAddress, EmployeeDateJoin, EmployeeGender,
                EmployeePassword);
                display << “Employee Updated”;
                end procedure;
                close EmployeeData_Connection();
                ShowEmployees();
                DataReset();
            }
            catch(exception)
            {
                display << execption;
            }
        }
}
```



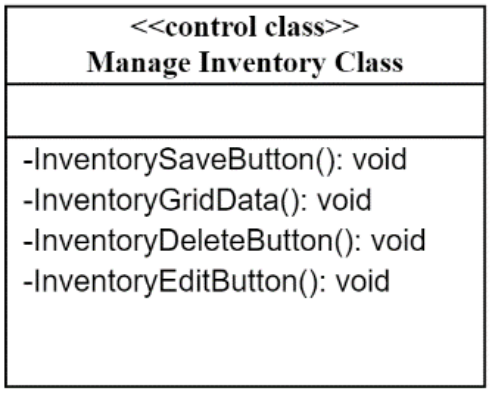

```
void ManufacturerSaveButton()
{
    if (EnterManufacturerNameTb.Text == " " || EnterManufacturerPhoneTb.Text == " " ||
    EnterManufacturerAddressTb.Text == " ")
    {
        display << "Need missing info";
    }
    else
    {
        try
        {
            open ManufacturerData_Connection();
            insert into ManufacturerTableData;
            procedure Save(ManufacturerNumber, ManufacturerName,
            ManufacturerPhone, ManufacturerAddress, ManufacturerDateJoin);
            display << "Manufacturer Saved";
            end procedure;
            close ManufacturerData_Connection();
            ShowManufacturers();
            DataReset();
        }
        catch(exception)
        {
            display << exception;
```

```
                }
        }
}
set integer DataKey to 0;

void ManufacturerGridData()
{
        set EnterManufacturerNameTb.Text to ManufacturerGridData.Rows[0].Cells[1];
        set EnterManufacturerPhoneNumberTb.Text to ManufacturerGridData.Rows[0].Cells[2];
        set EnterManufacturerAddressTb.Text to ManufacturerGridData.Rows[0].Cells[3];
        set ManufacturerDateJoined.Text to ManufacturerGridData.Rows[0].Cells[4];
        if(EnterManufacturerNameTb.Text == " ")
        {
                set DataKey to 0;
        }
        else
        {
                set DataKey to convert(ManufacturerGridData.rows[0].cells[0]);
        }
}
void ManufacturerDeleteButton()
{
        If(DataKey == 0)
        {
                display << "Choose Manufacturer First";
        }
        else
        {
                try
                {
                        open ManufacturerData_Connection();
```

```
                    Delete from ManufacturerTableData;
                    procedure Delete(ManufacturerNumber, ManufacturerName,
                    ManufacturerPhone, ManufacturerAddress, ManufacturerDateJoin);
                    display << “Manufacturer Deleted”;
                    end procedure;
                    close ManufacturerData_Connection();
                    ShowManufacturers();
                    DataReset();
                }
                catch(exception)
                {
                    Display << execption;
                }
        }
}
void ManufacturerEditButton()
{
        if (EnterManufacturerNameTb.Text == “” || EnterManufacturerPhoneTb.Text == “ ” ||
        EnterManufacturerAddressTb.Text == “ ”)
        {
                display << “Need missing info”;
        }
        else
        {
                try
                {
                    open ManufacturerData_Connection();
                    Update ManufacturerTableData;
                    procedure Update(ManufacturerNumber, ManufacturerName,
                    ManufacturerPhone, ManufacturerAddress, ManufacturerDateJoin);
                    display << “Manufacturer Updated”;
```

```
                    end procedure;
                    close ManufacturerData_Connection();
                    ShowManufacturers();
                    DataReset();
                }
                catch(exception)
                {
                    display << execption;
                }
        }
}
```

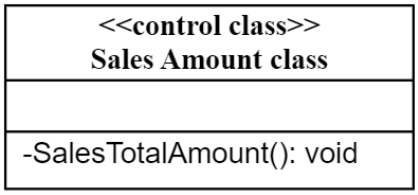



```
void InventorySaveButton()
{
        if (EnterItemNameTb.Text == " " || EnterItemPriceTb.Text == " " ||
        EnterTypeCb.SelectedIndex == -1 || ItemManufacturerNumber.Text == "" ||
        EnterItemQuantity.Text == " ")
        {
            display << "Need missing info";
        }
        else
        {
            try
            {
```

```
                    open InventoryData_Connection();
                    insert into InventoryTableData;
                    procedure Save(ItemNumber, ItemName, ItemType, ItemQuantity,
                    ItemPrice, ItemManufacturerNumber, ItemManufacturerName);
                    display << “Item Saved”;
                    end procedure;
                    close InventoryData_Connection();
                    ShowInventory();
                    DataReset();
                }
                catch(exception)
                {
                    display << exception;
                }
        }
}
void InventoryGridData()
{
        set EnterItemNameTb.Text to InventoryGridData.Rows[0].Cells[1];
        set EnterItemTypeCb.SelectedItem to InventoryGridData.Rows[0].Cells[2];
        set EnterItemQuantityTb.Text to InventoryGridData.Rows[0].Cells[3];
        set EnterItemPriceTb.Text to InventoryGridData.Rows[0].Cells[4];
        set ItemManufacturerNumberCb to InventoryGridData.Rows[0].Cells[5];
        set ItemManufacturerNameTb.Text to InventoryGridData.Rows[0].Cells[6];
                if(EnterItemNameTb.Text == “ ”)
                {
                    set DataKey to 0;
                }
                else
                {
                    set DataKey to convert(InventoryGridData.rows[0].cells[0]);
```

```
                }
}
void InventoryDeleteButton()
{
        if(DataKey == 0)
        {
                display << “Choose Item First”;
        }
        else
        {
                try
                {
                        open InventoryData_Connection();
                        Delete from InventoryTableData;
                        procedure Delete(ItemNumber, ItemName, ItemType, ItemQuantity,
                        ItemPrice, ItemManufacturerNumber, ItemManufacturerName);
                        display << “Manufacturer Deleted”;
                        end procedure;
                        close InventoryData_Connection();
                        ShowInventory();
                        DataReset();
                }
        }
                catch(exception)
                {
                        display << execption;
                }
        }
}
void InventoryEditButton()
{
```

```
        if (EnterItemNameTb.Text == " " || EnterItemPriceTb.Text == " " ||
        EnterTypeCb.SelectedIndex == -1 || ItemManufacturerNumber.Text == "" ||
        EnterItemQuantity.Text == " ")
        {
                display << "Need missing info";
        }
        else
        {
                try
                {
                        open InventoryData_Connection();
                        Update InventoryTableData;
                        procedure Update(ItemNumber, ItemName, ItemType, ItemQuantity,
                        ItemPrice, ItemManufacturerNumber, ItemManufacturerName);
                        display << "Inventory Updated";
                        end procedure;
                        close InventoryData_Connection();
                        ShowInventory();
                        DataReset();
                }
                catch(exception)
                {
                        Display << execption;
                }
        }
}
```



void **SalesTotalAmount**();

{

open Transaction_Connection();

Select Sum(Billings) from TransactionTableData;

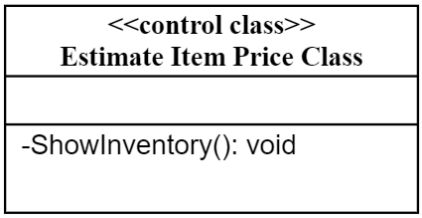
procedure Sum(Billings)

display << Sum(Billings,)

end procedure;

close Transaction_Connection();

}



void **ShowInventory()**

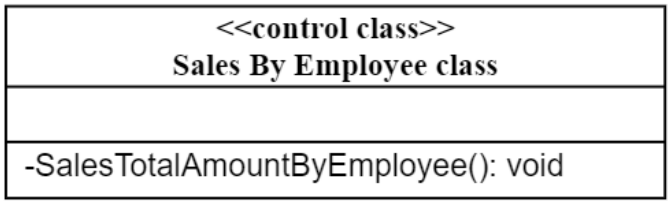
{

open InventoryData_Connection();

select * from InventoryTableData;

close InventoryData_Connection();

}



void **SalesTotalAmountByEmployee**()

{

open Transaction_Connection();

Select Sum(Billings, EmployeeName) from TransactionTableData;

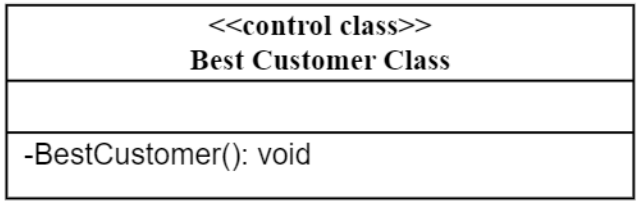
procedure Sum(Billings, EmployeeName);

Display << Sum(Billings, EmployeeName);

end procedure;

close Transaction_Connection();

}



void **BestCustomer**()

{

try

{

open Transaction_Connection();

Select Max(Billings, CustomerName) from TranscationTableData;

procedure Max(Billings, CustomerName);

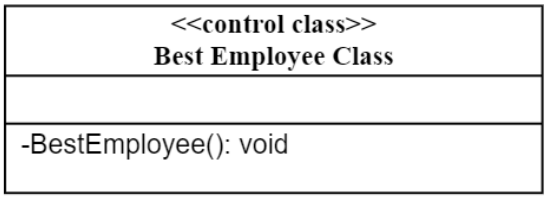
display << Max(Billings, CustomerName);

end procedure;

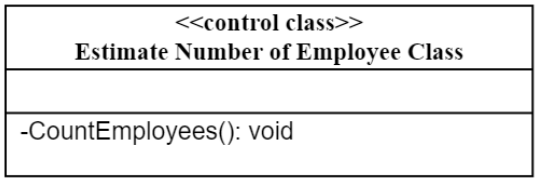
close Transaction_Connection();

}

```
        catch (exception)
        {
                close Transaction_Connection();
        }
}
```




```
void BestEmployee()
{
        try
        {
                open Transaction_Connection();
                Select Max(Billings, EmployeeName) from TransactionTableData;
                procedure Max(Billings, EmployeeName);
                Display << Max(Billings, EmployeeName);
                end procedure;
                close Transaction_Connection();
        }
        catch (exception)
        {
                close Transaction_Connection();
        }
}
```



void **CountEmployees**()

{

open EmployeeData_Connection();

Select Count(*) from EmployeeTableData;

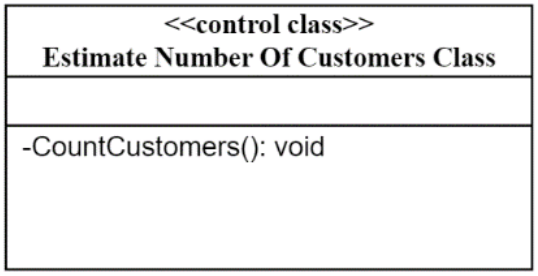
procedure Count(EmployeeTableData);

display << Count(EmployeeTableData);

end procedure;

close EmployeeData_Connection();

}



void **CountCustomers**()

{

open Customer_Connection();

Select Count(*) from CustomerTableData;

procedure Count(CustomerTableData);

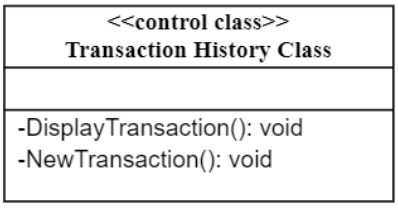
display << Count(CustomerTableData);

end procedure;

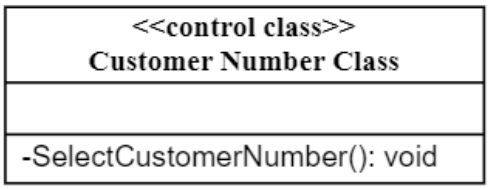
close Customer_Connection();

}

Control Classes: User




```
void DisplayTransactions()
{
        open Transaction_Connection();
        Select * from TransactionTableData;
        display << TransactionTableData;
        close Transaction_Connection();
}
void NewTransaction()
{
        try
        {
                open Transaction_Connection();
                insert into TransactionsTableData;
                procedure save(EmployeeName, CustNumber, CustName, CustDateJoin);
                display << save(EmployeeName, CustNumber, CustName, CustDateJoin);
                end procedure;
                close Transaction_Connection();
                DisplayTransactions();
        }
        catch (exception)
        {
                display << exception;
        }
}
```



void **SelectCustomerNumber()**

{

open Customer_Connection();

Select CustomerNumber from CustomerTableData;

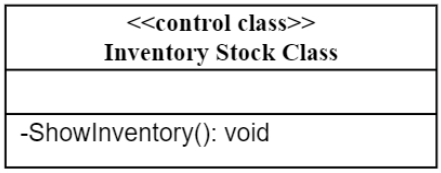
procedure Select(EmployeeName);

display << EmployeeName;

end procedure;

close Customer_Connection();

}



void **ShowInventory()**

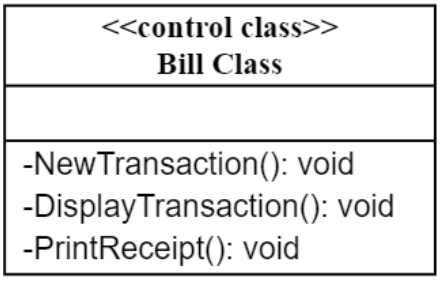
{

open InventoryData_Connection();

select * from InventoryTableData;

close InventoryData_Connection();

}



void **NewTransaction**()

```
{
        try
        {
                open Transaction_Connection();
                insert into TransactionsTableData;
                procedure save(EmployeeName, CustNumber, CustName, CustDateJoin);
                display << save(EmployeeName, CustNumber, CustName, CustDateJoin);
                end procedure;
                close Transaction_Connection();
                DisplayTransactions();
        }
        catch (exception)
        {
                display << exception;
        }
}
```

void **DisplayTransactions**()

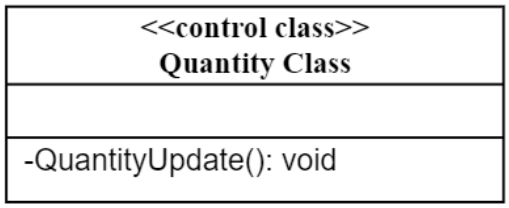
```
{
        open Transaction_Connection();
        Select * from TransactionTableData;
        display << TransactionTableData;
        close Transaction_Connection();
        PrintReceipt();
}
```

void **PrintReceipt**()

{

print << DisplayTransactions;

}



void **QuantityUpdate**()

{

try

{

open Customer_Connection();

Update InventoryTableData;

procedure Update(InventoryTableData);

display << Update(InventoryTableData);

end procedure;

close Customer_Connection();

ShowInventory();

}

catch (exception)

{

display << exception;

}

}

VIII. Implementation

Finding the right programming language was a difficult decision for our group. Most group members only know how to use C++. When we chose our project, we took notice that it will take a long time to create such a program due to having a difficult time in finding solutions using C++. We were willing to use C++, but we had to use another language for portability and for efficiency. Instead, we decided to use C# with SQL in Visual Studio.

For the programming practices we applied for this project is to simplify the code. We named variables differently for every function to make the code easier to read. For some of the window forms in the program, we use the same algorithm to make the code be consistent for the specific window's functions. However, we change variables to connect to a data to a specific table in the SQL database to make the code easier to read.

Our team had to reuse some code from another project. Realistically, we cannot start from scratch since some team members were unable to fully contribute to the project due to work, other schoolwork, and for personal reasons. There was little time to complete this project and we needed to make something as fast as possible. We believe that every component can be reusable for any project with some little modifications, due to the portability of C#. Which we consider our project to be portable.

The version control system (VCS) we used is GitHub. GitHub is one of the most used VCS and we believed that it will be useful to make changes in our project. However, the only downside is that it is somewhat complex to upload and share projects. Some of our members did not like how we had to use GitBash and Command prompt since it was confusing to upload changes in our project. For others, it was the first time learning how to use GitHub as it was difficult to upload projects, rather than just dragging and dropping the project in the browser. What we like about GitHub is that it is hard to lose a previous version of a project. We also like how well it is integrated with Visual Studio. The alternatively, we could just upload the files in zip by email, but it could be less secure due to the risk of losing files.

IX. Test Cases and Results

Test Case: 1

Test Scenario: Testing the security for the user to input a username and password in the user log in screen.

Test Steps:

- User enters wrong user and password.
- User clicks log in.
- Message box will appear stating “Wrong Username or Password.”
- User leaves username and password blank.
- User clicks log in.
- Message box will appear stating “Please enter both Username and Password.”
- User enters correct username and password.
- User will be prompted to a transactions screen.

Prerequisites: A registered account for the software with a username and password.

IDE: Visual Studio 2022

Expected Results: The user login is an authentication window. If user enters incorrect username and password, they will be prevented from accessing the user contents from the software.

Actual Results: As expected.

Test Status- Pass/Fail: Pass

Test Case: 2

Test scenario: Testing the functionality of the “Transaction” window after user logs in.

Test Steps:

- User selects customer number.
- Customer name is displayed below in a textbox below “Customer Name” label.
- Customer selects an item through a grid known as “Inventory Stock”.
- Item name is displayed below in a textbox below “Item Selected” label.
- Item price is displayed in a textbox below “Cost” label.
- User clicks on “Add to Bill” button.
- Message box will appear stating “Enter Correct Quantity”.
- User inputs any quantity.
- Message box will appear stating “Item Updated”.
- Item is added to a bill.
- Customer clicks on “Print” button.

- The software will display a receipt for user and the option to print it.
- The software will add transaction to a grid to display transaction history for the user.
- User clicks “Log out” label.
- User is prompted to the login screen.

Prerequisites: A registered user account is required to access the contents of this window.

IDE: Visual Studio 2022

Expected Results: The Transactions window is a window exclusively for user accounts. The software should run the software flawlessly and calculating the correct price for items added to bill.

Actual Results: Inputting letters in the “Quantity” textbox causes the software to crash. If the user forgets to choose “Customer #” then the sales will not update in the history of transactions window.

Test Status- Pass/Fail: Pass with minor errors.

Test Case: 3

Test scenario: Testing the security for the admin to input a password in the user log in screen.

Test Steps:

- Admin enters wrong password.
- Admin click log in.
- Message box will appear stating “Wrong Admin password.”
- Admin enters no password.
- Message box will appear stating “Please enter a password.”
- Admin enters correct password.
- Admin will be prompted to the home screen menu of the software.

Prerequisites: Admin will be required to enter a unique password.

IDE: Visual Studio 2022

Expected Results: The admin login is an authentication window. If the admin enters the incorrect password, the admin will be prevented from accessing the admin contents on the software.

Actual Results: As expected

Test Status- Pass/Fail: Pass

Test Case: 4

Test scenario: Testing the functionality of the interface of the "HomeScreenMenu" window.

Test Steps:

- "HomeScreenMenu" window will display a summary of a store's status.
- "HomeScreenMenu" window will display the best seller based on the transactions a user makes.
- "HomeScreenMenu" window will display the best customer based on the transactions a user makes.
- "HomeScreenMenu" window will display the number of different types of items in the inventory.
- "HomeScreenMenu" window will display the number of customers that have an account.
- "HomeScreenMenu" window will display the total number of sales.
- "HomeScreenMenu" window will display the total number of sales by employee.
- The admin choses a different employee.
- "HomeScreenMenu" window will display the total number of sales from the chosen employee.
- Admin chooses other menus in the software.
- Admin is prompted to the chosen menu in the software.

Prerequisites: Admin must enter the correct password to get access to this menu.

IDE: Visual Studio 2022

Expected Results: The software should be able to display the correct information for the admin.

Actual Results: The software displays the updated price only if the user choses "Customer #" in the "Transactions" windows. If user forgets to change "Customer #" in the transactions window, then the software will not update the total number of sales from the chosen employee.

Test Status- Pass/Fail: Pass with small errors.

Test Case: 5

Test scenario: Testing the functionality of the interface "Inventory" window.

Test steps:

- “Inventory” window will display a grid with the list of items in the inventory.
- Admin inputs information in every textbox in the “Inventory” interface window.
- Admin clicks save.
- Message box will display “Item Added” notifying the admin that an item is added.
- Item is added in the “Inventory List” grid.
- Admin clicks on an item in the “Inventory List” grid.
- Admin changes the information in the textbox.
- Admin clicks on edit.
- Message box displays “Inventory Updated” notifying the admin that item has been updated.
- The chosen edited item is updated in the “Inventory List” grid.
- Admin clicks on an item in the “Inventory List” grid.
- Admin clicks “Delete” button.
- Message box displays “Item Deleted” notifying the admin that an item is deleted.
- The chosen item is deleted off the “Inventory List” grid.

Prerequisites: Admin must enter the correct password to get access to this menu.

IDE: Visual Studio 2022

Expected Results: The software should be able to add, edit, or delete information. The software should be able to display the edited information for the admin.

Actual Results: The software successfully displays correct information. However, the software crashes when inputting letters in the cost of items textbox and in the .

Test Status- Pass/Fail: Pass with small errors.

Test Case: 6

Test scenario: Testing the functionality of the interface “Customers” window.

Test steps:

- Admin clicks on “Customers” label.
- Admin inputs information in every textbox in the “Customer” interface window.
- Admin clicks save.

- Message box will display “Customer Added” notifying the admin that a customer is added.
- Customer is added in the “Customer List” grid.
- Admin clicks on a customer in the “Customer List” grid.
- The chosen customer info is displayed in the textboxes.
- Admin edits the displayed information in the textboxes.
- Admin clicks on edit.
- Message box will display “Customer Updated” notifying the admin that the chosen customer info is edited.
- The chosen customer info is edited in the “Customer List” grid.
- Admin clicks on a customer on the “Customer List” grid.
- The chosen customer info is displayed in the textboxes.
- Admin clicks on delete button.
- Message box will display “Customer Deleted” notifying the admin that the chosen customer info is deleted.
- The chosen customer is removed off the “Customer List” grid.

Prerequisites: Admin must enter the correct password to get access to this menu.

IDE: Visual Studio 2022

Expected Results: The software should allow admin to add, edit, or delete information. The software should be able to display the edited information for the admin.

Actual Results: As expected.

Test Status- Pass/Fail: Pass

Test Case: 7

Test scenario: Testing the functionality of the interface “Manufacturers” window.

Test steps:

- Admin clicks on “Manufacturers” label.
- Admin inputs information in every textbox in the “Manufacturers” interface window.
- Admin clicks save.

- Message box will display “Manufacturer Added” notifying the admin that a Manufacturer is added.
- Admin clicks on a manufacturer in the “Manufacturer List” grid.
- The chosen manufacturer info is displayed in the textboxes.
- Admin edits the displayed information in the textboxes.
- Admin clicks on edit.
- Message box will display “Manufacturer Updated” notifying the admin that the chosen manufacturer info is edited.
- The chosen manufacturer info is edited in the “Manufacturer List” grid.
- Admin clicks on a manufacturer on the “Manufacturer List” grid.
- The chosen Manufacturer info is displayed in the textboxes.
- Admin clicks on delete button.
- Message box will display “Manufacturer Deleted” notifying the admin that the chosen manufacturer info is deleted.
