WareHouse Project - Nir Banchik, HackerU Project.

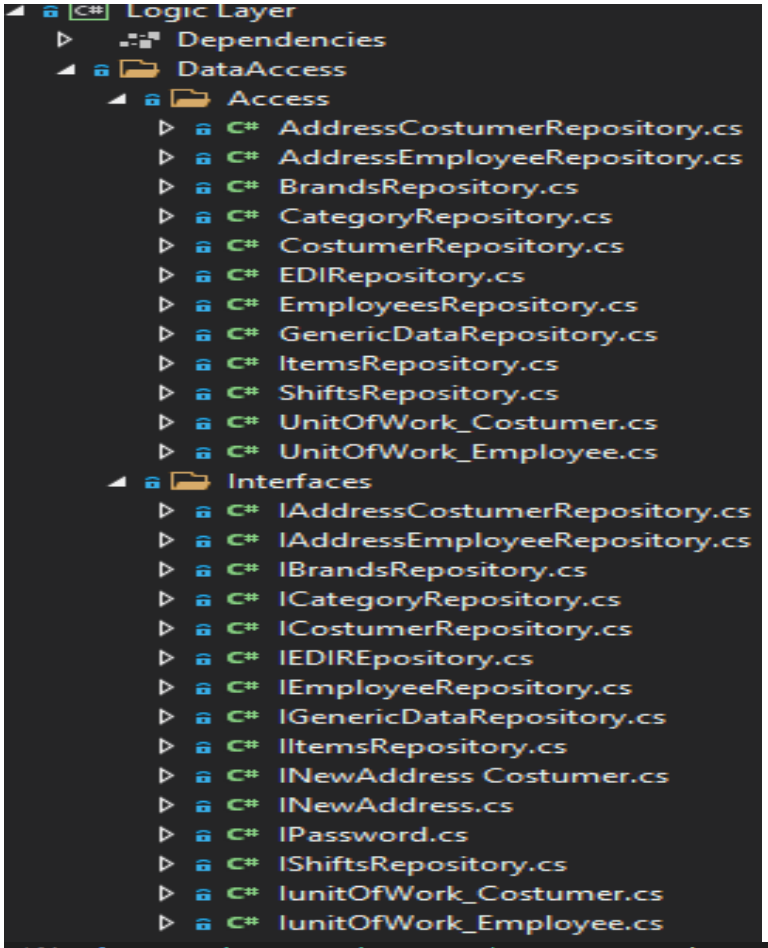
This project came to show the ability to create a Backend System for WareHouse use.
The ability of the system is to manage employees,customers,items and inventory.

This Project Uses:
C# as a programming language.
MS-sql as DataBase
WPF as UI.

In this project I use the “Repository” design pattern with the “Unit of work” design pattern :
I choose to use this pattern for the versatility and generic capabilities of this pattern.
In my design for every ability and table the system has, I've created a specific repository that inherits from a generic repository.
Each repository holds methods for accessing the database to decouple the Database access and the rest of the program.The repository is used for methods that are frequently used,such as get all records,add new records or complex querying the DB.

The Unit of work comes to hand as a buffer between each user of the system.
Holds all the repositories and holds the “CompleteAsync” “and ”DisposeAsync” methods for saving data in the DB and disposing of all the resources of the context.

I have used two types of units, one for employees and one for customers.(they work the same with the difference that the customer's unit holds less access to the db.):



The repository:

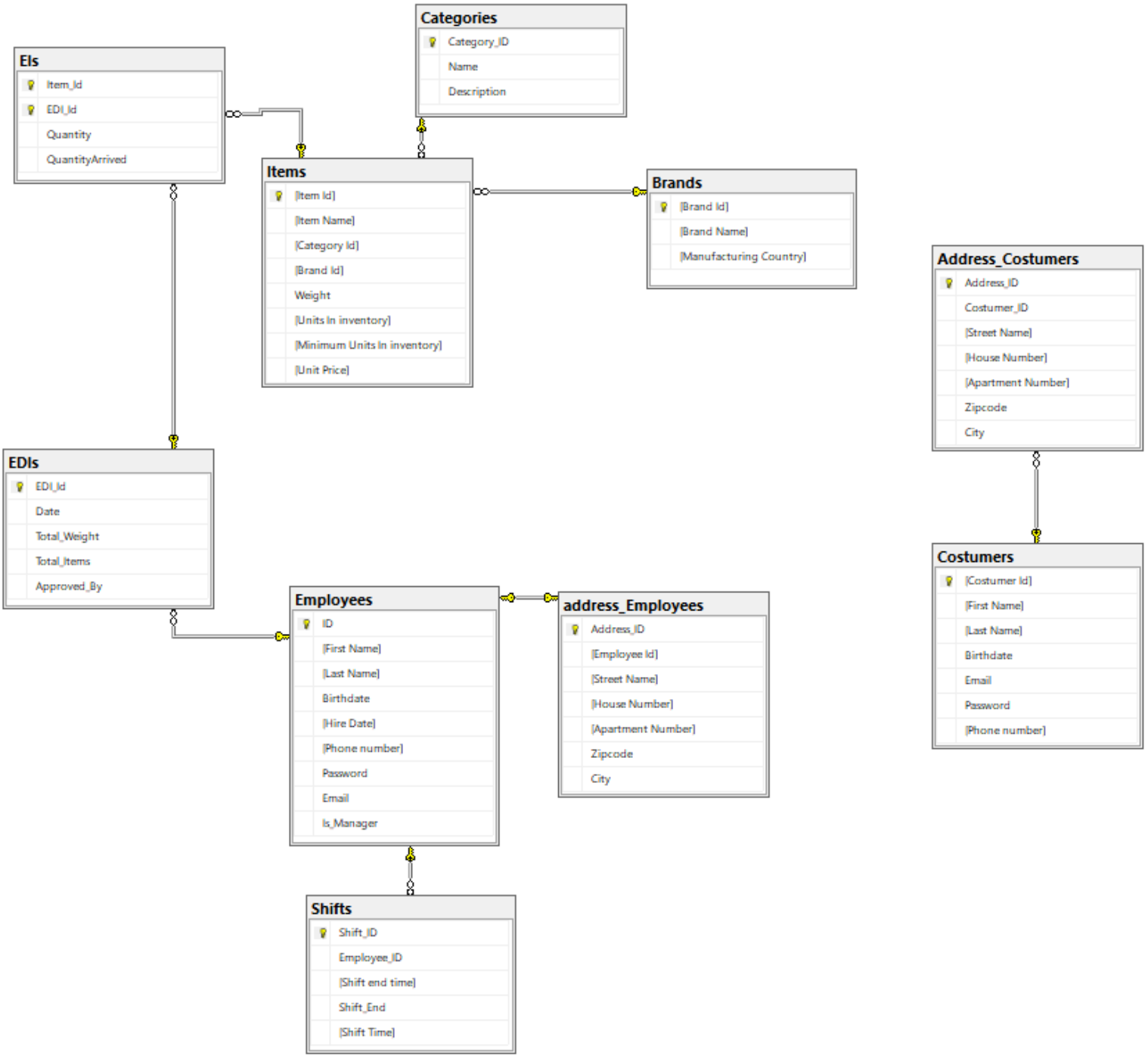
```
public class GenericDataRepository<T> :DbContext, IGenericDataRepository<T> where T : class
{
    protected DbContext context;
    internal DbSet<T> dbSet;

    9 references
    public GenericDataRepository(DbContext context)
    {
        this.context = context;
        this.dbSet = context.Set<T>();


    }
    7 references
    public virtual async Task<bool> Add(T entity)
    {
        await dbSet.AddAsync(entity);
        return true;
    }

    3 references
    public virtual async Task Delete(int id)
    {

        dbSet.Remove(await dbSet.FindAsync(id));

    }

    21 references
    public virtual async Task<ICollection<T>> GetAllAsync()
    {
        return await dbSet.ToListAsync();
    }

    10 references
    public virtual async Task<ICollection<T>> GetByCondition(Expression<Func<T, bool>> predicate)
    {
        return await dbSet.Where(predicate).ToListAsync();
    }
```

