Name - Asis Rai

Student ID Number – 6528683

340CT – Software Quality and Process Management

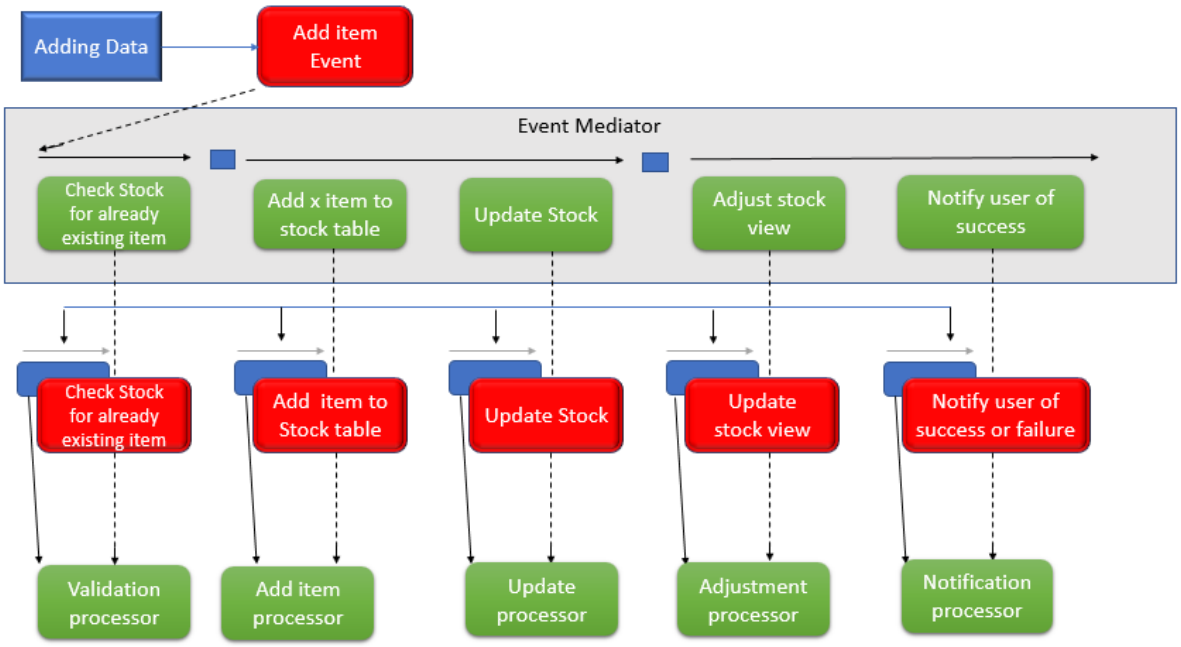
Module Leader – Yih-Ling Hedley

Task 1

Annotated Mediator diagrams of chosen functionality with annotation:

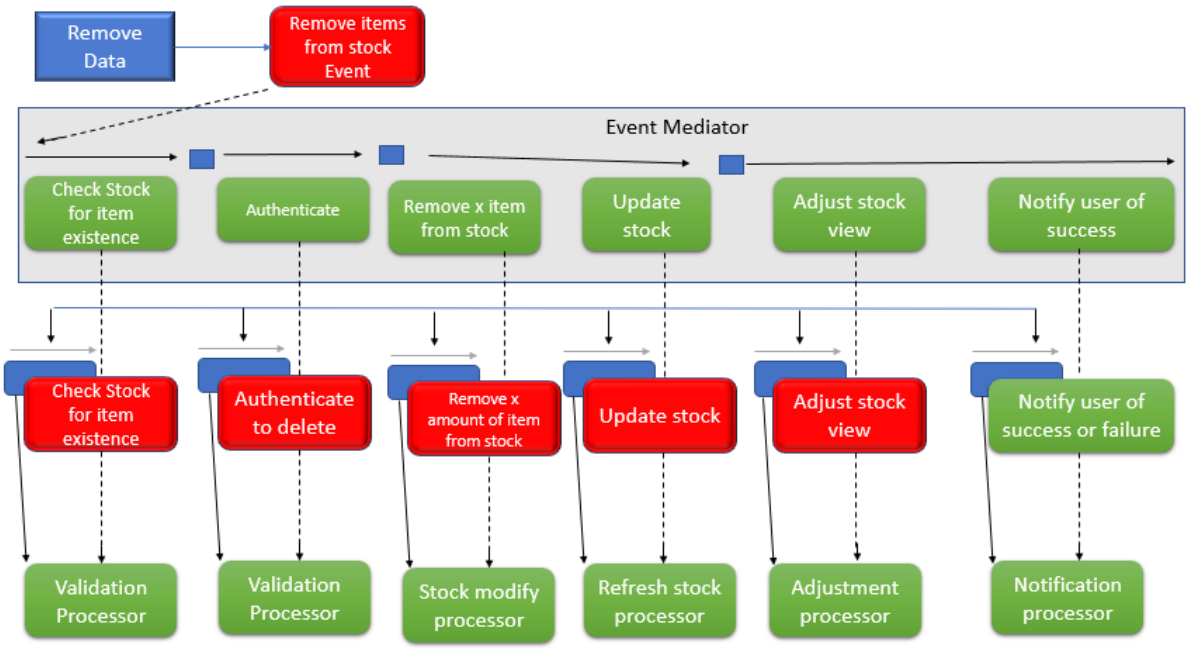
User Story 1 & 3: Adding, removing (Only Manager), Updating and Searching stock items for Manager and Stock Assistant.

Adding:



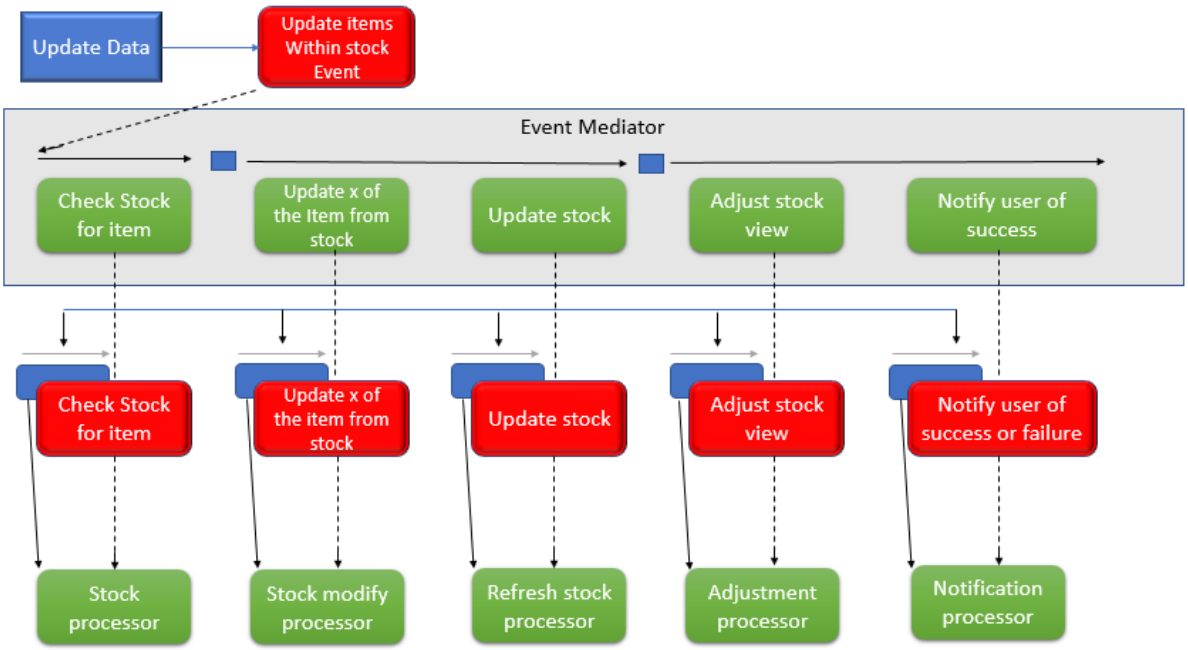
- Annotation: The Event Mediator Diagram above is the process when the user tries to add items into the database.
  - The first process of the process to be executed by the mediator is the validation process which will check if the details entered by the user already exists in the database. If it does, then it will notify the user with a message box saying the item couldn't be added because it already exists in the system. This prevents duplication.
  - If items entered are not in the database already, the mediator will execute the second processor which will Add item into the database. This will take all the values entered by the user and put them into their right tables. After the second processor is finished, the third processor is activated because the steps are concurrent. The third processor will update the stock which means that the new item is added below the items that may be already in the table. This will make sure that every item gets a unique id.
  - Fourth processor is called Adjustment processor, this will get the updated stock from the Table and display it on the Gird View of the user to see the most updated items in the table and finally the Fifth processor is executed concurrently, which will notify the user of success item being added to the system, or failure if there Is any other error such as connection problem with the database, through a Message box pop up.

Removing:



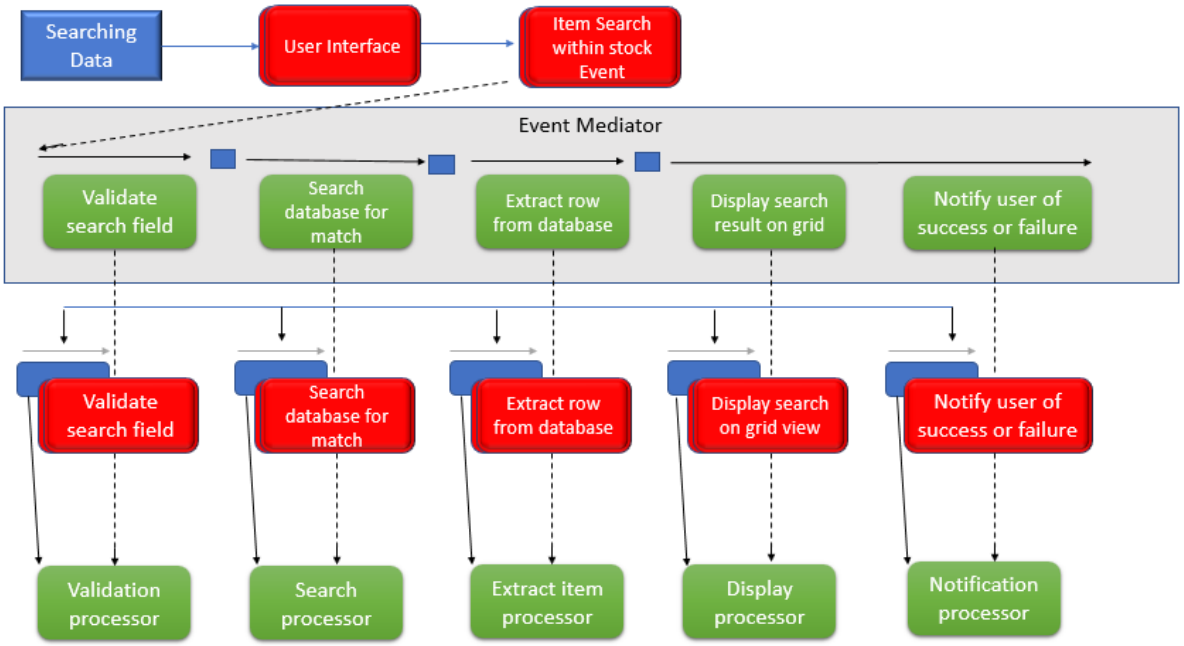
- <u>Annotation:</u> The Event Mediator Diagram above is the process when the user tries to Remove items from the database.
  - The first process will check if the item/items the user is trying to delete does exists in the system.
  - The second process will execute if the first is successful, it will ask for the user credentials to confirm that only Manager is able to delete it.
  - The third process will execute if the user authentication is successful, this processor will take the item name and item code entered in the data and find it on the database, and delete it.
  - The fourth process runs concurrently with the third, this process will update the database/stock when the item is deleted.
  - The fifth process will get the updated stock from the table it deleted from and display it on the Gird View of the user to see the most updated items in the table and finally the sixth processor is executed concurrently, which will notify the user of success item being delete on the system, or failure if there Is any other error such as connection problem with the database, through a Message box pop up.

Updating:



- <u>Annotation:</u> The Event Mediator Diagram above is the process when the user tries to update items in the database.
  - The first process will check if the item/items the user is trying to update does exists in the system.
  - If the first processor is successful, this processor will take the items entered in the data and find it on the database, and update it.
  - The third process runs concurrently with the second, this process will update the database/stock when the item stock is updated.
  - The fourth process will get the updated stock from the table it updated from and display it on the Gird View of the user to see the most updated items in the table and finally the Fifth processor is executed concurrently, which will notify the user of success item being updated on the system, or failure if there Is any other error such as connection problem with the database, through a Message box pop up.

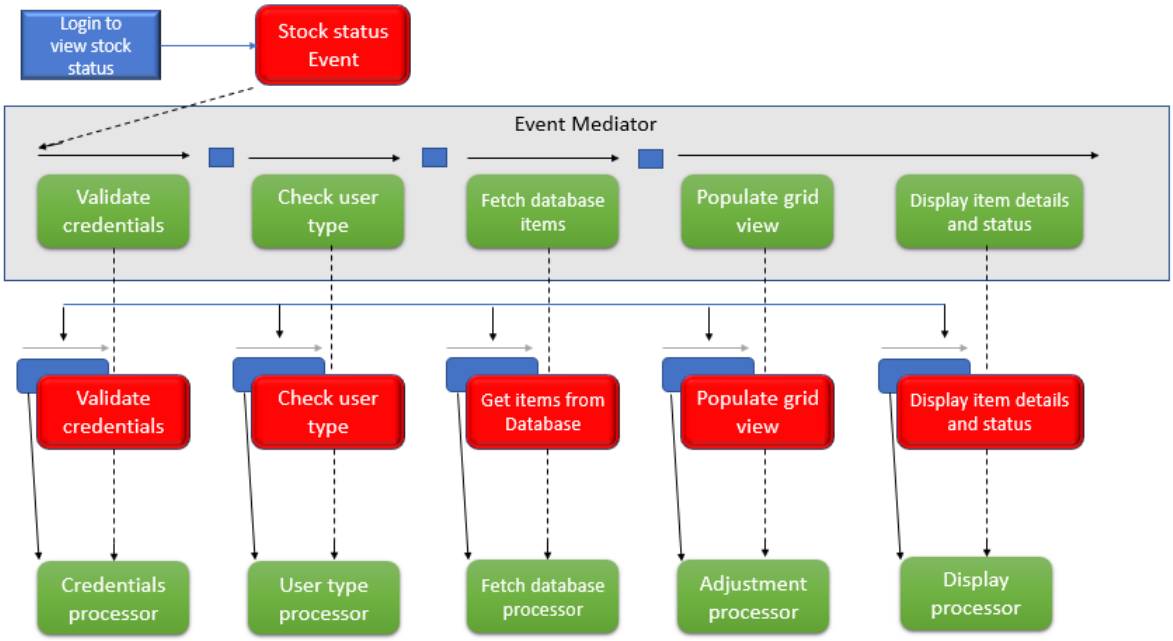
Searching:



- <u>Annotation:</u> The Event Mediator Diagram above is the process when the user tries to search items from the database.
  - The first process validates/checks if the user has left search field empty, because if there is nothing, there is nothing to search for.
  - The second process will check if the item/items the user is trying to search for, does exists in the system.
  - The third process will get the data set from the database if it exists.
  - The fourth process will get the searched data set from the database and display it on the Gird View of the user to and finally the Fifth processor is executed concurrently, which will notify the user of success item being found on the system, or failure if there Is any other error such as connection problem with the database, through a Message box pop up.