- The chosen manufacturer is removed off the “Manufacturer List” grid.

Prerequisites: Admin must enter the correct password to get access to this menu.

IDE: Visual Studio 2022

Expected Results: The software should allow admin to add, edit, or delete information. The software should be able to display the edited information for the admin.

Actual Results: As expected.

Test Status- Pass/Fail: Pass

Test Case: 8

Test scenario: Testing the functionality of the interface “Employees” window.

Test steps:

- Admin clicks on “Employees” label.
- Admin inputs information in every textbox in the “Employees” interface window.
- Admin clicks save.

- Message box will display “Employee Added” notifying the admin that an employee is added.
- Adding employees also adds accounts for user logins.
- Admin clicks on an employee in the “Employee List” grid.
- The chosen employee info is displayed in the textboxes.
- Admin edits the displayed information in the textboxes.
- Admin clicks on edit.
- Message box will display “Employee Updated” notifying the admin that the chosen employee info is edited.
- The chosen employee info is edited in the “Employee List” grid.
- Admin clicks on an employee on the “Employee List” grid.
- The chosen employee info is displayed in the textboxes.
- Admin clicks on delete button.
- Message box will display “Employee Deleted” notifying the admin that the chosen Employee info is deleted.
- The chosen Employee is removed off the “Employee List” grid.

Prerequisites: Admin must enter the correct password to get access to this menu.

IDE: Visual Studio 2022

Expected Results: The software should allow admin to add, edit, or delete information. The software should be able to display the edited information for the admin.

Actual Results: As expected.

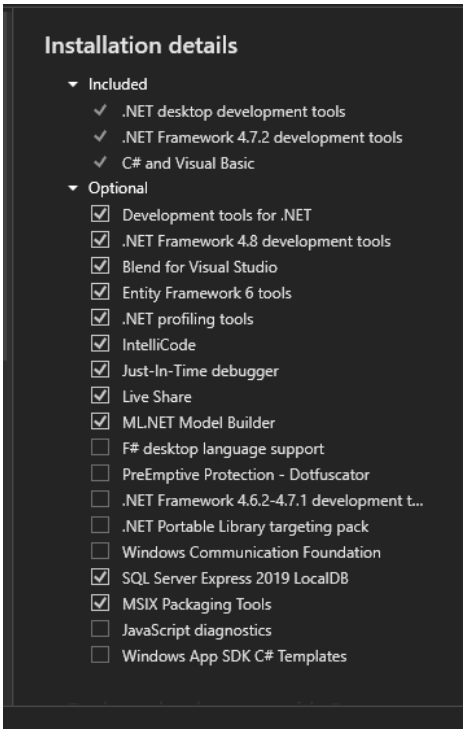
Test Status- Pass/Fail: Pass

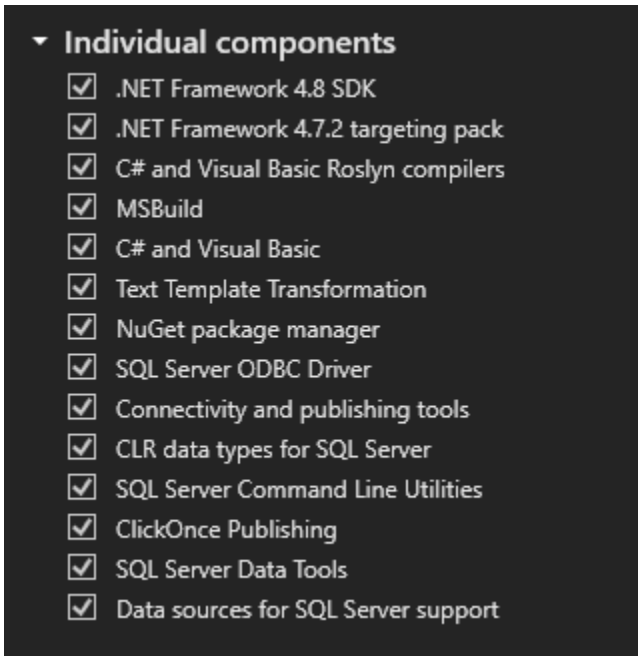
X. User Documentation

Software materials used:

Windows 10

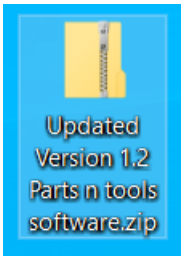
Visual Studio Community 2022



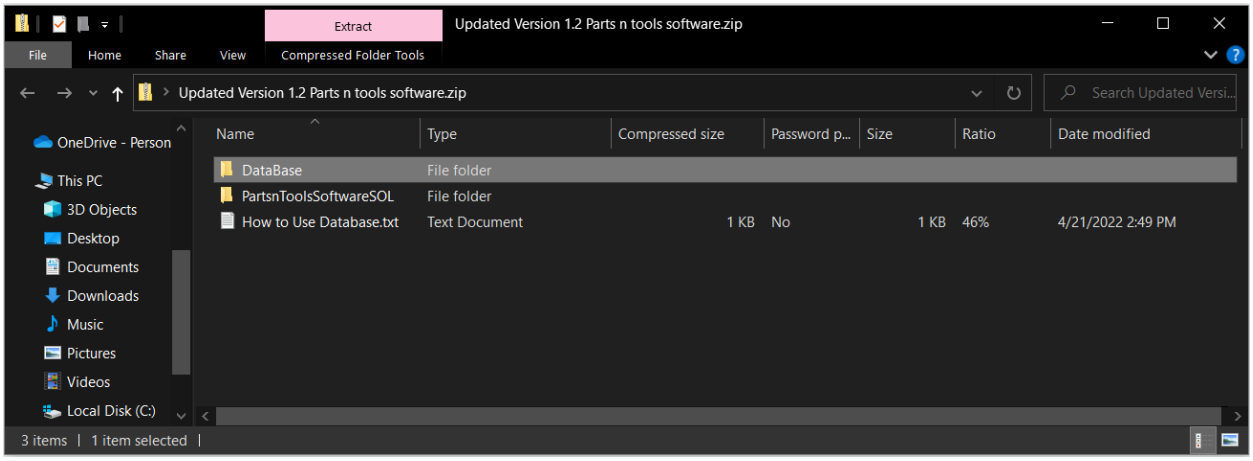


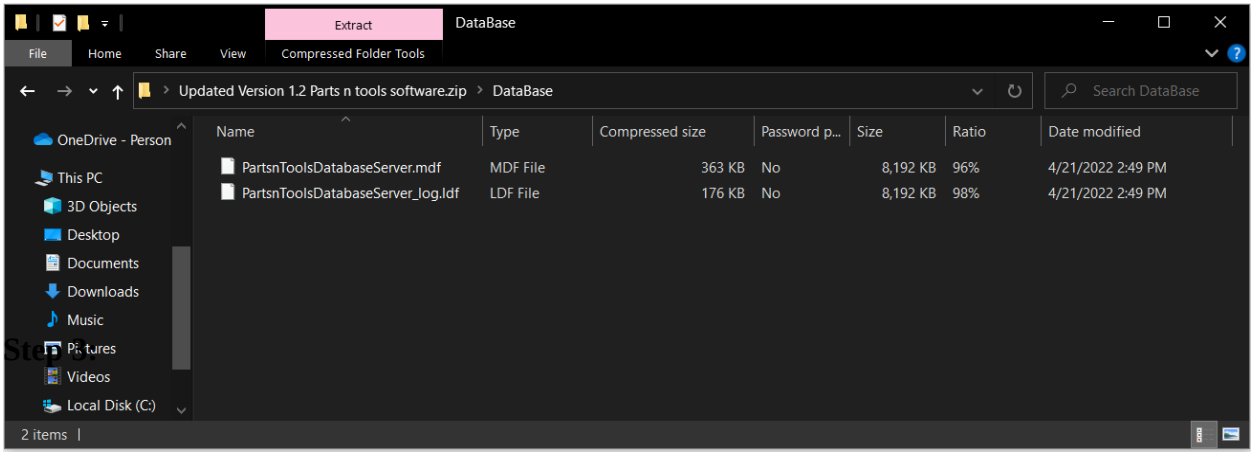
How to run:

Step 1: Click on the file “PartsnTools Ver 1.2”.



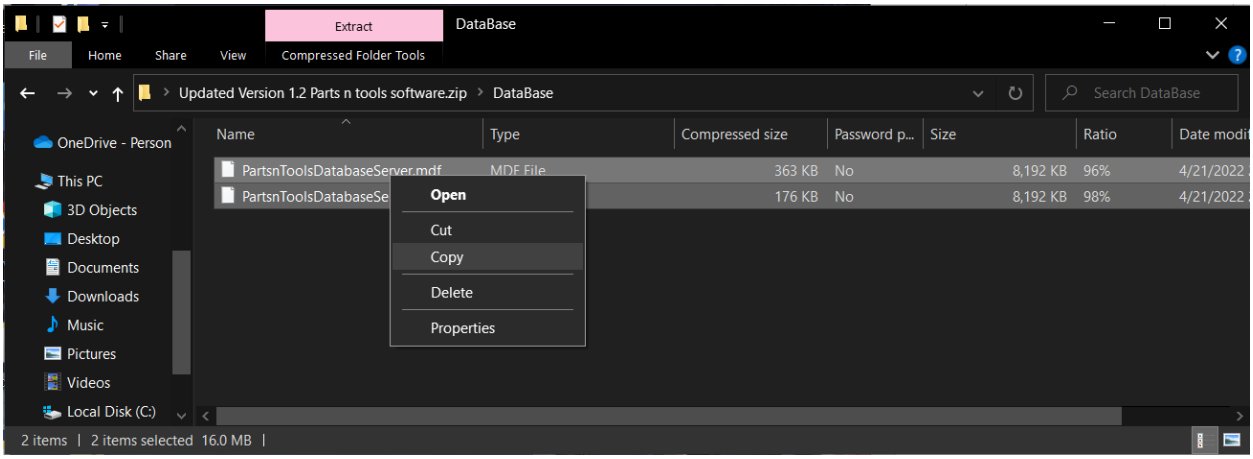
Step 2: Click on the “DataBase” file. (Will include a note pad of instructions on how to use the database).

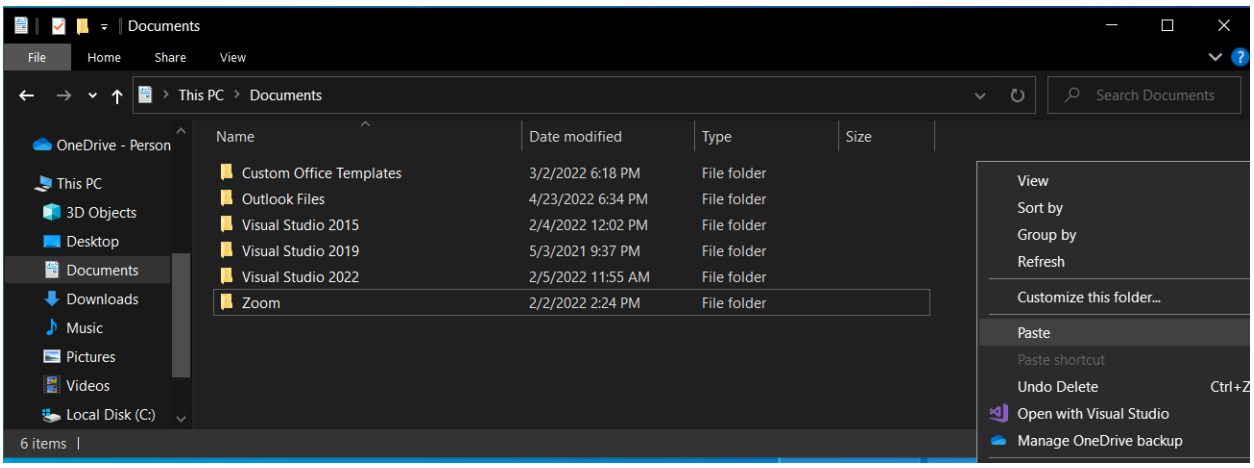


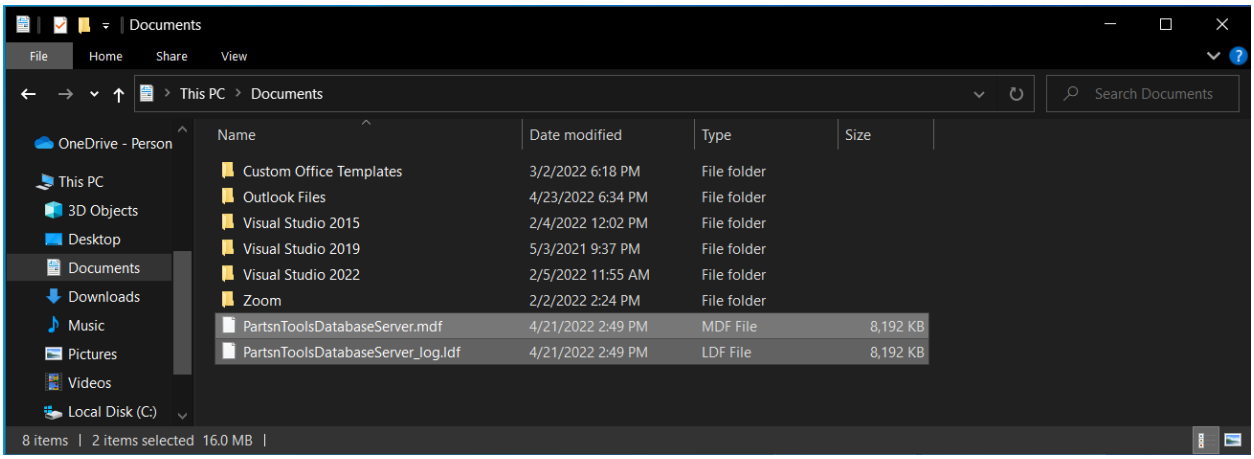


Step 3

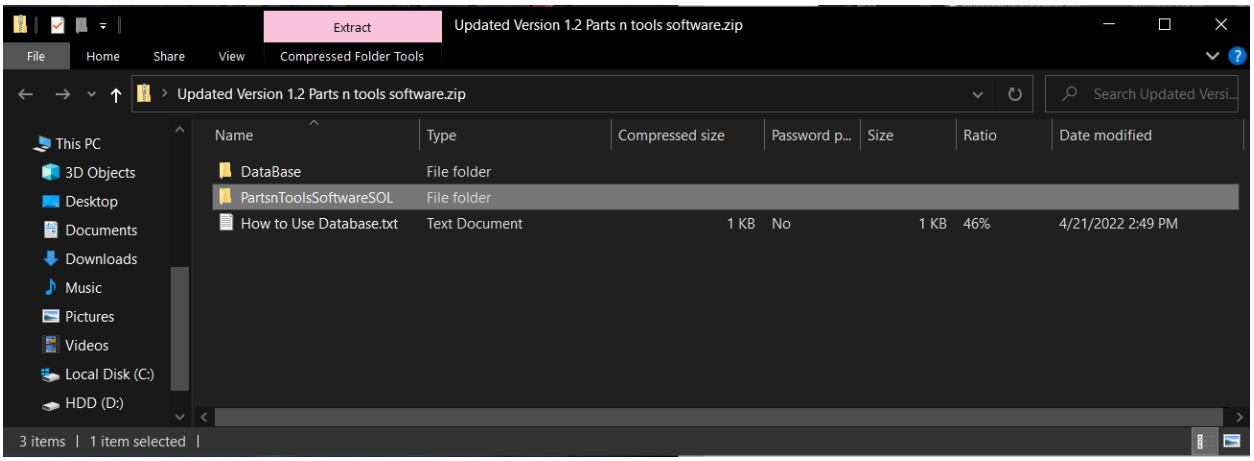
Step 3: Copy and paste both files into the documents file.

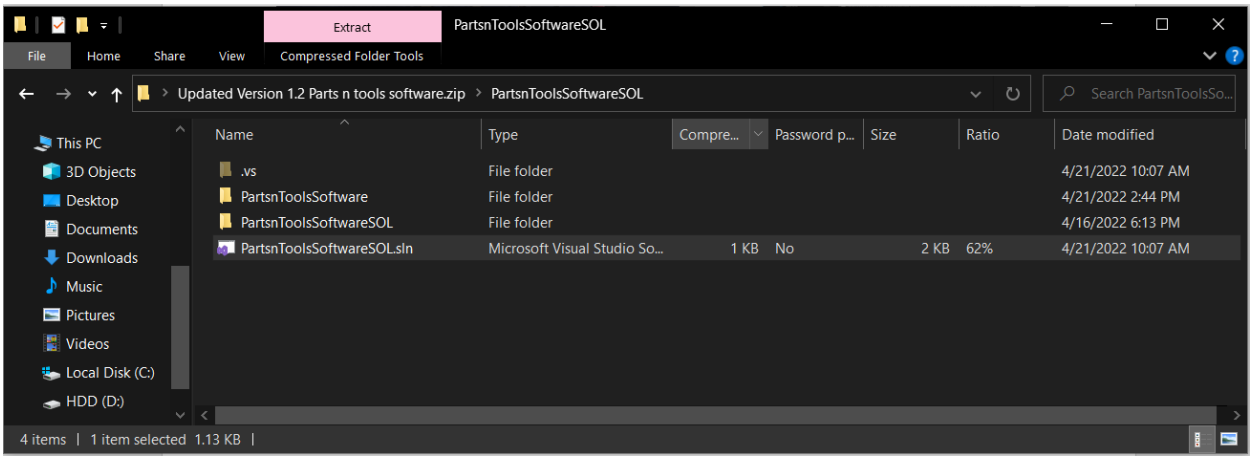


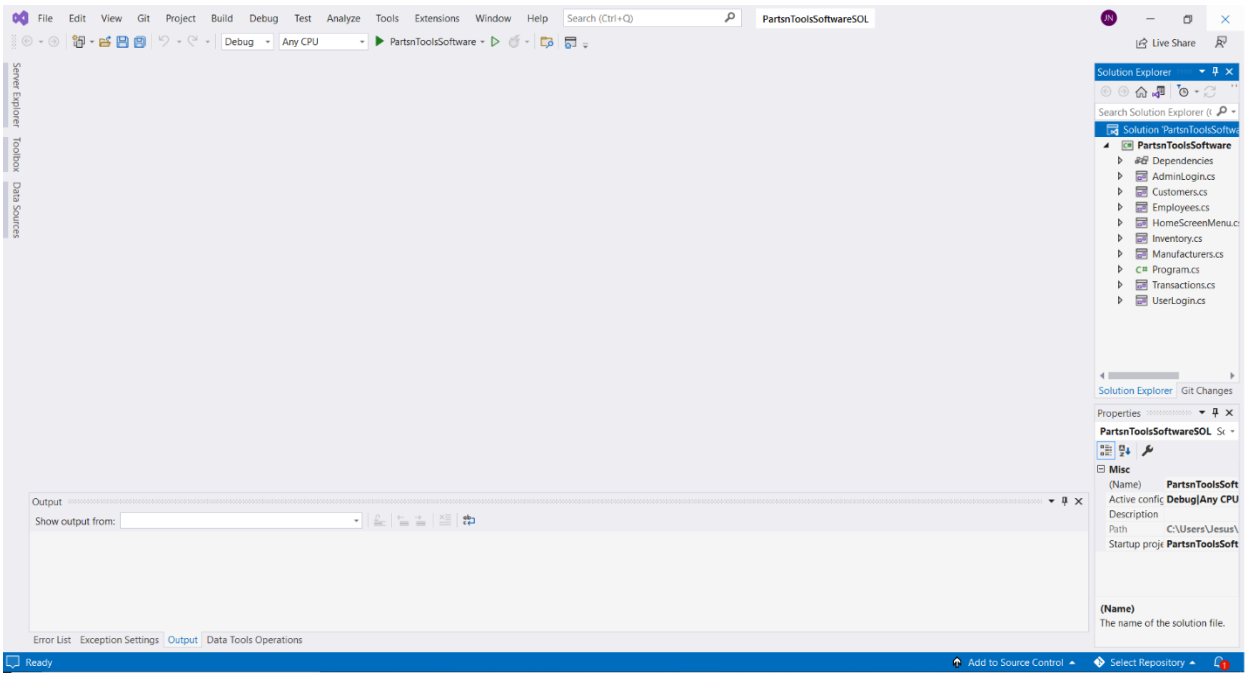

Documents
File
Home
Share
View
This PC
Documents
Search Documents
OneDrive - Person
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Name
Date modified
Type
Size
Custom Office Templates 3/2/2022 6:18 PM File folder
Outlook Files 4/23/2022 6:34 PM File folder
Visual Studio 2015 2/4/2022 12:02 PM File folder
Visual Studio 2019 5/3/2021 9:37 PM File folder
Visual Studio 2022 2/5/2022 11:55 AM File folder
Zoom 2/2/2022 2:24 PM File folder
View
Sort by
Group by
Refresh
Customize this folder...
Paste
Paste shortcut
Undo Delete Ctrl+Z
Open with Visual Studio
Manage OneDrive backup
6 items


Documents
File
Home
Share
View
This PC
Documents
Search Documents
OneDrive - Person
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Name
Date modified
Type
Size
Custom Office Templates 3/2/2022 6:18 PM File folder
Outlook Files 4/23/2022 6:34 PM File folder
Visual Studio 2015 2/4/2022 12:02 PM File folder
Visual Studio 2019 5/3/2021 9:37 PM File folder
Visual Studio 2022 2/5/2022 11:55 AM File folder
Zoom 2/2/2022 2:24 PM File folder
PartsnToolsDatabaseServer.mdf 4/21/2022 2:49 PM MDF File 8,192 KB
PartsnToolsDatabaseServer_log.ldf 4/21/2022 2:49 PM LDF File 8,192 KB
8 items | 2 items selected 16.0 MB

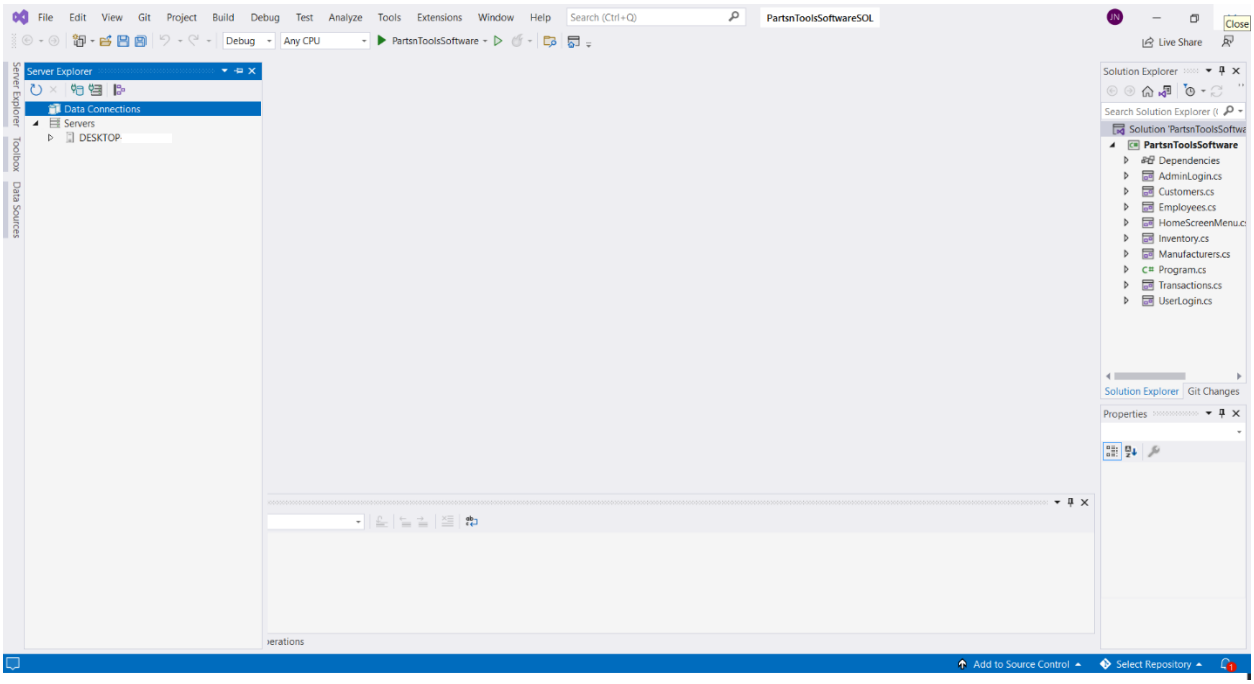
Step 4: Open “PartsnToolsSoftwareSOL” file and click on the “PartsnToolsSoftwareSOL.sln” solution.