The Unit of work:

```
public class UnitOfWork_Employee : IunitOfWork_Employee, IAsyncDisposable
{
    readonly DbContext context;

    2 references
    public UnitOfWork_Employee()
    {
        this.context = new FactoryDbContext();
        this.addressEmployee = new AddressEmployeeRepository(context);
        this.brands = new BrandsRepository(context);
        this.category = new CategoryRepository(context);
        this.employee = new EmployeeRepository(context);
        this.items = new ItemsRepository(context);
        this.shifts = new ShiftsRepository(context);
        this.customers = new customersRepository(context);
        this.addresscustomers = new AddresscustomersRepository(context);
        this.EDI = new EDIRepository(context);
    }

    4 references
    public IAddressEmployeeRepository addressEmployee { get; private set; }
    11 references
    public IBrandsRepository brands { get; private set; }
    11 references
    public ICategoryRepository category { get; private set; }
    10 references
    public IEmployeeRepository employee { get; private set; }
    8 references
    public IItemsRepository items { get; private set; }
    9 references
    public IShiftsRepository shifts { get; private set; }
    3 references
    public IcustomersRepository customers { get; private set; }
    3 references
    public IAddresscustomersRepository addresscustomers { get; private set; }
    9 references
    public IEDIREpository EDI { get; private set; }

    18 references
    public async Task CompleteAsync()
    {
        await context.SaveChangesAsync();
    }

    1 reference
    public async ValueTask DisposeAsync()
```

Model:

In my model I have 10 tables.



To start the program:

Run the Factory.sql file to populate and create the Database.(no need for any migrations)
In my system there are three types of users- Manager,Employee and customer.

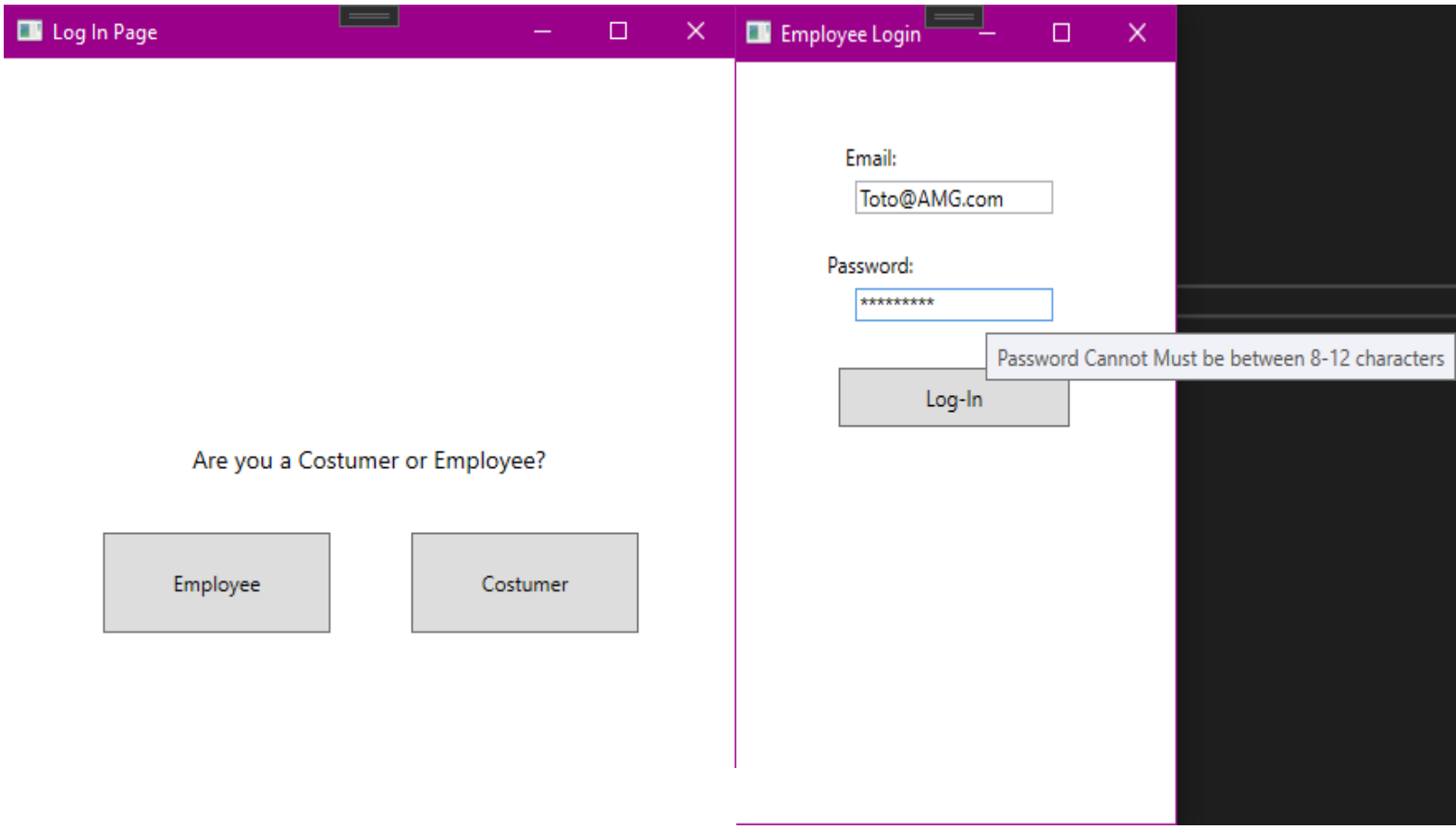
You can choose which type of user to log in with.

Manager :
Email:Toto@AMG.com
Password:123123123

Employee:
Email:TScott@aol.com
Password:123123123

Customer:
Email: Mchen@Yahoo.com
Password: 123123123

The login process checks if there is a user with the email/pass combination.
Every check of a password is encrypted.
Passwords must be between 8-12 characters.