User Story 4: All staff be able to Login using their credentials and view stock status

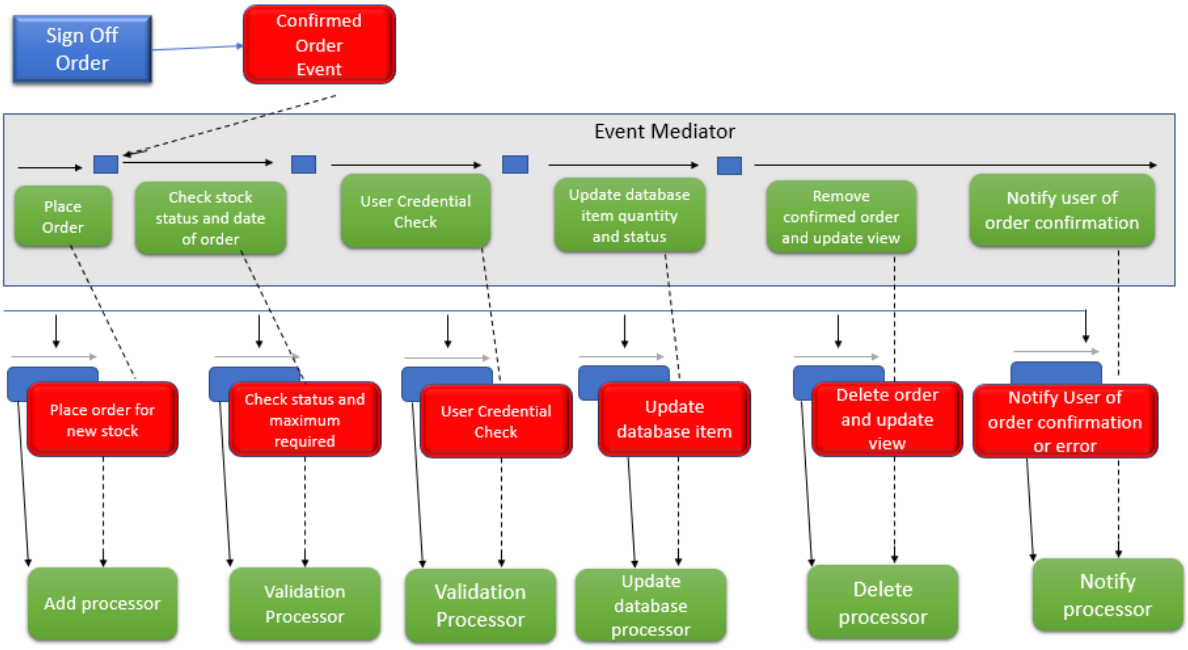
Login to view stock status:



- Annotation: The Event Mediator Diagram above is the process when the user has to Login to view stock status.
  - The first process of the process to be executed by the mediator is the validation process which will check if the input fields are left empty, if they are then it will tell the user what field is missing value i.e. password.
  - If credentials entered are in database, the mediator will execute the second processor which will check which user type it is, Manager, Stock control assistant or other Staff.
  - The third processor gets all the items from the Stock table and passes it to the Grid View.
  - Fourth processor is called Adjustment processor, this will get all stock from the Table required and display it on the Gird View of the user.
  - Fifth processor is executed concurrently, this will notify the user if the items were found or failure if there Is any other error such as connection problem with the database, through a Message box pop up.

User Story 2: **Place Orders if item stock low and sign off arrived orders**

Place Orders if item stock low and sign off arrived orders:



- <u>Annotation:</u> The Event Mediator Diagram above is the process when a member of staff will approve or delete an awaiting order that has been delivered by the suppliers.
  - A staff will enter all the field required for the item to be place and add process is executed.
  - Validation process is then excited to make sure that status of the item's which the order is placed of is changed to 'New Stock ordered' and also the ordered quantity shouldn't exceed the maximum required quantity.
  - User validation Is performed after to make sure that it is the Stock Control Assistant who is ordering new stocks because only they can order new stocks.
  - Update database processor is now executed, which makes sure that the new stock is ordered and inserted into the Pending Order Table and the item status in the Stock table is changed to 'New Stock Ordered'.
  - Delete processor is now executed, this makes sure that the item just ordered is now deleted from the Orders table because new stock of the item is ordered and not needed in the Orders table which needs new stocks to be ordered.

Task 2

Examples of commented source code of chosen functionality which consists of the THREE user stories:

User Story 1 & 3: Adding, removing (Only Manager), Updating and Searching stock items for Manager and Stock Assistant.

Adding

```
private void button1_Click(object sender, EventArgs e) //Add item button to trigger Add event
    {
        checkifitemexsists();
    }

 private void checkifitemexsists() //validation process
    {
        mediator.itemduplication_validation(textBox1.Text, textBox2.Text);//checks if the item already exists in Stock Table
        addtoStockTable(); //calls this function, which calls the add to stock table mediator

    }


private void addtoStockTable() //add process
    {
        //calling the mediator to perform the add process, adding to StockTable
        mediator.Add_Stocktable(textBox1.Text, textBox2.Text, textBox3.Text, textBox4.Text, dateTimePicker1.Text, textBox6.Text, textBox7.Text, textBox8.Text);

    }


private void getPrimaryKey() //Give new id to the new item
    {
        using (SqlConnection Connection = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-\(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30"))
        {
            try
            {
                Connection.Open();
                SqlCommand cmd = new SqlCommand(@"SELECT MAX(Id)+1 FROM StockTable;", Connection);
                Connection.Close();

            }

            catch (Exception ex)
            {
                MessageBox.Show("Unexpected Error has occured: " + ex.Message);
            }
        }
    }
```

Searching

```
private void button2_Click(object sender, EventArgs e) //search button
    {
        mediator.search_mediator(textBox1.Text, textBox2.Text); //search mediator with item name and item code to search

        //SqlDataReader rd = checkid.ExecuteReader();
        con = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-\(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30");
        SqlDataAdapter checkup = new SqlDataAdapter("SELECT * FROM [StockTable] WHERE itemname ='" + textBox9.Text + "'", con); //this will get all the data
        DataTable sd = new DataTable();

        checkup.Fill(sd);
        dataGridView1.DataSource = sd;
    }
```

Deleting

```
private void button5_Click(object sender, EventArgs e) //remove button to trigger delete event
    {
        DataGridViewRow row = dataGridView1.CurrentCell.OwningRow;//grab a reference to the current row
        mediator.delete_StockTable(row); //calls the delete process from delete mediator to delete the row of the StockTable
        dataGridView1.DataSource = mediator.refresh_StockTable(); //calls the refresh mediator to get the updated data from the Stock Table after the delete mediator has deleted the selected row
        dataGridView1.Update();//updates the data grid with new updated data from the Stock Table
    }
```

Updating

```
private void button3_Click(object sender, EventArgs e) //update button to trigger update event
    {
        mediator.textbox_validation_StockTable(textBox1.Text, textBox2.Text, textBox3.Text, textBox4.Text, dateTimePicker1.Text, textBox6.Text, textBox7.Text, textBox8.Text); //validates if data is entered in the text boxes of the values to be updated into the Stock Table
        mediator.itemduplication_validation(textBox1.Text, textBox1.Text); //checks if the data entered is already inside the StockTable
        mediator.update_StockTable(textBox1.Text,textBox2.Text, textBox3.Text, textBox4.Text, dateTimePicker1.Text, textBox6.Text); //updates the data entered in text boxes into the StockTable
        dataGridView1.DataSource = mediator.refresh_StockTable(); //refreshes the Stocktable and puts it into the Data Grid view
        dataGridView1.Update(); //Data grid view gets updated with the updated data
    }
```

User Story 4: All staff be able to Login using their credentials and view stock status

Manager Login:

```
private void button1_Click(object sender, EventArgs e) //manager login button
    {
        mediator.textbox_Validation_LoginForm(textUsername.Text, textPassword.Text); //Validation process to check if username and password is entered
        mediator.mmanager_login_validation(textUsername.Text, textPassword.Text); //checks to see if the entered username and password matches with the one stored in database
        mainui main = new mainui();
        main.Show(); //opens assistant interface form
        this.Close(); //form is closed when all proceeses are fini
    }
```

Any Other Staff Login:

```
private void button2_Click(object sender, EventArgs e) //staff login button
    {
        mediator.textbox_Validation_LoginForm(textUsername.Text, textPassword.Text); //Validation process to check if username and password is entered
        mediator.loginandapprove_validation(textUsername.Text, textPassword.Text); //checks to see if the entered username and password matches with the one stored in database
        MainUI.assistantmain main = new MainUI.assistantmain();
        main.Show(); //opens assistant interface form
        this.Close(); //form is closed when all proceeses are finished
    }
```

View stock status:

```
private void button4_Click_1(object sender, EventArgs e) //view all items button to trigger an event
  {
    DataGridViewRow row = dataGridView1.CurrentCell.OwningRow;//grab a reference to the current row
    dataGridView1.DataSource = mediator.refresh_StockTable();//calls the refresh mediator to get the updated data from the Stock Table
    dataGridView1.Update(); //updates the data grid with new updated data from the Stock Table
  }

  private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e) //Grid Display
  {
    if (e.RowIndex >= 0)
    {
      //gets a collection that contains all the rows
      DataGridViewRow row = this.dataGridView1.Rows[e.RowIndex];
      //populate the textbox from specific value of the coordinates of column and row.
      textBox1.Text = row.Cells[1].Value.ToString();
```

```
            textBox2.Text = row.Cells[2].Value.ToString();
            textBox3.Text = row.Cells[3].Value.ToString();
            textBox4.Text = row.Cells[4].Value.ToString();
            dateTimePicker1.Text = row.Cells[5].Value.ToString();
            textBox6.Text = row.Cells[6].Value.ToString();
            textBox7.Text = row.Cells[7].Value.ToString();
            textBox8.Text = row.Cells[8].Value.ToString();
        }
    }

    private void button6_Click(object sender, EventArgs e) //clear button
    {
        textBox1.Clear();
        textBox2.Clear();
        textBox3.Clear();
        textBox4.Clear();
        dateTimePicker1.ResetText();
        textBox6.Clear();
        textBox7.Clear();
        textBox8.Clear();
    }
}
```

User Story 2: **Place Orders if item stock low and sign off arrived orders**

Viewing all items in stock that are low:

```
private void viewallitems_Click(object sender, EventArgs e) //vieww all items button
    {
        con = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-\(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30");
        SqlDataAdapter checkup = new SqlDataAdapter("SELECT * FROM StockTable", con); //this will get all the data
        DataTable sd = new DataTable();

        checkup.Fill(sd);
        dataGridView1.DataSource = sd;

        DataTable sd1 = new DataTable();
        //sd1 = sd.DefaultView.ToTable(true, "itemcode", "itemname", "itemquantity", "stockarrivaldate", "minimumrequired", "maximumrequired", "stockstatus", "stockordered");
        sd1 = sd.DefaultView.ToTable(true, "itemcode", "itemname", "itemquantity", "stockarrivaldate", "maximumrequired", "itemstatus");

        dataGridView1.DataSource = sd1;
    }
```

Stock Control Assistant button to place a new stock Order:

```
private void button1_Click(object sender, EventArgs e) //place a new oder button
    {

        using (SqlConnection Connection = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-\(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30"))
        {
            int a = int.Parse(textBox3.Text);
            int b = int.Parse(textBox4.Text);

            if (textBox1.Text == String.Empty || textBox2.Text == String.Empty || textBox3.Text == String.Empty || textBox4.Text == String.Empty || textBox5.Text == String.Empty || a > b)
            {
                MessageBox.Show("Error, All item deails must be entered", "Value Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            else
            {
                try
                {
                    Connection.Open();
                    SqlCommand cmd = new SqlCommand(@"INSERT INTO OrderTable ([itemcode], [itemname], [itemquantity], [stockarrivaldate], [maximumrequired], [orderstatus]) VALUES (@itemcode, @itemname, @itemquantity, @stockarrivaldate, @maximumrequired, @orderstatus);", Connection);

                    //string query = "UPDATE StockTable SET orderstatus = @orderstatus2  where orderstatus2 = @orderstatus2 ";
                    //SqlCommand cmd2 = new SqlCommand(query, con);

                    cmd.Parameters.AddWithValue("@itemcode", textBox1.Text);
                    cmd.Parameters.AddWithValue("@itemname", textBox2.Text);
                    cmd.Parameters.AddWithValue("@itemquantity", textBox3.Text);
                    cmd.Parameters.AddWithValue("@stockarrivaldate", dateTimePicker1.Text);
                    cmd.Parameters.AddWithValue("@orderstatus", textBox5.Text);
                    cmd.Parameters.AddWithValue("@maximumrequired", textBox4.Text);



                    int i = cmd.ExecuteNonQuery();
                    Connection.Close();

                    if (i == 1)

                    {
                        MessageBox.Show("New Stock has been ordered");
                        getPrimaryKey();
                        updatestatus();

```

```
                    textBox1.Clear();
                    textBox2.Clear();
                    textBox3.Clear();
                    dateTimePicker1.ResetText();
                    textBox4.Clear();
                    textBox5.Clear();


                }
                else
                {
                    MessageBox.Show("Stock could not be ordered, Please Try again");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Unexpected Error has occured:" + ex.Message);

            }
        }
    }

}


private void updatestatus() //update item status //only executes if adding a new order is successful
    {

        con = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-\(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30");

        {

            string query = "UPDATE StockTable SET itemstatus = @itemstatus  where itemname = @itemname";
            SqlCommand cmd = new SqlCommand(query, con);
            cmd.Parameters.AddWithValue("@itemstatus", textBox5.Text);
            cmd.Parameters.AddWithValue("@itemname", textBox2.Text);
            cmd.Connection.Open();

            try
            {
                int i = cmd.ExecuteNonQuery();
                cmd.Connection.Close();

                if (i == 1)
                {
```

```
                    MessageBox.Show("Stock Staus has been updated");

                }
            }
            catch (Exception ex)
            {
                throw new Exception("Unexpected Error has occured: " + ex.Message);
            }

        }

            }

        private void getPrimaryKey() //Give new id to the new item
            {
                using (SqlConnection Connection = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-\(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30"))
                {
                    try
                    {
                        Connection.Open();
                        SqlCommand cmd = new SqlCommand(@"SELECT MAX(Id)+1 FROM OrderTable;", Connection);
                        Connection.Close();

                    }

                    catch (Exception ex)
                    {
                        MessageBox.Show("Unexpected Error has occured: " + ex.Message);
                    }
                }
            }
```

<u>Stock Assistant Approving if the Order placed matched with the Order arrived</u>

```
private void button1_Click(object sender, EventArgs e) //approve button
{
    AssitantUI.approval f2 = new AssitantUI.approval(this.approved); //Passing a delegate to an update method in Approval form, passing approved function from checkorder form to approval form
                                                    //so (approved) function can be exceuted from a different form(class), and approved function can remain private, meaning low coupling between two forms
}

private void approved() //if approved
{
```

```
    mediator.textbox_validation_StockTable2(textBox1.Text, textBox2.Text, textBox3.Text, dateTimePicker1.Text, textBox5.Text); //validates if data is entered in the text boxes of the values to be updated into the Stock Table
    mediator.itemduplication_validation(textBox1.Text, textBox1.Text); //checks if the data entered is already inside the StockTable
    mediator.update_StockTable(textBox1.Text, textBox2.Text, textBox3.Text, dateTimePicker1.Text, textBox4.Text, textBox5.Text); //updates the data entered in text boxes into the StockTable
    deleteorder();
    dataGridView1.DataSource = mediator.refresh_OrderTable(); //refreshes the Stocktable and puts it into the Data Grid view
    dataGridView1.Update(); //Data grid view gets updated with the updated data
}

public void deleteorder() //deletes the item in Order Table which  is being added to the StockTable
{
    DataGridViewRow row = dataGridView1.CurrentCell.OwningRow;//grab a reference to the current row //validation inside the process
    mediator.delete_OrderTable(row); //calls the delete process from delete mediator to delete the row of the Order TableTable
    dataGridView1.DataSource = mediator.refresh_OrderTable(); //calls the refresh mediator to get the updated data from the Order Table after the delete mediator has deleted the selected row
    dataGridView1.Update();//updates the data grid with new updated data from the Stock Table
}
```

- <u>Stock Control Assistant deletes an arrived order:</u>

```
private void button2_Click(object sender, EventArgs e) //delete button
{

    AssitantUI.approval f2 = new AssitantUI.approval(this.deletebygrid); //delegating to authenticate usercredtianls
    //if authentication is complete, run deletebygrid method
    f2.Show();

}

private void deletebygrid() //mediator to delete the values of Order Table
{
    DataGridViewRow row = dataGridView1.CurrentCell.OwningRow;//grab a reference to the current row
    mediator.delete_OrderTable(row); //calls the delete process from delete mediator to delete the row of the Order TableTable
    dataGridView1.DataSource = mediator.refresh_OrderTable(); //calls the refresh mediator to get the updated data from the Order Table after the delete mediator has deleted the selected row
```

```
15.    dataGridView1.Update();//updates the data grid with new updated data from the Stock Table
16. }
```

- <u>Authentication: To approve or delete stock control assistant must sign in with their credentials</u>

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.ComponentModel;
4.  using System.Data;
5.  using System.Drawing;
6.  using System.Linq;
7.  using System.Text;
8.  using System.Threading.Tasks;
9.  using System.Windows.Forms;
10. using System.Data.SqlClient;
11.
12. namespace _SCM_System.AssitantUI
13. {
14.    public partial class approval : Form
15.    {
16.       private Mediator mediator = new Mediator();
17.       private readonly Action _approver; //method that doesn't have any parameters and does not return any value, making private and readonly. For delegation for a process from another form
18.       public approval(Action approver)
19.       {
20.          _approver = approver; //linking the private class and public class together
21.          InitializeComponent();
22.          textUsername.KeyPress += new KeyPressEventHandler(CheckEnter);
23.          textPassword.KeyPress += new KeyPressEventHandler(CheckEnter);
24.       }
25.
26.       private void button3_Click(object sender, EventArgs e) //exit button
27.       {
28.          System.Environment.Exit(0);
29.       }
30.
31.       private void button2_Click(object sender, EventArgs e) //cancel button
32.       {
33.          ManagerUI.checkorder main = new ManagerUI.checkorder();
34.          main.Show();
35.          this.Close();
36.       }
37.
38.       private void CheckEnter(object sender, KeyPressEventArgs e) //when enter is pressed, this button (button1_Click) is clicked
39.       {
40.          if (e.KeyChar == (char)13)
41.          {
42.             button1_Click(this, new EventArgs());
```