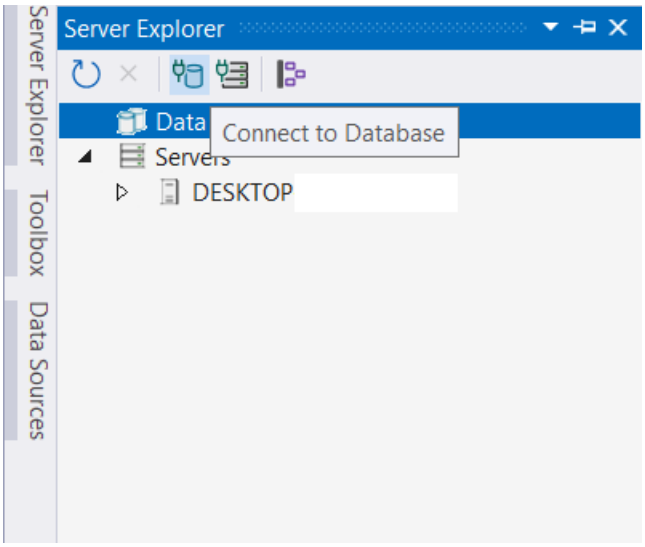


Step 5: Click on “Server Explorer”.

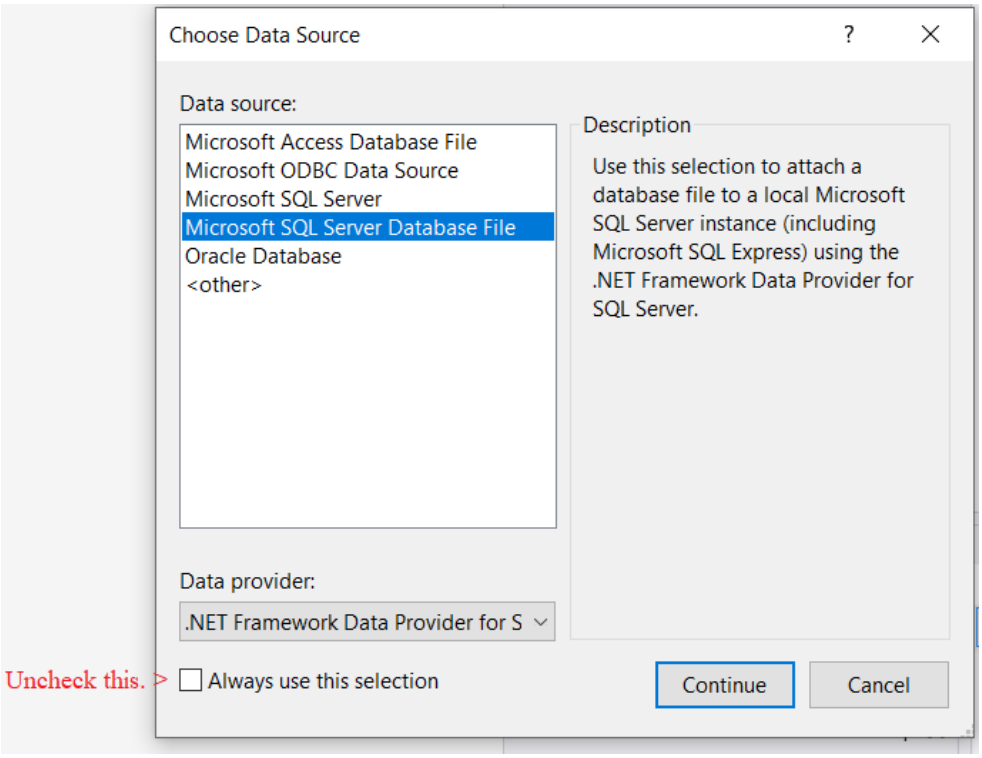


Step 6: Click on “Connect to Database”

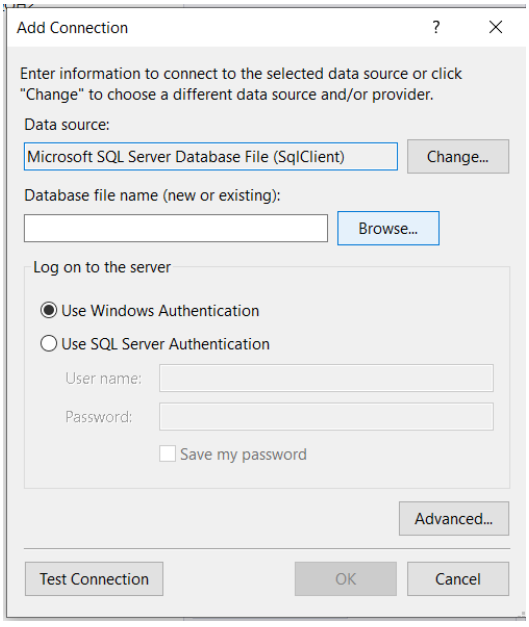
Connect to Database icon: 

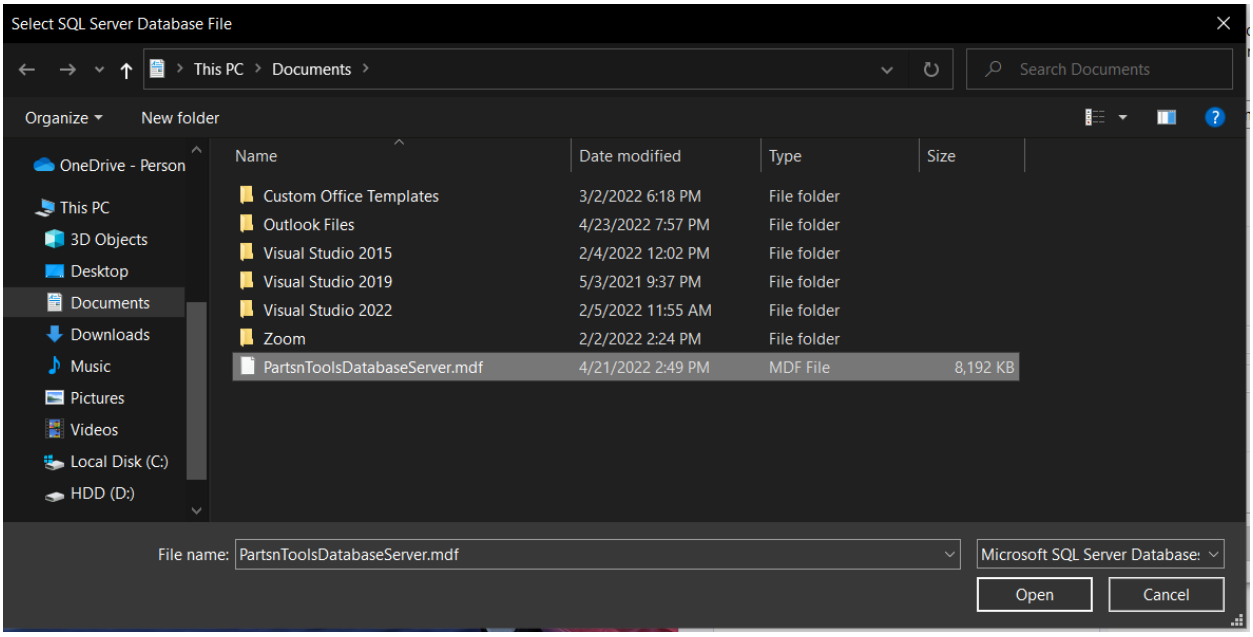


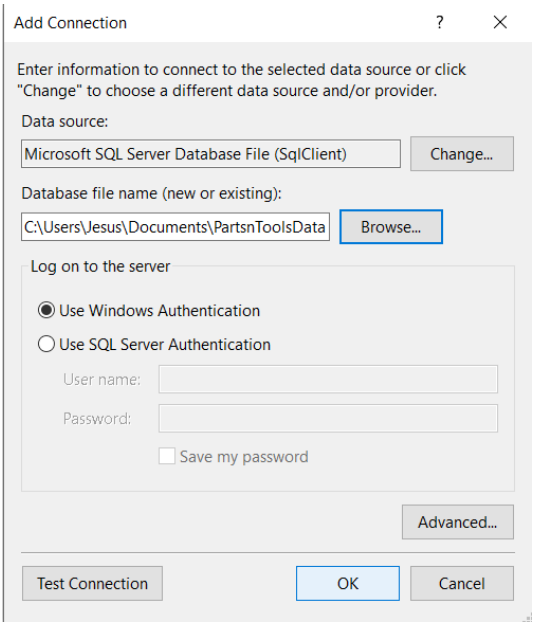
Step 7: Select “Microsoft SQL Server Database File”, uncheck the “Always use this selection” box and hit continue.



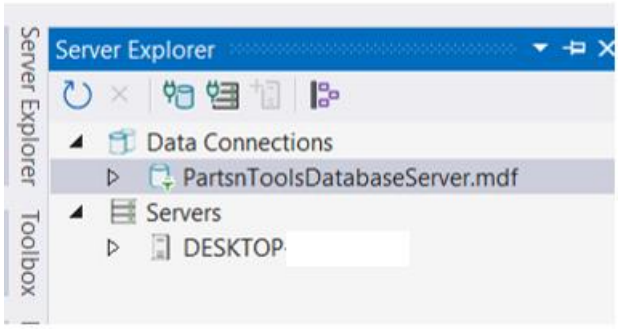
Step 8: Click on “browse” and then find the database in the documents folder. Then click on “Open” and then click ok.



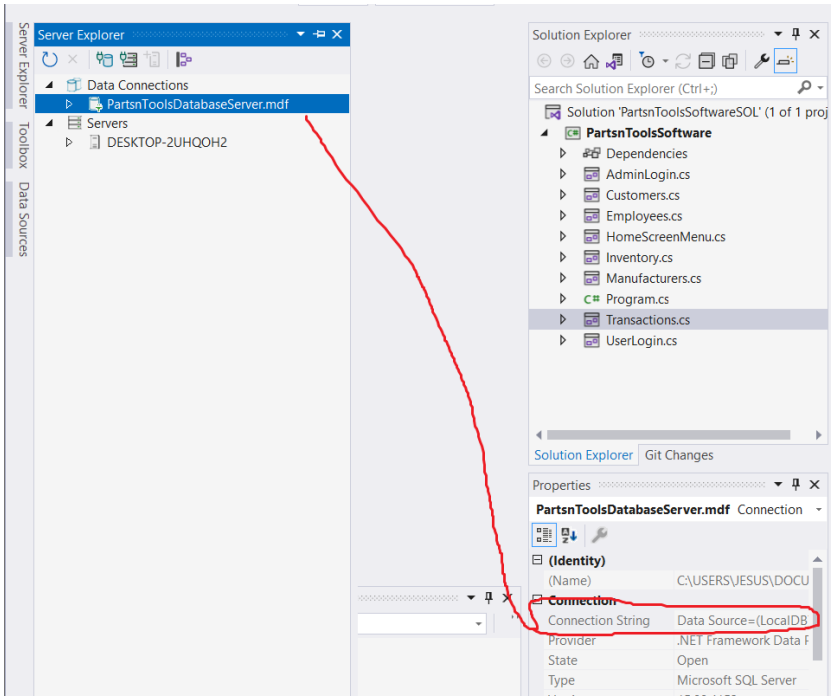




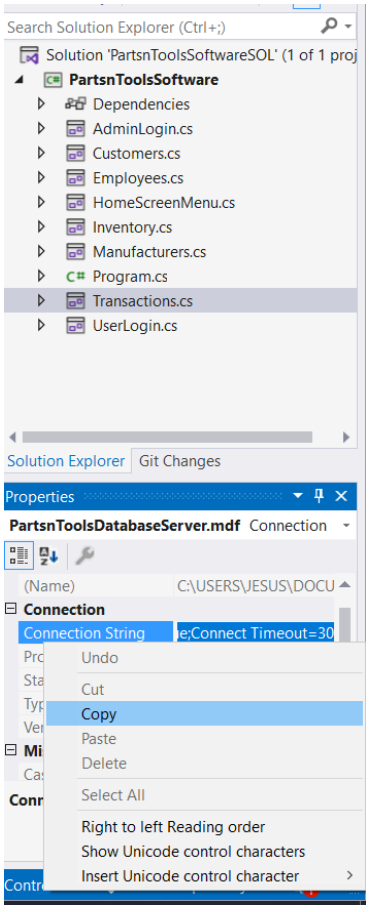
Step 9: Database connection should appear in “Server Explorer”.



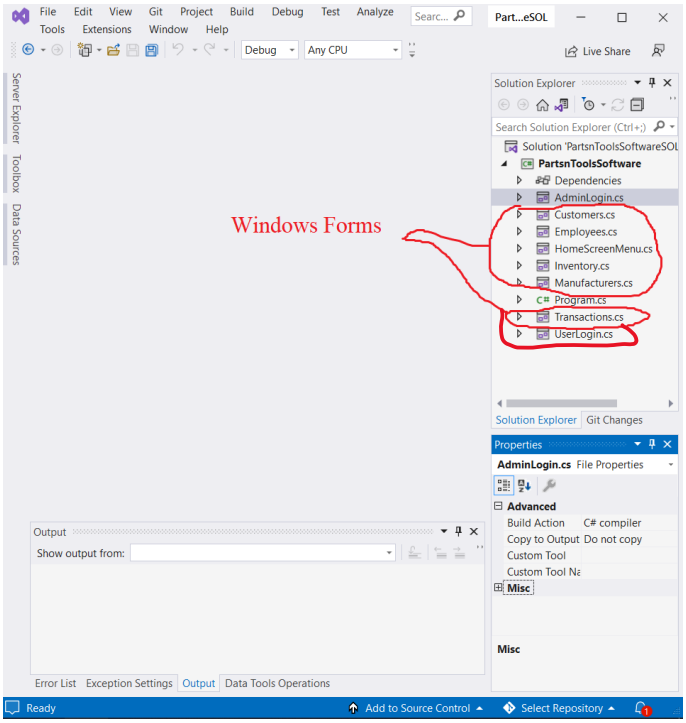
Step 10: Click on the “PartsnToolsDatabaseSever.mdf” in “Server Explorer” and then look for the “Connection String” in properties.



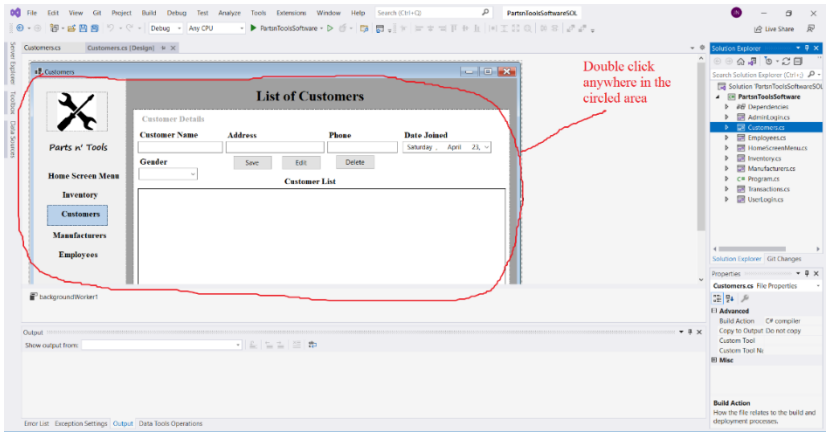
Step 11: Copy the “Connection String”.

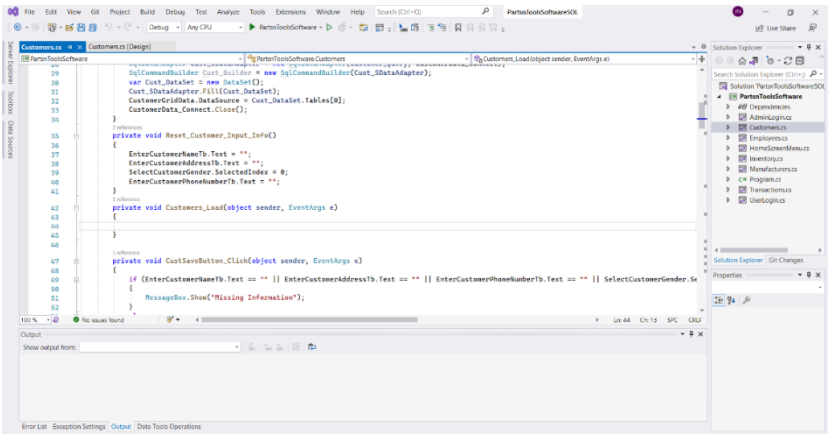


Step 12: Double click on one of the following circled windows forms.

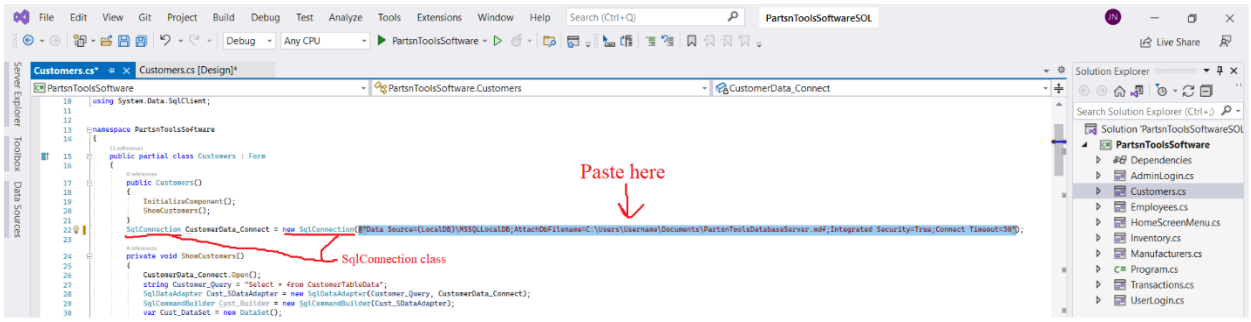


Step13: Double click on any spot of the window to get access of the coding.

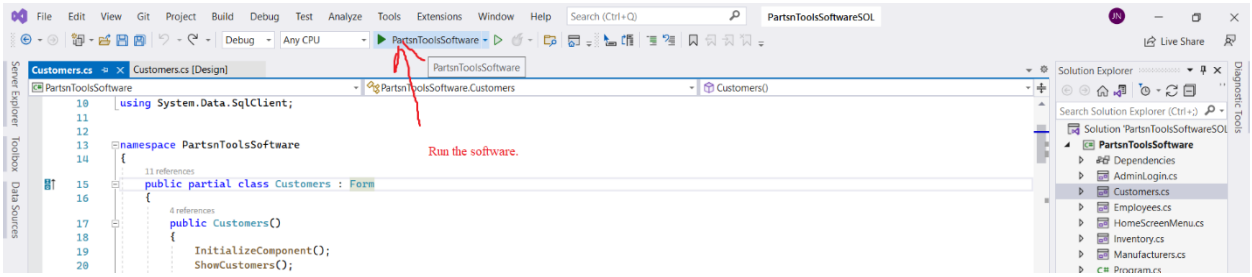




Step 14: Paste the copied “Connection String” on every “SqlConnection” class for the following windows forms: “UserLogin.cs”, “Customer.cs”, “Employees.cs”, “HomeScreenMenu.cs”, “Inventory.cs”, “Manufacturer.cs”, and “Transactions.cs”.



Step 15: Set up is done, run the software.

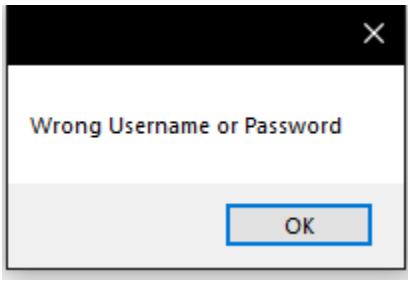


Running the software

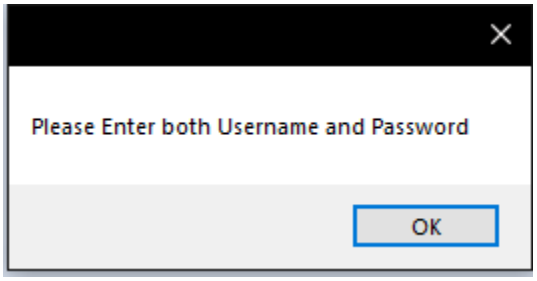
Authentication security

User Login: When running the software, the user/admin will be prompted to a “User Login” screen. The “UserLogin” is for users only. The “Admin” label login is for administrators only.

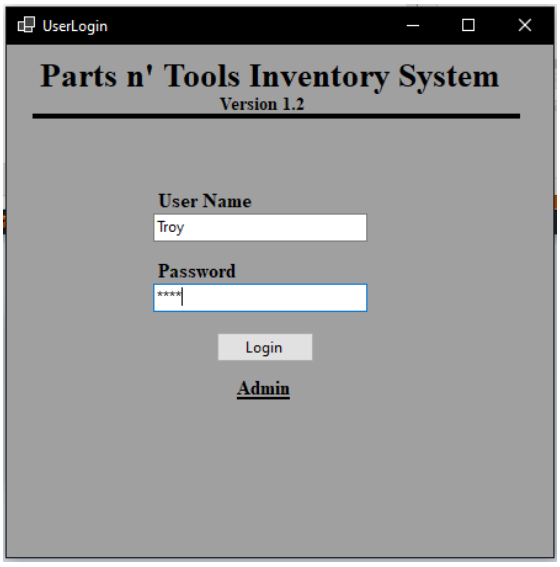
Users will be unable to login to access user contents when entering the wrong password. A message box will notify the user that a wrong username or password was entered.



Likewise, if a user enters no username or password, then a message box will notify the user to enter a username and password.

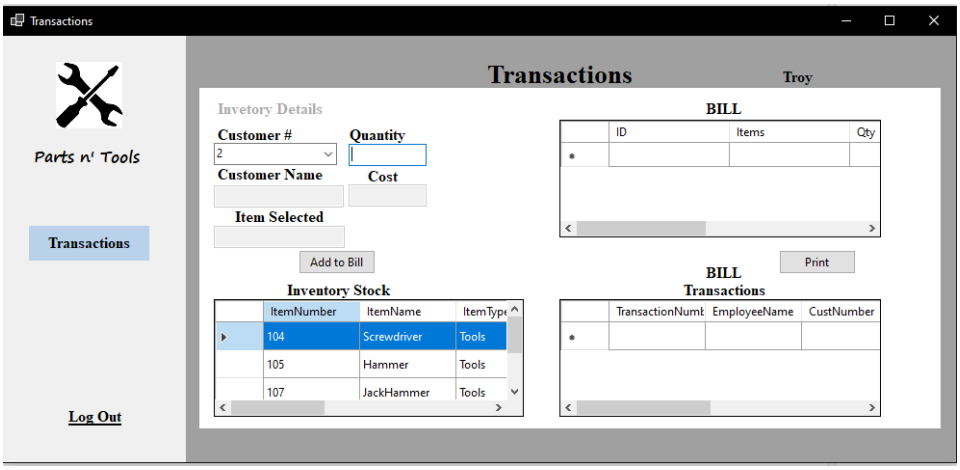


When the user inputs a password, the letters will be hidden and replaced by “*”.