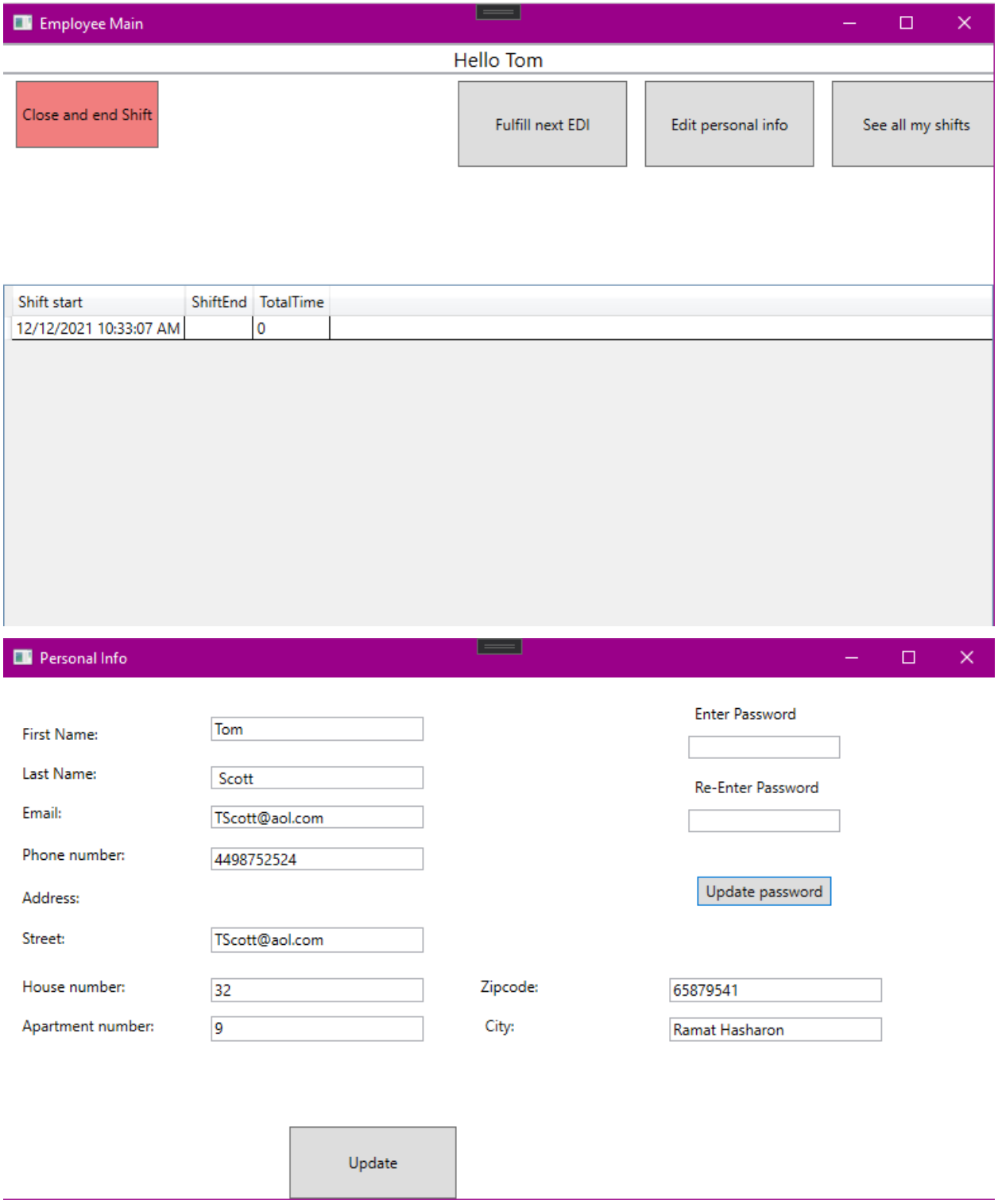


Customer Menu:

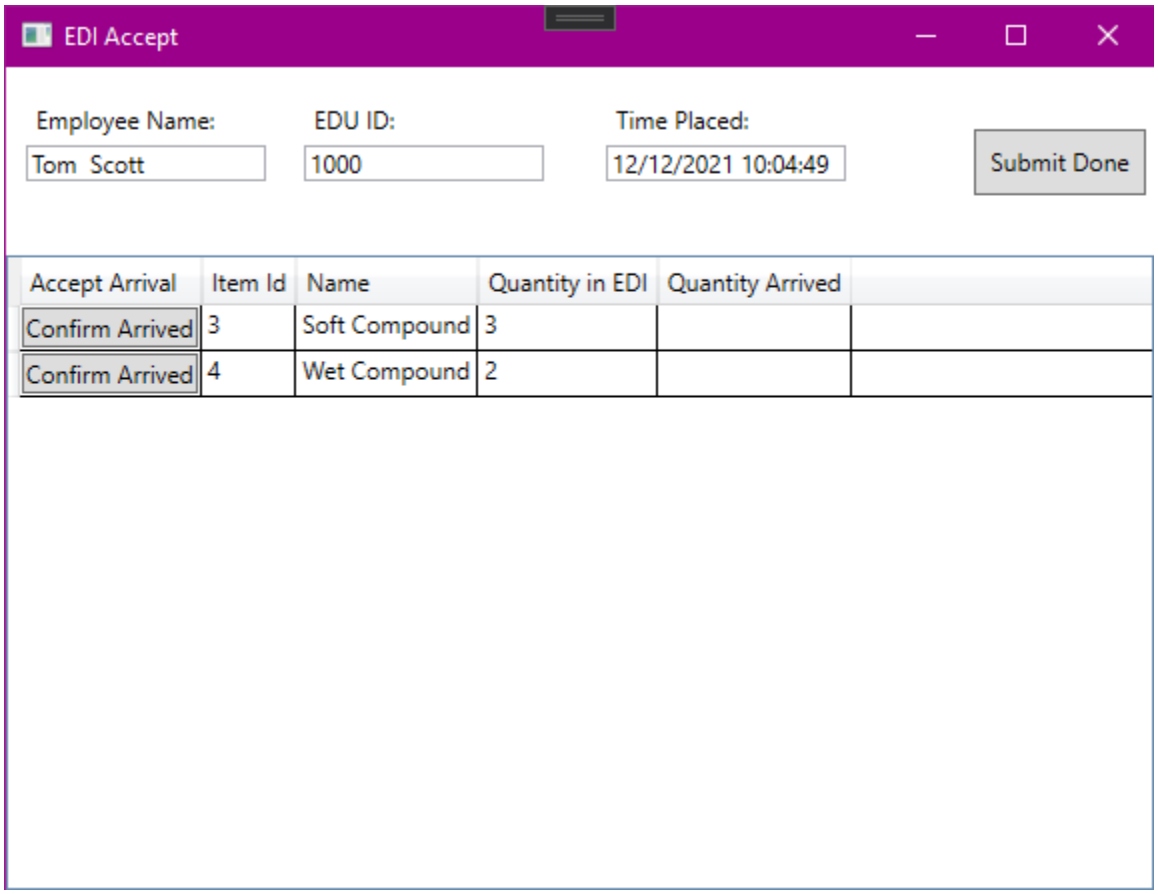
In the customer menu the customer has the ability to create orders from the warehouse(not implemented),see past orders(not implemented) edit personal information

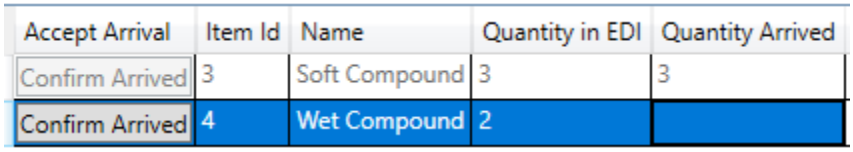
Employee Menu:

In the employee menu,the employee can see his own shifts, edit personal data and change their password, and the main ability of the employee is accepting EDI to the warehouse.