```
43.         }
44.     }
45.
46.
47.     void button1_Click(object sender, EventArgs e) //approve button
48.     {
49.         mediator.textbox_Validation_LoginForm(textUsername.Text, textPassword.Text); //Val
    idation process to check if username and password is entered
50.         mediator.loginandapprove_validation(textUsername.Text, textPassword.Text); //check
    s to see if the entered username and password matches with the one stored in database
51.         _approver(); //this will run any methods passed by other forms which required authent
    ication to proceed
52.         this.Close(); //form is closed when all proceeses are finished
53.
54.
55.     }
56.
57.
58.     private void approval_Load(object sender, EventArgs e)
59.     {
60.
61.     }
62.   }
63. }
```

Evidence of Mediator being implemented with annotation:

- Main Mediator class which calls the required processes that needs to be executed from the required forms: Mediator.cs

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading.Tasks;
6.  using System.Windows.Forms;
7.  using System.Data;
8.
9.  namespace _SCM_System
10. {
11.   public class Mediator
12.   {
13.     private deletemediator deleteitemprocess;
14.     private AddItem additemprocess;
15.     private updatemediator updateprocess;
16.     private Searchitem searchprocess;
17.     private Validation validationprocess;
18.     private RefreshStock adjustmentprocess;
19.
20.
21.     public Mediator()
22.     {
23.       //setting mediator processes
24.       deleteitemprocess = new deletemediator();
```

```
            additemprocess = new AddItem();
            updateprocess = new updatemediator();
            validationprocess = new Validation();
            searchprocess = new Searchitem();
            adjustmentprocess = new RefreshStock();


        }

        public void delete_StockTable(DataGridViewRow row) // setting delete mediator  #Number of process: 1
        {
            //delete processes executed by the mediator
            deleteitemprocess.ExecuteStockTable(row); //delete process for StockTable
        }

        public void delete_OrderTable(DataGridView row) //delete from order Table
        {
            deleteitemprocess.Delete_OrderTable(row); //process to delete from Order Table #Number of process: 2
        }

        //public void delete_Pendingorder(DataGridView row) //delete from Pending Order
        //{
            //deleteitemprocess.ExecutePendingTable(row); //delete process for Pending Table #Number of process: 3
        //}

        public void Add_Stocktable(string itemcode, string itemname, string itemprice, string itemquantity, string stock
    arrivaldate, string minimumrequired, string maximumrequired, string staffcheck) //Add into StockTable with values

        {
            //add processes executed by the mediator
            additemprocess.ExecuteAddStocktable(itemcode, itemname, itemprice, itemquantity, stockarrivaldate, mini
    mumrequired, maximumrequired, staffcheck); //add item into Stock Table #Number of process: 4
        }

        public void Add_OrderTable(string itemcode, string itemname, string itemquantity, string stockarrivaldate, stri
    ng maximumrequired, string orderstatus) //add into Order Table with values
        {
            additemprocess.ExecuteAddOrderTable(itemcode, itemname, itemquantity, stockarrivaldate, maximumreq
    uired, orderstatus); //add item into OrderTable #Number of process:  5
        }

        public void update_StockTable(string itemcode, string itemname, string itemprice, string itemquantity, string st
    ockarrivaldate, string itemstatus)  //update StockTable with values
        {
            updateprocess.UpdateStockTable(itemcode, itemname, itemprice, itemquantity, stockarrivaldate, itemstatu
    s); //update process to update Stock Table #Number of process: 6
        }

        public void search_mediator(string itemcode, string itemname) //searches by name or code
        {
            searchprocess.searchbynameorcode(itemcode, itemname); //search by name or code #Number of process:
    7
        }

        public void itemduplication_validation(string itemcode, string itemname) //checks if item name or code is alre
    ady in the table  #Number of process: 8
        {
            validationprocess.itemduplicationvalidation(itemcode, itemname); //setting up duplicatiomn validatiom in t
    he database when item is added  //duplication validation process executed by the mediator
        }

        public void textbox_Validation_LoginForm(string textUsername, string textPassword) //checks username or pa
    ssword is entered #Number of process: 12
```

```
        {
            validationprocess.textbox_Validation_LoginForm(textUsername, textPassword); //setting up value validation to check something is entered in username or password text box
        }

        public void mmanager_login_validation(string textUsername, string textPassword) //checks username or password is entered for manager #Number of process: 13
        {
            validationprocess.mmanager_login_validation(textUsername, textPassword); //setting up value validation to check something is entered in username or password text box
        }


        public void textbox_validation_StockTable(string textBox1, string textBox2, string textBox3, string textBox4, string DateTimepicker1, string textBox6, string textBox7, string textBox8) //checks if every boxes are not empty
        {
            validationprocess.textbox_validation_StockTable(textBox1, textBox2, textBox3, textBox4, DateTimepicker1, textBox6, textBox7, textBox8);  // #Number of process: 9
            //text validation process executed by the mediator //setting text box validation
        }

        public void textbox_validation_StockTable2(string textBox1, string textBox2, string textBox3, string DateTimepicker1, string textBox5)//checks if every boxes are not empty
        {
            validationprocess.textbox_validation_StockTable2(textBox1, textBox2, textBox3, DateTimepicker1, textBox5); //text validation process executed by the mediator //setting text box validation in Pending Orders Form
        }

        public void loginandapprove_validation(string textUsername, string textPassword) //validates if entered username and password is correct
        {
            validationprocess.loginandapprovevalidation(textUsername, textPassword); //#Number of process: 10
            //validation process for Login and Approve //this process can be used for Logging in and authencating credtianls again when approving orders
        }
        public DataTable refresh_StockTable()//get updated data from Stock Table
        {
            return adjustmentprocess.refresh_StockTable(); //#Number of process: 11
        }

        public DataTable refresh_OrderTable() //get updated data from Order table
        {
            adjustmentprocess.refresh_OrderTable(); //#Number of process:12
        }


    }
}
```

- <u>Validation process</u> which contains all the validation processes, which the main Mediator class calls: Validation.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;
```

```
11.
12. namespace _SCM_System
13. {
14.   class Validation
15.   {
16.     //private readonly Action _getdatato; //method that doesn't have any parameters and does not return any val
    ue, making private and readonly. For delegation for a process from another Update
17.     //public getdatato(Action getdatato) //making method public so it can be used into the itemduplicationvalidati
    on process
18.     //{
19.       //_getdatato = getdatato; //linking the pivate and public methods together
20.
21.     //}
22.
23.
24.     SqlConnection con;
25.     string Connection = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-
    \(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30";
26.
27.     public void textbox_validation_StockTable(string textBox1, string textBox2, string textBox3, string textBox4, stri
    ng DateTimepicker1, string textBox6, string textBox7, string textBox8) //text box validation
28.     {
29.       if (textBox1 == String.Empty || textBox2 == String.Empty || textBox3 == String.Empty || textBox4 == String.
    Empty || DateTimepicker1 == String.Empty || textBox6 == String.Empty || textBox7 == String.Empty || textBox8 =
    = String.Empty)
30.         {
31.           MessageBox.Show("Make sure every value is filled", "Value Error", MessageBoxButtons.OK, MessageBo
    xIcon.Error);
32.         }
33.     }
34.
35.     public void textbox_validation_StockTable2(string textBox1, string textBox2, string textBox3, string DateTimepi
    cker1, string textBox5) //text box validation
36.     {
37.       if (textBox1 == String.Empty || textBox2 == String.Empty || textBox3 == String.Empty || DateTimepicker1 =
    = String.Empty || textBox5 == String.Empty)
38.       {
39.         MessageBox.Show("Make sure every value is filled", "Value Error", MessageBoxButtons.OK, MessageBoxI
    con.Error);
40.       }
41.     }
42.
43.     public void textbox_Validation_LoginForm(string textUsername, string textPassword) //text box validation for L
    ogin Form
44.     {
45.       if (textUsername == String.Empty || textPassword == String.Empty)
46.       {
47.         MessageBox.Show("Make sure Username or Password is filled", "Value Error", MessageBoxButtons.OK, M
    essageBoxIcon.Error);
48.       }
49.     }
50.
51.     public void itemduplicationvalidation(string itemcode, string itemname) //item duplication validation
52.     {
53.       SqlCommand cmd; //setting new sql command object
54.       string validate = @"SELECT COUNT(*) from StockTable (where itemcode like @itemcode AND itemname like
    @itemname";
55.
56.       using (con = new SqlConnection(Connection))
57.       {
58.         try
59.         {
60.           con.Open();
```

```
61.                 cmd = new SqlCommand(validate, con);
62.                 cmd.Parameters.AddWithValue("@itemcode", itemname);
63.                 cmd.Parameters.AddWithValue("@itemname", itemcode);
64.                 int userCount = (int)cmd.ExecuteScalar();
65.
66.                 if (userCount > 0)
67.                 {
68.                     MessageBox.Show("Item already exists in the table", "Value Error", MessageBoxButtons.OK, Message
    BoxIcon.Error);
69.                 }
70.
71.                 else
72.                 {
73.                     //this holds UpdateStockTable process from Update Mediator
74.                     //_getdatato(); //if the data is found non existent in Stock Table, update value to Stock Table
75.                 }
76.
77.             }
78.             catch (Exception ex)
79.             {
80.                 MessageBox.Show("Unexpected Error has occured:" + ex.Message);
81.
82.             }
83.         }
84.
85.     }
86.
87.     public void loginandapprovevalidation(string textUsername, string textPassword) //validation process for Login
     and Approve
88.                                                     //this process can be used for Logging in and authencating credti
    anls again when approving orders
89.     {
90.         using (SqlConnection Connection = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDb
    Filename=C:\340CT\340CT---Asis-Rai-
    \(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30"))
91.         {
92.             if (textUsername == String.Empty)
93.             {
94.                 MessageBox.Show("Please enter user name", "Input Error", MessageBoxButtons.OK, MessageBoxIcon.E
    xclamation);
95.                 //textUsername.Focus();
96.             }
97.
98.             else if (textPassword == String.Empty)
99.             {
100.                MessageBox.Show("Please enter password", "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Ex
    clamation);
101.                //textPassword.Focus();
102.            }
103.
104.            else
105.            {
106.                try
107.                {
108.                    Connection.Open();
109.                    SqlCommand cmd = new SqlCommand(@"SELECT Count(*) FROM StaffTable WHERE username=@un
    ame and password=@pass", Connection);
110.                    cmd.Parameters.AddWithValue("@uname", textUsername);
111.                    cmd.Parameters.AddWithValue("@pass", textPassword);
112.                    int result = (int)cmd.ExecuteScalar();
113.
114.                    if (result > 0)
115.                    {
```

```
                    //_approver(); //now this updates the existing Pending Order form instances
                    //this.Close();
                    Connection.Close(); //closes the connection
                }
                else
                {
                    MessageBox.Show("Incorrect crediantials, please try again");
                }
            }

            catch (Exception ex)
            {
                MessageBox.Show("Unexpected error:" + ex.Message);
            }
        }
    }
}

    public void mmanager_login_validation(string textUsername, string textPassword) //validation process for Login and Approve
                                                                    //this process can be used for Logging in and authencating credtianls again when approving orders
    {
        using (SqlConnection Connection = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-\(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30"))
        {
            if (textUsername == String.Empty)
            {
                MessageBox.Show("Please enter user name", "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                //textUsername.Focus();
            }

            else if (textPassword == String.Empty)
            {
                MessageBox.Show("Please enter password", "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                //textPassword.Focus();
            }

            else
            {
                try
                {
                    Connection.Open();
                    SqlCommand cmd = new SqlCommand(@"SELECT Count(*) FROM ManagerTable WHERE username=@uname and password=@pass", Connection);
                    cmd.Parameters.AddWithValue("@uname", textUsername);
                    cmd.Parameters.AddWithValue("@pass", textPassword);
                    int result = (int)cmd.ExecuteScalar();

                    if (result > 0)
                    {
                        //_approver(); //now this updates the existing Pending Order form instances
                        //this.Close();
                        Connection.Close(); //closes the connection
                    }
                    else
                    {
                        MessageBox.Show("Incorrect crediantials, please try again");
                    }
                }
```

```
172.
173.                catch (Exception ex)
174.                {
175.                    MessageBox.Show("Unexpected error:" + ex.Message);
176.                }
177.            }
178.        }
179.    }
180.
181.  }
182. }
```

- <u>Refresh Stock class</u> which contains all the refresh processes, which the main Mediator class calls: RefreshStock.cs

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading.Tasks;
6.  using System.Data.SqlClient;
7.  using System.Windows.Forms;
8.  using System.Data;
9.
10. namespace _SCM_System
11. {
12.    class RefreshStock
13.    {
14.
15.        string Connection = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-
    \(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30";
16.
17.        public DataTable refresh_StockTable()
18.        {
19.            DataTable data = new System.Data.DataTable();
20.
21.            try
22.            {
23.                SqlDataAdapter dataAdapter = new System.Data.SqlClient.SqlDataAdapter("SELECT * FROM StockTable",
    Connection); //pass in the select command and the conncetion string
24.                data.Locale = System.Globalization.CultureInfo.InvariantCulture;
25.                dataAdapter.Fill(data); //fill the table
26.            }
27.
28.            catch (System.Data.SqlClient.SqlException ex)
29.            {
30.                MessageBox.Show(ex.Message); //show a useful message to the user of the program
31.            }
32.
33.            return data;
34.        }
35.
36.
37.        public DataTable refresh_OrderTable()
38.        {
39.            DataTable data = new System.Data.DataTable();
40.
41.            try
42.            {
```

```
            SqlDataAdapter dataAdapter = new System.Data.SqlClient.SqlDataAdapter("SELECT * FROM OrderTable", Connection); //pass in the select command and the conncetion string
            data.Locale = System.Globalization.CultureInfo.InvariantCulture;
            dataAdapter.Fill(data); //fill the table
        }

        catch (System.Data.SqlClient.SqlException ex)
        {
            MessageBox.Show(ex.Message); //show a useful message to the user of the program
        }

        return data;
    }
  }
}

```

- Add Item class which contains all the add item processes, which the main Mediator class calls: AddItem.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace _SCM_System
{
  class AddItem
  {
    SqlConnection con;
    string Connection = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-\(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30";

    public void ExecuteAddStocktable(string itemcode, string itemname, string itemprice, string itemquantity, string stockarrivaldate, string minimumrequired, string maximumrequired, string staffcheck) //for Stock form
    {
      SqlCommand cmd; //setting new sql command object
      string add = @"INSERT INTO StockTable ([itemcode], [itemname], [itemprice], [itemquantity], [stockarrivaldate], [minimumrequired], [maximumrequired], [staffcheck])
                    VALUES (@itemcode, @itemname, @itemprice, @itemquantity, @stockarrivaldate, @minimumrequired, @maximumrequired, @staffcheck)";

      using (con = new SqlConnection(Connection))
      {
        try
        {
          con.Open(); //open connection
          //Read value from forms
          cmd = new SqlCommand(add, con);
          cmd.Parameters.AddWithValue("@itemcode", itemcode);
          cmd.Parameters.AddWithValue("@itemname", itemname);
          cmd.Parameters.AddWithValue("@itemprice", itemprice);
          cmd.Parameters.AddWithValue("@itemquantity", itemquantity);
          cmd.Parameters.AddWithValue("@stockarrivaldate", stockarrivaldate);
          cmd.Parameters.AddWithValue("@minimumrequired", minimumrequired);
          cmd.Parameters.AddWithValue("@maximumrequired", maximumrequired);
          cmd.Parameters.AddWithValue("@staffcheck", staffcheck);

          //Append values into StockTable
```

```
39.                 cmd.ExecuteNonQuery();
40.
41.                 int i = cmd.ExecuteNonQuery();
42.
43.                 if (i == 1)
44.                 {
45.                     MessageBox.Show("Item has been registered");
46.
47.                 }
48.                 else
49.                 {
50.                     MessageBox.Show("Item couldn't be added, Please Try again");
51.                 }
52.             }
53.
54.             catch (Exception ex)
55.             {
56.                 //gives error if there is a connection issue
57.                 MessageBox.Show("Unexpected Error has occured, Items couldn't be added into StockTable:" + ex.Mes
    sage);
58.             }
59.
60.         }
61.     }
62.
63.     public void ExecuteAddOrderTable(string itemcode, string itemname, string itemquantity, string stockarrivalda
    te, string maximumrequired, string orderstatus) //for oder form
64.     {
65.         SqlCommand cmd; //setting new sql command object
66.         string add = @"INSERT INTO OrderTable ([itemcode], [itemname], [itemquantity], [stockarrivaldate], [maxim
    umrequired], [orderstatus])
67.                     VALUES ((@itemcode, @itemname, @itemquantity, @stockarrivaldate, @maximumrequired, @or
    derstatus)";
68.
69.         using (con = new SqlConnection(Connection))
70.         {
71.             try
72.             {
73.                 con.Open(); //open connection
74.                 //Read value from forms
75.                 cmd = new SqlCommand(add, con);
76.                 cmd.Parameters.AddWithValue("@itemcode", itemcode);
77.                 cmd.Parameters.AddWithValue("@itemname", itemname);
78.                 cmd.Parameters.AddWithValue("@itemquantity", itemquantity);
79.                 cmd.Parameters.AddWithValue("@stockarrivaldate", stockarrivaldate);
80.                 cmd.Parameters.AddWithValue("@maximumrequired", maximumrequired);
81.                 cmd.Parameters.AddWithValue("@orderstatus", orderstatus);
82.
83.                 //Append values into StockTable
84.                 cmd.ExecuteNonQuery();
85.
86.                 int i = cmd.ExecuteNonQuery();
87.
88.                 if (i == 1)
89.                 {
90.                     MessageBox.Show("Item has been registered");
91.
92.                 }
93.                 else
94.                 {
95.                     MessageBox.Show("Item couldn't be added, Please Try again");
96.                 }
97.
```