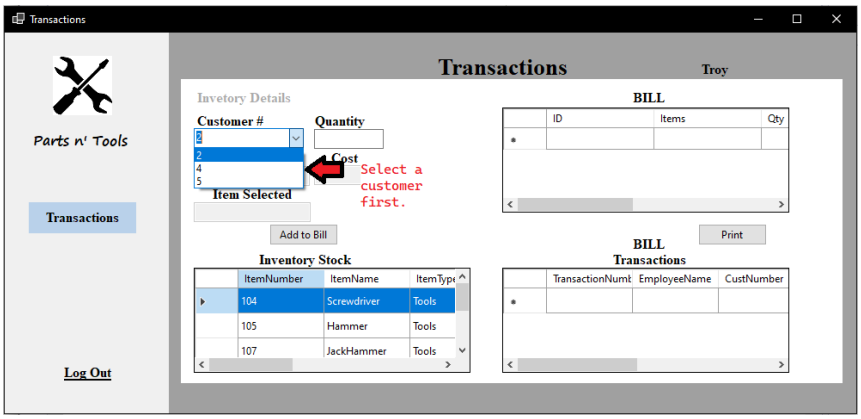


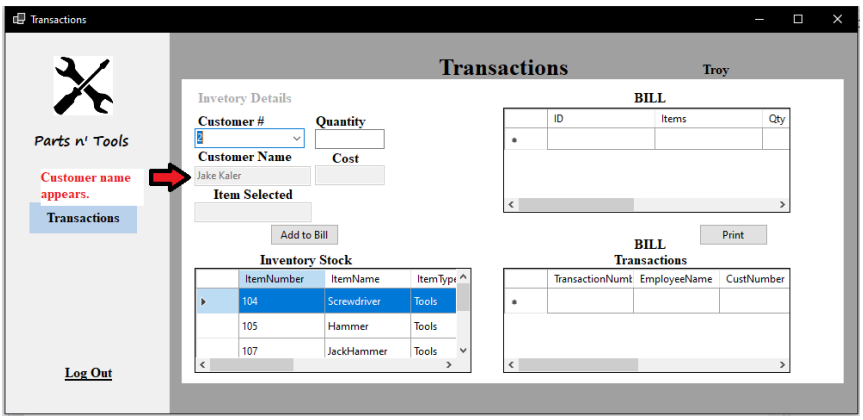
User software contents

Transactions: After a user has successfully entered the correct username and password, they will be prompted to a transactions window. Which is exclusively for user accounts. To run this window properly, here are the steps to follow.

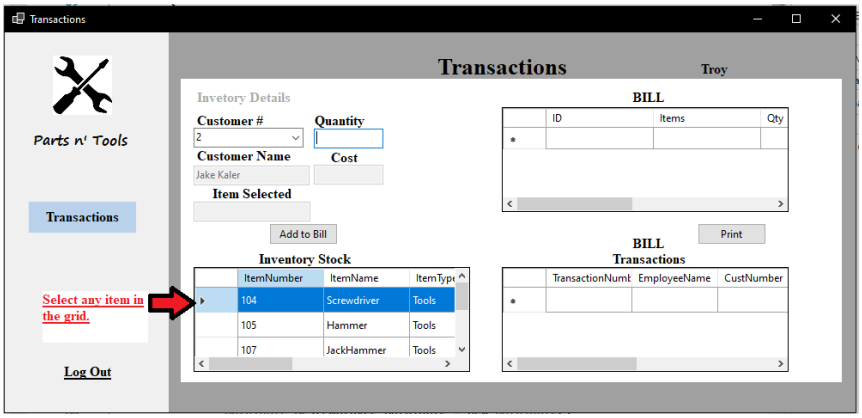


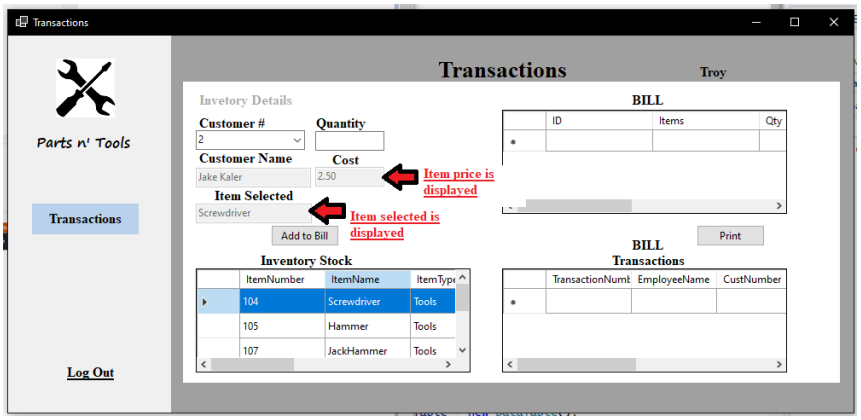
Step 1: User must choose a customer number below the "Customer #" label in the software. A customer's name will appear under the "Customer Name" textbox. (**Warning:** If no customer is selected, it will not process the transactions history for the employee.)



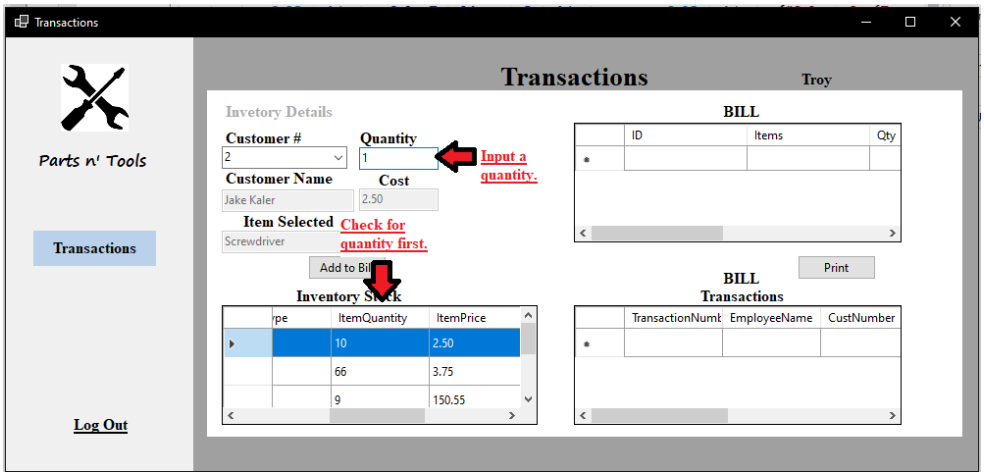


Step 2: Select an item in the “Inventory Stock” grid.

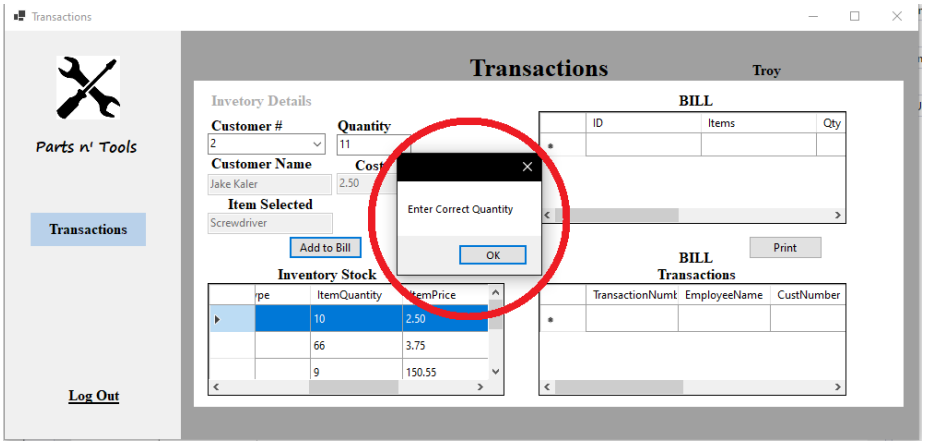




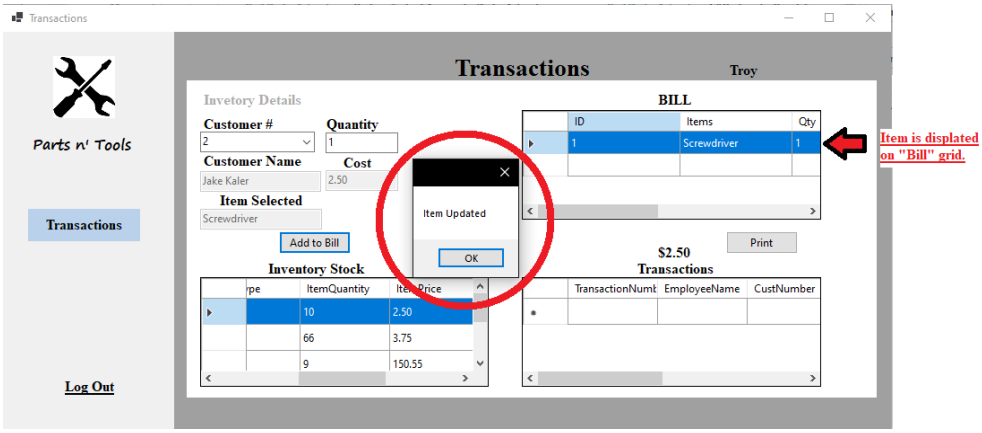
Step 3: Inputting a quantity and clicking “Add to Bill” button. Before the clicking “Add to Bill” button, make sure that the user quantity is less than the “ItemQuantity” in the grid.



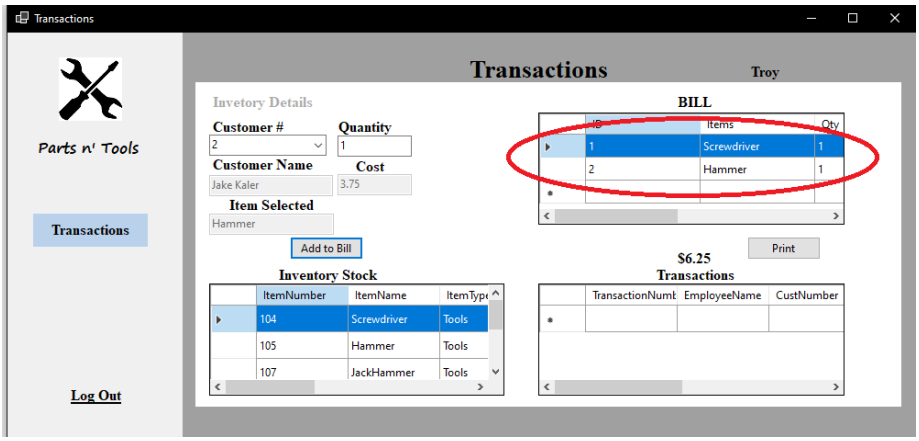
When user inputs a value for quantity higher than “ItemQuantity” in the grid, a message box notifies the user to “Enter Correct Quantity”.



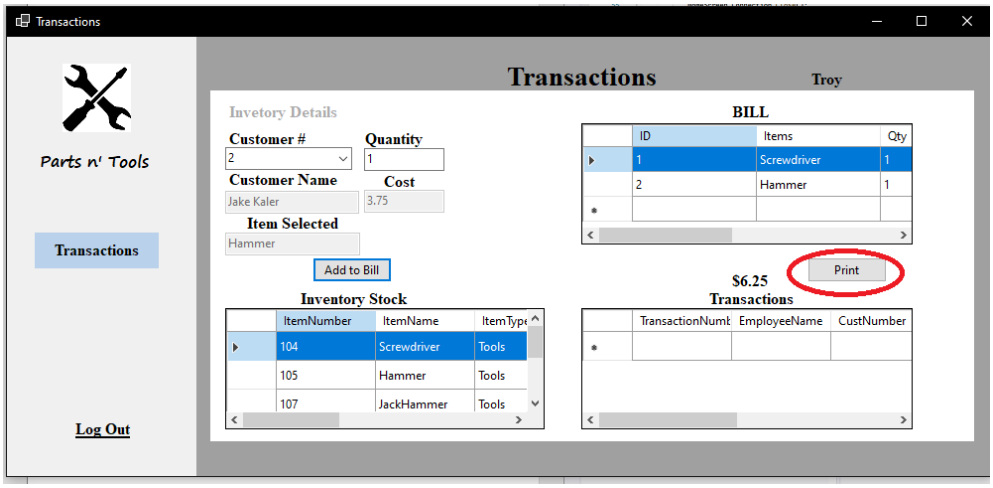
When user inputs the correct quantity, a message box displays “Item Updated”. The selected item will be displayed on the top right grid named “Bill”.

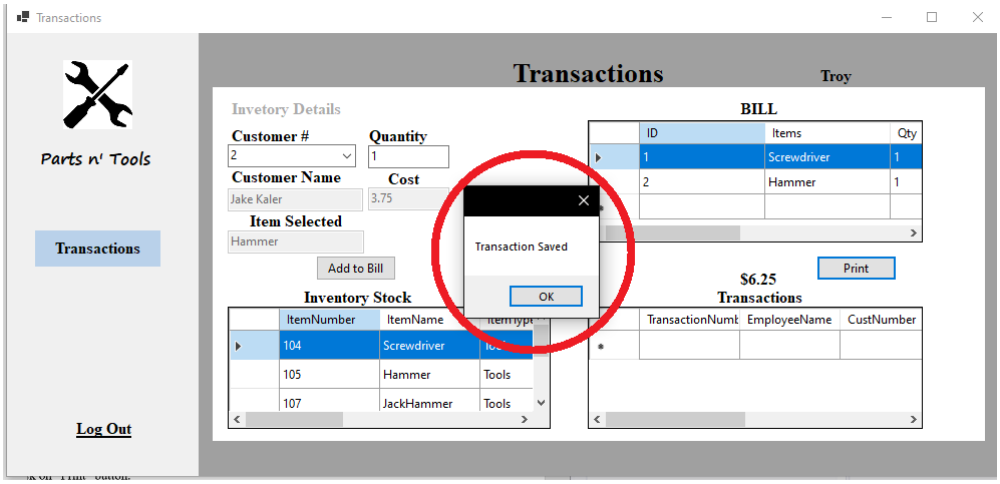


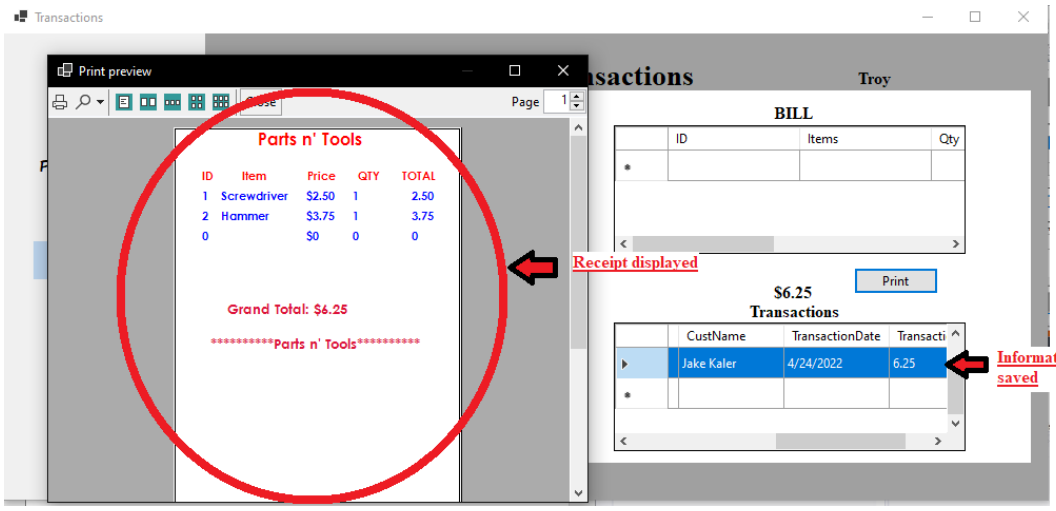
Optionally, users can add more items to add to the “Bill” grid.



Step 4: When a user clicks on the “Print” button, a message box will display “Transaction Saved”. Meaning the transaction information will be saved on the bottom corner grid known as “Transactions”. Afterwards, the software will display a receipt and the ability to print it out.



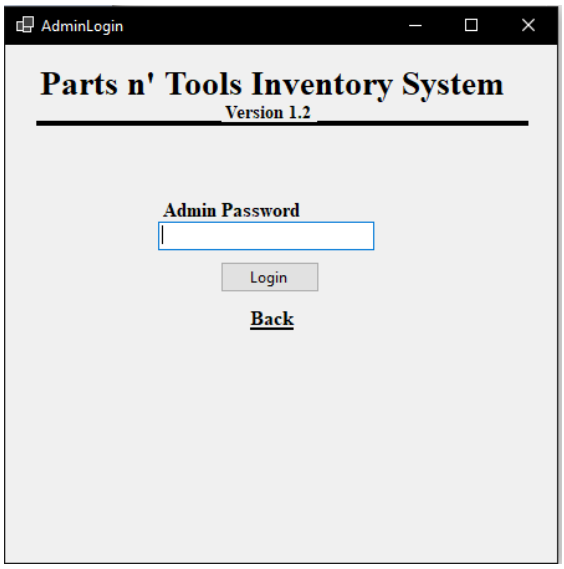




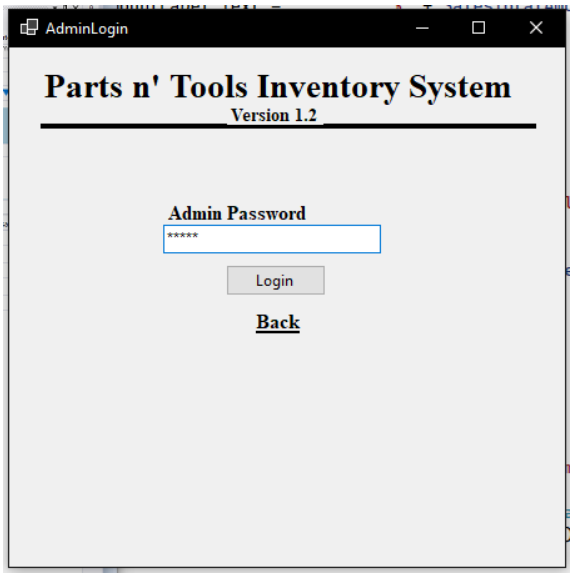
Administrator privileges

Authentication security

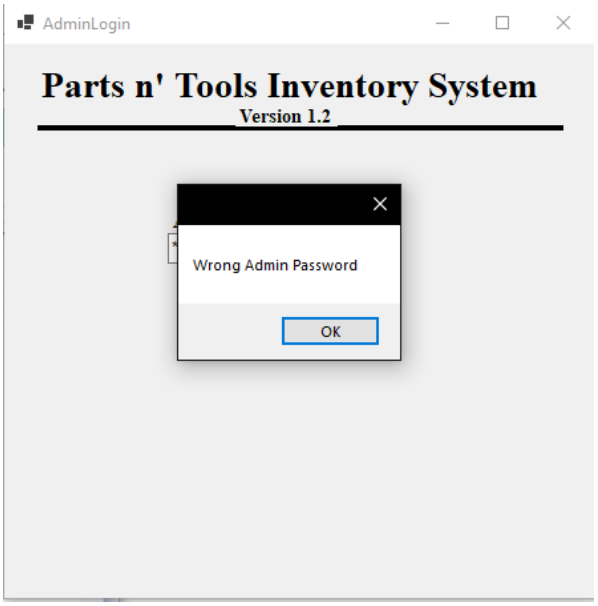
Admin Login: The administrator login functions the same as the user log in, only difference is that the admin must only enter a password.



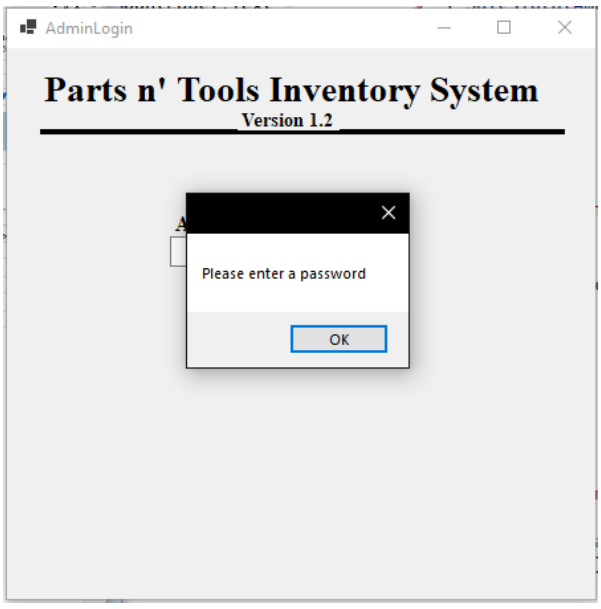
Password for the administrator strings will be hidden with “*”. The back button returns to the “User login” window.