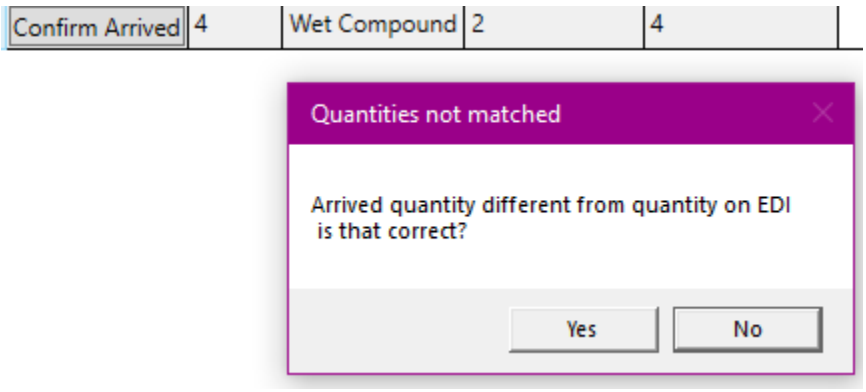


When the employee presses the “fulfill next EDI” , the employee will get the next unfulfilled in-EDI.
The employee needs to “ enter the number of items in the “real” shipment and accept the number.





If the quantity arrived is not the same as quantity in EDI a message
Will pop and ask from the employee to accept its not an error

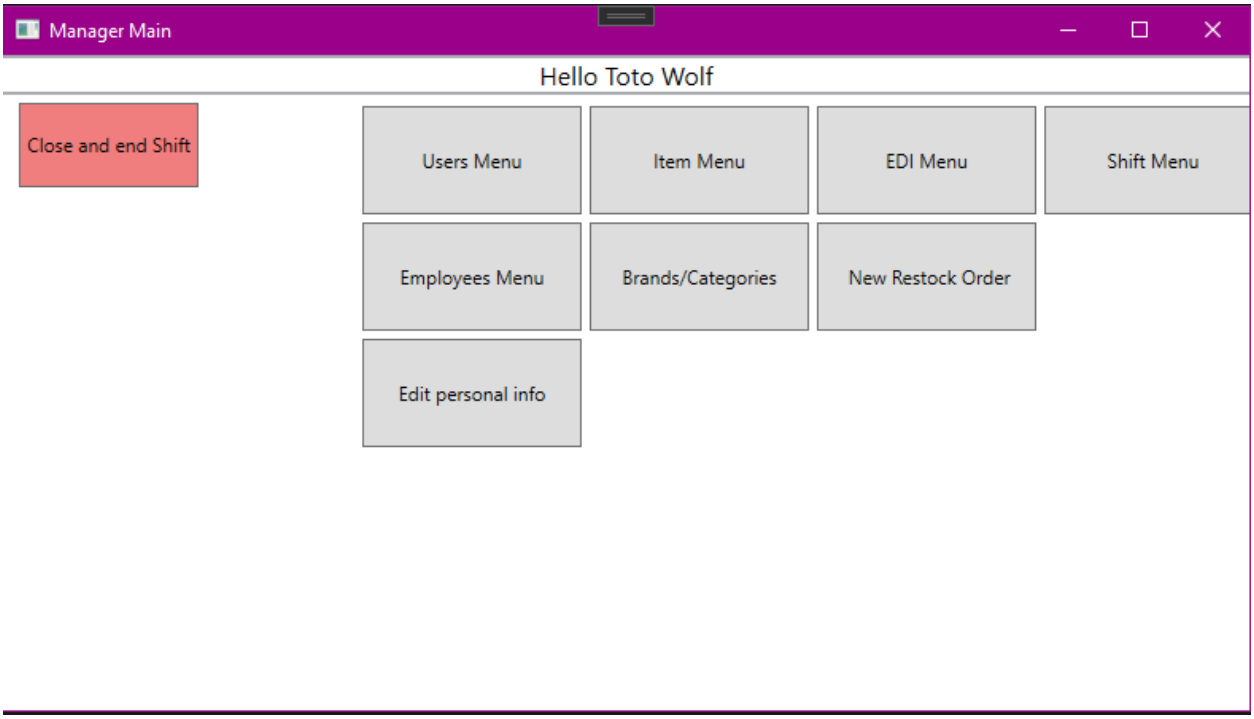


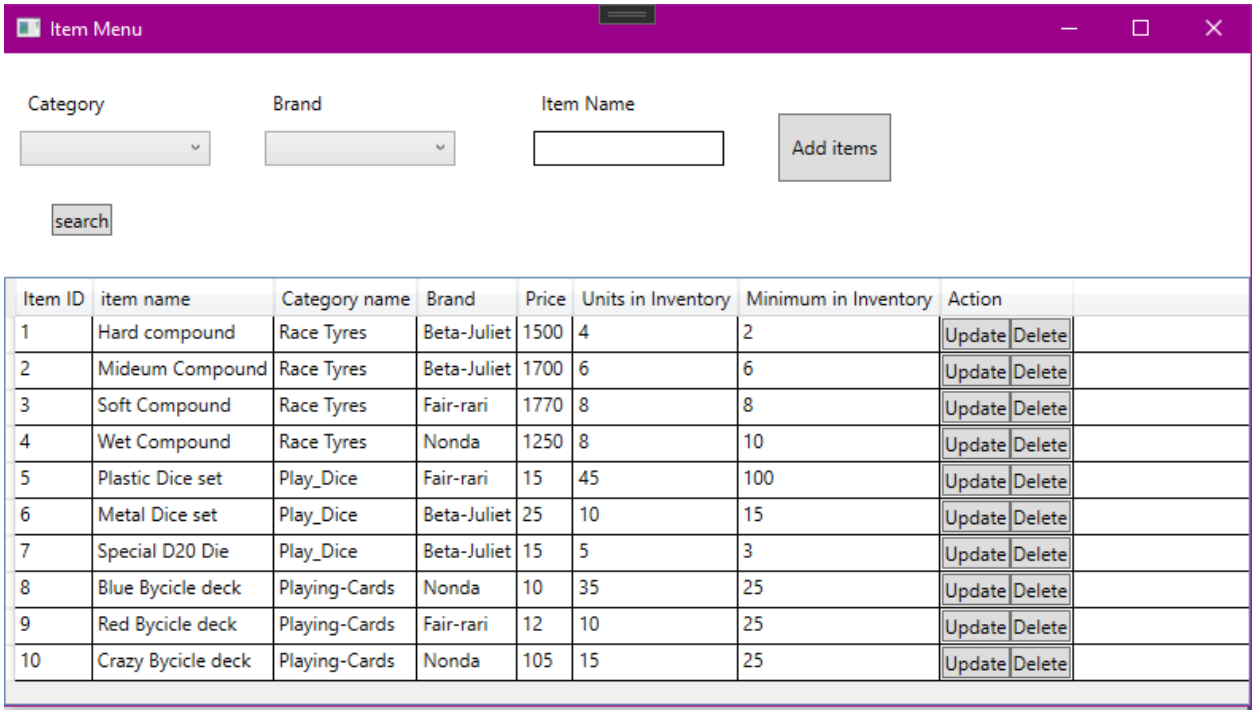
When the employee finishes counting all items, the items will update in inventory.

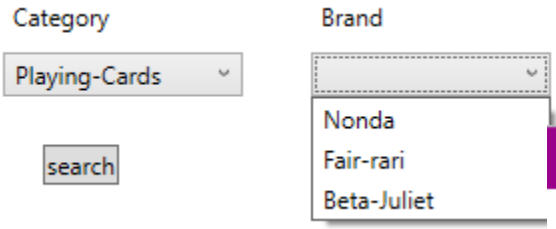
Manager Menu:

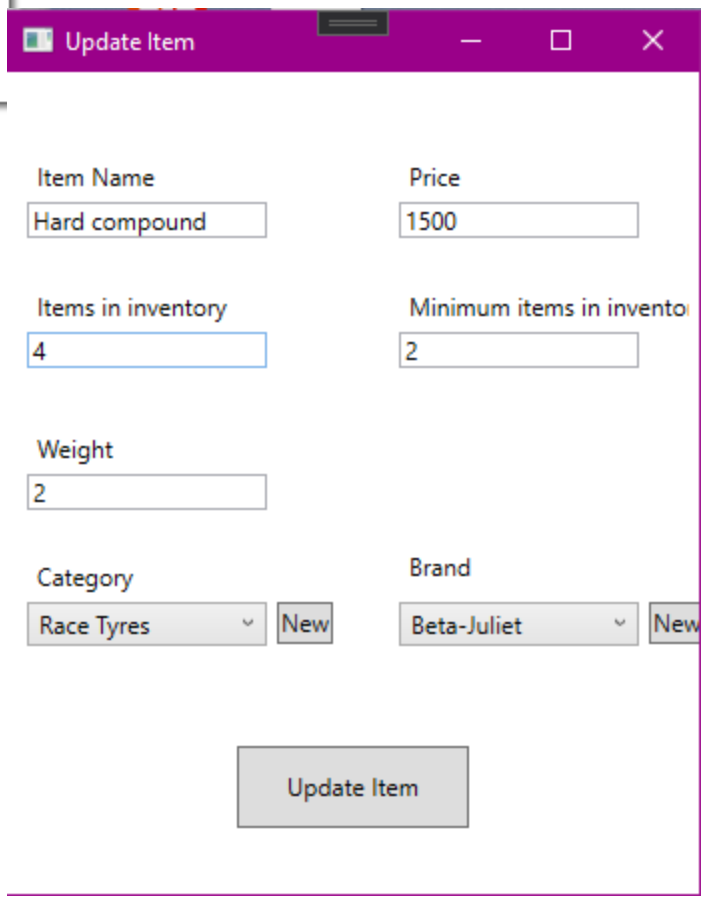
The manager can:

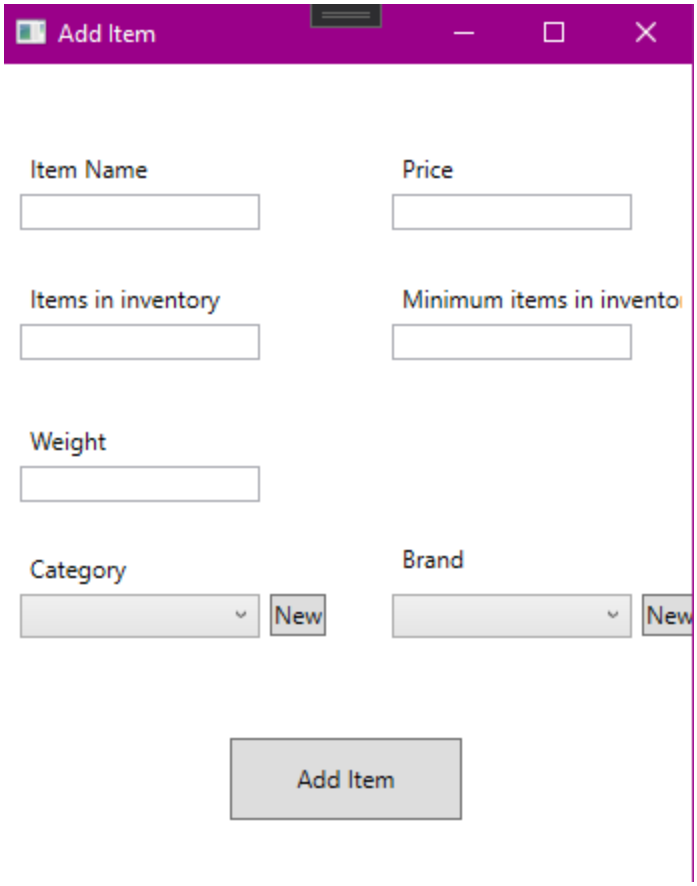
1.Add/update/delete Items in warehouse
2.Add/update/delete Brands and Categories
3.Add/edit employees and Customer
4.see shifts for all employees.
5.create a new Restock order for item.
6. See all EDI's that have not been fulfilled and "closed ones and who fulfilled them.



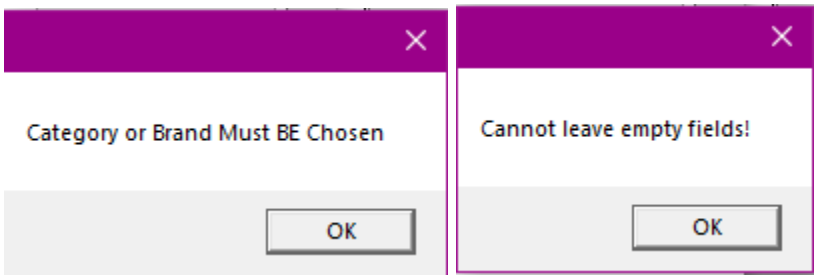

Item Menu
Category
Brand
Item Name
Add items
search
Item ID
item name
Category name
Brand
Price
Units in Inventory
Minimum in Inventory
Action
1 Hard compound Race Tyres Beta-Juliet 1500 4 2 Update Delete
2 Mideum Compound Race Tyres Beta-Juliet 1700 6 6 Update Delete
3 Soft Compound Race Tyres Fair-rari 1770 8 8 Update Delete
4 Wet Compound Race Tyres Nonda 1250 8 10 Update Delete
5 Plastic Dice set Play_Dice Fair-rari 15 45 100 Update Delete
6 Metal Dice set Play_Dice Beta-Juliet 25 10 15 Update Delete
7 Special D20 Die Play_Dice Beta-Juliet 15 5 3 Update Delete
8 Blue Bycicle deck Playing-Cards Nonda 10 35 25 Update Delete
9 Red Bycicle deck Playing-Cards Fair-rari 12 10 25 Update Delete
10 Crazy Bycicle deck Playing-Cards Nonda 105 15 25 Update Delete

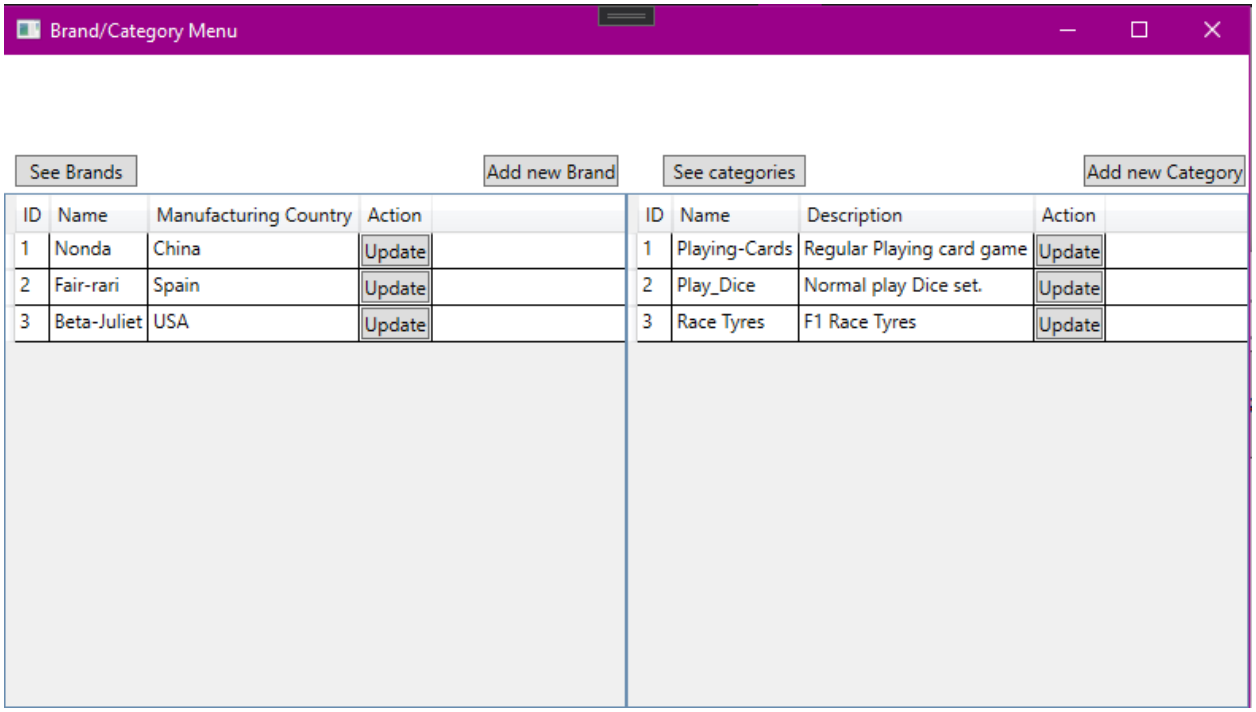

Category
Brand
Playing-Cards
search
Nonda
Fair-rari
Beta-Juliet


Update Item
Item Name
Hard compound
Price
1500
Items in inventory
4
Minimum items in invento
2
Weight
2
Category
Race Tyres
New
Brand
Beta-Juliet
New
Update Item

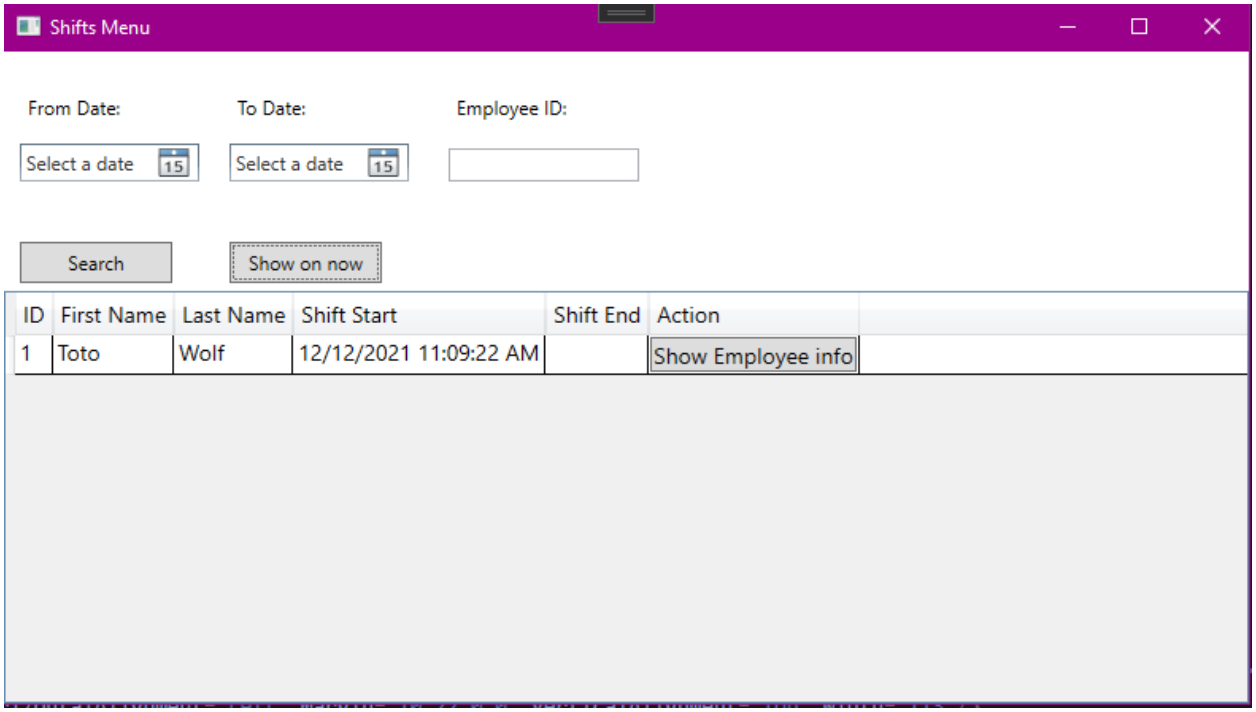


On the item menu you can search by Brand or category and item name.
Through the new item screen you can choose an already exists brand and category or create new one.
You can't finish creating without choosing brand and category or empty fields



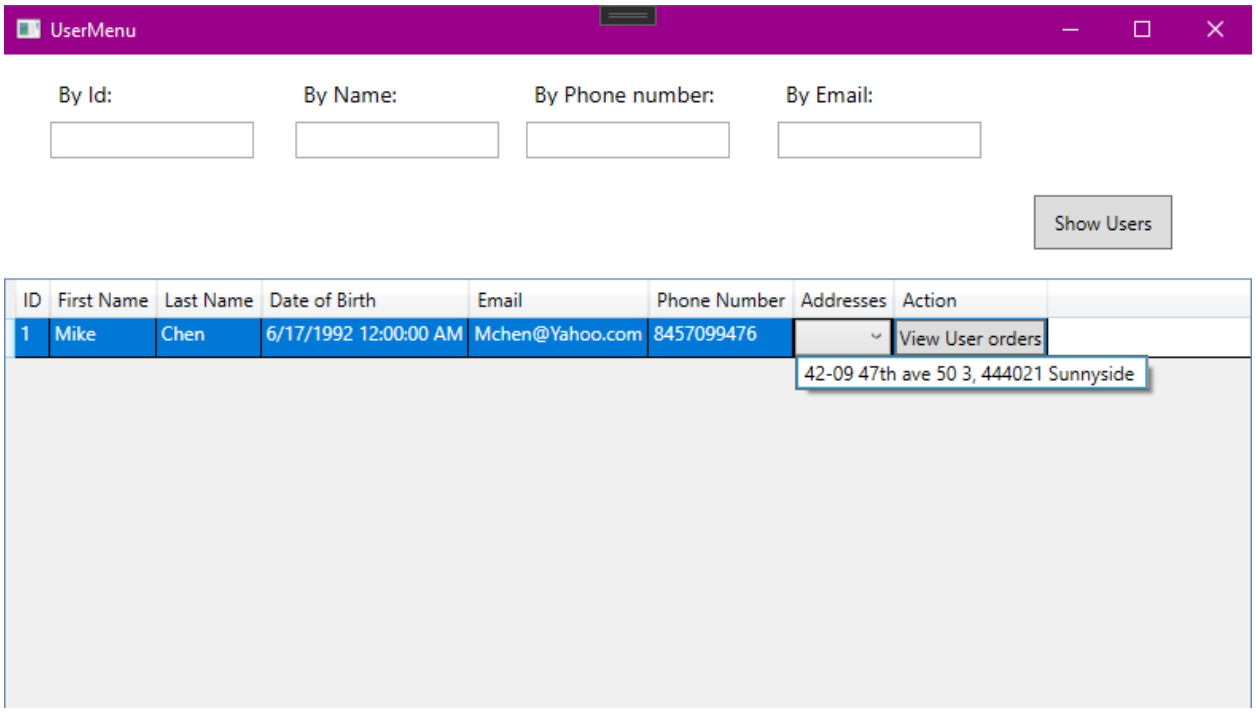


On the Brands/Category you can update,search and add new brands and categories.

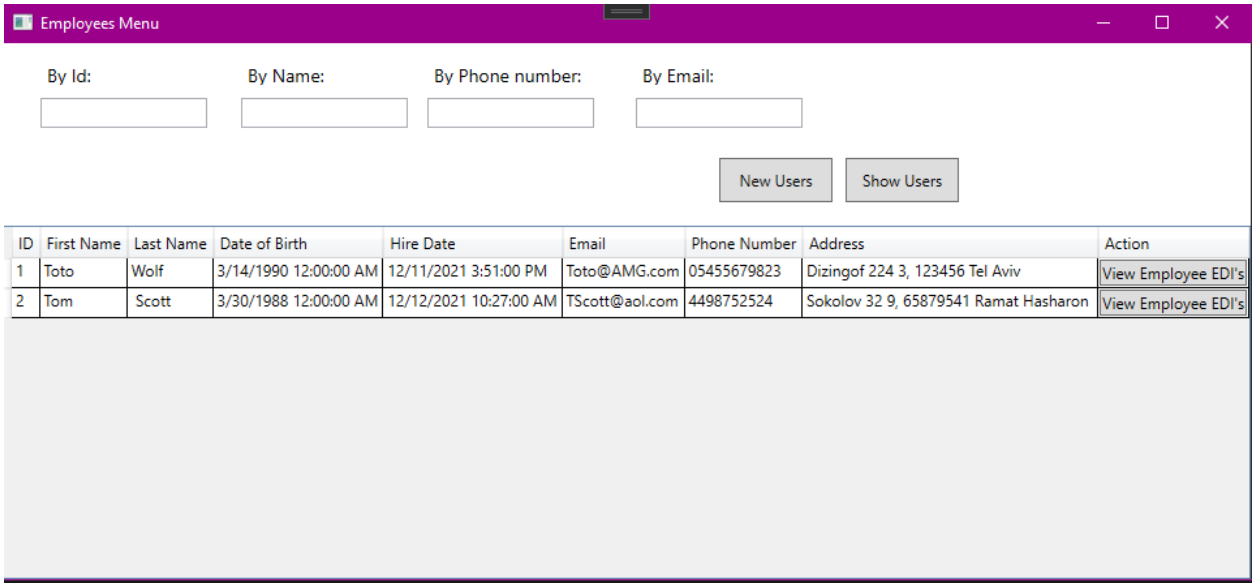


On the shift menu you can see who is on now.And search by employee id or date of shifts

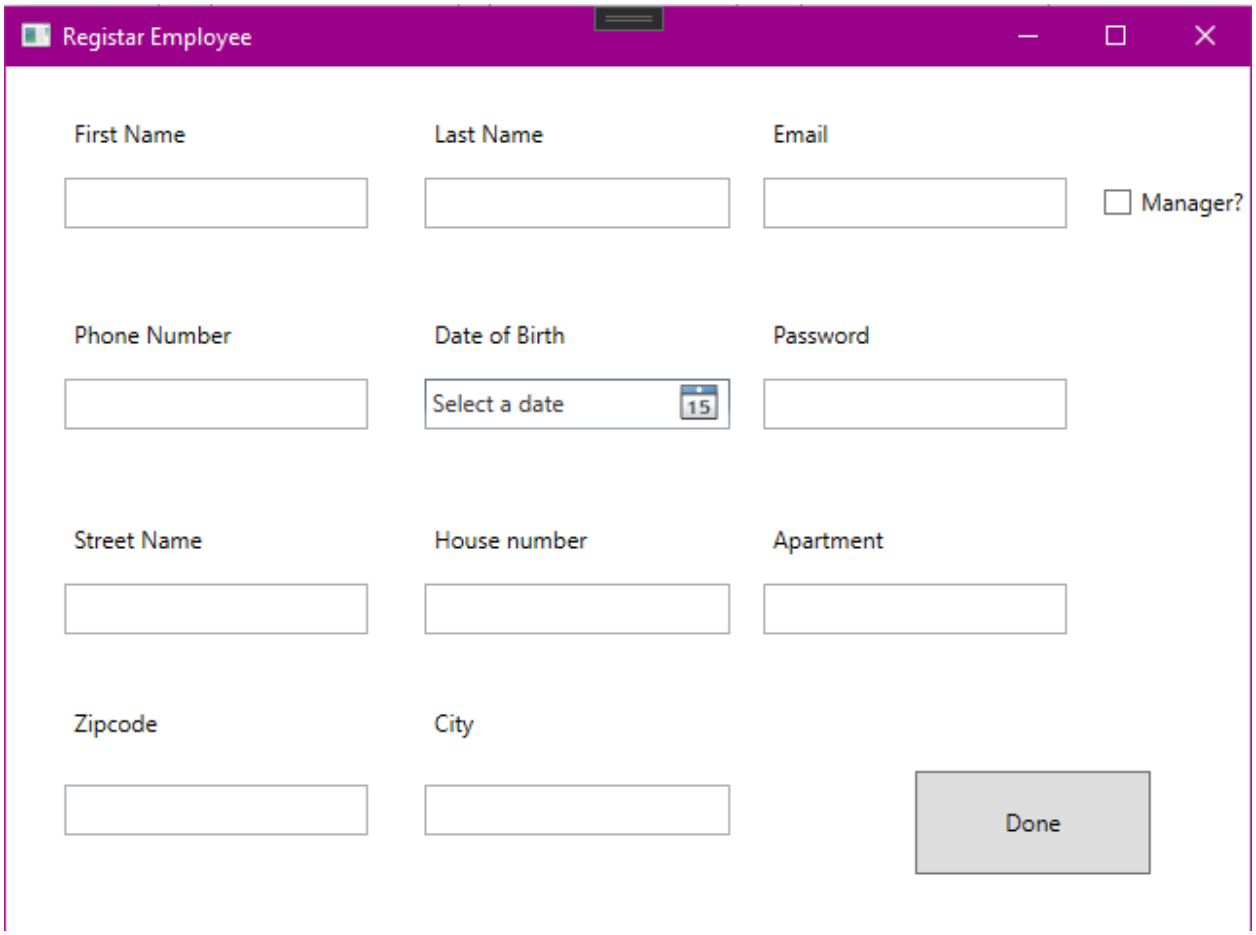
---

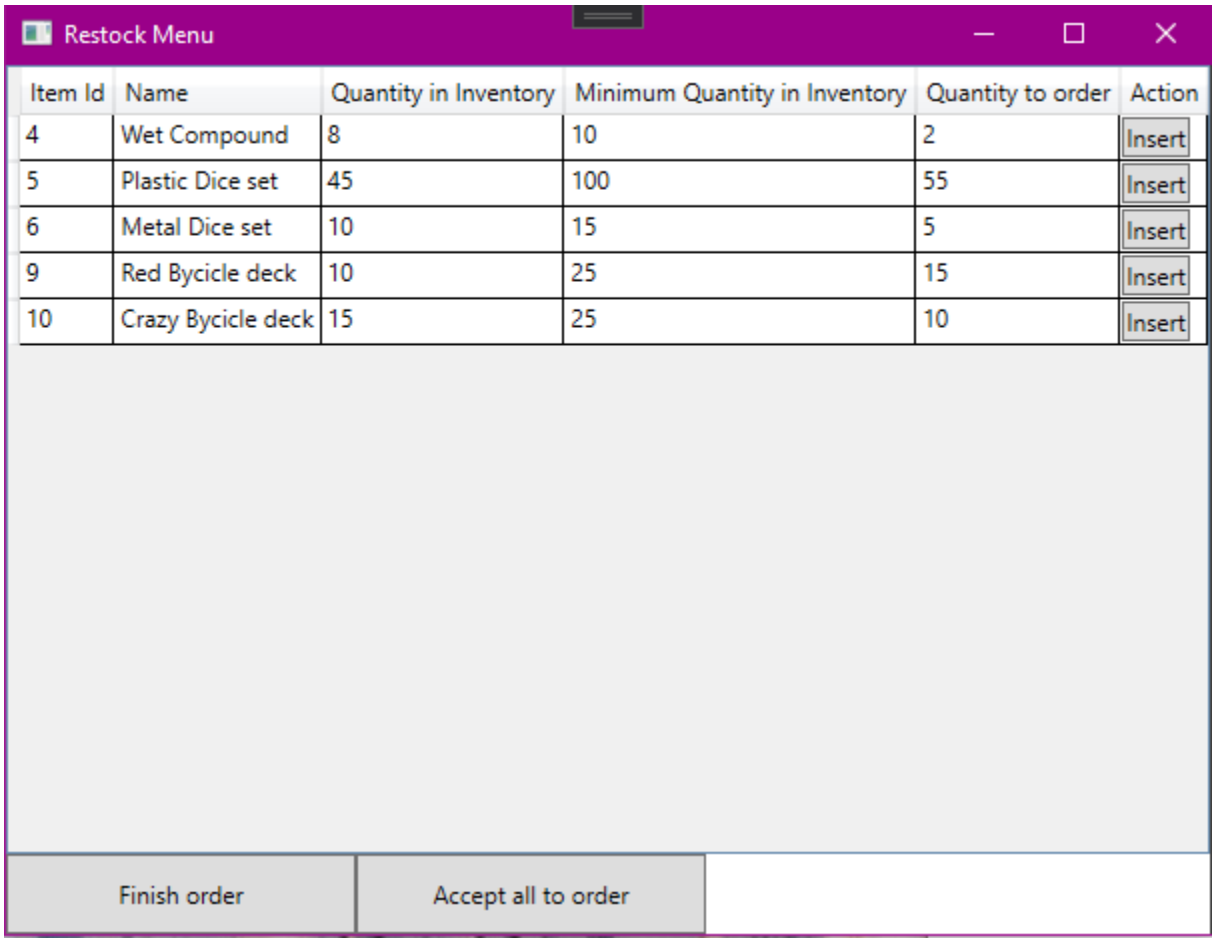


On the user menu you can see all users or search by parameters.

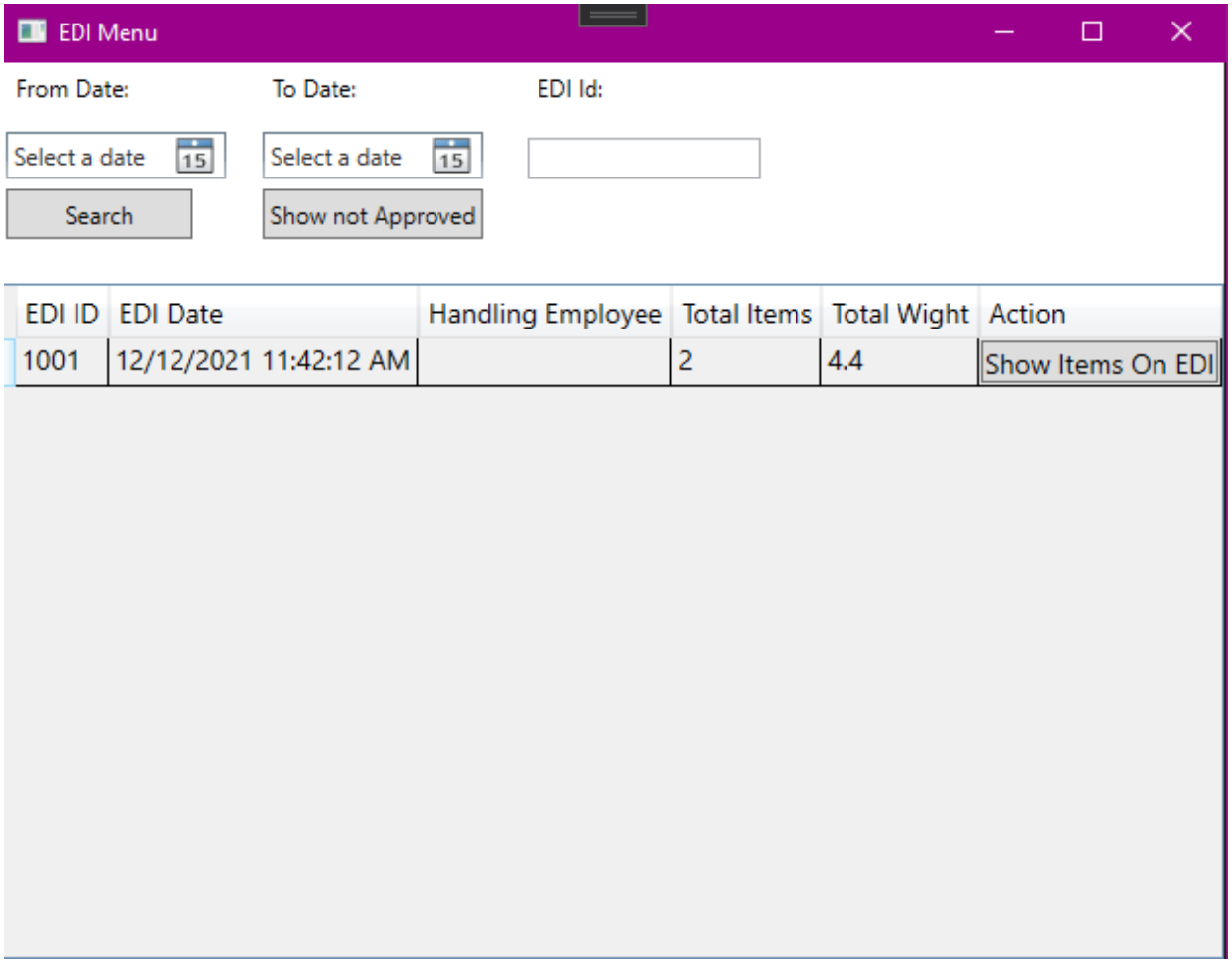


On the employee page you can see all employees and create new users.(same as register of customers)





When entering the Restock menu you can see all the items in the warehouse with quantity in inventory lower than minimum quantity.
You can't see items that need to be ordered that have already been ordered but EDI has not fulfilled yet.
You can choose the amount you want to order on “Quantity to order” with the default on the difference between the amount in inventory and minimum amount.
You can accept all items to order like that or order individual items.



On the EDI menu you can see all in-EDI by date and by id.
You can show only EDI's that have not been accepted by the employee yet.
On arrived edi you can see the Quantity arrived .