```
            }

            catch (Exception ex)
            {
                //gives error if there is a connection issue
                MessageBox.Show("Unexpected Error has occured, Items couldn't be added into OrderTable:" + ex.Mes
sage);
            }

        }
    }

  }
}
```

- <u>Delete item class</u> which contains all the Delete item processes, which the main Mediator class calls: DeleteItem.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace _SCM_System
{
    class deletemediator
    {
        public void ExecuteStockTable(DataGridViewRow row)
        {
            SqlConnection con;
            string Connection = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-
Rai-\(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30";

            //this will grab the value from the item name of the selected record
            string code = row.Cells["itemcode"].Value.ToString();

            //This will grab the value from the item name field of the selected record
            string name = row.Cells["itemname"].Value.ToString();

            // This will grab the value from the item quantity field of the selected record
            string quantity = row.Cells["itemquantity"].Value.ToString();

            // This will grab the value from the stock arrival date field of the selected record
            string arrival = row.Cells["stockarrivaldate"].Value.ToString();


            // This will grab the value from the minimum required date field of the selected record
            string Minimum = row.Cells["minimumrequired"].Value.ToString();

            // This will grab the value from the maximum required field of the selected record
            string Maximum = row.Cells["maximumrequired"].Value.ToString();

            //string Staff = row.Cells["staffcheck"].Value.ToString();
            string status = row.Cells["itemtstatus"].Value.ToString();

            //Messahe box will pop up for user confirmation to delete
            DialogResult result = MessageBox.Show("Do you really want to delete " + name + " " + quantity + ", record "
+ code, "Message", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

```

```
43.         string deleteState = @"Delete from StockTable where itemcode = '" + code + "'";//this is the sql to delete th
    e records from the sql table
44.
45.         if (result == DialogResult.Yes) //runs if the user decides to delete ||VALIDATION||          {
46.           using (con = new SqlConnection(Connection)) //uses the connection stated at the top
47.           {
48.             try
49.             {
50.               con.Open(); //using the connection stated at the top, open connection the the database
51.               SqlCommand cmd = new SqlCommand(deleteState, con); //take the command to delete and connecti
    on
52.               cmd.ExecuteNonQuery();//taking the command and connection, delete the selected records
53.
54.
55.             }
56.             catch (Exception ex)
57.             {
58.               MessageBox.Show(ex.Message);//runs if the code above fails(connection, other errors)
59.             }
60.           }
61.       }
62.     }
63.
64.     public void Delete_OrderTable(DataGridViewRow row)
65.     {
66.       SqlConnection con;
67.       string Connection = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-
    \(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30";
68.
69.       //this will grab the value from the item name of the selected record
70.       string code = row.Cells["itemcode"].Value.ToString();
71.
72.       //This will grab the value from the item name field of the selected record
73.       string name = row.Cells["itemname"].Value.ToString();
74.
75.       // This will grab the value from the item quantity field of the selected record
76.       string quantity = row.Cells["itemquantity"].Value.ToString();
77.
78.       // This will grab the value from the stock arrival date field of the selected record
79.       string arrival = row.Cells["stockarrivaldate"].Value.ToString();
80.
81.       // This will grab the value from the maximum required field of the selected record
82.       string Maximum = row.Cells["maximumrequired"].Value.ToString();
83.
84.       //string Staff = row.Cells["staffcheck"].Value.ToString();
85.       string status = row.Cells["orderstatus"].Value.ToString();
86.
87.       //Messahe box will pop up for user confirmation to delete
88.       DialogResult result = MessageBox.Show("Do you really want to delete " + name + " " + quantity + ", record " +
    code, "Message", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
89.
90.       string deleteState = @"Delete from OrderTable where itemname = '" + name + "'";//this is the sql to delete the
     records from the sql table
91.
92.       if (result == DialogResult.Yes) //runs if the user decides to delete ||VALIDATION||          {
93.         using (con = new SqlConnection(Connection)) //uses the connection stated at the top
94.         {
95.           try
96.           {
97.             con.Open(); //using the connection stated at the top, open connection the the database
98.             SqlCommand cmd = new SqlCommand(deleteState, con); //take the command to delete and connectio
    n
99.             cmd.ExecuteNonQuery();//taking the command and connection, delete the selected records
```

```
100.
101.
102.            }
103.            catch (Exception ex)
104.            {
105.                MessageBox.Show(ex.Message);//runs if the code above fails(connection, other errors)
106.            }
107.        }
108.    }
109. }
```

- <u>Update Item class</u> which contains all the Update item processes, which the main Mediator class calls: Update.cs

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading.Tasks;
6.  using System.Data.SqlClient;
7.  using System.Windows.Forms;
8.
9.  namespace _SCM_System
10. {
11.     class updatemediator
12.     {
13.         SqlConnection con;
14.         string Connection = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-
    \(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30";
15.
16.
17.
18.         //Passing a delegate to an update method in Approval form, passing approved function from checkorder form
    to approval form
19.         //so (approved) function can be exceuted from a different form(class), and approved function can remain priv
    ate, meaning low coupling between two forms
20.         //Validation f1 = new Validation(this.UpdateStockTable);
21.         public void UpdateStockTable(string itemcode, string itemname, string itemprice, string itemquantity, string st
    ockarrivaldate, string itemstatus) //this will update stock table
22.         {
23.             SqlCommand cmd; //setting new sql command object
24.             string update = @"UPDATE StockTable SET itemcode = @itemcode, itemname=@itemname, itemquantity =
    @itemquantity, stockarrivaldate = @stockarrivaldate, itemstatus = @itemstatus  where itemname = @itemname";

25.
26.             using (con = new SqlConnection(Connection))
27.             {
28.                 try
29.                 {
30.                     con.Open(); //open connection
31.                             //Read value from forms
32.                     cmd = new SqlCommand(update, con);
33.                     cmd.Parameters.AddWithValue("@itemcode", itemcode);
34.                     cmd.Parameters.AddWithValue("@itemname", itemname);
35.                     cmd.Parameters.AddWithValue("@itemquantity", itemquantity);
36.                     cmd.Parameters.AddWithValue("@stockarrivaldate", stockarrivaldate);
37.                     cmd.Parameters.AddWithValue("@itemstatus", itemstatus);
38.
39.                     //Append values into StockTable
40.                     cmd.ExecuteNonQuery();
41.                     con.Close();
42.
```

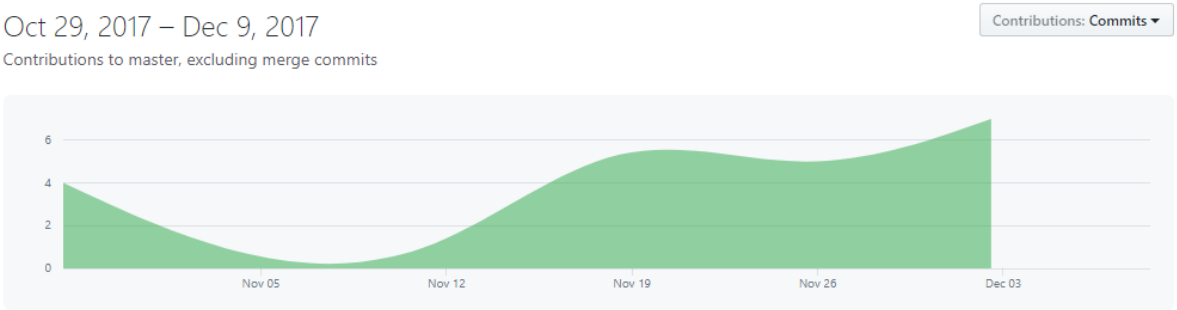
```
43.             }
44.             catch (Exception ex)
45.             {
46.                 //gives error if there is a connection issue
47.                 throw new Exception("Unexpected Error has occured: Items couldn't be updated" + ex.Message);
48.             }
49.
50.         }
51.     }
52.   }
53. }
```

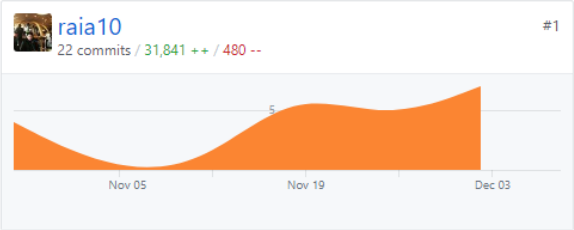
- Notification class which contains all the notification processes, which the main Mediator class calls: Notification.cs

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading.Tasks;
6.  using System.Windows.Forms;
7.  using System.Data.SqlClient;
8.
9.  namespace _SCM_System
10. {
11.   class Notification
12.   {
13.     SqlConnection con;
14.     string Connection = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\340CT\340CT---Asis-Rai-\(SCM)System\(SCM)System\(SCM)System.mdf;Integrated Security=True;Connect Timeout=30";
15.
16.     public void notify_finished_adding(string itemcode, string itemname, string itemprice, string itemquantity, string stockarrivaldate, string minimumrequired, string maximumrequired, string staffcheck)
17.     {
18.         SqlCommand cmd; //setting new sql command object
19.         string add = @"INSERT INTO StockTable ([itemcode], [itemname], [itemprice], [itemquantity], [stockarrivaldate], [minimumrequired], [maximumrequired], [staffcheck])
20.                     VALUES (@itemcode, @itemname, @itemprice, @itemquantity, @stockarrivaldate, @minimumrequired, @maximumrequired, @staffcheck)";
21.
22.         using (con = new SqlConnection(Connection))
23.         {
24.             try
25.             {
26.                 con.Open(); //open connection
27.                 //Read value from forms
28.                 cmd = new SqlCommand(add, con);
29.                 cmd.Parameters.AddWithValue("@itemcode", itemcode);
30.                 cmd.Parameters.AddWithValue("@itemname", itemname);
31.                 cmd.Parameters.AddWithValue("@itemprice", itemprice);
32.                 cmd.Parameters.AddWithValue("@itemquantity", itemquantity);
33.                 cmd.Parameters.AddWithValue("@stockarrivaldate", stockarrivaldate);
34.                 cmd.Parameters.AddWithValue("@minimumrequired", minimumrequired);
35.                 cmd.Parameters.AddWithValue("@maximumrequired", maximumrequired);
36.                 cmd.Parameters.AddWithValue("@staffcheck", staffcheck);
37.
38.                 //Append values into StockTable
39.                 cmd.ExecuteNonQuery();
40.             }
41.
42.             catch (Exception ex)
43.             {
```

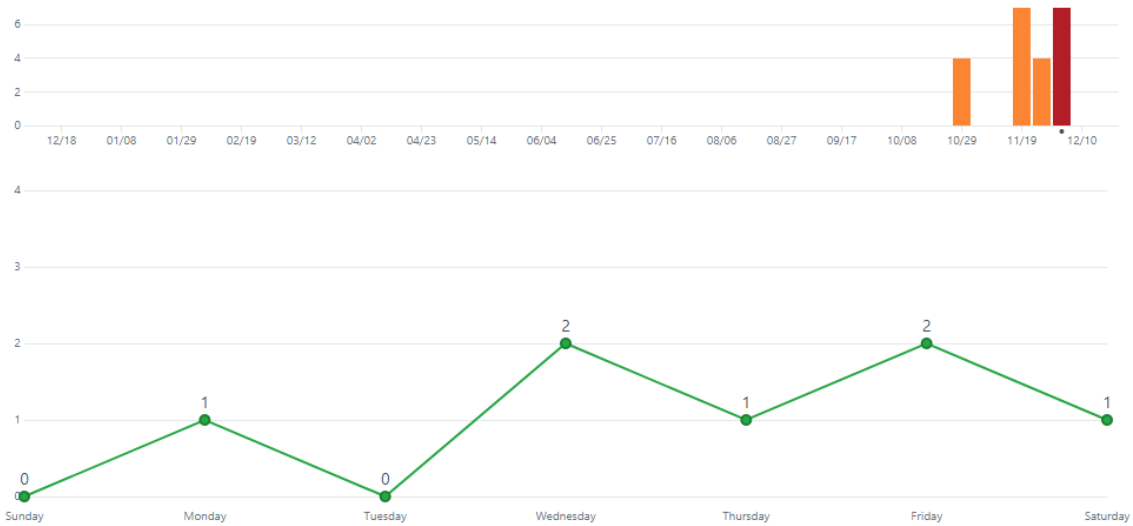
```
            //gives error if there is a connection issue
            MessageBox.Show("Unexpected Error has occured, Items couldn't be added into StockTable:" + ex.Mes
sage);
        }
    }
  }
 }
}
```

Evidenced of version control tool (GITHUB)