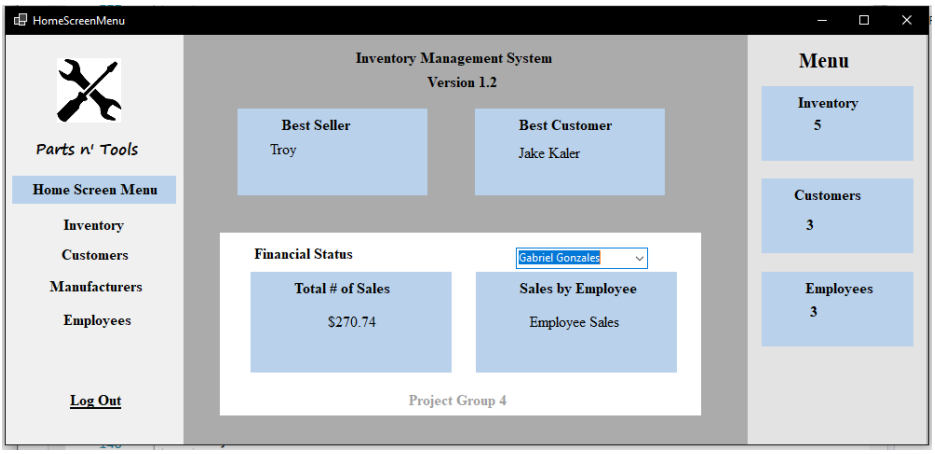


If admin enters the wrong password:



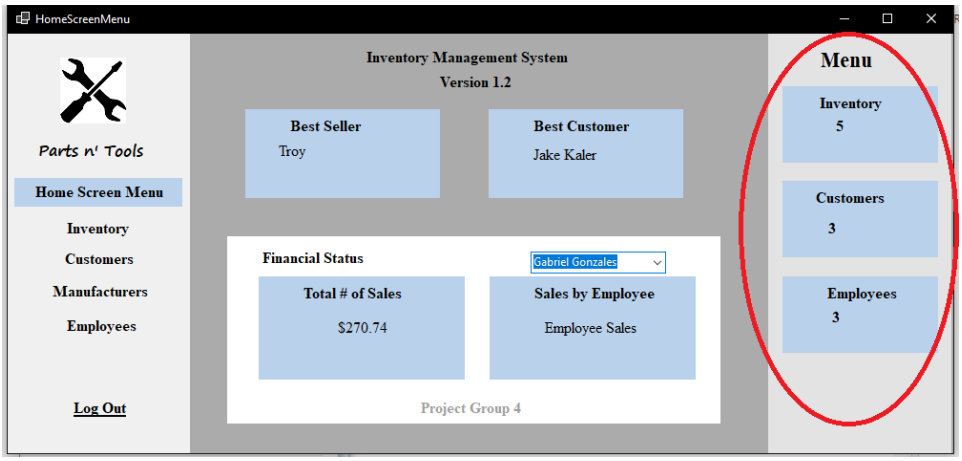
If admin enters no password:





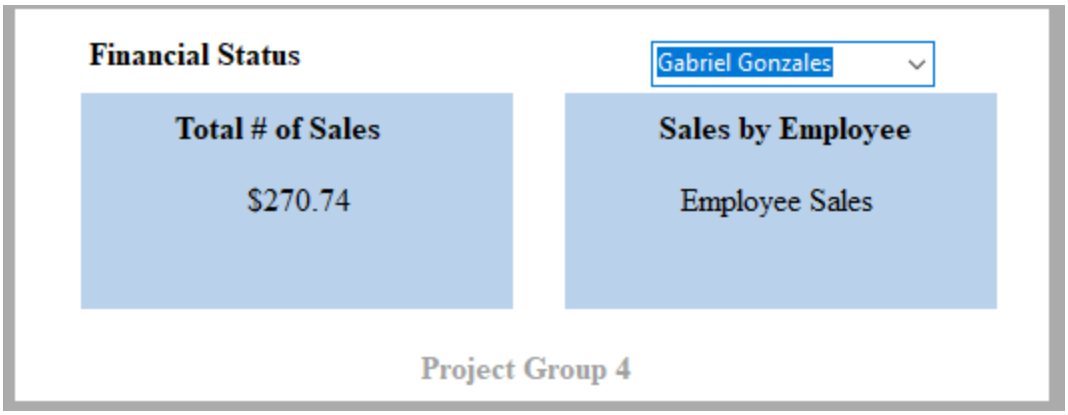
If admin enters the correct password, they will be prompted to the “HomeScreenMenu” window.

The “Best Seller” and the “Best Customer” boxes display the names of those who contributed on the most purchases/sales. The data is collected from users who log in and makes purchases for customers from the “Transactions” window.

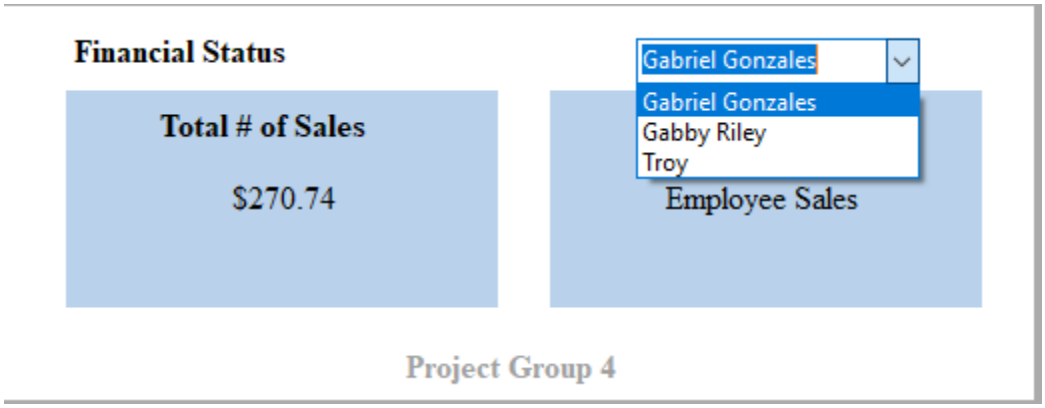


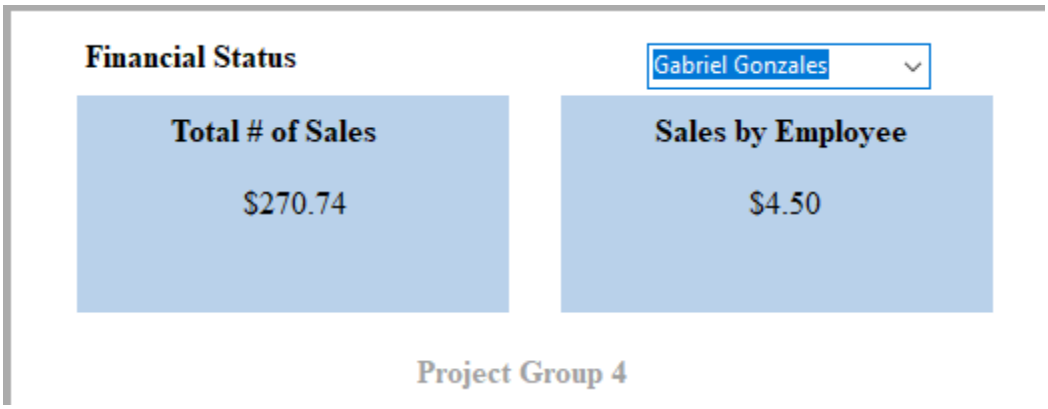
The circled area is a summary of data from other interfaces that the manager has entered.

- “Inventory” data collects the different types of items in the “Inventory Interface”, not the total amount of items.
- “Customers” data collects the total number of customers in the “Customers interface.”
- “Employees” data collects the total number of customers in the “Employees interface”



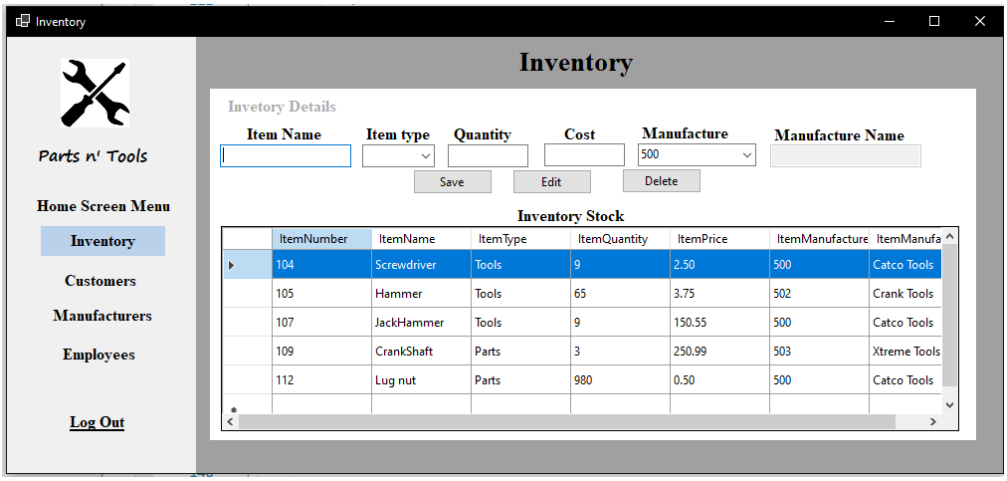
The “Total # of Sales” box collects data from the “Transaction” from every purchase from each user and adds them up to get the total sales of the software.





The admin must choose an employee’s name on the combo box to display the total number of sales the selected employee has made.

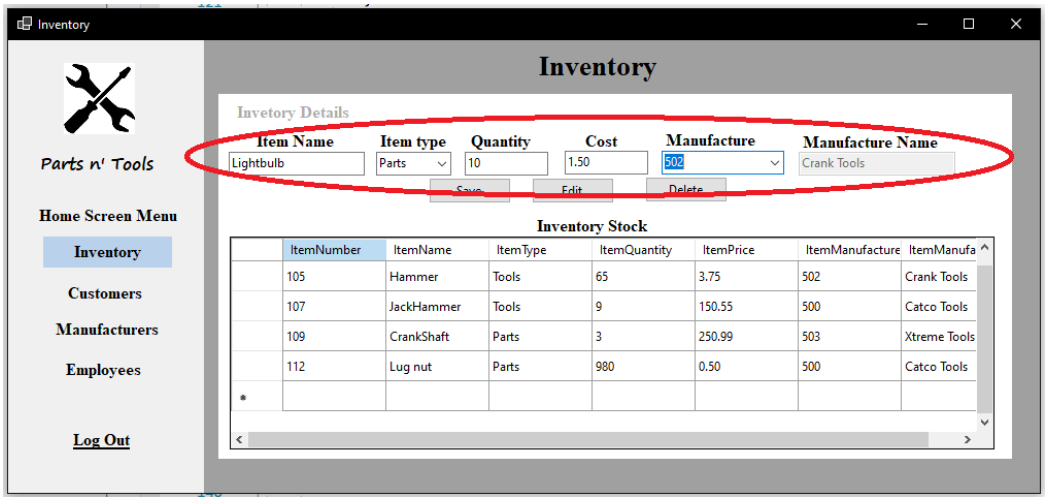
Inventory Interface



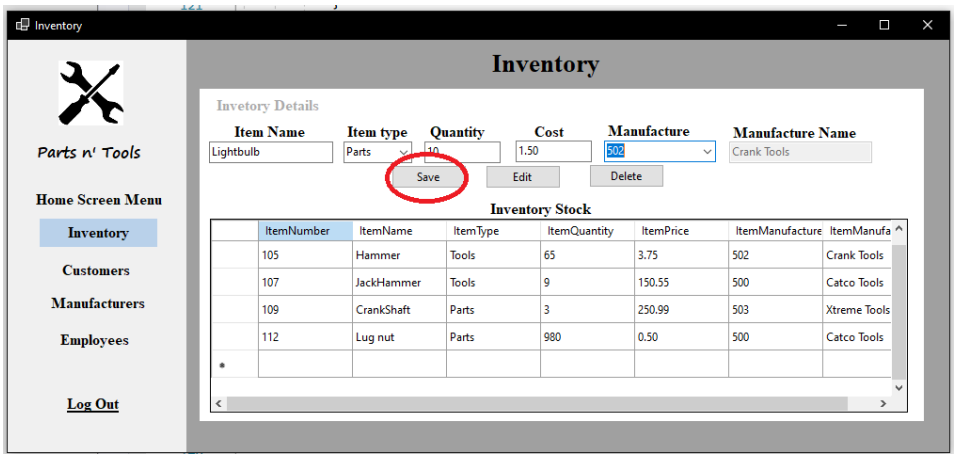
The administrator can save, edit, or delete items in the grid.

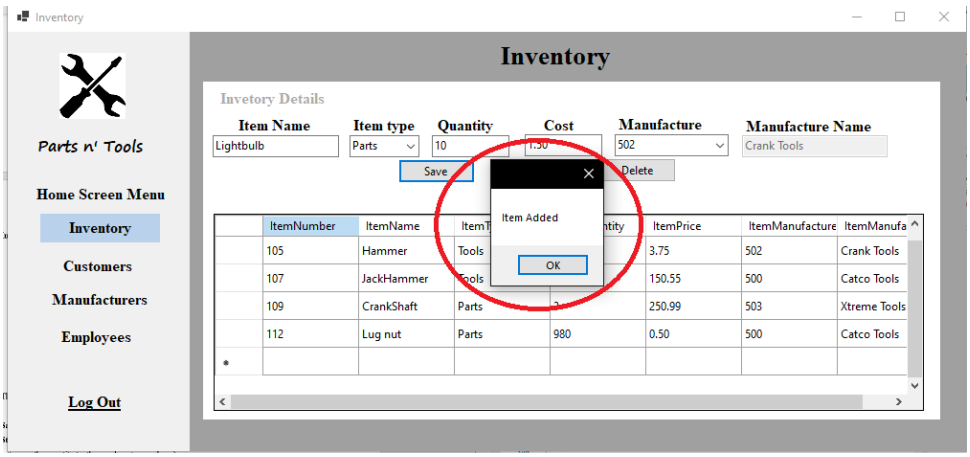
Saving an item

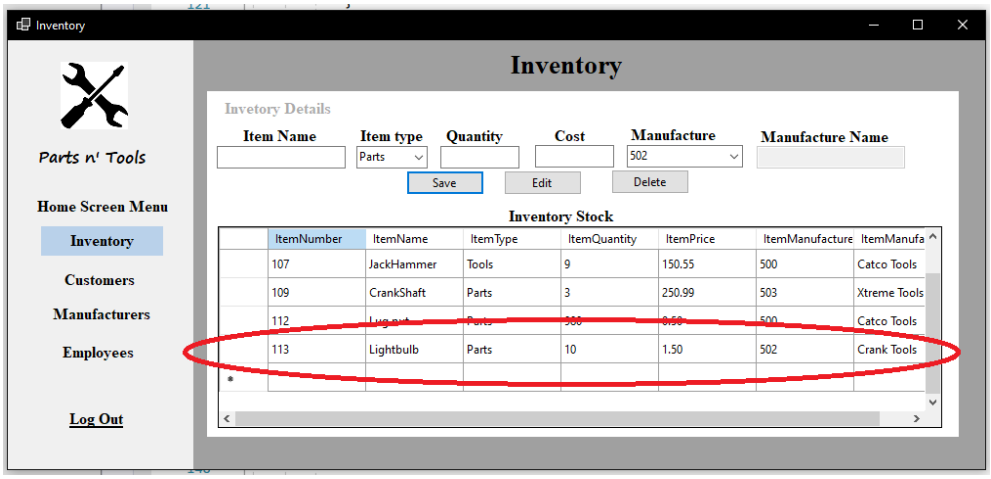
Step 1: The administrator must fill every information in the boxes. (**Warning:** Do not enter letters on the quantity textbox, only enter numbers.)



Step 2: Click on “Save” and item should be added in the “Inventory Stock” grid.

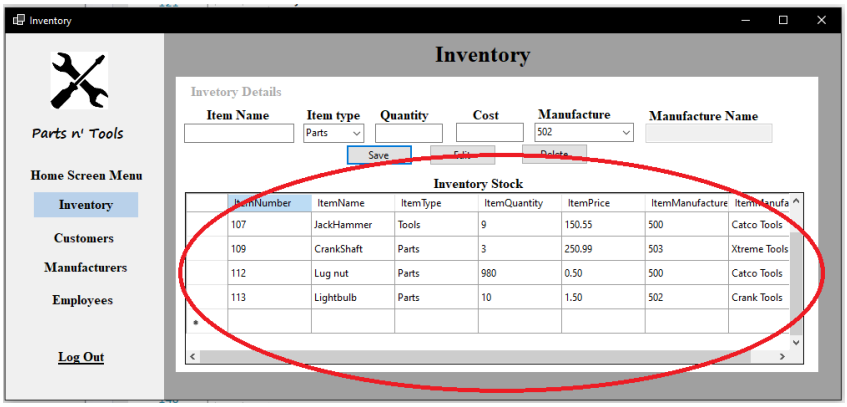




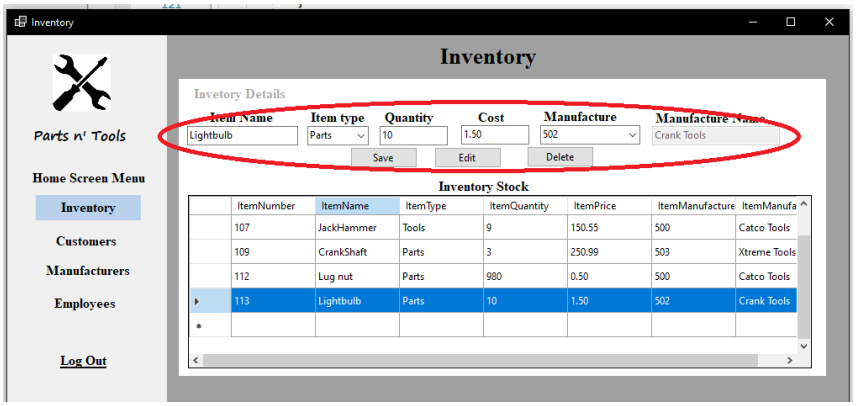


Editing an Item

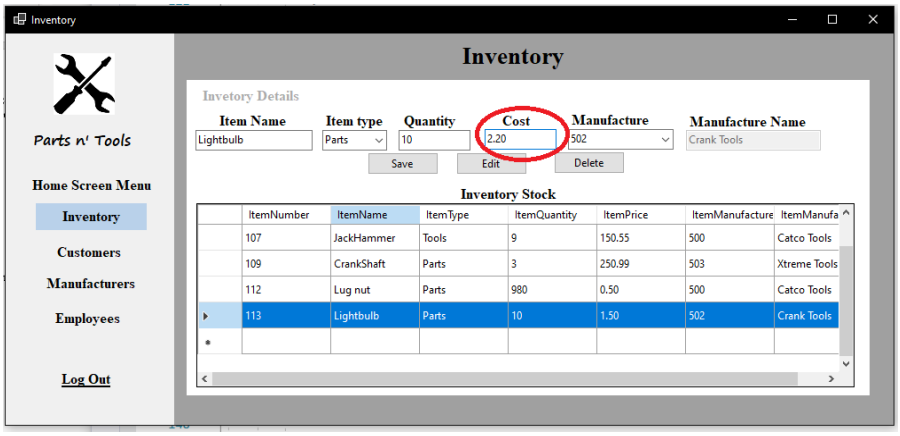
Step 1: Click on an item in the “Inventory Stock” grid.



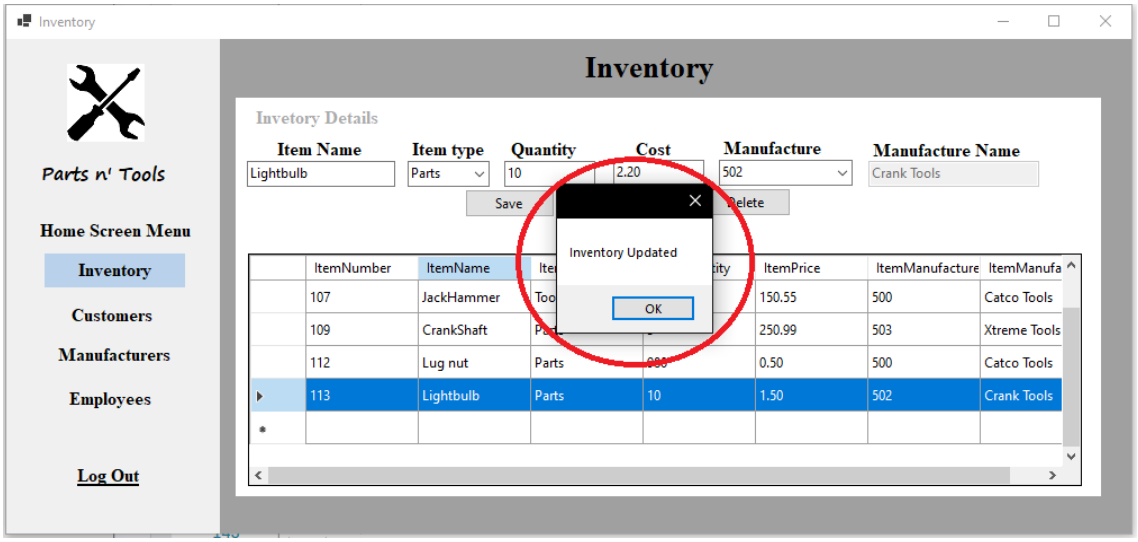
The selected item should appear on the textboxes. In this case Lightbulb is selected.

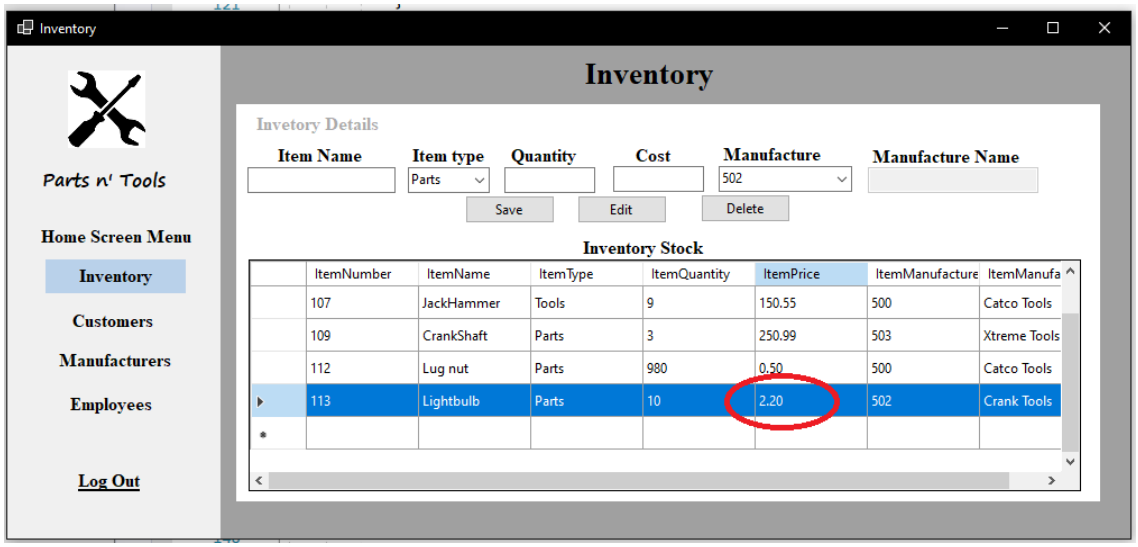


Step 2: Change any information of the item in the text boxes. In this case, cost of item will be changed. (You can optionally change everything at once)



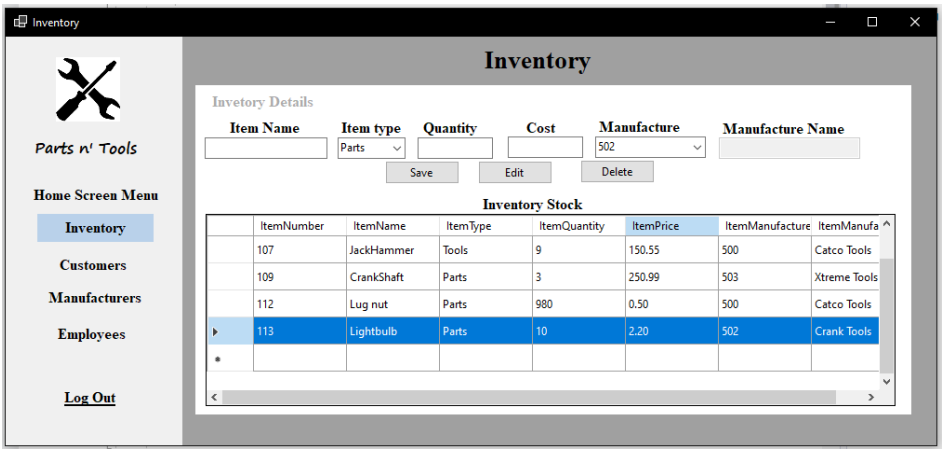
Step 3: Click on the “Edit” button and the item will be updated.

