**Methods Document**

## Login Methods

**Login Class**

Create Login Method for User to login.

**CheckValidUsernamePasswordField(string username, string password)**

//This method checks the username and password field is valid. Can only enter Alphabets, Numbers, //periods, and underscore. Cannot be longer than 15 characters.

Captures user information and validate if valid user.

Cannot be invalid.

Cannot be more than 15 characters.

Do this method before validating user with userlist.

No need to obtain information about a file when the username and password is invalid.

**Deserialize method()**

//This method de-serialize a binary file from the the filesystem. After de-serialization, it saves the file //into a List of User Objects.

Create a new List of User Objects to keep track of users.

Deserialize Binary File from filesystem and set to List of Objects.

Return List of Objects.

Add Exception Handling (FileNotFound Exception Serialization Exception,etc…)

**Serialize method(User user)**

//This method serialize a User to a binary file. The method first deserializes the users and gets the //current list of user. The method then adds the added user to the List of User Objects. The list is then //serialized onto a binary file.

Add parameter for User objects. User objects contain username and password.

Deserialize Current Binary File. It will return a List of current users.

Check if User is valid

Check if User is current on binary file

{

Do not serialize object.

}

Else //User is not in file

{

Obtain deserialize list.

Add Addition user to de-serialized list

Return de-serialized list.

}

Add Exception Handling (FileNotFound Exception Serialization Exception,etc…)

**CheckValidUser(User user, List<User> userlistarray)**

//This method checks if the User is in the binary file. Binary file is deserialize and put into a List of User Objects. The method checks to see if User in the List of User Objects.

Use parameter for Current User object and List<User> list.

User file is de-serialized and converted to List of user Objects.

Username and password is valid against List of user Objects.

If username and password matches

{

Return true;

}

Else Create Message saying username is not found.

{

Return false

}

Obtain current list of users by running Deserialize method.

## Admin Methods

**User Class**

**AddUser(User user)**

//This method adds to User to List of User object. Binary file, of current user, has to be deserialized to a List of User Objects. User is then added to List of User object and serialized.

Binary File is deserialized into a List of Objects

If(User is not in List of Objects)

{

User is added to de-serialized List of Objects.

Serialize a List of User Objects to file.

}

Else

{

Show Message stating the User is already in the system.

}

**RemoveUser(User user, List<User> userlist)**

//This method removes a User from the List of Users. Temp list is used to keep track of User that are not removed.

Create new List of Users as temp list.

Current list will be saved to a List of Objects.

If(User is in userlist)

{

//Do nothing

}

}

Else

{

Add user to templist of User Objects.

}

Return temp list of objects.

**AdminUser Class**

**AddProduct(Product product, List<Product> productlist)**

//This method adds a product to a list of Products

If(product is not in product list)

**{**

Add product to product arraylist.

}

Else

{

Show Message stating the User is already in the system.

}

**RemoveProduct(Product product, List<Product productlist)**

//This method removes a product from a list of products.

{

If(product is in list) Compare Product Key fields(Key fields not decided) to product list.

{

Remove Product;

}

Else

{

MessageBox stating that the product does not exist.

}

**AddAdminUser(AdminUser adminuser, List<Admin> adminuserlist )**

//This method adds an admin user to admin Userlist

Deserialize current list of Admin Users.

If( AdminUser is not in AdminUserlist)

{

Serialize a Admin User to and add to AdminUserlist.

}

Else

{

Show Message stating the User is already in the system.

}

**RemoveAdminUser(Adminuser adminuser)**

//This method remove an admin user to admin Userlist

Run UserNamePassword valid method to ensure user field has valid characters. Just letters and alphabet.

If (Admin user is in adminuserlist)

{

If( Valid admin userlist)

{

Remove admin user.

}

}

Else

{

Message box saying Admin userlist isn’t found.

}

**IsAdminUser(string username, string password)**

//This method checks for a valid Admin User

Add parameter for username and password

If ( username and password is Admin User list)

{

return true

}

else

{

return false

}

## Product

Add category as Enum

Create Product class that has Properties to get and set feature of a product.

Add interface later.

Product class

Int weight=0;

Public Product()

{

}

Public int Weight(get;set;);

**Sort List<Product> CategoryMethod(category, List of Products)**

{

If(category==”All”)

{

Return list of products.

}

Create a new list<products>

Iterate through list of products with for loop.

{

If category is found in list of products

{

Add category to new list.

}

}

Return list.

}

**Update ProductMethod(Product product, List<Products> product list, update column ,update columnvalue)**

{

//Maybe get all products before hand.

Productlist samplelist = productlist.

Iterate through product list

{

If(product is in product list)//use product id)

{

Update column in productsample list with column value.

Update property of Product object for value entered in parameter.

Update QueryMethod(Product)

}

}

Return productsample list.

}

**Update QueryMethod(Product)**

{

Need SQL Connection.

String updatequery=”UPDATE …”

}

**UpdateMultipleProducts()**

{

}

**Public Product SearchforProduct(SearchQueryProductname, category, List of Products)**

{

Iterate through productlist

{

If(SearchQueryProductname =list.productname && category = list.category)

{

Return that product

}

Return null;

}

}

Public void removeproductfromList(**ArrayList productidsList, List<product> products, DataGridview datagrid)**

**{**

**May run productid methods in method.**

**Make new list of products.**

**Iterate through products list**

**Iterate through productid list**

**If(productidList.productid!=products.productid)**

**{**

**Newproductlist.add(productidList.productid);**

**}**

**Class.ProductList = newProductList;**

**Int datagridView = RemoveProduct(ArrayList productidsList, DatagridView datagridview)**

**Product.filldata(Selectquery, datagridView);**

**//Return new list of products;**

**}**

**Public DatagridView RemoveProduct(ArrayList productidsList, DatagridView datagridview)**

{

//Get current Productfrom datagrid.

//Find currentUser and current product for order to cancel.

delete query string=””;

For loop for each productidsList;

{

Convert current ProductID to integer.

If(i==0)

{

Construct delete query string =”Delete from table where ProductID IN (productid)”

}

Else

{

deleteQueryString+=”, “ + productID+

}

}

delete query string+=”);”

Execute command using SQL command.

**removeProductFromDatabase(string deletequery)**;

Int datagridview = Run filldata method to populate datagrid.

Return datagridview;

}

Public void **removeProductFromDatabase(**delete query string**)?**

{

Execute command using SQL command.

}

# Orders( when a customer buys an item, an order is generated)

Order List will contain username, List<Products> productsbought, quantity);

Public class Orders

{

List of Orders Properties.(get;set;)

User property

List of productsbought property

Public Orders()

{

}

**Public Orders(User user, List<Products> products, quantity)**

{

Initialize orders.

Run Add Order Method();

}

**Public void AddOrders(User user, Product product,quantity)**

{

Bool userfound =false;

Iterate through orderlist

{

See if(user is in OrderList)

{

Find user and add product to orderslist.

Bool userfound=true;

}

}

If(userfound==false)

{

List of Orders.

Add User(Customer) and product to List of products to order list based off of quantity.

}

}

**Public void Add Multiple Orders()**

**Public List<Order> CancelOrder(List<Orders> orderslist, List<Products> products, DatagridView** datagridview**)**

{

Get current order from datagridview.

Find currentUser and current product for order to cancel.

//Iterate through datagridviews. And find product id that equal what in the product list.

List<Orders>neworderlist = new List<Orders>();

List<int> ordernumberlist in list = new List<int>();

Iterate through datagrid selected rows

{

Remove selected rows from datagridView

* foreach (DataGridViewRow row in dgvGlobalAssosciation.SelectedRows)

        if (!row.IsNewRow)

            dgvGlobalAssosciation.Rows.Remove(row);

**Iterate through datagridview selected rows**

**{**

**Ordernumberlist.add( datagrid,selectedrow.Cell[5].Value.toString();**

**Datagridview1.Rows.Remove(Datagridview1.Rows(Datagridview1.SelectedCells.Item(i).RowIndex)**

**}**

}

List<Orders> finalorderlist = new List<Order<>();

Iterate through List of Orders

Iterate through ordernumber list

If(ordernumber!= List.Ordernumber)

{

Finalorderlist.add(Orders[i];)

}

//Remove from List<Orders>

Return list of orders.( finalorderlist)

}

Public void findProductIDfromOrders(List<Orders> orderlist, this.datagrid)

{

List<Orders>neworderlist = new List<Orders>();

Iterate through datagrid selected rows

{

Remove selected rows from datagridView

}

}

**Public void CancelMultipleOrders()**

{

}

**Public void CancelAllOrders()**

{

}

**Public void ViewAllOrders(List<Orders> orders)**

{

Return orderlist

Iterate through orderlist

Return usernames and Product in list of orders.

}

**Public void ViewCustomerOrders(User user List<Orders> orders,)**

{

List<Orders finalorderlist = new List<Orders>();

Iterate through orderslist

{

If user is found in orderslist

{

Return product list in orderslist object.

}

}

}

**Public void update orders(Order orders, dateordered, date updated)**

{

}

**Public getAllCurrentOrder()**

{

Return static orderlist.

}

**Public view displayOrderinGrid(DataGrid datagrid, List<Orders> orderlist)**

{

Create columns for orders.

Display orders prelist.

}

**Register click methods()**

Assign values in textbox

To create customer.

Then run customer.AddCustomer which add customer to CustomerList.

# Customer Class

Create properties for fields below.

Int customerid

String firstname;

String lastname

Date Lastpurchase date

Date LastLogin

Int numberofOrders

Int numberofProducts

List<Orders> customerorders.

List<Products> product list

Public static List of Customers

Customer()

{

Assign all data members

}

Public void AddCustomer(Customer customer)

{

CustomerList.Add(customer)

Add customer to Customer Database.

}

Public void ViewCustomer(List<Customers> customerlist, this,datagrid.)

{

Get all customers from database

SQLCommand

}

Register(All fields of customers,)

{

}

# Account

Get Username and email if found

View Orders Button goes to View Orders Form/

Change email button

**Change email method()**

Change password button

**Change password method();**