- Regular commits showing the changes in code, Additions and deletions



Commits on Dec 9, 2017

Merge branch 'AllUserStoriesComplete'
raia10 committed a day ago
34e4678

All 3 User Stories and Mediators - DONE
raia10 committed a day ago
8817a72

Commits on Dec 8, 2017

Userautentication added - DONE
raia10 committed 2 days ago
165976c

Merge branch 'ApproveOrderUI'
raia10 committed 2 days ago
d127c8a

Approval Validation - DONE
raia10 committed 2 days ago
fd52f20

Commits on Dec 7, 2017

Merge branch 'Finaltouches'
raia10 committed 3 days ago
ca06687

ALL 3 User Stories - DONE
raia10 committed 3 days ago
3fb0a22

Commits on Dec 6, 2017

Merge branch 'OrderScreen'
raia10 committed 4 days ago
a0aad11

ApproveOrder- Front End & Back End = DONE
raia10 committed 4 days ago
35e02dd

ApproveOrder- Front End & Back End = DONE
raia10 committed 4 days ago
b62b9b9

Commits on Dec 4, 2017

Stock-Update, Clear, DATA GRID - DONE
raia10 committed 6 days ago
be5968a

Commits on Nov 27, 2017

Stock Remove item- DONE
raia10 committed 13 days ago
5cacd5b

added removing item-done
raia10 committed 13 days ago
0814f69

front end design -DONE
raia10 committed 13 days ago
aa4ec02

Stock front & back end-DONE
raia10 committed 13 days ago
e0c6746

Commits on Nov 25, 2017

stock screen ui - done
raia10 committed 15 days ago
777b243

Staff Table and Login FORM- DONE
raia10 committed 15 days ago
1d0bd0e

Manager Login - DONE
raia10 committed 15 days ago
9582a3d

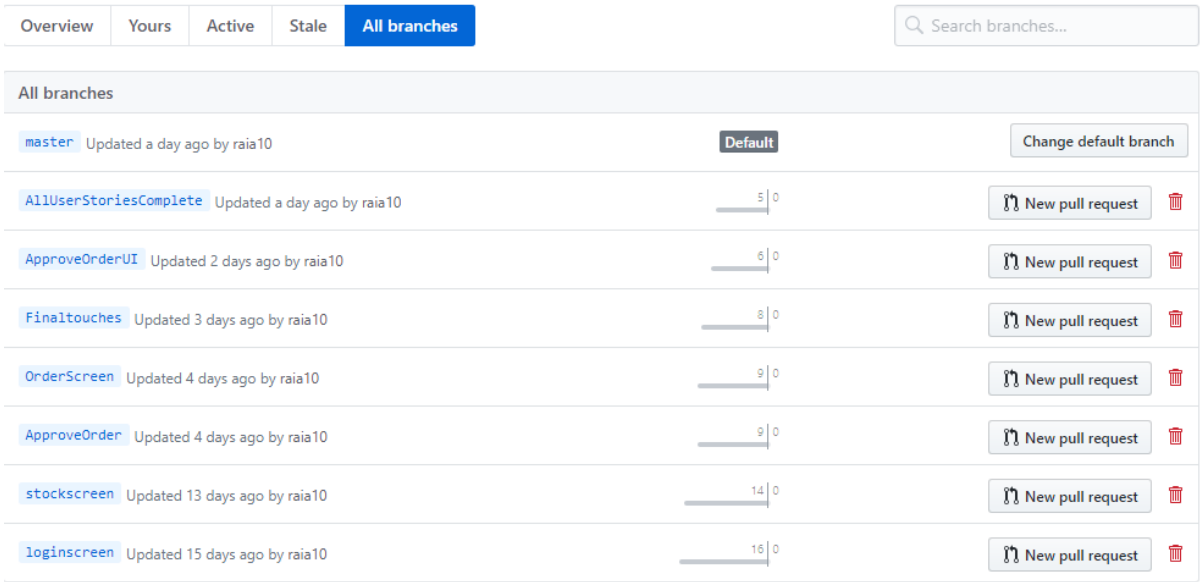
Commits on Nov 24, 2017

form design done
raia10 committed 15 days ago
9f4b4e9

Login Screen
raia10 committed 16 days ago
c3c69cb

task update
raia10 committed 16 days ago
1bbbdfc

Project Start
raia10 committed 16 days ago
5d2d889

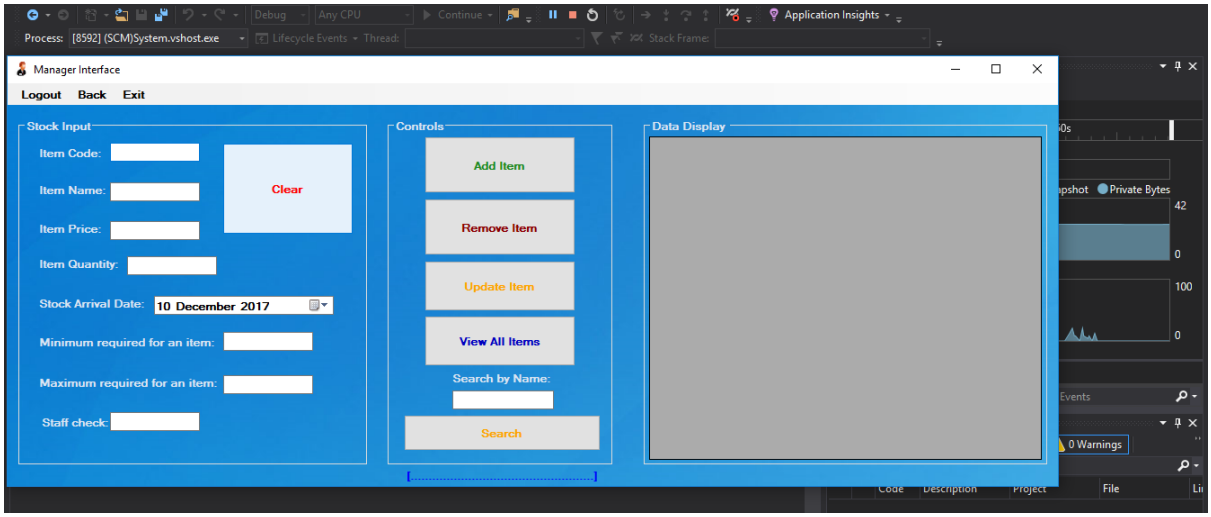


- Different branches used to control different versions of the SCM system

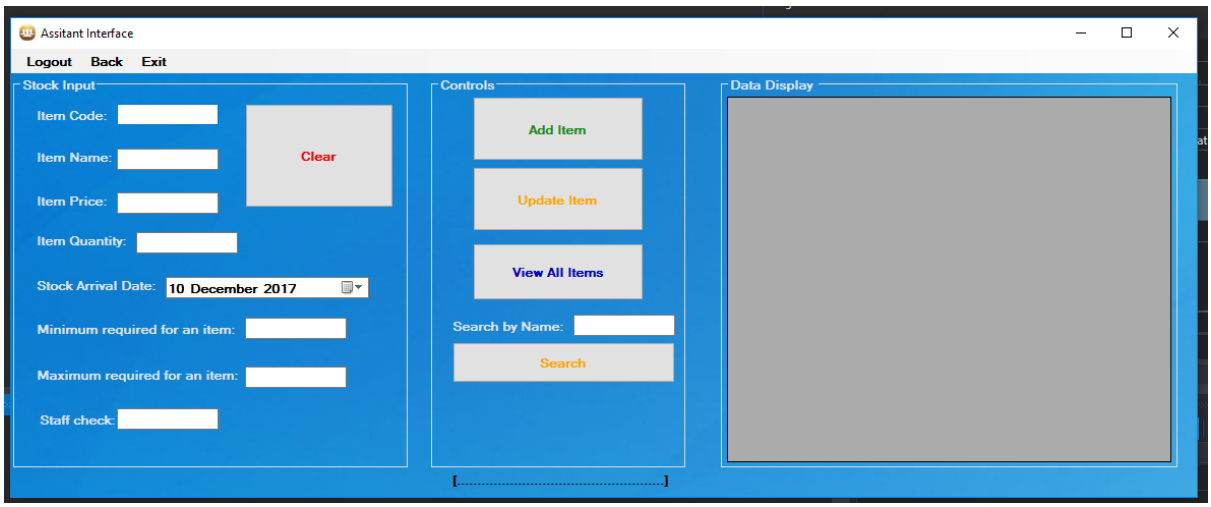
Task 3

Program Output (Screenshots) with annotation of the THREE user stories.

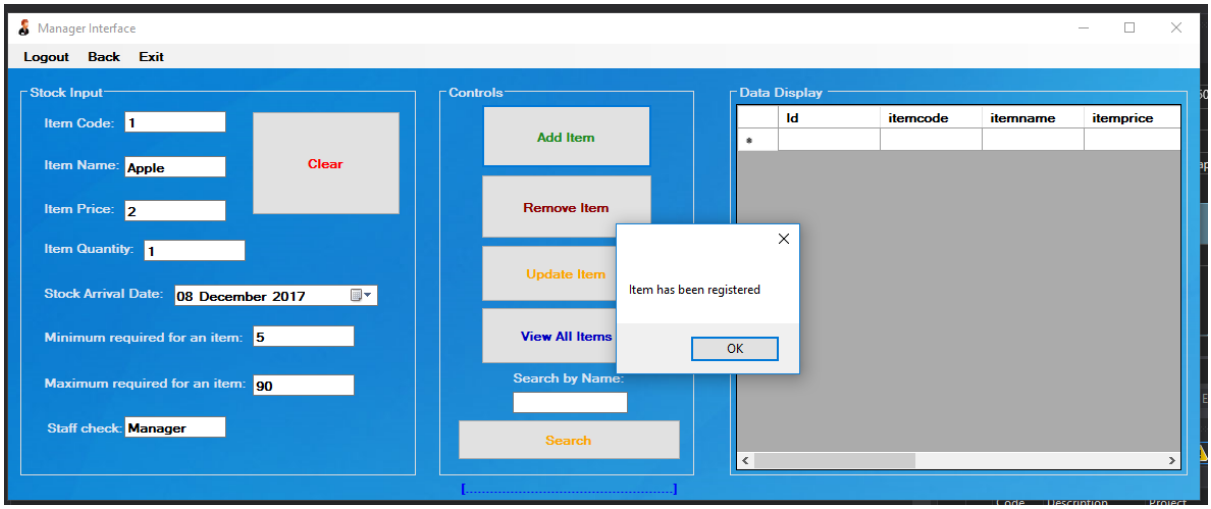
User Story 1 & 3: Adding, removing (Only Manager), Updating and Searching stock items for Manager and Stock Assistant.



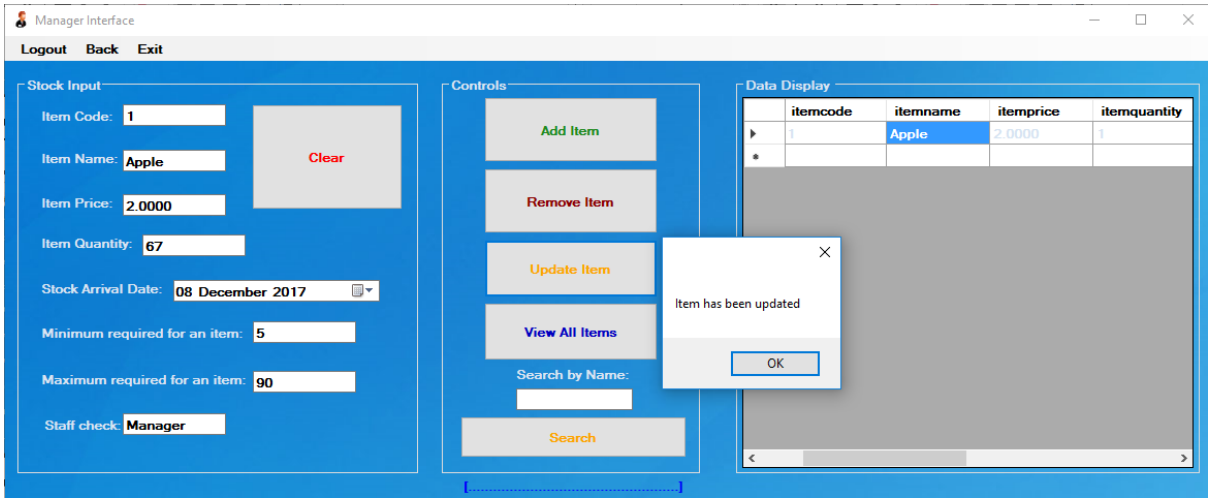
- Manager can Add, Remove, Update and Search for items within the database. Only Manager can delete items in the database.



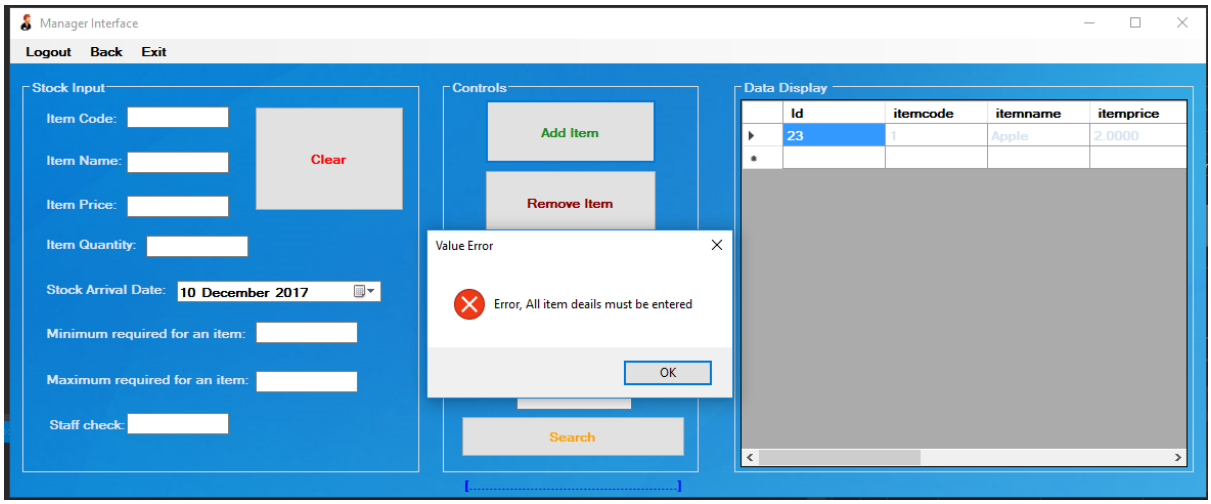
- Stock Assistant Can Add, Update and Search for items within the database. Stock Assistant cannot delete an item from the database.



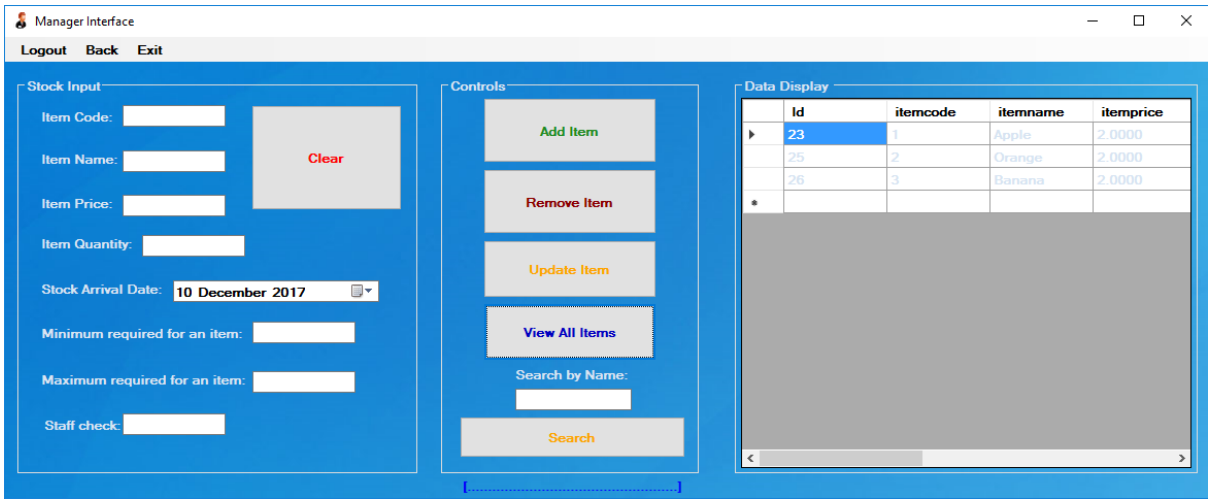
- Both users can add new item into the database of their choice. Message box will pop up on the screen to confirm the item has been added to notify the user.



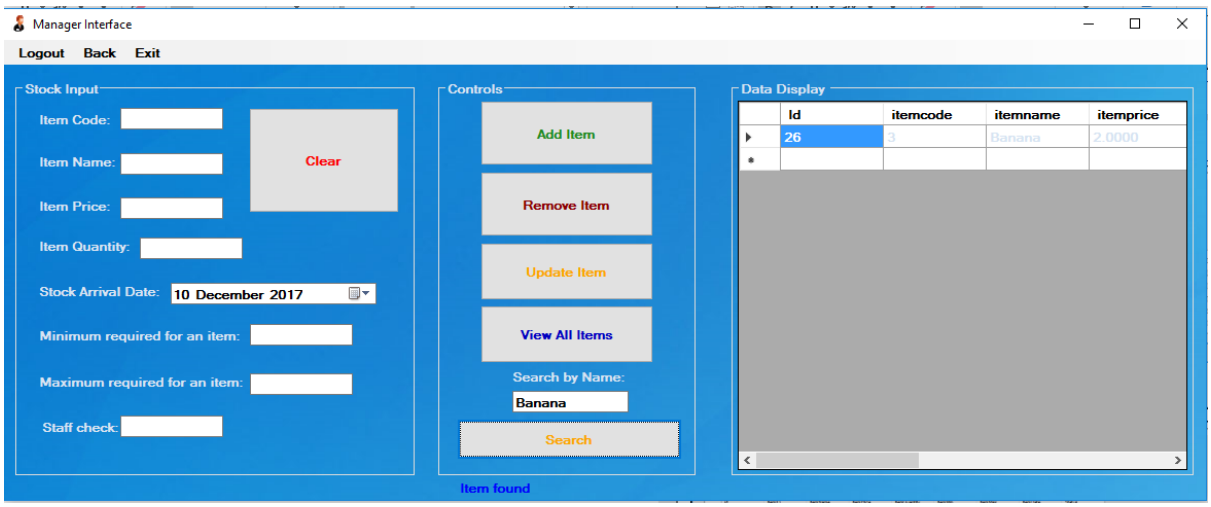
- Both users can update values of existing items in the database. In the screenshot the item quantity of the item name 'Apple' has been changed from 1 to 67. A massage box has appeared to let the user know that the change was executed and was successful.



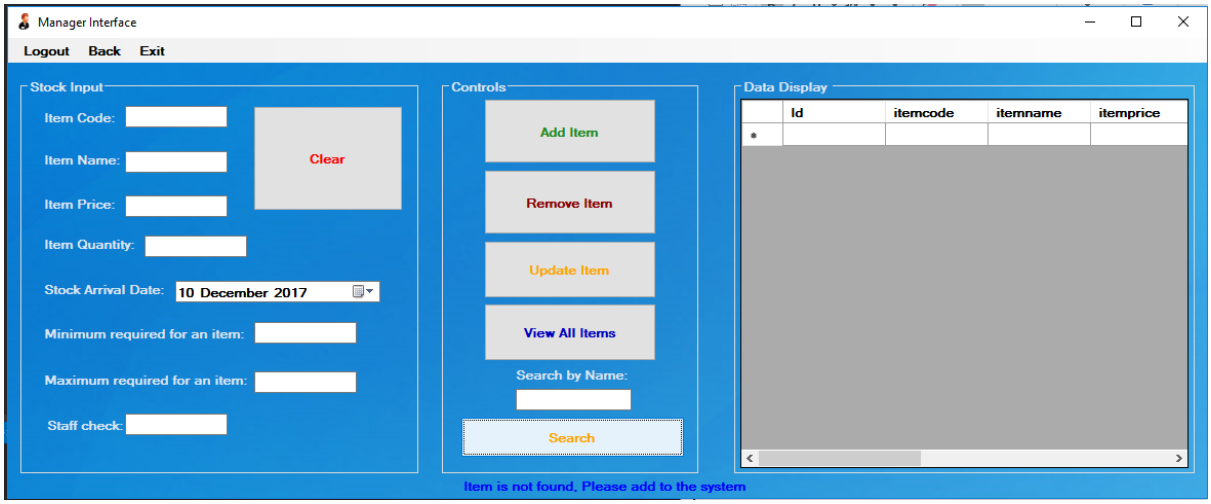
- Validation process, which makes sure that all the fields are filled when the user tries to add a new item.