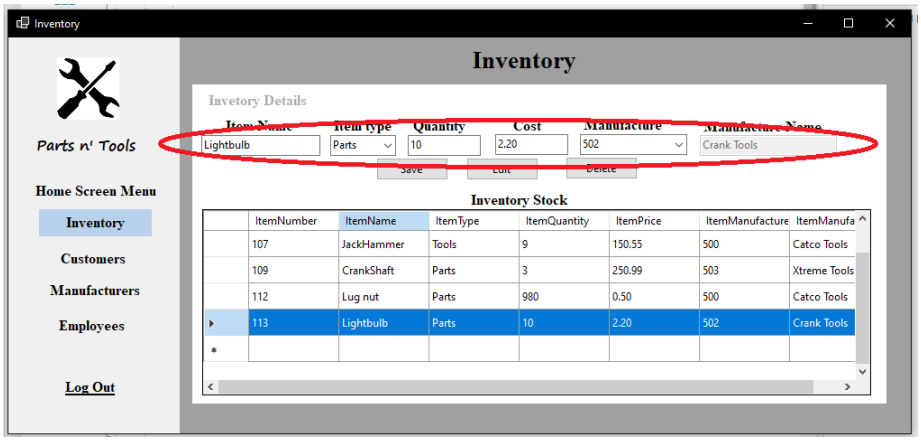




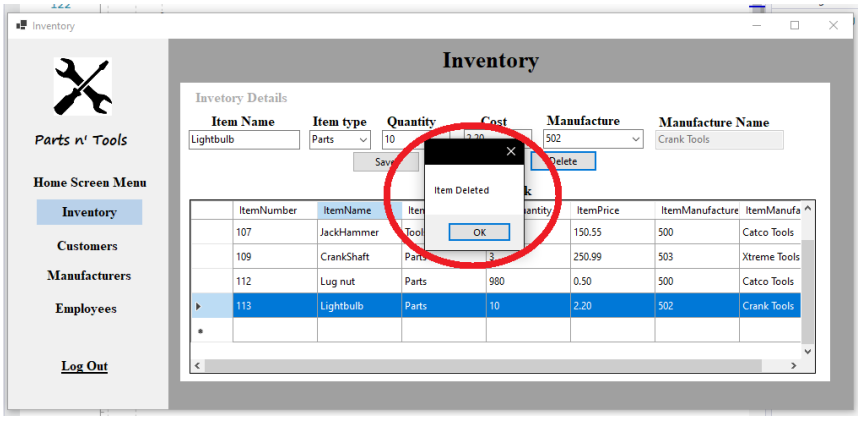
Deleting an item from the Inventory interface

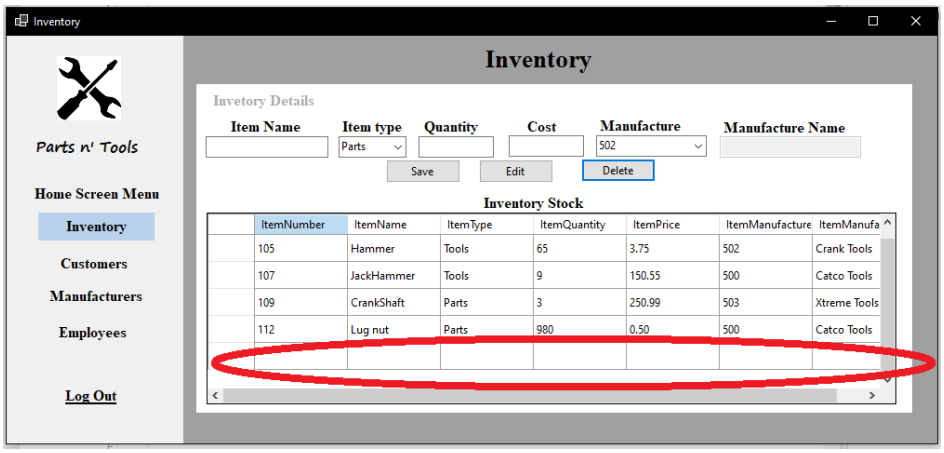
Step 1: Click on an item on the “Inventory Stock” interface. In this case, “Lightbulb” is selected.



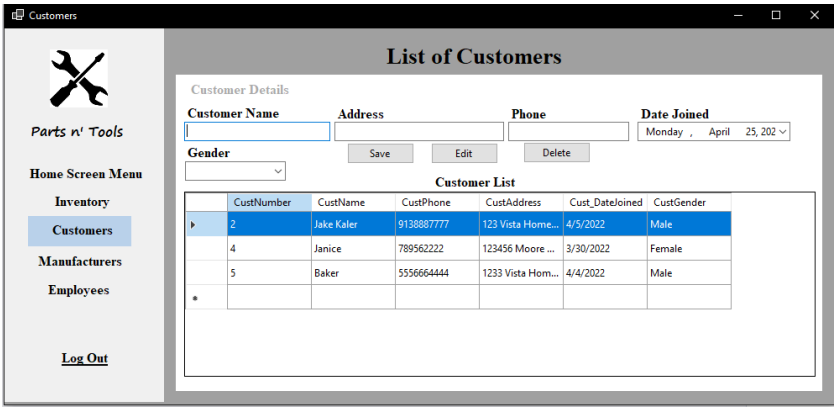


Step 2: Click on the “Delete” button and the selected item will be removed off the “Inventory Stock” interface.



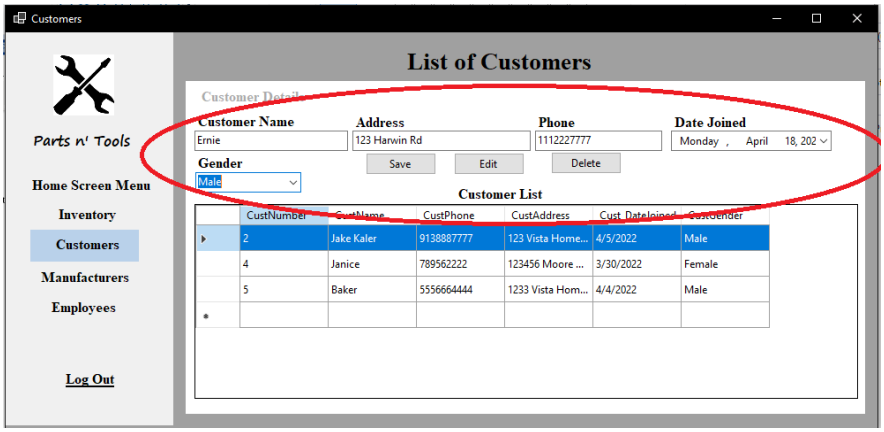


Customer Interface

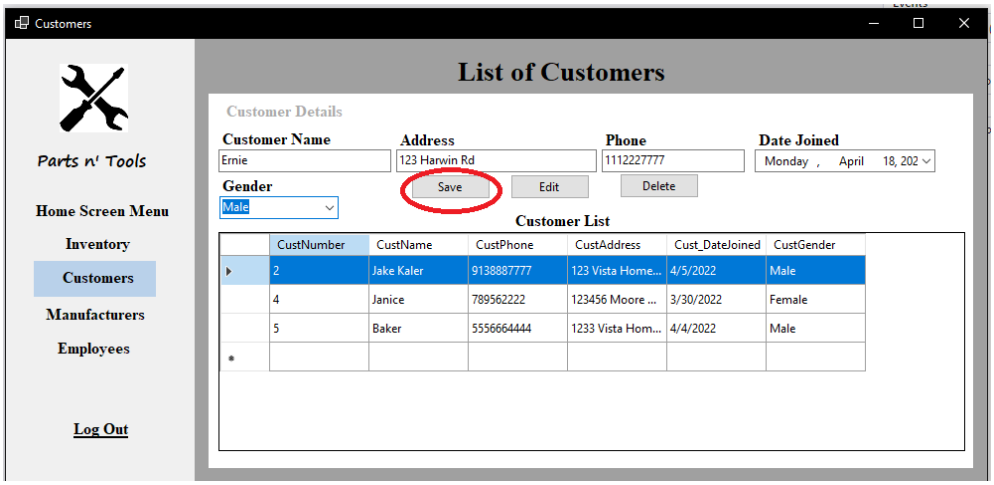


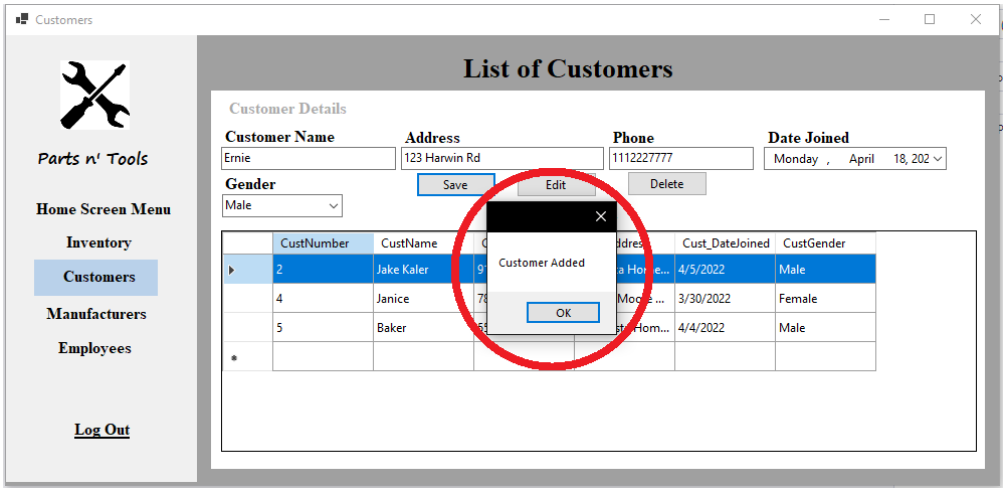
Saving a customer.

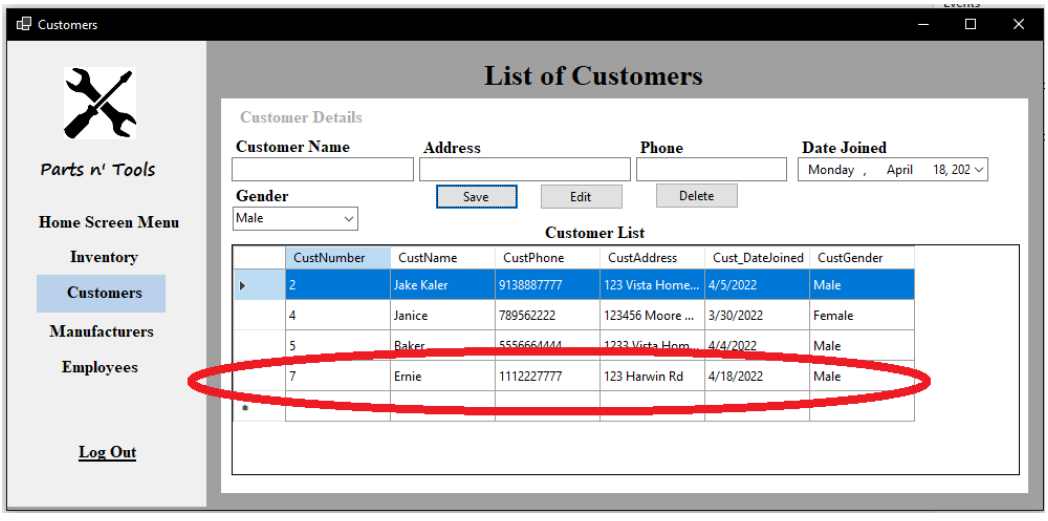
Step 1: The administrator must fill every information in the boxes.



Step 2: Click on “Save” and the customer should be added in the “Customer List” grid.

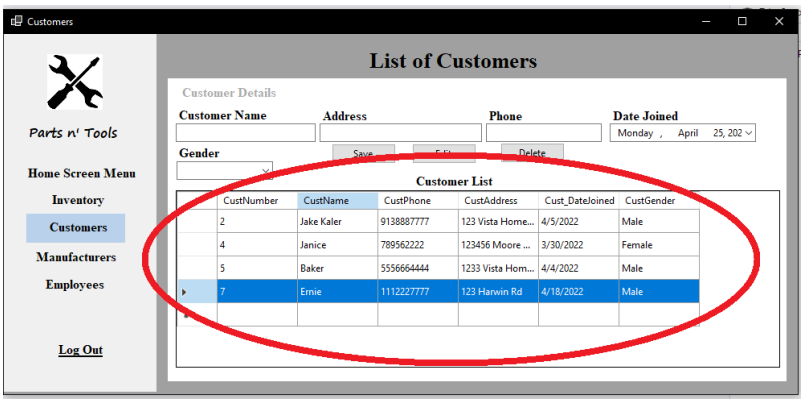




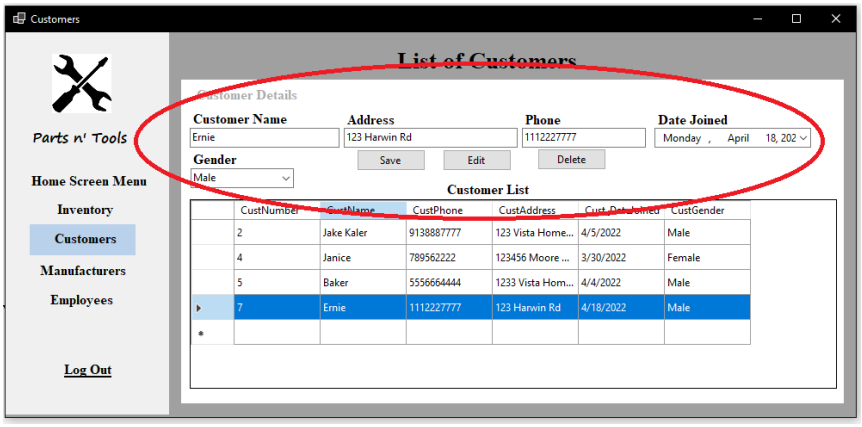


Editing customer information

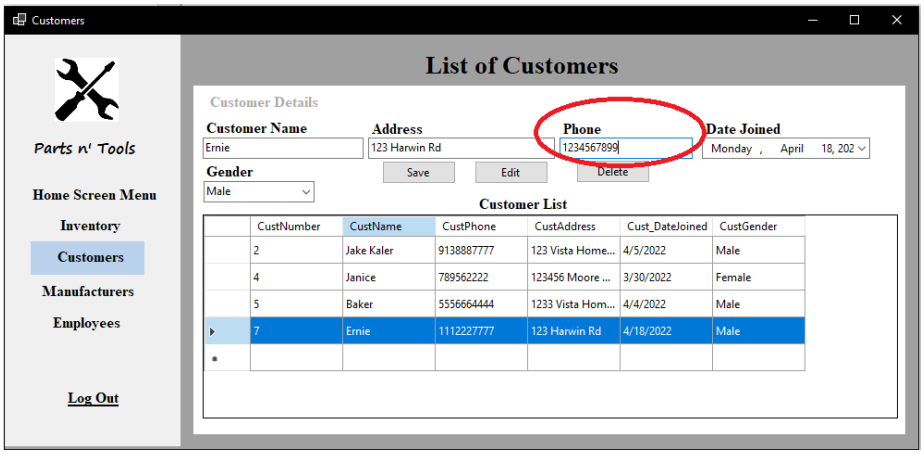
Step 1: Click on a customer in the “Customer List” grid.



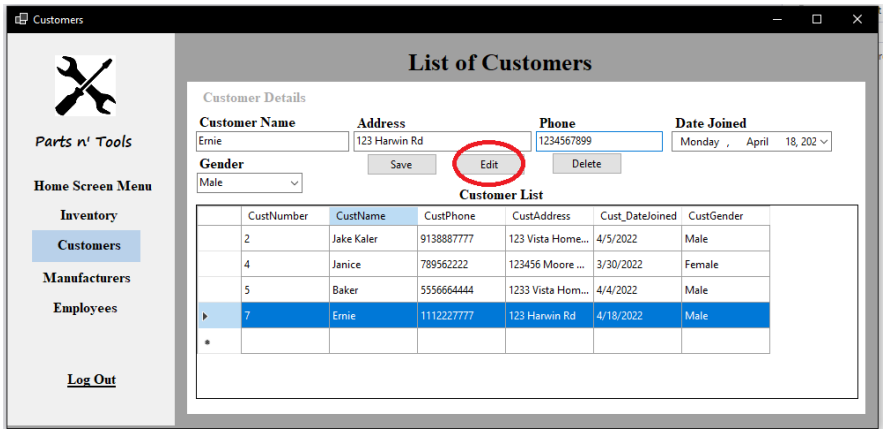
The selected customer should appear on the textboxes. In this case Ernie is selected.

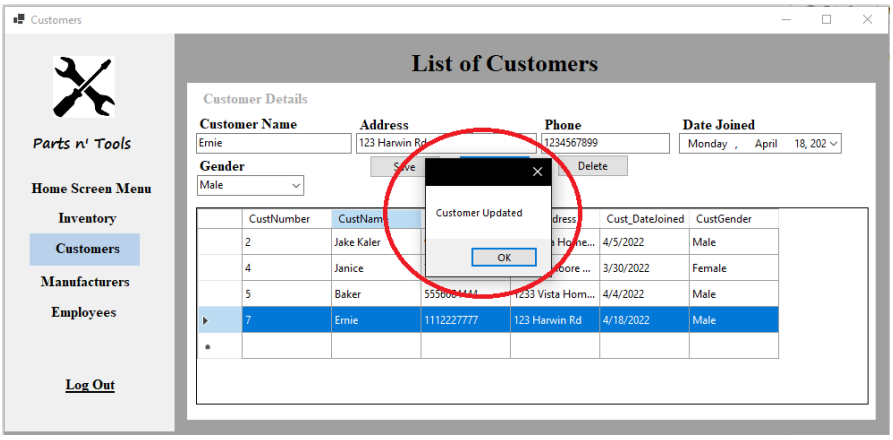


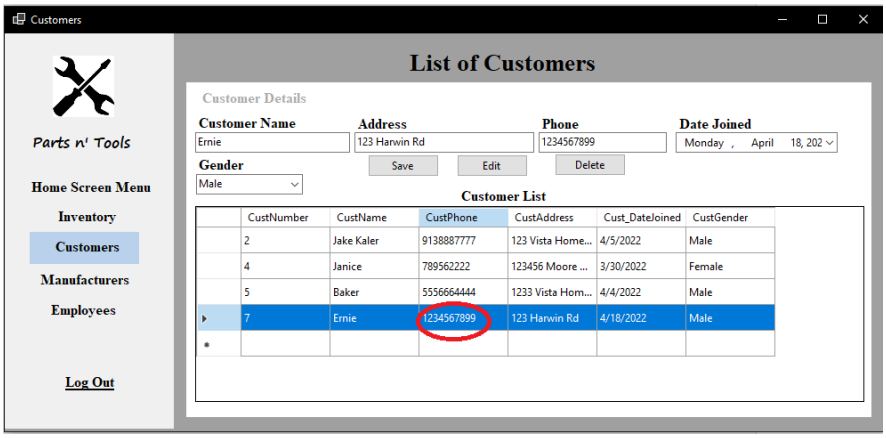
Step 2: Change any information displayed on the boxes. In this case phone number is being changed. (You can optionally change everything at once)



Step 3: Click on the “Edit” button and the selected customer information is updated.

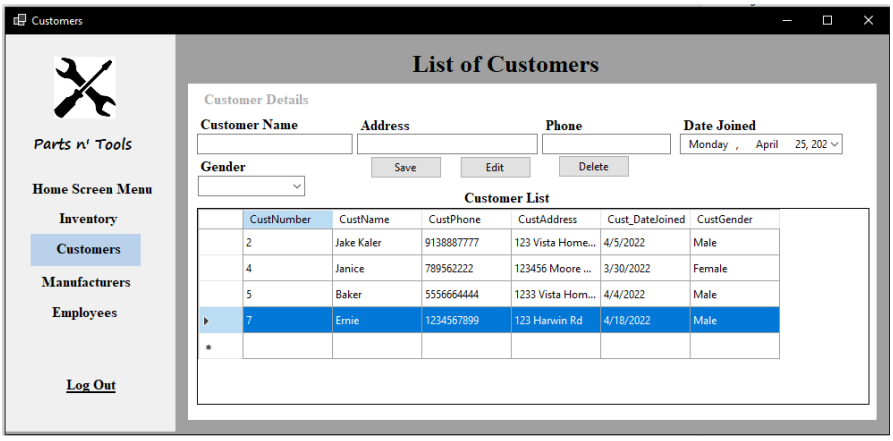


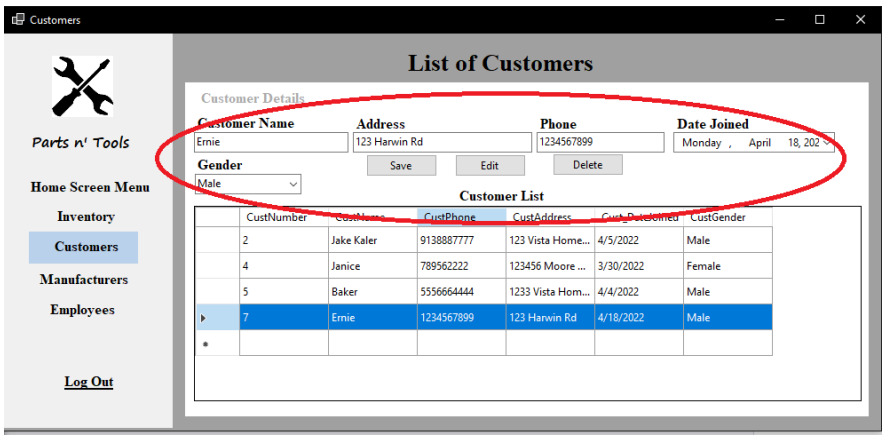




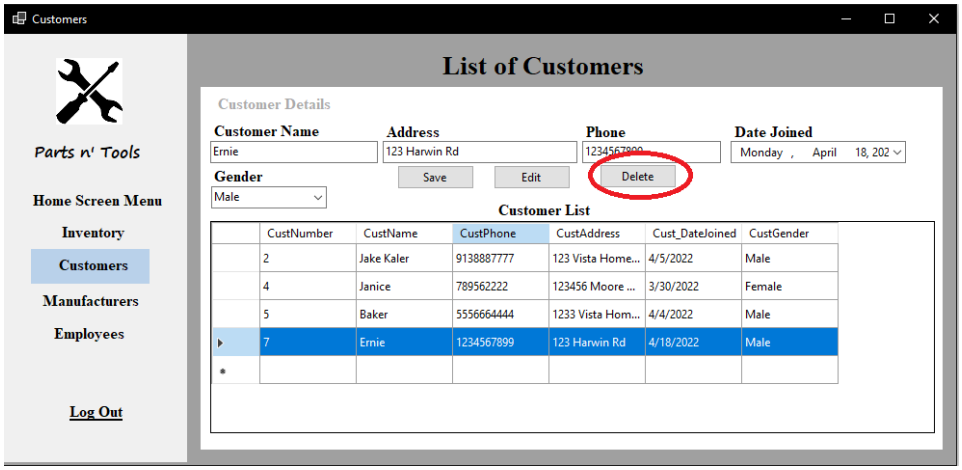
Deleting a customer

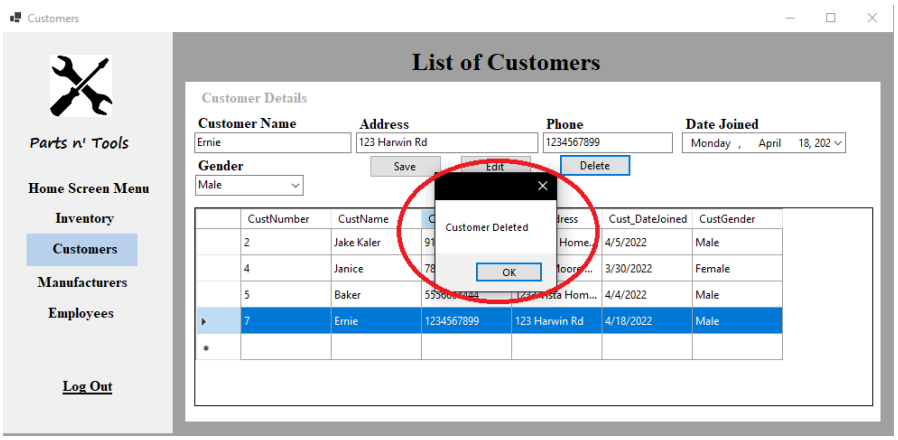
Step 1: Click on a customer in the “Customer List” grid. In this case Ernie is selected.

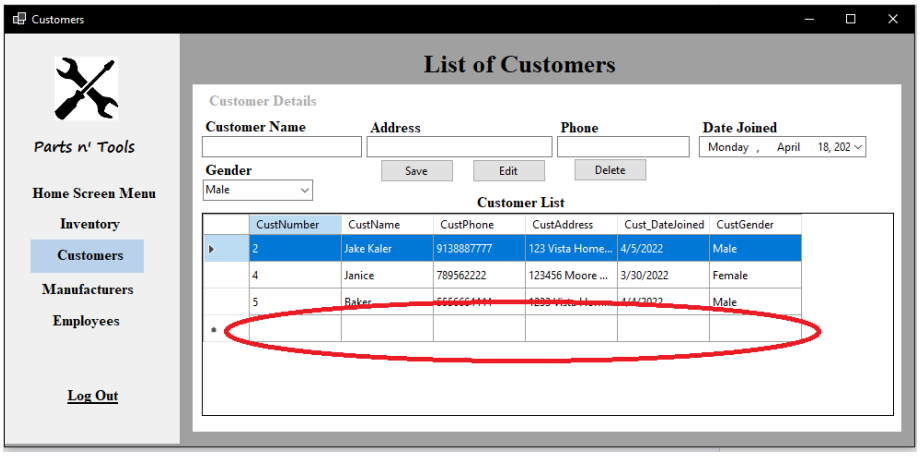




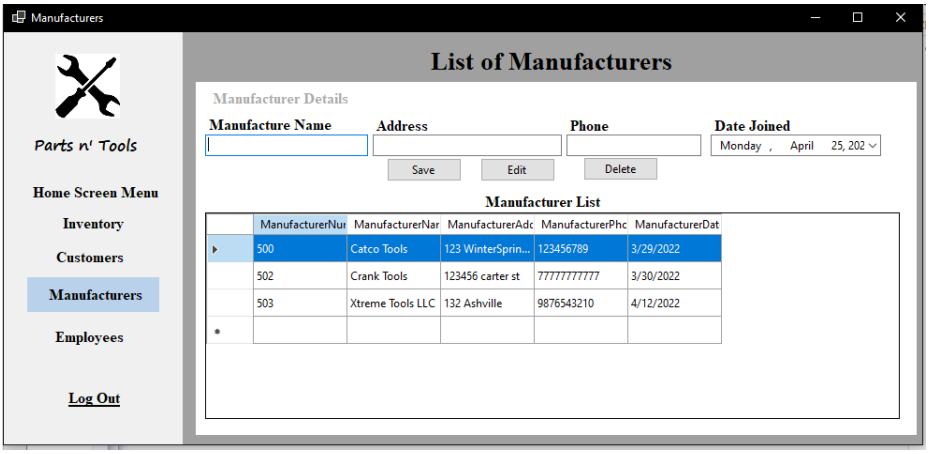
Step 2: Click on delete and the selected customer will be removed off the “Customer List” grid.




Customers
Parts n' Tools
Home Screen Menu
Inventory
Customers
Manufacturers
Employees
Log Out
List of Customers
Customer Details
Customer Name
Ernie
Address
123 Harwin Rd
Phone
1234567899
Date Joined
Monday , April 18, 202
Gender
Male
Save
Edit
Delete
Customer Deleted
OK
CustNumber
CustName
Cust_DateJoined
CustGender
2
Jake Kaler
4/5/2022
Male
4
Janice
3/30/2022
Female
5
Baker
4/4/2022
Male
7
Ernie
1234567899
123 Harwin Rd
4/18/2022
Male

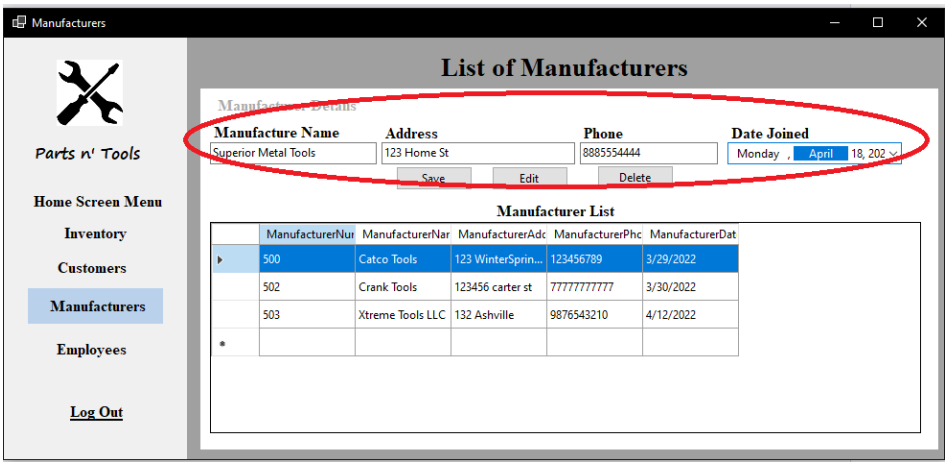

Customers
Parts n' Tools
Home Screen Menu
Inventory
Customers
Manufacturers
Employees
Log Out
List of Customers
Customer Details
Customer Name
Address
Phone
Date Joined
Monday , April 18, 202
Gender
Male
Save
Edit
Delete
Customer List
CustNumber
CustName
CustPhone
CustAddress
Cust_DateJoined
CustGender
2
Jake Kaler
9138887777
123 Vista Home...
4/5/2022
Male
4
Janice
789562222
123456 Moore ...
3/30/2022
Female
5
Baker
Male

Manufacturer interface

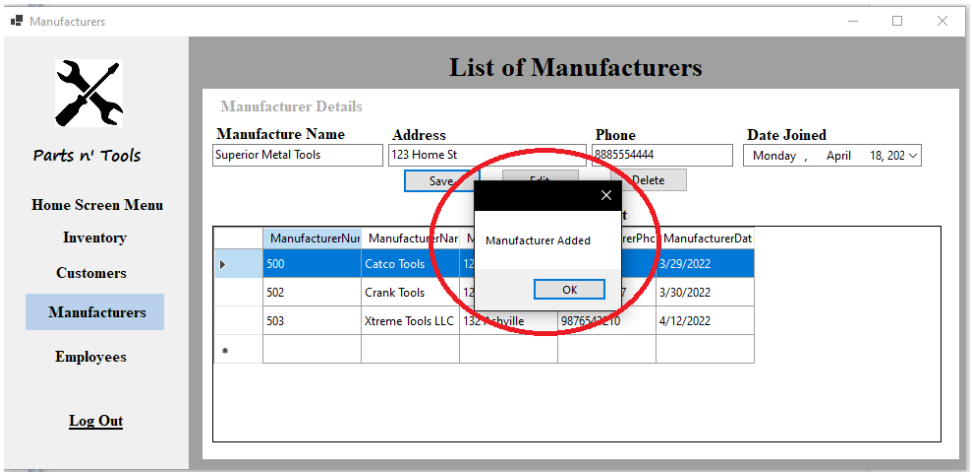


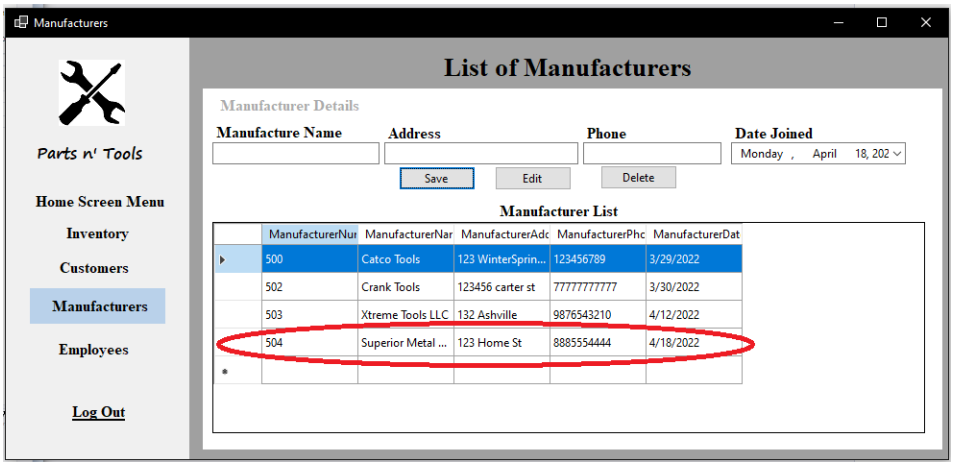
Saving a manufacturer

Step 1: The administrator must fill every information in the boxes.



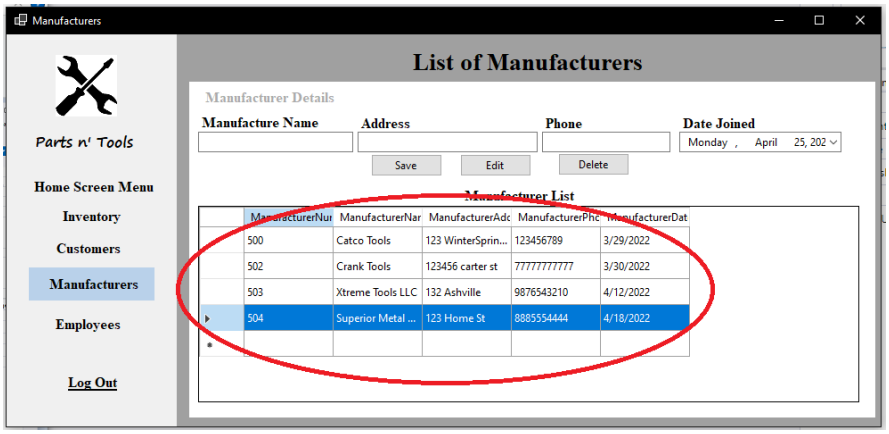
Step 2: Click on “Save” and the manufacturer should be added in the “Manufacturer List” grid.

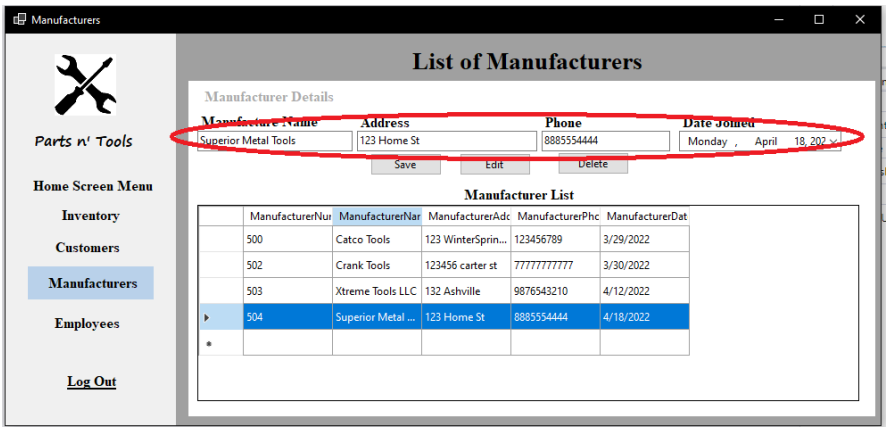




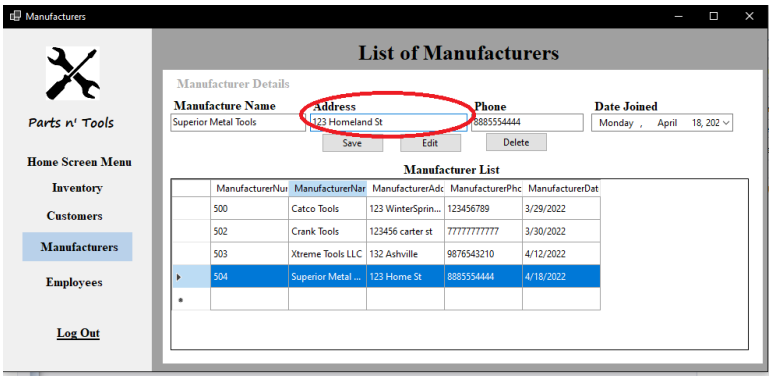
Editing manufacturer information

Step 1: Click on a manufacturer in the "Manufacturer List" grid. In this case Superior Metal Tools is selected.

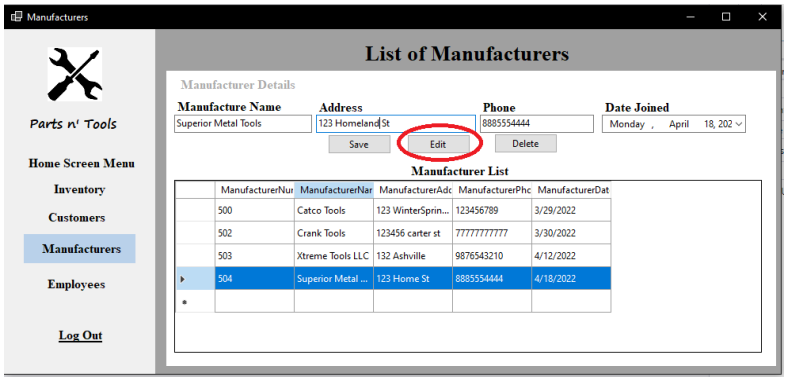


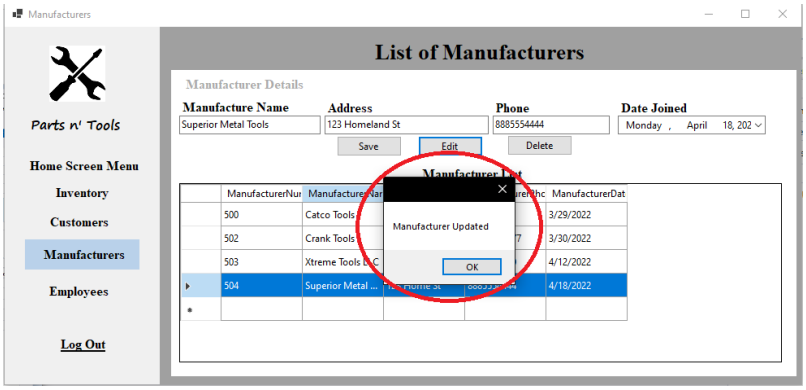


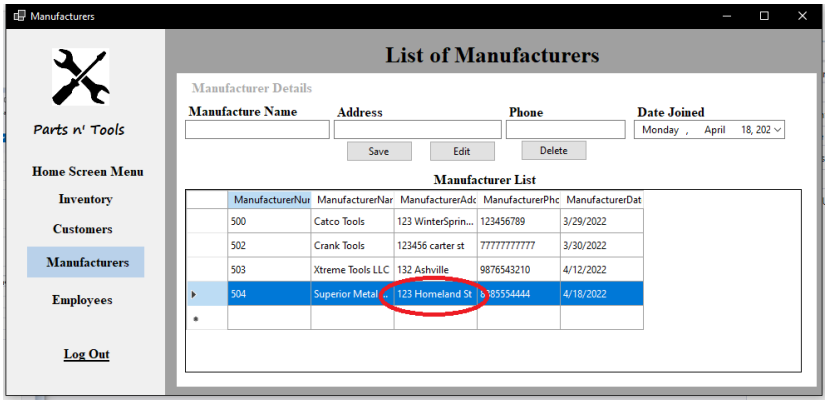
Step 2: Change any information displayed on the boxes. In this case address is being changed. (You can optionally change everything at once)



Step 3: Click on the "Edit" button and the selected manufacturer information is updated.

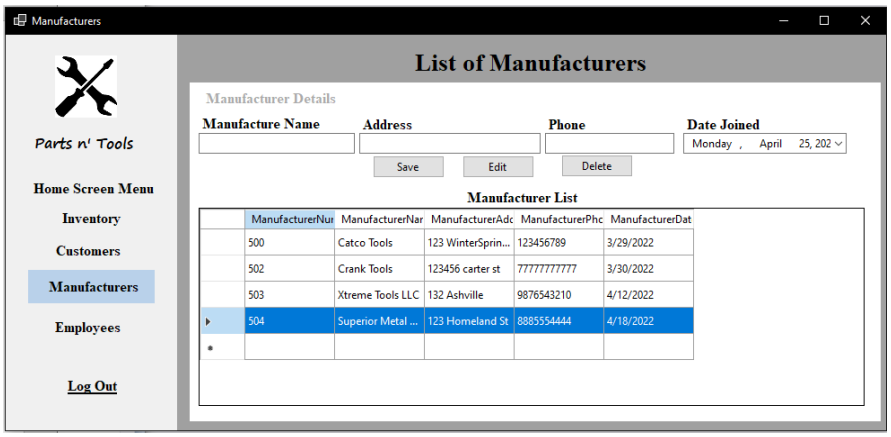


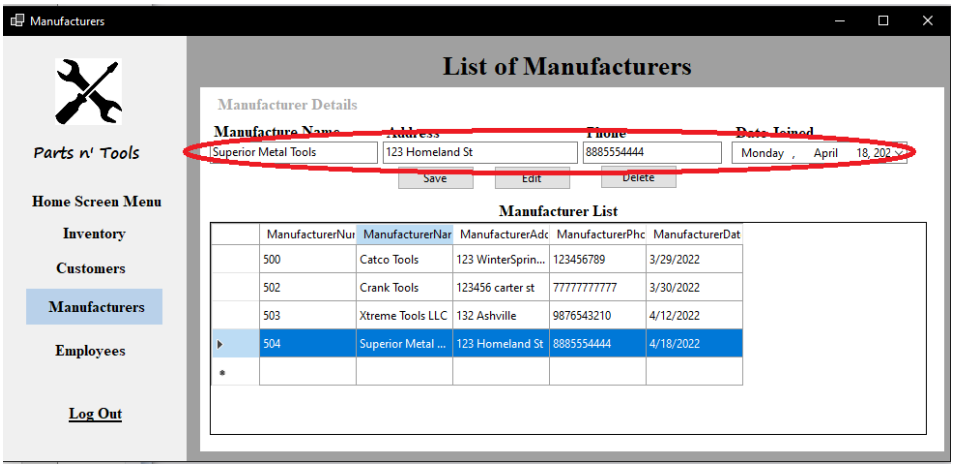




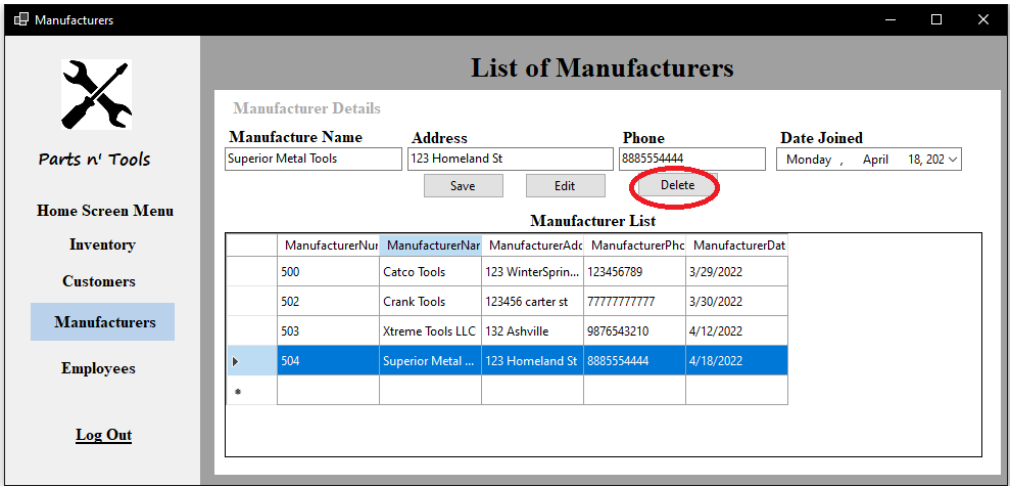
Deleting a manufacturer

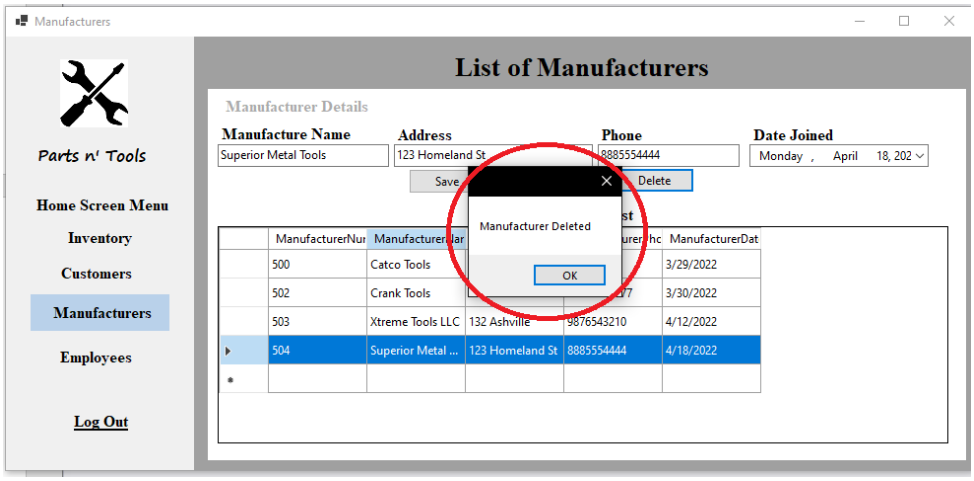
Step 1: Click on a manufacturer in the “Manufacturer List” grid. In this case Superior Metal Tools is selected.

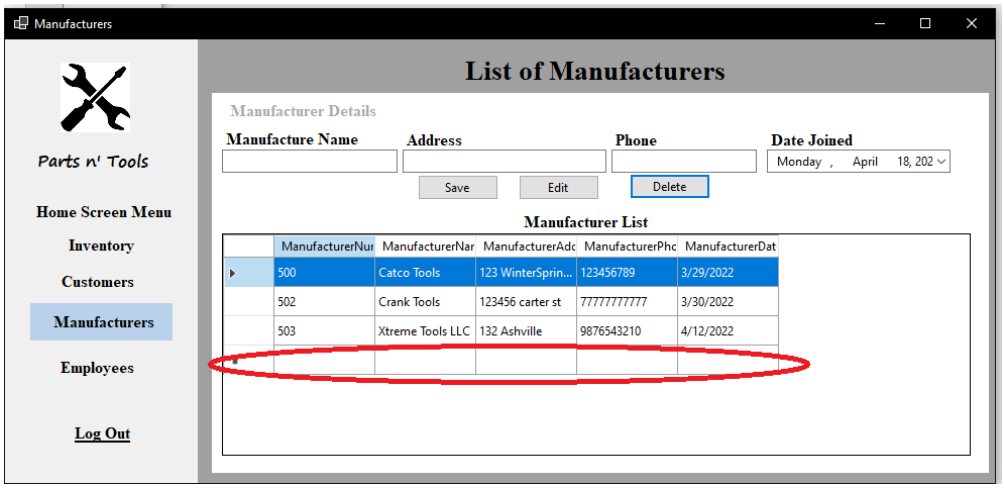




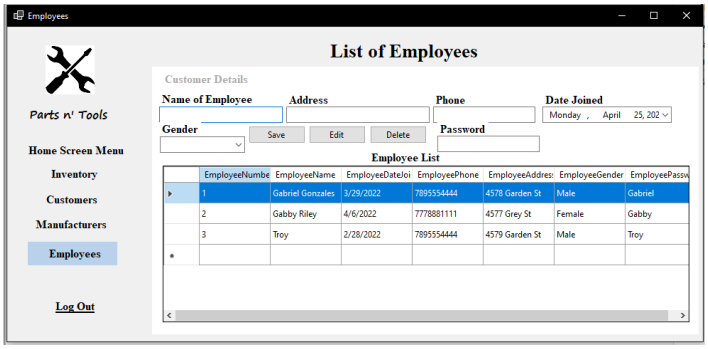
Step 2: Click on delete and the selected manufacturer will be removed off the “Manufacturer List” grid.




Manufacturers
Parts n' Tools
Home Screen Menu
Inventory
Customers
Manufacturers
Employees
Log Out
List of Manufacturers
Manufacturer Details
Manufacture Name
Address
Phone
Date Joined
Superior Metal Tools
123 Homeland St
8885554444
Monday , April 18, 202
Save
Delete
Manufacturer Deleted
OK
ManufacturerNur
ManufacturerNar
ManufacturerDat
500
Catco Tools
3/29/2022
502
Crank Tools
3/30/2022
503
Xtreme Tools LLC
132 Ashville
9876543210
4/12/2022
504
Superior Metal ...
123 Homeland St
8885554444
4/18/2022

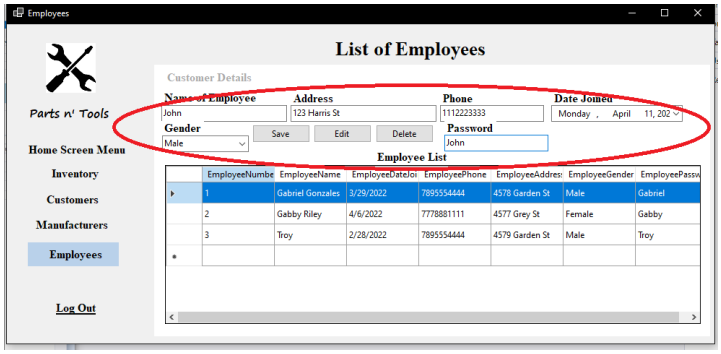

Manufacturers
Parts n' Tools
Home Screen Menu
Inventory
Customers
Manufacturers
Employees
Log Out
List of Manufacturers
Manufacturer Details
Manufacture Name
Address
Phone
Date Joined
Monday , April 18, 202
Save
Edit
Delete
Manufacturer List
ManufacturerNur
ManufacturerNar
ManufacturerAdc
ManufacturerPhc
ManufacturerDat
500
Catco Tools
123 WinterSprin...
123456789
3/29/2022
502
Crank Tools
123456 carter st
77777777777
3/30/2022
503
Xtreme Tools LLC
132 Ashville
9876543210
4/12/2022

Employee interface

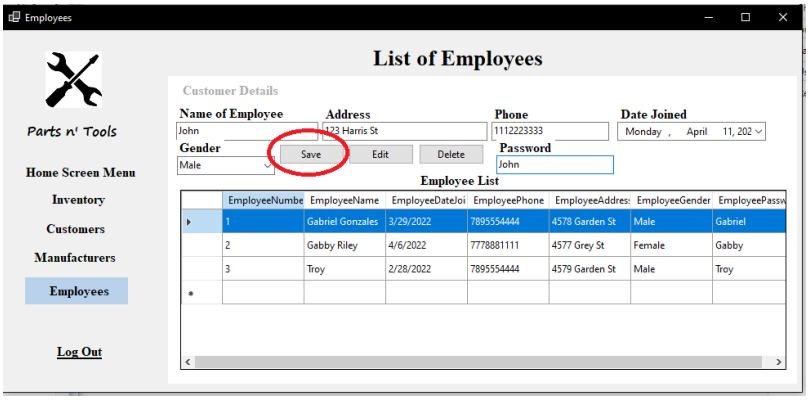


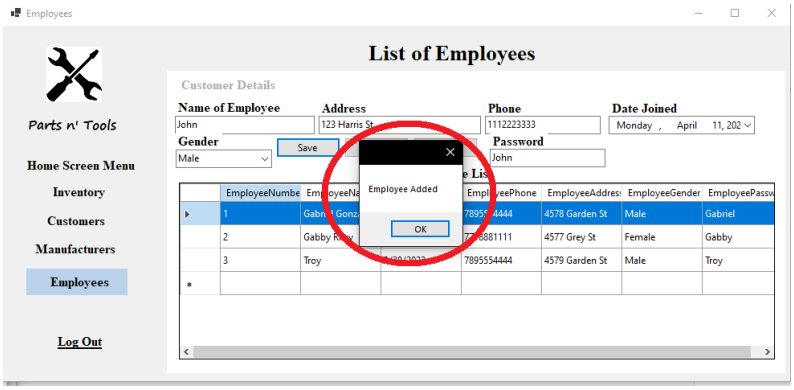
Saving an employee.

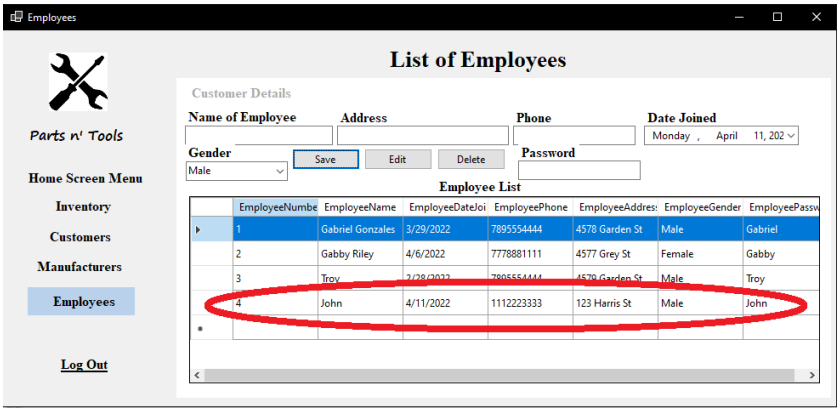
Step 1: The administrator must fill every information in the boxes.



Step 2: Click on "Save" and the employee should be added in the "Employee List" grid. (**Note:** Adding employees in the grid will create more user accounts.)

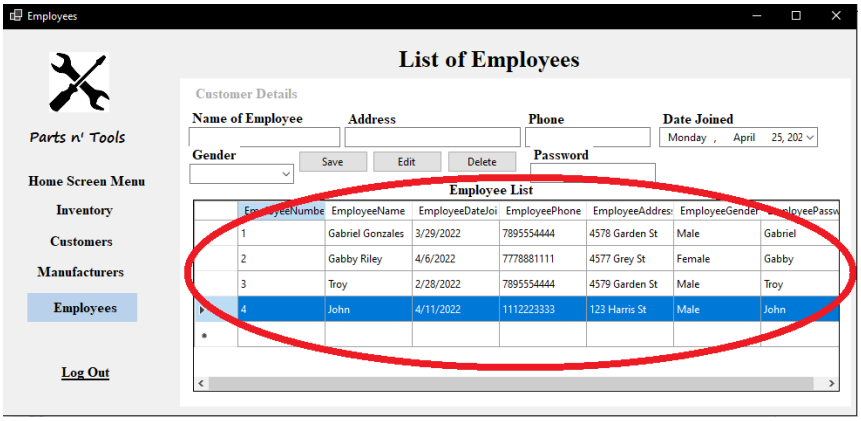


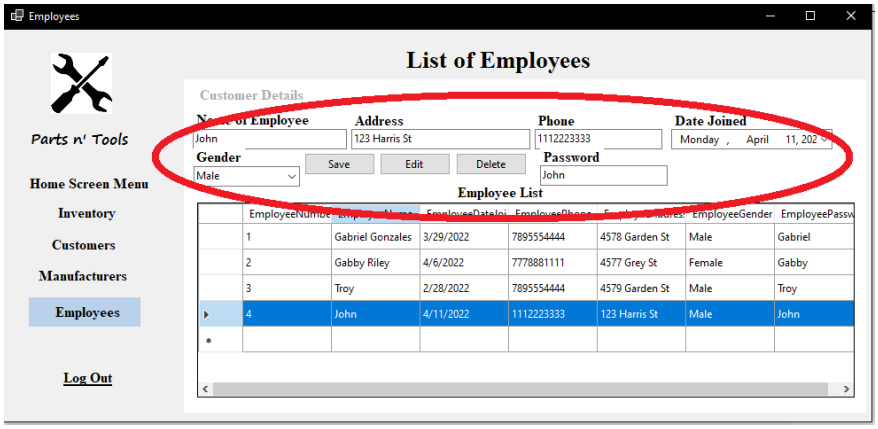




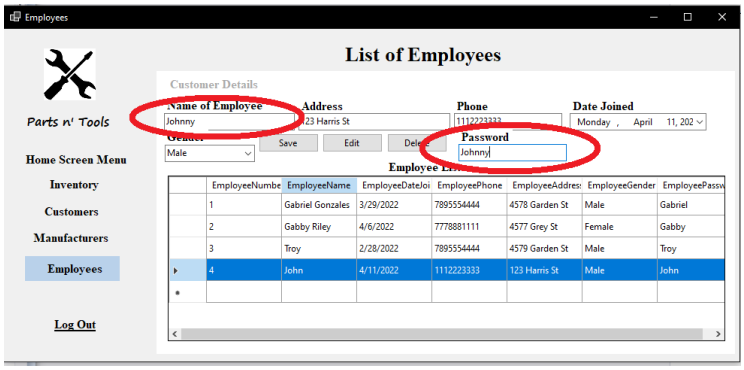
Editing Employee information

Step 1: Click on an employee in the "Employee List" grid. In this case John is selected.

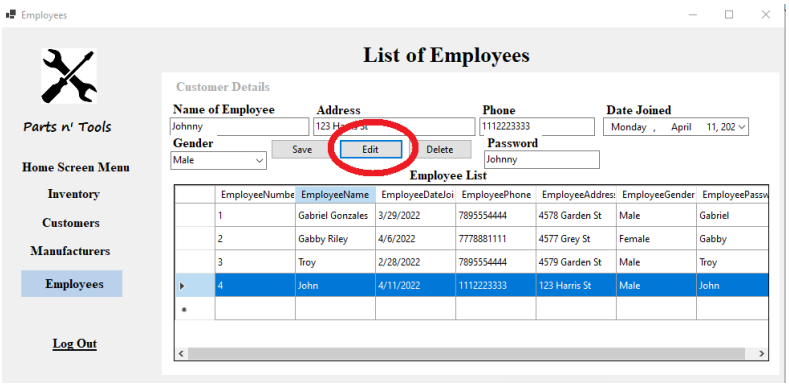


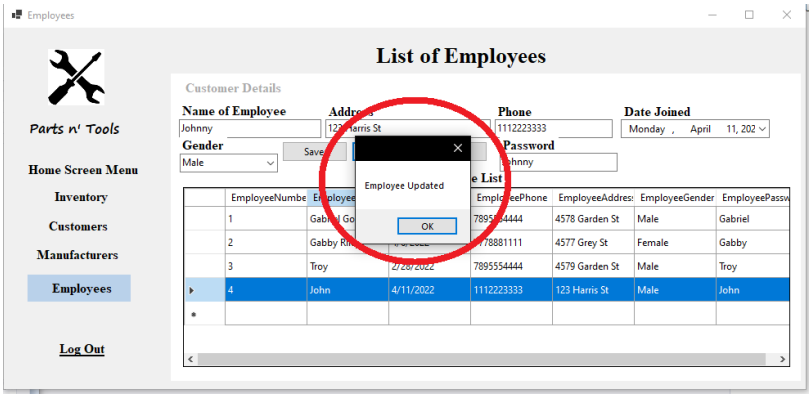


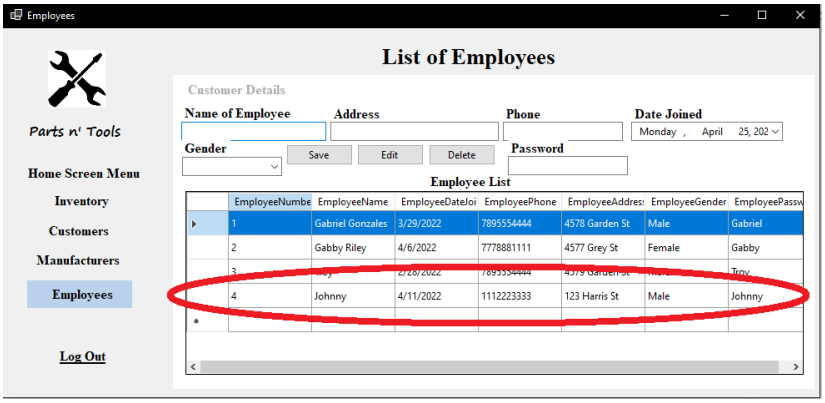
Step 2: Change any information displayed on the boxes. In this case name and password is being changed. You can optionally change everything at once. (**Note:** when changing the password, it will also change when the user tries to log in.)



Step 3: Click on the "Edit" button and the selected employee information is updated.

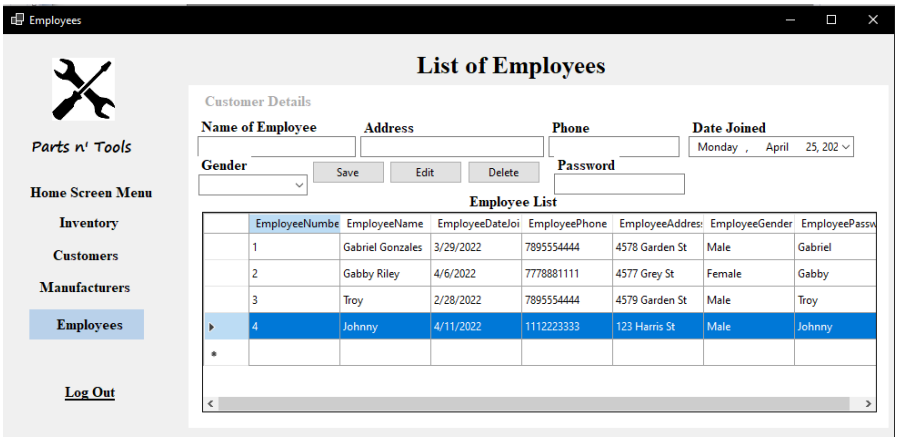


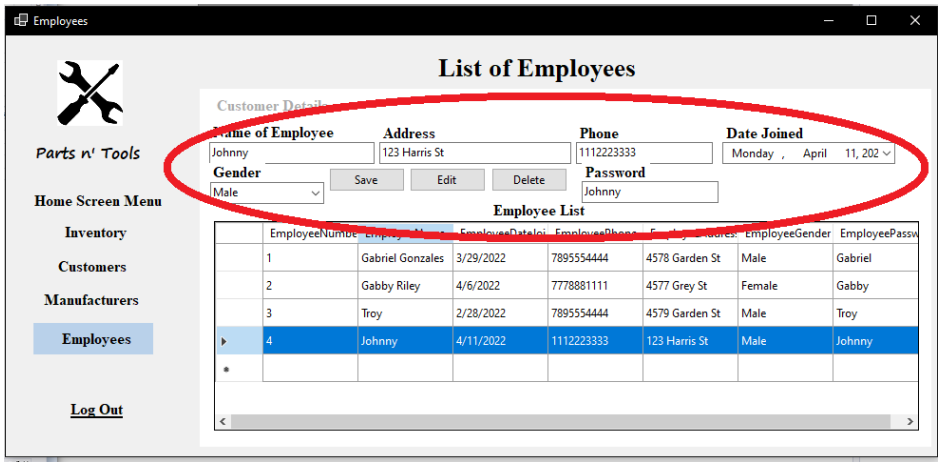




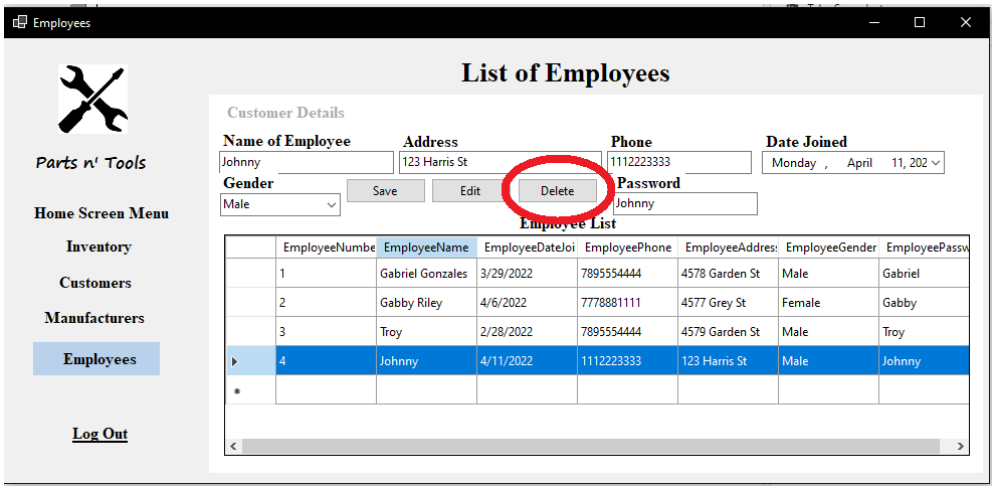
Deleting an employee

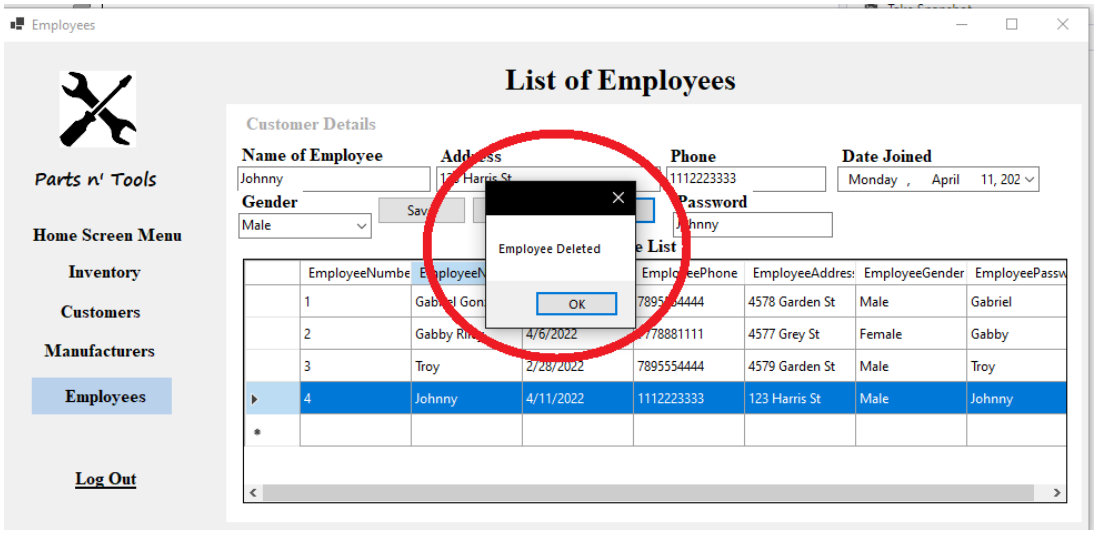
Step 1: Click on an employee in the “Employee List” grid. In this case Johnny is selected.

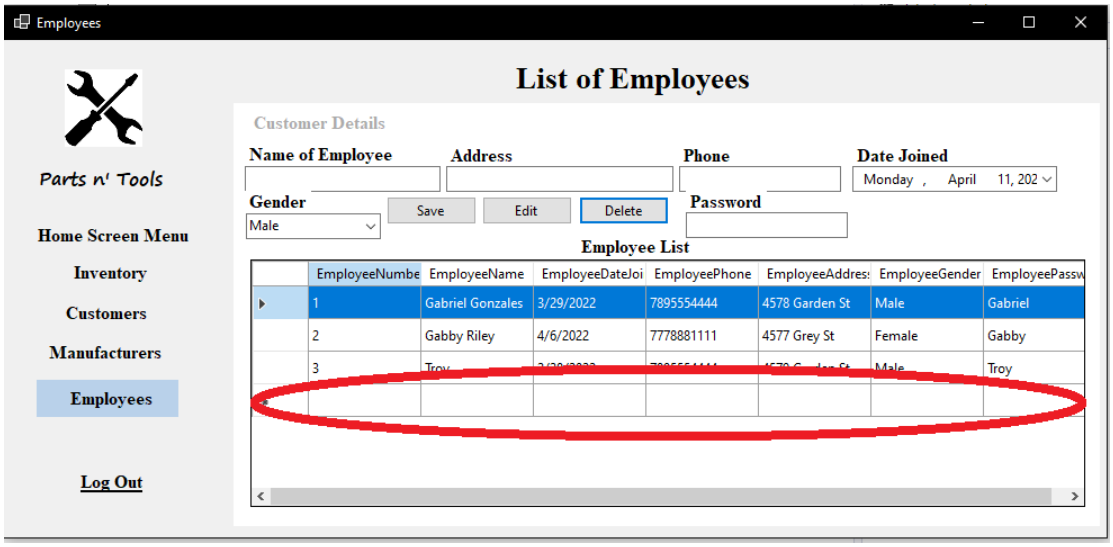




Step 2: Click on delete and the selected manufacturer will be removed off the “Manufacturer List” grid




Employees
List of Employees
Parts n' Tools
Home Screen Menu
Inventory
Customers
Manufacturers
Employees
Log Out
Customer Details
Name of Employee
Address
Phone
Date Joined
Gender
Password
Employee Deleted
OK
Employee List
EmployeeNumbe
EmployeeName
EmployeeDateJoi
EmployeePhone
EmployeeAddres
EmployeeGender
EmployeePassw
1 Gabriel Gonzales 3/29/2022 7895554444 4578 Garden St Male Gabriel
2 Gabby Riley 4/6/2022 7778881111 4577 Grey St Female Gabby
3 Troy 2/28/2022 7895554444 4579 Garden St Male Troy
4 Johnny 4/11/2022 1112223333 123 Harris St Male Johnny


Employees
List of Employees
Parts n' Tools
Home Screen Menu
Inventory
Customers
Manufacturers
Employees
Log Out
Customer Details
Name of Employee
Address
Phone
Date Joined
Gender
Save
Edit
Delete
Password
Employee List
EmployeeNumbe
EmployeeName
EmployeeDateJoi
EmployeePhone
EmployeeAddres
EmployeeGender
EmployeePassw
1 Gabriel Gonzales 3/29/2022 7895554444 4578 Garden St Male Gabriel
2 Gabby Riley 4/6/2022 7778881111 4577 Grey St Female Gabby
3 Troy Male Troy

Work cited

(Code/Algorithm citation)

MyCodeSpace. (2021, August 27). *Pharmacy Management System C#.NET and SQL Server - YouTube*. Youtube. Retrieved February 28, 2022, from https://www.youtube.com/watch?v=ogS0SfW1pm0