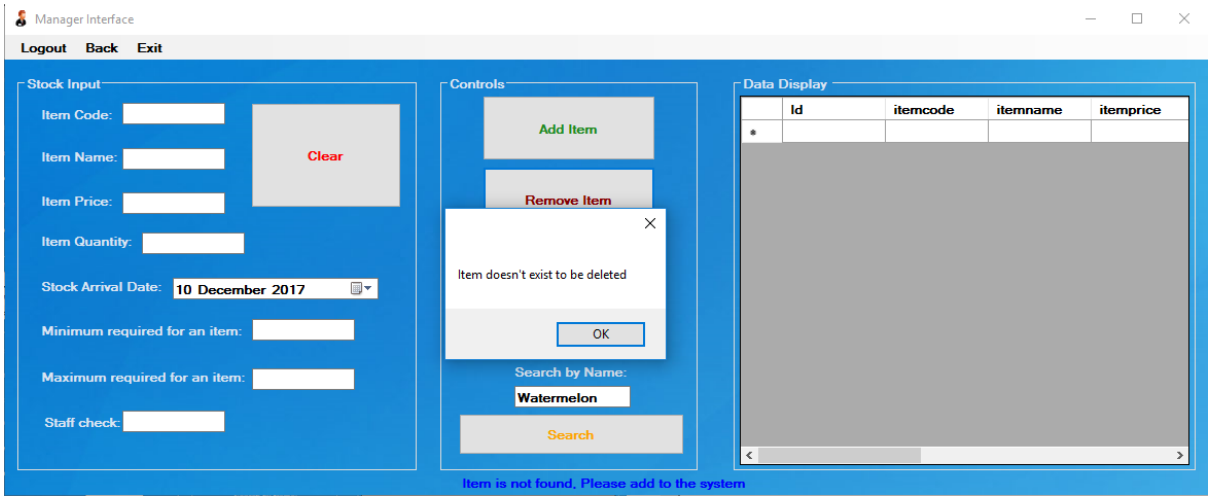
- This feature is when the database is loaded with massive number of items. The user types an 'item name' or 'item code'. For example, in this context 'Banana is searched'. Below is the result.



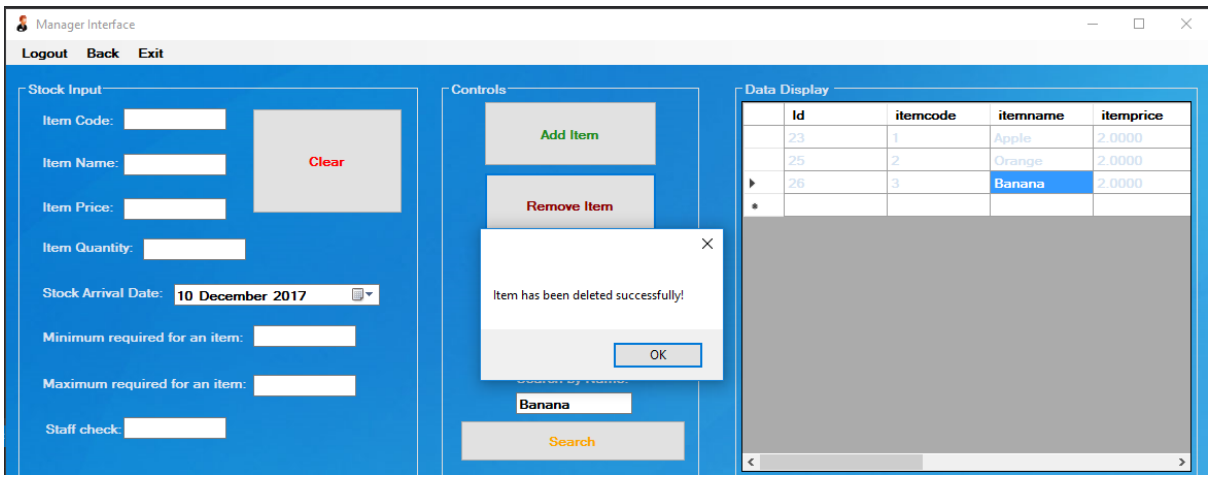
- As expected, only banana is found in the database and displayed on the Data Grid View. The label underneath is also changed to 'Item found', because the item was found in the database to notify the user.



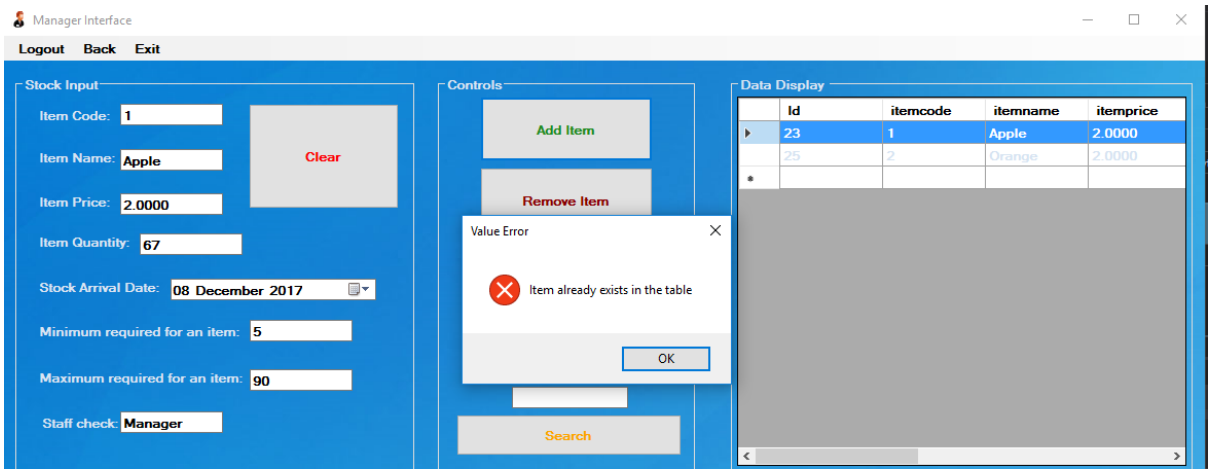
- If the item is not found it would say 'Item not found. Please add to the system' to notify the user.



- If the user enters an item to delete which is not in the database, the system should use the validation processor to alert the user of the item not being in the database. In this context, Watermelon is being asked to delete but it doesn't exist in the system, therefore an error message is thrown to notify the user.

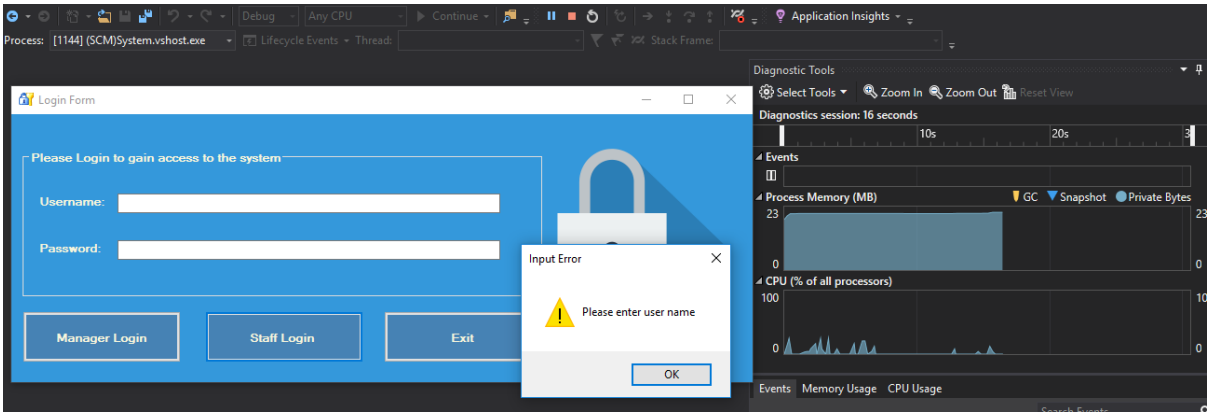


- If the user enters an item to delete, which exists in the system then the item is deleted. In this context, 'Banana' item is deleted as it exists in the system.

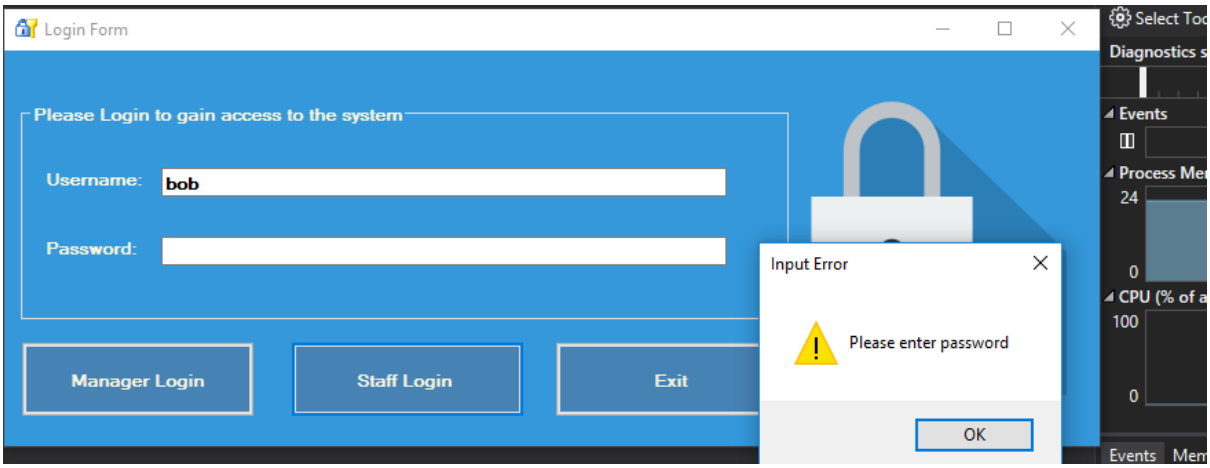


- This validation processor makes sure that the item with same name and code is not entered twice, to prevent duplication of items. In this context, user is trying to add the same item 'Apple' and code '1' into the system which already exists, and the processor will throw an error message to notify the user, item already exists in the system and cannot be added.

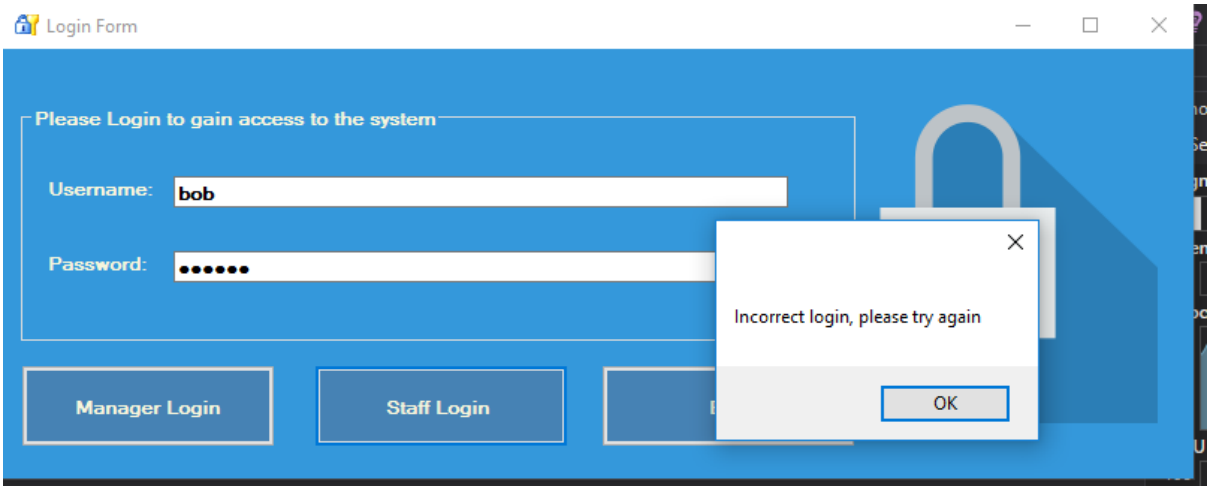
User Story 4: All staff be able to Login using their credentials and view stock status



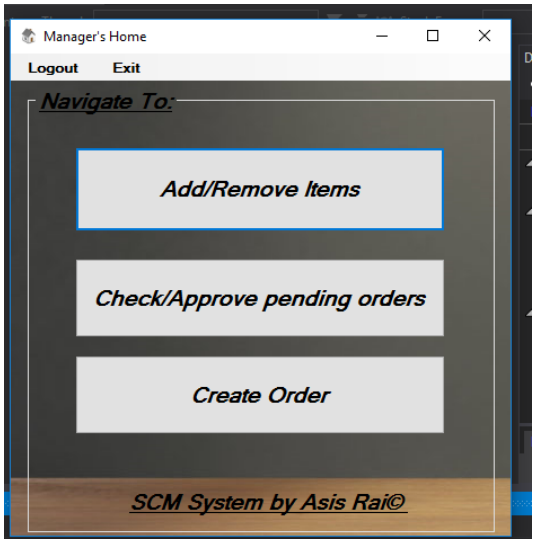
- This is the main Login form when the system starts, If the manager wants to login then he/she should just enter their credentials and press 'Manager Login' and if other staff wants to Login then they should enter their credentials and press 'Staff Login'. This is a validation processor to make sure fields are not empty. The user has to enter a username to login.

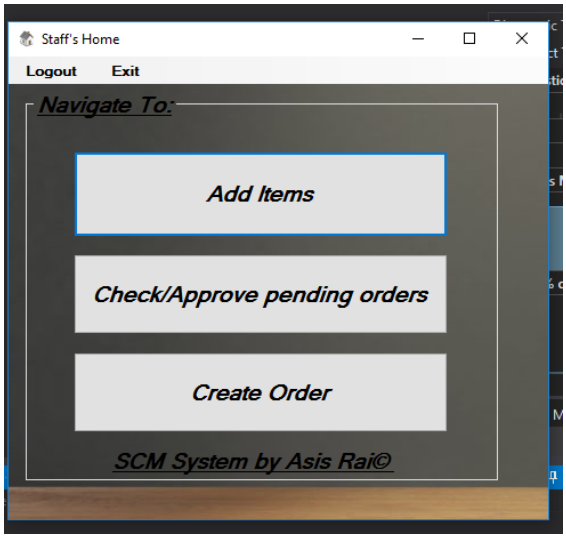


- Similarly, if only the username is entered without entering the password then the validation processor will throw an error message to notify the user that password also must be entered to Login.

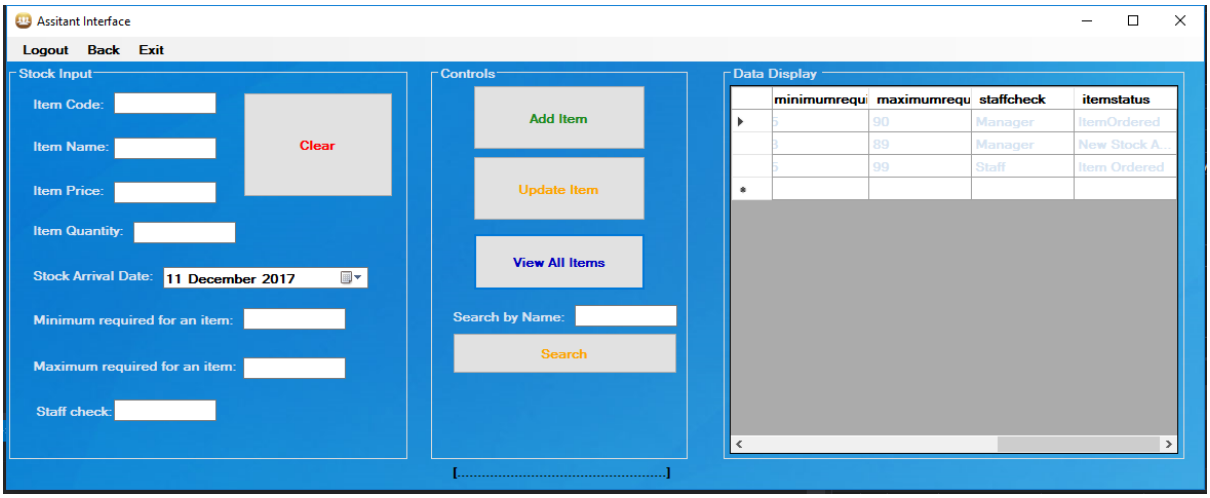


- If the entered credentials are wrong then the validation processor will make sure that user checks their credentials again and entered the correct credentials, the processor will throw an error message to notify the user.

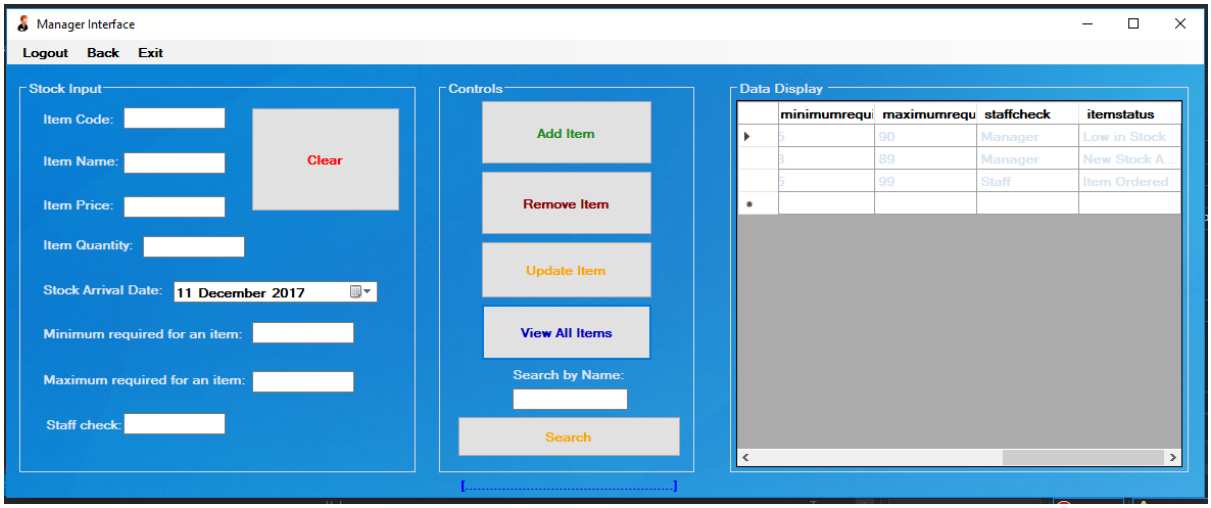




- These are two different interfaces for different user types. If the manager logs into the system, then he/she will be directed to 'Managers Home' interface where he/she can only access the controls that is allowed. Similarly, when Staff Logs into the system then he/she will be directed to 'Staff's Home' interface, where he/she can only have access to the controls that is allowed. In this context, the staff wants to check the status of items in the database therefore he/she can do that by going to 'Add Items'.
- Any staff can create new Order for new stock for items which are 'Low in Stock' by going to 'Create Order'.
- Stock Control assistants can go to 'Check/Approve pending orders' to cross check between the items ordered and the items that have arrived, then they can 'Delete' if the order is wrong otherwise they can choose to 'Approve', if the order is right and the item quantity(stock) of the selected item will be updated in the database.

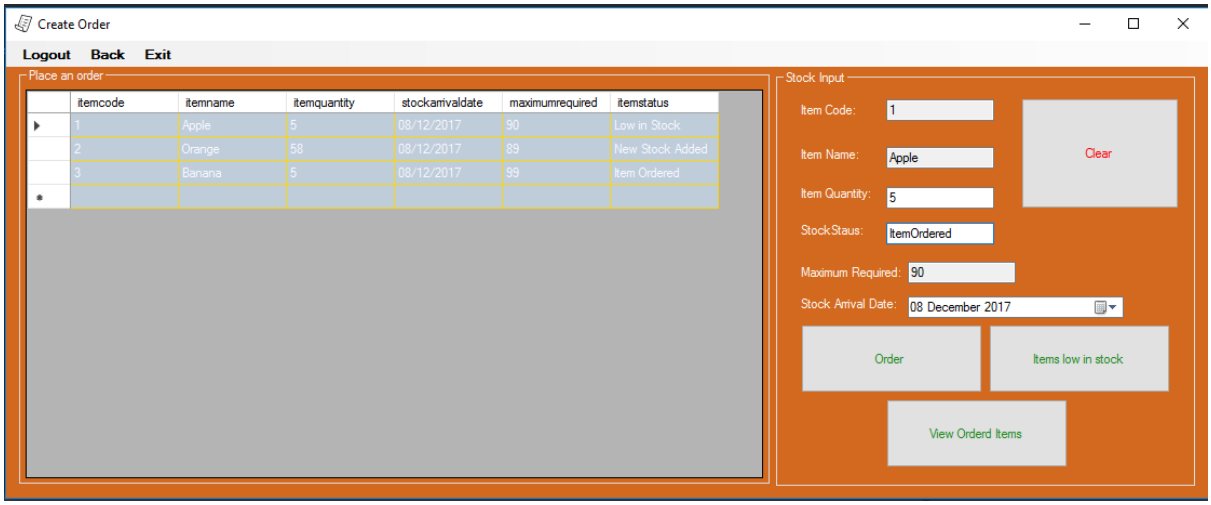


- This is where other Staffs can view all the items and their status by clicking on 'View All Items'. All item status is displayed on the Data Grid view.

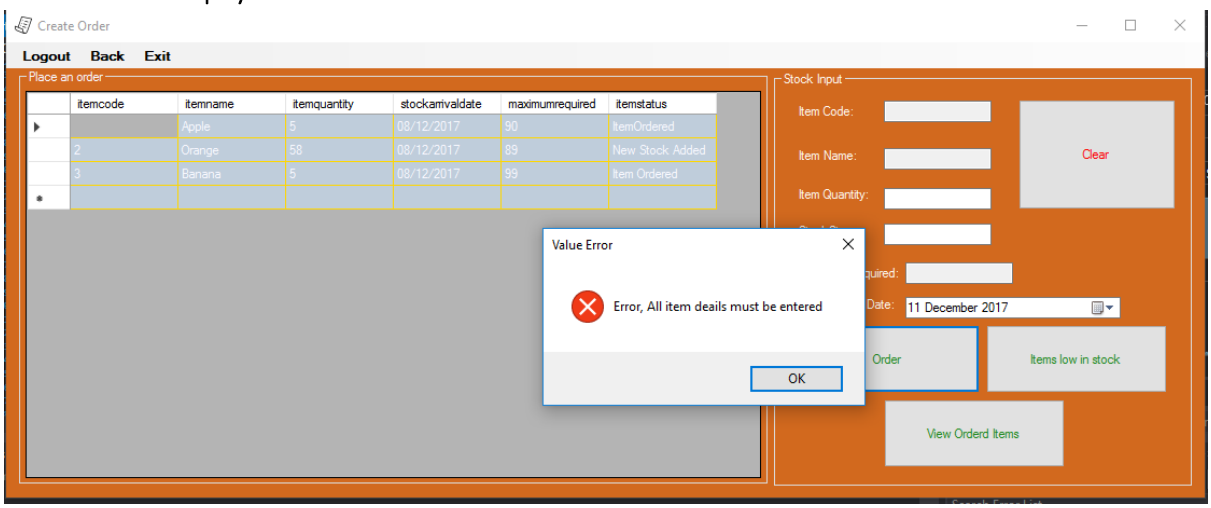


- This is where Manager can view all the items and their status by clicking on 'View All Items'. All item status is displayed on the Data Grid view. As sees on the screenshot, the manager has the option to delete any items from the database, however other staff do not.

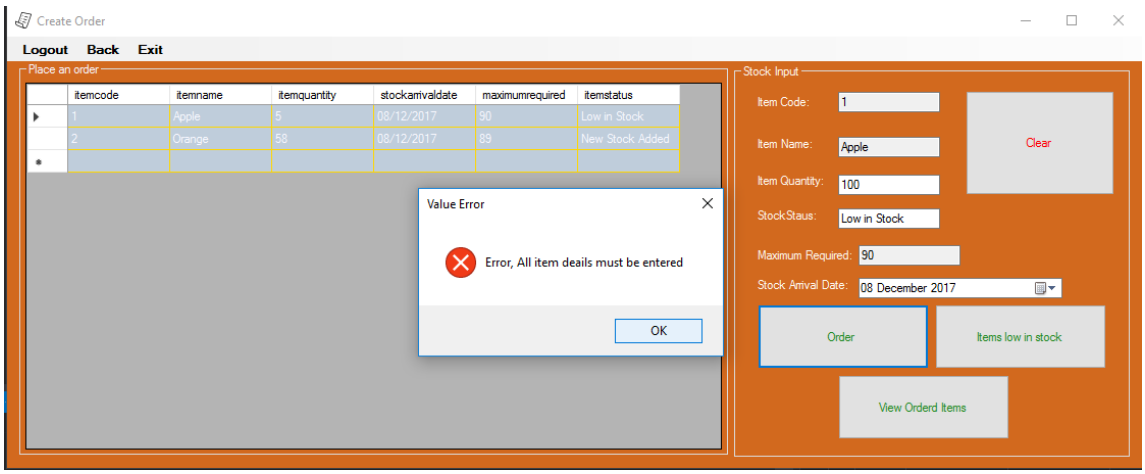
User Story 2: Place Orders if item stock low and sign off arrived orders



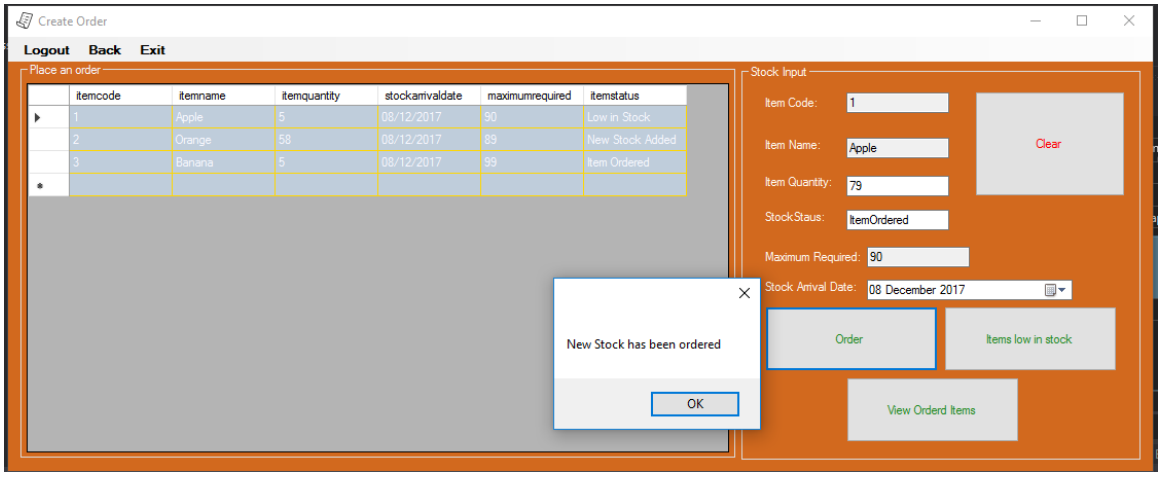
- This is where the Stock Control Assistant can create new orders for the items that are 'Low in Stock'.
- All the orders that are 'Low in Stock' will be displayed once the user presses 'Items low in stock', the data will be displayed on the Data Grid view.



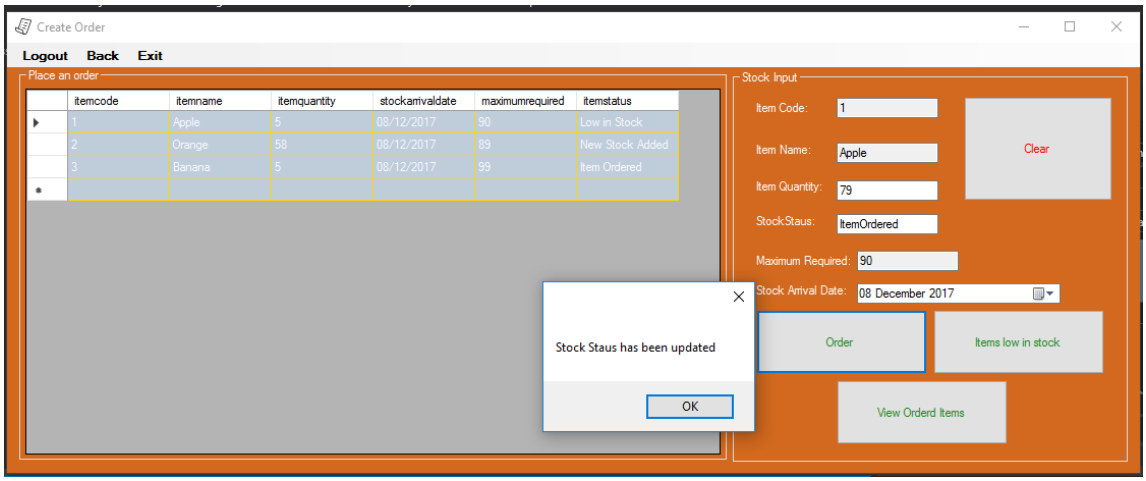
- Form contains a Validation processor, which makes sure that every item detail is filled to Oder a new stock of an item.



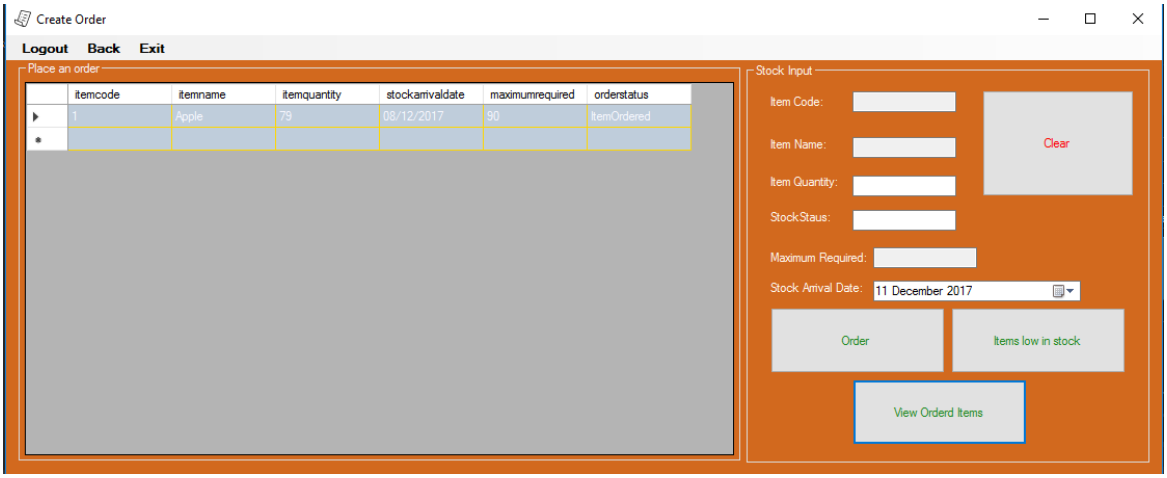
- This is another Validation processor, where it makes sure that the item quantity the user is ordering, cannot be more than the 'Maximum required'. This makes sure that the orders are placed rightly.



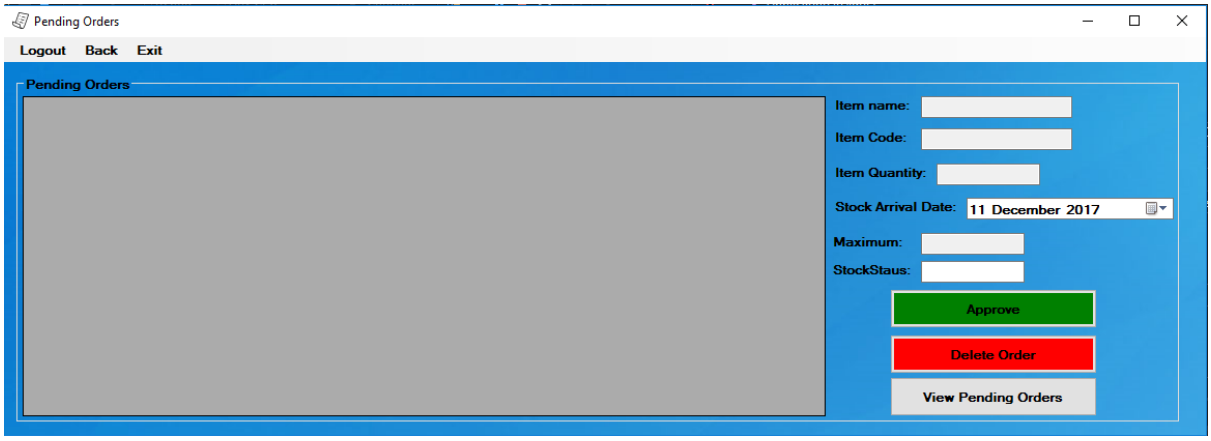
- After the user creates a new order successfully, the order processor notifies the user of the success.
- Notification processor will also alert the user with a message box pop up, to notify the user of the success.



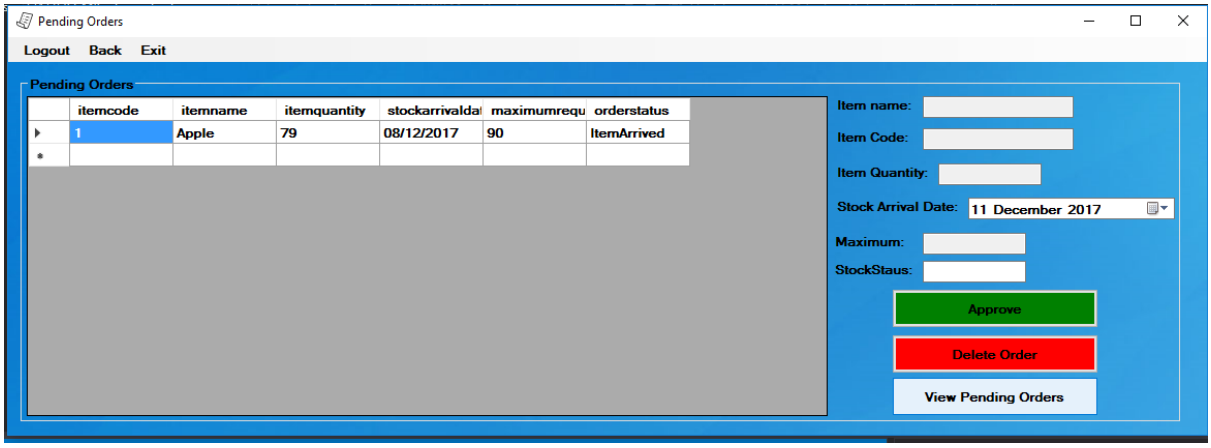
- The order processor also notifies the user, that the Order Status has been updated in the main Stock table for all users to see.
- Notification processor will also alert the user with a message box pop up, to notify the user of the success.



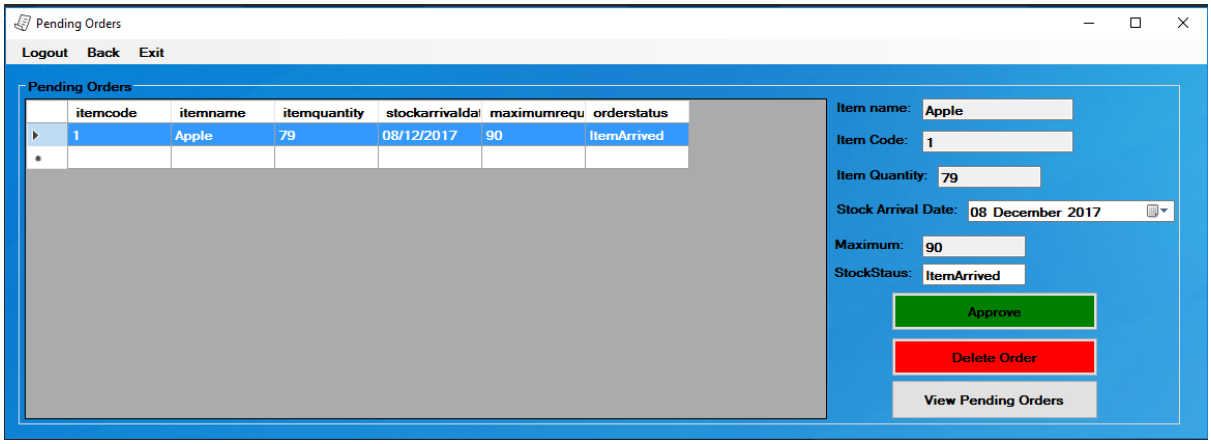
- The order placed has been added into the Orders Table and with its status changed to 'Item Ordered'. All users can see this by pressing 'View Ordered Items'.



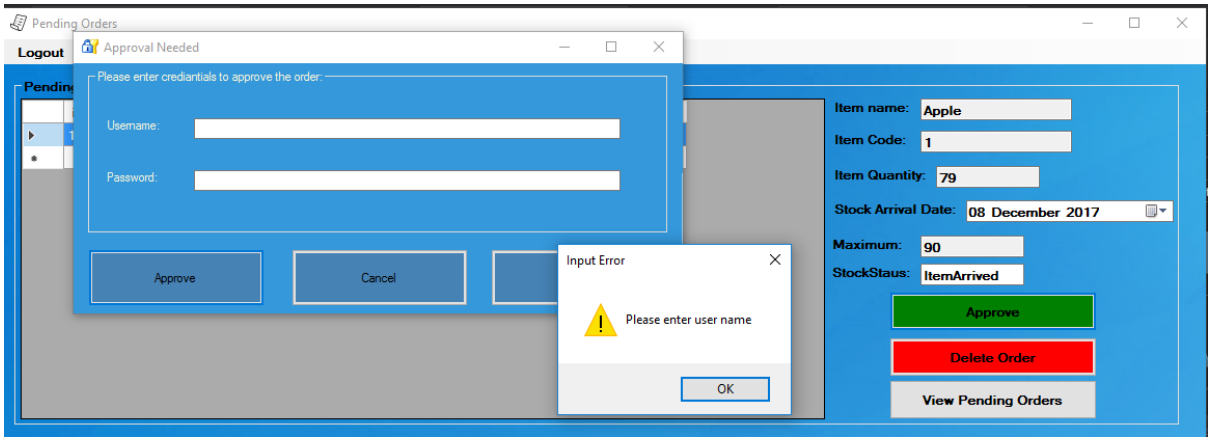
- The user goes back and selects 'View/Approve Pending orders'. Pending orders form will be opened where the 'Stock Control Assistant' will be able to 'View Pending Orders', select it and Approve or delete the order.



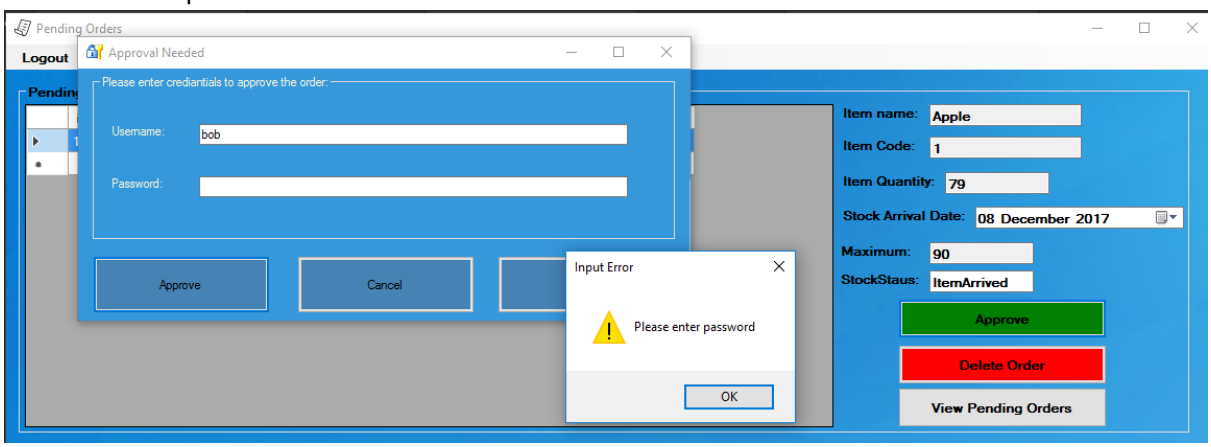
- When the user clicks 'View Pending Orders', the Data Grid will display all orders that have arrived from the suppliers. In this context, the order placed earlier by the user has now arrived and awaiting approval.



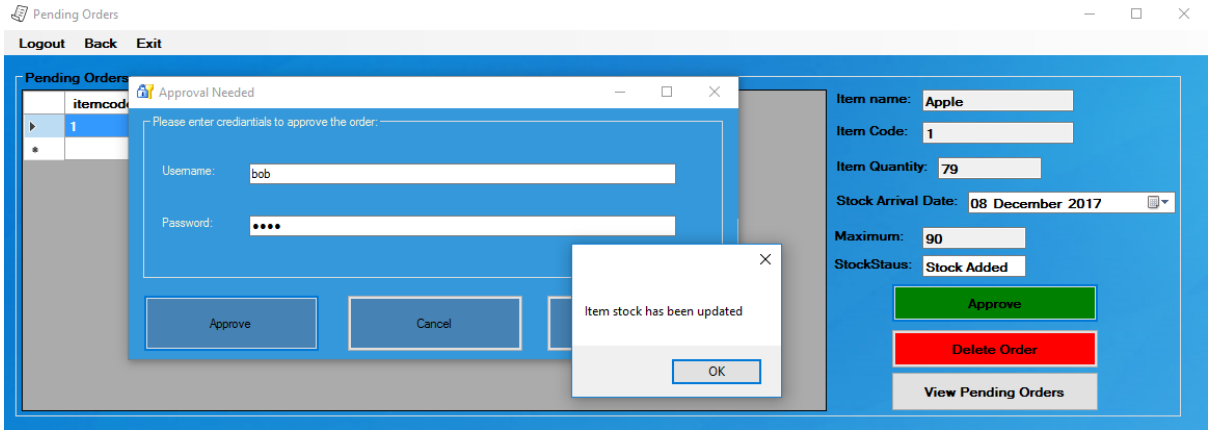
- The assistant selects the he/she wants which will automatically fill the text boxes in the form and presses 'Approve' button, this will activate the 'Authentication Processor'.



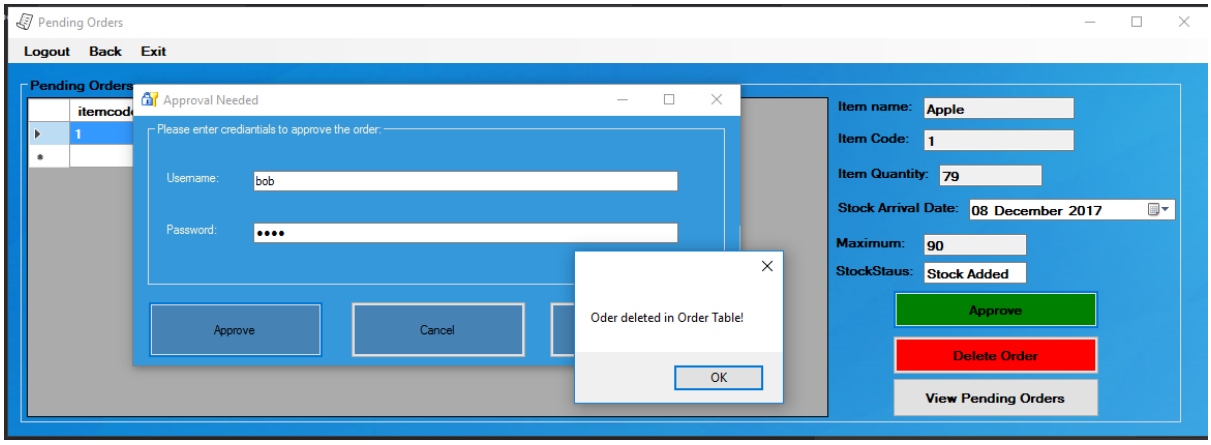
- The Authentication processor will ask to verify the 'Stock Control Assistant's' credentials again. If the field are left empty, validation processor will pop up a message box to notify the user that both fields must be filled to proceed.



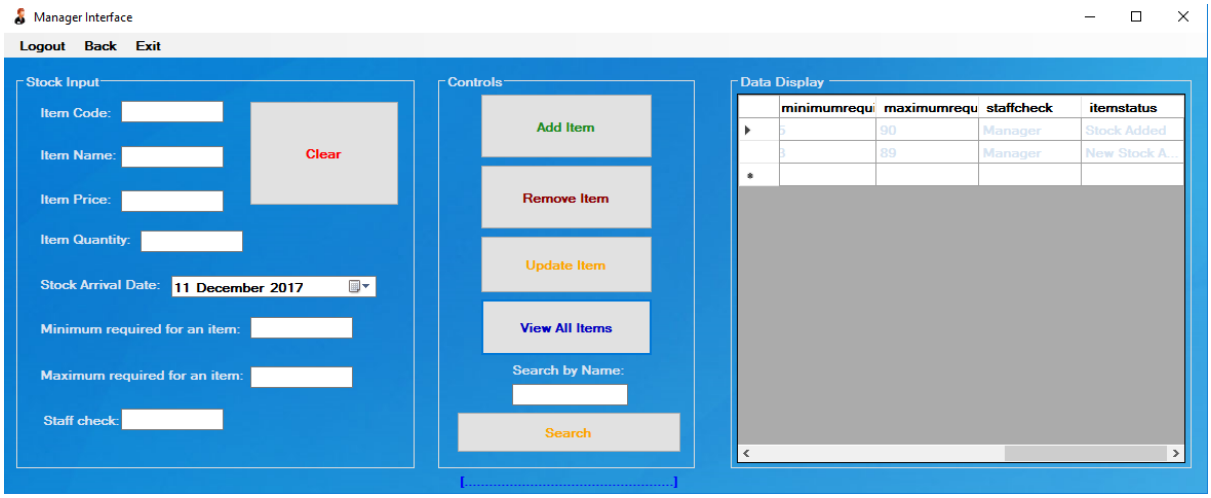
- The validation processor, will also notify the user that password also must be entered to proceed if only username is entered.



- If the entered credentials match with the credentials that was logged in with/credentials stored in the database of the 'Stock Control Assistant's', Update processor update the stock level of the selected item in the database.
- The update processor will also update the stock status from 'Item Arrived' to 'Stock Added' if the credentials are entered correctly.
- Notification processor will also alert the user with a message box pop up, to notify the user of the success.



- Delete processor will also delete the selected pending order from the current table because the 'Pending Order' was just approved by the stock control assistant.
- Notification processor will also alert the user with a message box pop up, to notify the user of the success.



- All staff are now able to see the most updated stock details in the system, including the new stock that was just added, 'Stock Added'.

Task 4

Critical Evaluation of chosen agile technique applied:

BREAKDOWN OF TASKS:

User Story ID	Priority	As a *<type of user>*	I want to *<perform some task>*	so that I can *<achieve some goal>*	Final Story?
1	High	Stock Control Assistants	Enter the details of the stock items	Add, Update and Search items stored in the database	Final Story
2	High	Stock Control Assistants	Place Orders if item stock low and sign off arrived orders	Restock items which are low and check if arrived orders are correct	Final Story
3	Medium	Store Manager	Add new items and delete existing items	Add new items and remove existing items in the database	Final Story
4	Medium	All Staff	Log-in to the sysem using using their unique credentials	View status of stocks in the database	Final Story
5	Low	Stock Control Assistants	Stocks are checked against the sales made	See stocks which are low and needs to be deleted	Rejected
6	Low	Store Manager	To be able to view stock reports	to see trends in sales	Rejected

User story 1: As a Stock control assistant, I want to be able to add, update and search for items in the database so that I can enter the details for the stock items to be stored in the database.

- Based on the architecture, the tasks may include:
- Create processes which allows to add items, update items, search for items and view the most up to date items.
- Create Interface which allows to view all the items in the database.
- Develop a database to hold all items.

- Based on the following Definition of Done for the user story:
- Automated Unit testing to test the credentials, to prove the user's identity
- Completion of the code
- Fully updated Documentation
-
- Tasks may include:
- To write tests for Automated Unit testing for user login

User story 2: As a Stock control assistant, I want to be able to re-stock the items that are low in stock and when the stock arrives, match the arrived stock with the ordered stock. Therefore, I can place orders for new items and sign off on orders that are arriving.

- Based on the architecture, the tasks may include:
- Add an Interface which allows user to view all the orders ordered that are pending and need for approval.
- Create a process which refreshes the pending orders
- Develop a Database to store the pending orders created
- Create a process which when confirmed pending orders, will delete it from the pending orders
- Create a process which will return/delete a pending order
- Create a process which will update the stock status of items when ordered/signed off
- Create a process where user can search for a pending order

- Based on the following Definition of Done for the user story:
- Automated Unit testing to test the credentials, to prove the user's identity
- Completion of the code
- Fully updated Documentation

- Tasks may include:
- To write tests for Automated Unit testing for user login

User story 3: As a store Manager, I want to be able to add and remove items in a database. Therefore, I can enter new items and delete items which are not needed.

- Based on the architecture, the tasks may include:
- Add an Interface which allows to view all the items in the database.
- Create processes to add, update, delete, search and refresh items.
- Develop a Database to store items.

- Based on the following Definition of Done for the user story:
- Automated Unit testing to test the credentials, to prove the user's identity
- Completion of the code
- Fully updated Documentation

- Tasks may include:
- To write tests for Automated Unit testing for user login

User story 4: As a store Staff, I want to be able to Log-in to the system with my unique username and password, so that I can view the stock status of items.

- Based on the architecture, the tasks may include:
- Add an Interface which allows to view all the items in the database.
- Create a process which refreshes the items in the database.
- Develop a Database which stores the items.

- Based on the following Definition of Done for the user story:
- Automated Unit testing to test the credentials, to prove the user's identity
- Completion of the code
- Fully updated Documentation

- Tasks may include:
- To write tests for Automated Unit testing for user login

Benefits of chosen agile technique:

- Very handy for planning out the tasks as it can give an estimate of how long it will take, therefore picking out tasks that are not time consuming and it also gives an idea of how difficult tasks are going to be, therefore helping you decide on more balanced tasks over the head ones and within your abilities.
- Helps to prioritise of tasks of user stories, allowing user stories to be develop and created over time, therefore completing the user stories on time.

Drawbacks of chosen agile technique:

- As User stories tasks become more detailed overtime, causing tasks to keep changing in search of detail, which would be a waste of time. It could also

Solution:

- I avoided the issue given by the chosen agile technique by understanding my user stories, by planning each task so that each functionality would be achieved on time and with no extension of tasks adding.

Limitations:

- Burndown chart is not part of the chosen agile technique, it would have been helped to visually to see how long it would have taken for each task, making more use of the spare time used to improve the functionalities even more.

Lessons learnt/Suggestions:

- Spent long time on one user story, having little time for the others and limiting others by not having detailed user stories than the main ones. I would suggest spent more time on other user stories to make them more detailed and creating a burn down chart so that the tasks of the user stories can be finished on time.

Task 5

Assessment of maturity of the project using CMMI Model for the chosen process areas: