Name - Asis Rai

Student ID Number – 6528683

340CT – Software Quality and Process Management

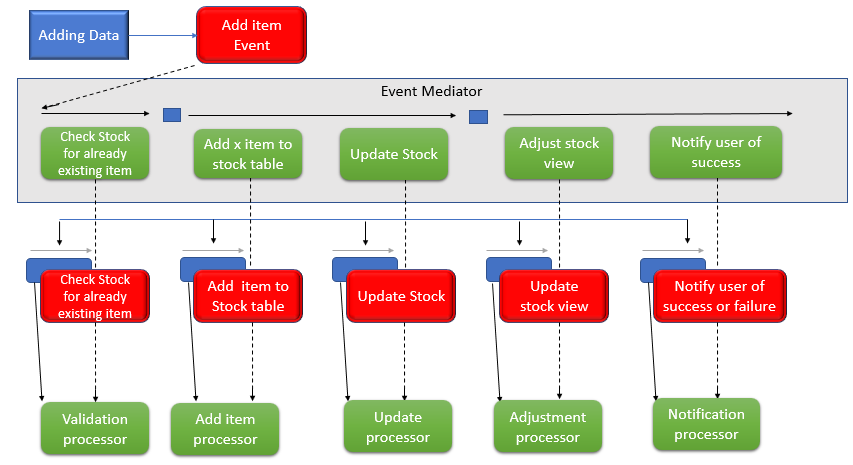
Module Leader – Yih-Ling Hedley

**Task 1**

Annotated Mediator diagrams of chosen functionality with annotation:

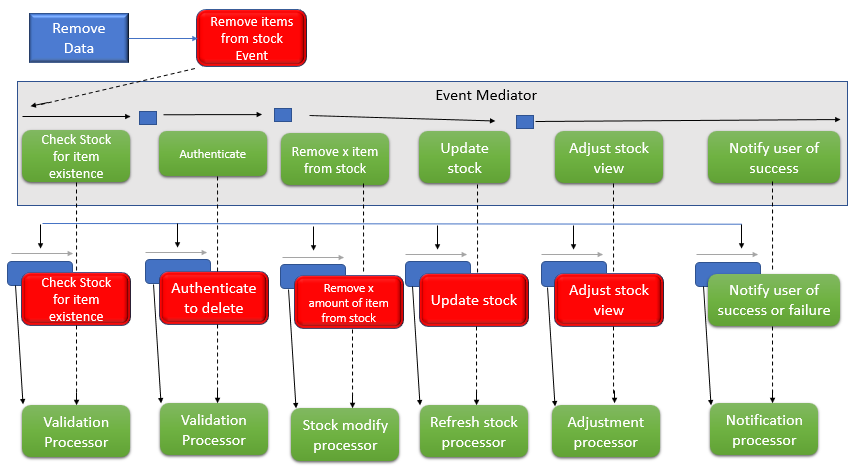
**User Story 1 & 3: Adding, removing (Only Manager), Updating and Searching stock items for Manager and Stock Assistant.**

**Adding:**



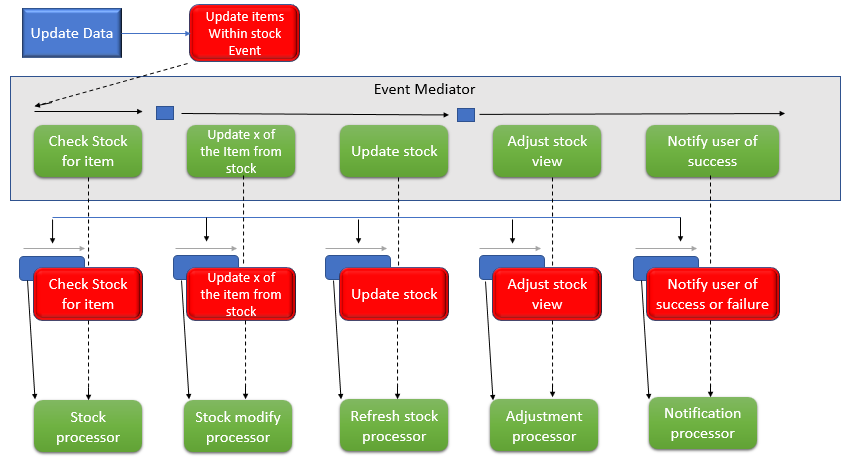
* **Annotation:**  The Event Mediator Diagram above is the process when the user tries to add items into the database.
* The first process of the process to be executed by the mediator is the validation process which will check if the details entered by the user already exists in the database. If it 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:**



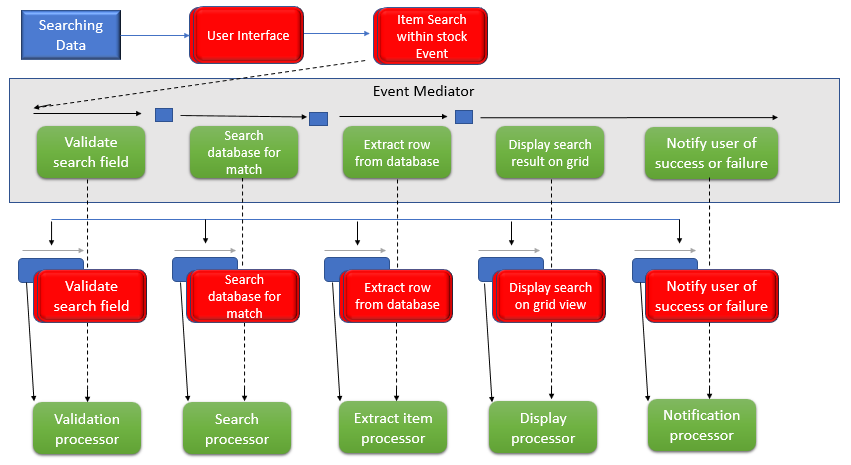
* **Annotation:** The Event Mediator Diagram above is the process when the user tries to Remove items from the database.
* The first process will check if the item/items the user is trying to delete does exists in the system.
* The second process will execute if the first is successful, it will ask for 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:**



* **Annotation:** 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.

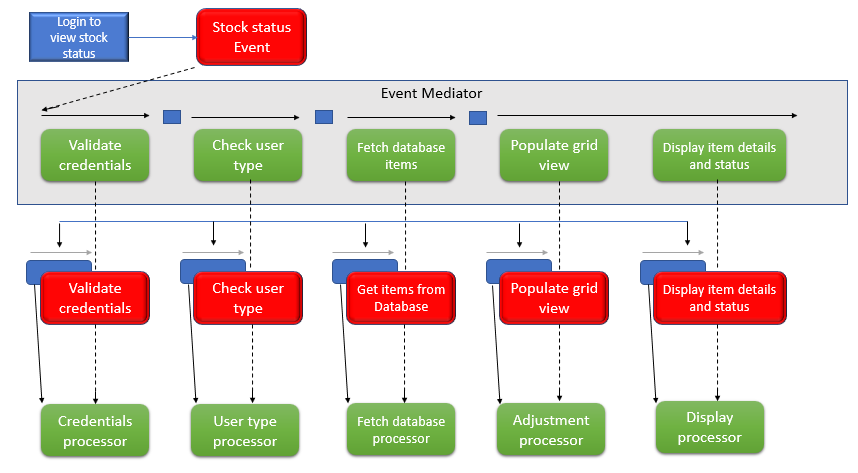
**Searching:**



* **Annotation:** 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**

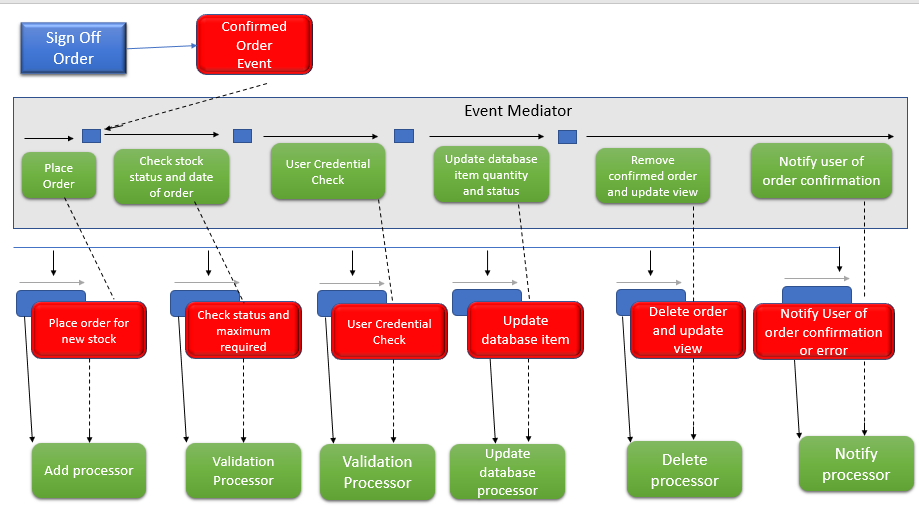
**Login to view stock status:**



* **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:**



* **Annotation:**  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**

1. **private** **void** button1\_Click(**object** sender, EventArgs e) //Add item button to trigger Add event
2. {
3. checkifitemexsists();
4. }
6. **private** **void** checkifitemexsists() //validation process
7. {
8. mediator.itemduplication\_validation(textBox1.Text, textBox2.Text);//checks if the item already exists in Stock Table
9. addtoStockTable(); //calls this function, which calls the add to stock table mediator
11. }

14. **private** **void** addtoStockTable() //add process
15. {
16. //calling the mediator to perform the add process, adding to StockTable
17. mediator.Add\_Stocktable(textBox1.Text, textBox2.Text, textBox3.Text, textBox4.Text, dateTimePicker1.Text, textBox6.Text, textBox7.Text, textBox8.Text);
19. }

22. **private** **void** getPrimaryKey() //Give new id to the new item
23. {
24. **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"))
25. {
26. **try**
27. {
28. Connection.Open();
29. SqlCommand cmd = **new** SqlCommand(@"SELECT MAX(Id)+1 FROM StockTable;", Connection);
30. Connection.Close();
32. }
34. **catch** (Exception ex)
35. {
36. MessageBox.Show("Unexpected Error has occured: " + ex.Message);
37. }
38. }
39. }

**Searching**

1. **private** **void** button2\_Click(**object** sender, EventArgs e) //search button
2. {
3. mediator.search\_mediator(textBox1.Text, textBox2.Text); //search mediator with item name and item code to search
5. //SqlDataReader rd = checkid.ExecuteReader();
6. 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");
7. SqlDataAdapter checkup = **new** SqlDataAdapter("SELECT \* FROM [StockTable] WHERE itemname ='" + textBox9.Text + "'", con); //this will get all the data
8. DataTable sd = **new** DataTable();
10. checkup.Fill(sd);
11. dataGridView1.DataSource = sd;
12. }

**Deleting**

1. **private** **void** button5\_Click(**object** sender, EventArgs e) //remove button to trigger delete event
2. {
3. DataGridViewRow row = dataGridView1.CurrentCell.OwningRow;//grab a reference to the current row
4. mediator.delete\_StockTable(row); //calls the delete process from delete mediator to delete the row of the StockTable
5. 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
6. dataGridView1.Update();//updates the data grid with new updated data from the Stock Table
7. }

**Updating**

1. **private** **void** button3\_Click(**object** sender, EventArgs e) //update button to trigger update event
2. {
3. 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
4. mediator.itemduplication\_validation(textBox1.Text, textBox1.Text); //checks if the data entered is already inside the StockTable
5. 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
6. dataGridView1.DataSource = mediator.refresh\_StockTable(); //refreshes the Stocktable and puts it into the Data Grid view
7. dataGridView1.Update(); //Data grid view gets updated with the updated data
8. }

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

**Manager Login:**

1. **private** **void** button1\_Click(**object** sender, EventArgs e) //manager login button
2. {
3. mediator.textbox\_Validation\_LoginForm(textUsername.Text, textPassword.Text); //Validation process to check if username and password is entered
4. mediator.mmanager\_login\_validation(textUsername.Text, textPassword.Text); //checks to see if the entered username and password matches with the one stored in database
5. mainui main = **new** mainui();
6. main.Show(); //opens assistant interface form
7. **this**.Close(); //form is closed when all proceeses are fini
8. }

**Any Other Staff Login:**

1. **private** **void** button2\_Click(**object** sender, EventArgs e) //staff login button
2. {
3. mediator.textbox\_Validation\_LoginForm(textUsername.Text, textPassword.Text); //Validation process to check if username and password is entered
4. mediator.loginandapprove\_validation(textUsername.Text, textPassword.Text); //checks to see if the entered username and password matches with the one stored in database
5. MainUI.assistantmain main = **new** MainUI.assistantmain();
6. main.Show(); //opens assistant interface form
7. **this**.Close(); //form is closed when all proceeses are finished
8. }

**View stock status:**

1. **private** **void** button4\_Click\_1(**object** sender, EventArgs e) //view all items button to trigger an event
2. {
3. DataGridViewRow row = dataGridView1.CurrentCell.OwningRow;//grab a reference to the current row
4. dataGridView1.DataSource = mediator.refresh\_StockTable();//calls the refresh mediator to get the updated data from the Stock Table
5. dataGridView1.Update(); //updates the data grid with new updated data from the Stock Table
6. }
8. **private** **void** dataGridView1\_CellContentClick(**object** sender, DataGridViewCellEventArgs e) //Grid Display
9. {
10. **if** (e.RowIndex >= 0)
11. {
12. //gets a collection that contains all the rows
13. DataGridViewRow row = **this**.dataGridView1.Rows[e.RowIndex];
14. //populate the textbox from specific value of the coordinates of column and row.
15. textBox1.Text = row.Cells[1].Value.ToString();
16. textBox2.Text = row.Cells[2].Value.ToString();
17. textBox3.Text = row.Cells[3].Value.ToString();
18. textBox4.Text = row.Cells[4].Value.ToString();
19. dateTimePicker1.Text = row.Cells[5].Value.ToString();
20. textBox6.Text = row.Cells[6].Value.ToString();
21. textBox7.Text = row.Cells[7].Value.ToString();
22. textBox8.Text = row.Cells[8].Value.ToString();
23. }
24. }
26. **private** **void** button6\_Click(**object** sender, EventArgs e) //clear button
27. {
28. textBox1.Clear();
29. textBox2.Clear();
30. textBox3.Clear();
31. textBox4.Clear();
32. dateTimePicker1.ResetText();
33. textBox6.Clear();
34. textBox7.Clear();
35. textBox8.Clear();
36. }
37. }

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

**Viewing all items in stock that are low:**

1. **private** **void** viewallitems\_Click(**object** sender, EventArgs e) //vieww all items button
2. {
3. 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");
4. SqlDataAdapter checkup = **new** SqlDataAdapter("SELECT \* FROM StockTable", con); //this will get all the data
5. DataTable sd = **new** DataTable();
7. checkup.Fill(sd);
8. dataGridView1.DataSource = sd;
10. DataTable sd1 = **new** DataTable();
11. //sd1 = sd.DefaultView.ToTable(true, "itemcode", "itemname", "itemquantity", "stockarrivaldate", "minimumrequired", "maximumrequired", "stockstatus", "stockordered");
12. sd1 = sd.DefaultView.ToTable(**true**, "itemcode", "itemname", "itemquantity", "stockarrivaldate", "maximumrequired", "itemstatus");
14. dataGridView1.DataSource = sd1;
15. }

**Stock Control Assistant button to place a new stock Order:**

1. **private** **void** button1\_Click(**object** sender, EventArgs e) //place a new oder button
2. {
4. **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"))
5. {
6. **int** a = **int**.Parse(textBox3.Text);
7. **int** b = **int**.Parse(textBox4.Text);
9. **if** (textBox1.Text == String.Empty || textBox2.Text == String.Empty || textBox3.Text == String.Empty || textBox4.Text == String.Empty || textBox5.Text == String.Empty || a > b)
10. {
11. MessageBox.Show("Error, All item deails must be entered", "Value Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
12. }
14. **else**
15. {
16. **try**
17. {
18. Connection.Open();
19. SqlCommand cmd = **new** SqlCommand(@"INSERT INTO OrderTable ([itemcode], [itemname], [itemquantity], [stockarrivaldate], [maximumrequired], [orderstatus]) VALUES (@itemcode, @itemname, @itemquantity, @stockarrivaldate, @maximumrequired, @orderstatus);", Connection);
21. //string query = "UPDATE StockTable SET orderstatus = @orderstatus2  where orderstatus2 = @orderstatus2 ";
22. //SqlCommand cmd2 = new SqlCommand(query, con);
24. cmd.Parameters.AddWithValue("@itemcode", textBox1.Text);
25. cmd.Parameters.AddWithValue("@itemname", textBox2.Text);
26. cmd.Parameters.AddWithValue("@itemquantity", textBox3.Text);
27. cmd.Parameters.AddWithValue("@stockarrivaldate", dateTimePicker1.Text);
28. cmd.Parameters.AddWithValue("@orderstatus", textBox5.Text);
29. cmd.Parameters.AddWithValue("@maximumrequired", textBox4.Text);


33. **int** i = cmd.ExecuteNonQuery();
34. Connection.Close();
36. **if** (i == 1)
38. {
39. MessageBox.Show("New Stock has been ordered");
40. getPrimaryKey();
41. updatestatus();
43. textBox1.Clear();
44. textBox2.Clear();
45. textBox3.Clear();
46. dateTimePicker1.ResetText();
47. textBox4.Clear();
48. textBox5.Clear();


52. }
53. **else**
54. {
55. MessageBox.Show("Stock could not be ordered, Please Try again");
56. }
57. }
58. **catch** (Exception ex)
59. {
60. MessageBox.Show("Unexpected Error has occured:" + ex.Message);
62. }
63. }
64. }
66. }

69. **private** **void** updatestatus() //update item status //only executes if adding a new order is successful
70. {
72. 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");
74. {
76. **string** query = "UPDATE StockTable SET itemstatus = @itemstatus  where itemname = @itemname";
77. SqlCommand cmd = **new** SqlCommand(query, con);
78. cmd.Parameters.AddWithValue("@itemstatus", textBox5.Text);
79. cmd.Parameters.AddWithValue("@itemname", textBox2.Text);
80. cmd.Connection.Open();
82. **try**
83. {
84. **int** i = cmd.ExecuteNonQuery();
85. cmd.Connection.Close();
87. **if** (i == 1)
88. {
89. MessageBox.Show("Stock Staus has been updated");
91. }
92. }
93. **catch** (Exception ex)
94. {
95. **throw** **new** Exception("Unexpected Error has occured: " + ex.Message);
96. }
98. }
100. }
102. **private** **void** getPrimaryKey() //Give new id to the new item
103. {
104. **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"))
105. {
106. **try**
107. {
108. Connection.Open();
109. SqlCommand cmd = **new** SqlCommand(@"SELECT MAX(Id)+1 FROM OrderTable;", Connection);
110. Connection.Close();
112. }
114. **catch** (Exception ex)
115. {
116. MessageBox.Show("Unexpected Error has occured: " + ex.Message);
117. }
118. }
119. }

**Stock Assistant Approving if the Order placed matched with the Order arrived**

1. **private** **void** button1\_Click(**object** sender, EventArgs e) //approve button
2. {
3. 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
4. //so (approved) function can be exceuted from a different form(class), and approved function can remain private, meaning low coupling between two forms
5. }
7. **private** **void** approved() //if approved
8. {
9. 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
10. mediator.itemduplication\_validation(textBox1.Text, textBox1.Text); //checks if the data entered is already inside the StockTable
11. 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
12. deleteorder();
13. dataGridView1.DataSource = mediator.refresh\_OrderTable(); //refreshes the Stocktable and puts it into the Data Grid view
14. dataGridView1.Update(); //Data grid view gets updated with the updated data
15. }
17. **public** **void** deleteorder() //deletes the item in Order Table which  is being added to the StockTable
18. {
19. DataGridViewRow row = dataGridView1.CurrentCell.OwningRow;//grab a reference to the current row //validation inside the process
20. mediator.delete\_OrderTable(row); //calls the delete process from delete mediator to delete the row of the Order TableTable
21. 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
22. dataGridView1.Update();//updates the data grid with new updated data from the Stock Table
23. }

* **Stock Control Assistant deletes an arrived order:**

1. **private** **void** button2\_Click(**object** sender, EventArgs e) //delete button
2. {
4. AssitantUI.approval f2 = **new** AssitantUI.approval(**this**.deletebygrid); //delegating to authenticate usercredtianls
5. //if authentication is complete, run deletebygrid method
6. f2.Show();
8. }
10. **private** **void** deletebygrid() //mediator to delete the values of Order Table
11. {
12. DataGridViewRow row = dataGridView1.CurrentCell.OwningRow;//grab a reference to the current row
13. mediator.delete\_OrderTable(row); //calls the delete process from delete mediator to delete the row of the Order TableTable
14. 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. }

* **Authentication: To approve or delete stock control assistant must sign in with their credentials**

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;
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. }
26. **private** **void** button3\_Click(**object** sender, EventArgs e) //exit button
27. {
28. System.Environment.Exit(0);
29. }
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. }
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. }

47. **void** button1\_Click(**object** sender, EventArgs e) //approve button
48. {
49. mediator.textbox\_Validation\_LoginForm(textUsername.Text, textPassword.Text); //Validation process to check if username and password is entered
50. mediator.loginandapprove\_validation(textUsername.Text, textPassword.Text); //checks 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 authentication to proceed
52. **this**.Close(); //form is closed when all proceeses are finished

55. }

58. **private** **void** approval\_Load(**object** sender, EventArgs e)
59. {
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;
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;

21. **public** Mediator()
22. {
23. //setting mediator processes
24. deleteitemprocess = **new** deletemediator();
25. additemprocess = **new** AddItem();
26. updateprocess = **new** updatemediator();
27. validationprocess = **new** Validation();
28. searchprocess = **new** Searchitem();
29. adjustmentprocess = **new** RefreshStock();

32. }
34. **public** **void** delete\_StockTable(DataGridViewRow row) // setting delete mediator  #Number of process: 1
35. {
36. //delete processes executed by the mediator
37. deleteitemprocess.ExecuteStockTable(row); //delete process for StockTable
38. }
40. **public** **void** delete\_OrderTable(DataGridView row) //delete from order Table
41. {
42. deleteitemprocess.Delete\_OrderTable(row); //process to delete from Order Table #Number of process: 2
43. }
45. public void delete\_Pendingorder(DataGridView row) //delete from Pending Order
46. {
47. deleteitemprocess.ExecutePendingTable(row); //delete process for Pending Table #Number of process: 3
48. }
50. **public** **void** Add\_Stocktable(**string** itemcode, **string** itemname, **string** itemprice, **string** itemquantity, **string** stockarrivaldate, **string** minimumrequired, **string** maximumrequired, **string** staffcheck) //Add into StockTable with values
51. {
52. //add processes executed by the mediator
53. additemprocess.ExecuteAddStocktable(itemcode, itemname, itemprice, itemquantity, stockarrivaldate, minimumrequired, maximumrequired, staffcheck); //add item into Stock Table #Number of process: 4
54. }
56. **public** **void** Add\_OrderTable(**string** itemcode, **string** itemname, **string** itemquantity, **string** stockarrivaldate, **string** maximumrequired, **string** orderstatus) //add into Order Table with values
57. {
58. additemprocess.ExecuteAddOrderTable(itemcode, itemname, itemquantity, stockarrivaldate, maximumrequired, orderstatus); //add item into OrderTable #Number of process:  5
59. }
61. **public** **void** update\_StockTable(**string** itemcode, **string** itemname, **string** itemprice, **string** itemquantity, **string** stockarrivaldate, **string** itemstatus)  //update StockTable with values
62. {
63. updateprocess.UpdateStockTable(itemcode, itemname, itemprice, itemquantity, stockarrivaldate, itemstatus); //update process to update Stock Table #Number of process: 6
64. }
66. **public** **void** search\_mediator(**string** itemcode, **string** itemname) //searches by name or code
67. {
68. searchprocess.searchbynameorcode(itemcode, itemname); //search by name or code #Number of process: 7
69. }
71. **public** **void** itemduplication\_validation(**string** itemcode, **string** itemname) //checks if item name or code is already in the table  #Number of process: 8
72. {
73. validationprocess.itemduplicationvalidation(itemcode, itemname); //setting up duplicatiomn validatiom in the database when item is added  //duplication validation process executed by the mediator
74. }
76. **public** **void** textbox\_Validation\_LoginForm(**string** textUsername, **string** textPassword) //checks username or password is entered #Number of process: 12
77. {
78. validationprocess.textbox\_Validation\_LoginForm(textUsername, textPassword); //setting up value validation to check something is entered in username or password text box
79. }
81. **public** **void** mmanager\_login\_validation(**string** textUsername, **string** textPassword) //checks username or password is entered for manager #Number of process: 13
82. {
83. validationprocess.mmanager\_login\_validation(textUsername, textPassword); //setting up value validation to check something is entered in username or password text box
84. }

87. **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
88. {
89. validationprocess.textbox\_validation\_StockTable(textBox1, textBox2, textBox3, textBox4, DateTimepicker1, textBox6, textBox7, textBox8);  // #Number of process: 9
90. //text validation process executed by the mediator //setting text box validation
91. }
93. **public** **void** textbox\_validation\_StockTable2(**string** textBox1, **string** textBox2, **string** textBox3, **string** DateTimepicker1, **string** textBox5)//checks if every boxes are not empty
94. {
95. validationprocess.textbox\_validation\_StockTable2(textBox1, textBox2, textBox3, DateTimepicker1, textBox5); //text validation process executed by the mediator //setting text box validation in Pending Orders Form
96. }
98. **public** **void** loginandapprove\_validation(**string** textUsername, **string** textPassword) //validates if entered username and password is correct
99. {
100. validationprocess.loginandapprovevalidation(textUsername, textPassword); //#Number of process: 10
101. //validation process for Login and Approve //this process can be used for Logging in and authencating credtianls again when approving orders
102. }
103. **public** DataTable refresh\_StockTable()//get updated data from Stock Table
104. {
105. **return** adjustmentprocess.refresh\_StockTable(); //#Number of process: 11
106. }
108. **public** DataTable refresh\_OrderTable() //get updated data from Order table
109. {
110. adjustmentprocess.refresh\_OrderTable(); //#Number of process:12
111. }

114. }
115. }

* **Validation process which contains all the validation processes, which the main Mediator class calls: Validation.cs**

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;
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 value, 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 itemduplicationvalidation process
18. //{
19. //\_getdatato = getdatato; //linking the pivate and public methods together
21. //}

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";
27. **public** **void** textbox\_validation\_StockTable(**string** textBox1, **string** textBox2, **string** textBox3, **string** textBox4, **string** 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, MessageBoxIcon.Error);
32. }
33. }
35. **public** **void** textbox\_validation\_StockTable2(**string** textBox1, **string** textBox2, **string** textBox3, **string** DateTimepicker1, **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, MessageBoxIcon.Error);
40. }
41. }
43. **public** **void** textbox\_Validation\_LoginForm(**string** textUsername, **string** textPassword) //text box validation for Login 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, MessageBoxIcon.Error);
48. }
49. }
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";
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();
66. **if** (userCount > 0)
67. {
68. MessageBox.Show("Item already exists in the table", "Value Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
69. }
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. }
77. }
78. **catch** (Exception ex)
79. {
80. MessageBox.Show("Unexpected Error has occured:" + ex.Message);
82. }
83. }
85. }
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 credtianls again when approving orders
89. {
90. **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"))
91. {
92. **if** (textUsername == String.Empty)
93. {
94. MessageBox.Show("Please enter user name", "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
95. //textUsername.Focus();
96. }
98. **else** **if** (textPassword == String.Empty)
99. {
100. MessageBox.Show("Please enter password", "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
101. //textPassword.Focus();
102. }
104. **else**
105. {
106. **try**
107. {
108. Connection.Open();
109. SqlCommand cmd = **new** SqlCommand(@"SELECT Count(\*) FROM StaffTable WHERE username=@uname and password=@pass", Connection);
110. cmd.Parameters.AddWithValue("@uname", textUsername);
111. cmd.Parameters.AddWithValue("@pass", textPassword);
112. **int** result = (**int**)cmd.ExecuteScalar();
114. **if** (result > 0)
115. {
116. //\_approver(); //now this updates the existing Pending Order form instances
117. //this.Close();
118. Connection.Close(); //closes the connection
119. }
120. **else**
121. {
122. MessageBox.Show("Incorrect crediantials, please try again");
123. }
124. }
126. **catch** (Exception ex)
127. {
128. MessageBox.Show("Unexpected error:" + ex.Message);
129. }
130. }
131. }
132. }
134. **public** **void** mmanager\_login\_validation(**string** textUsername, **string** textPassword) //validation process for Login and Approve
135. //this process can be used for Logging in and authencating credtianls again when approving orders
136. {
137. **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"))
138. {
139. **if** (textUsername == String.Empty)
140. {
141. MessageBox.Show("Please enter user name", "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
142. //textUsername.Focus();
143. }
145. **else** **if** (textPassword == String.Empty)
146. {
147. MessageBox.Show("Please enter password", "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
148. //textPassword.Focus();
149. }
151. **else**
152. {
153. **try**
154. {
155. Connection.Open();
156. SqlCommand cmd = **new** SqlCommand(@"SELECT Count(\*) FROM ManagerTable WHERE username=@uname and password=@pass", Connection);
157. cmd.Parameters.AddWithValue("@uname", textUsername);
158. cmd.Parameters.AddWithValue("@pass", textPassword);
159. **int** result = (**int**)cmd.ExecuteScalar();
161. **if** (result > 0)
162. {
163. //\_approver(); //now this updates the existing Pending Order form instances
164. //this.Close();
165. Connection.Close(); //closes the connection
166. }
167. **else**
168. {
169. MessageBox.Show("Incorrect crediantials, please try again");
170. }
171. }
173. **catch** (Exception ex)
174. {
175. MessageBox.Show("Unexpected error:" + ex.Message);
176. }
177. }
178. }
179. }
181. }
182. }

* **Refresh Stock class 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;
10. **namespace** \_SCM\_System
11. {
12. **class** RefreshStock
13. {
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";
17. **public** DataTable refresh\_StockTable()
18. {
19. DataTable data = **new** System.Data.DataTable();
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. }
28. **catch** (System.Data.SqlClient.SqlException ex)
29. {
30. MessageBox.Show(ex.Message); //show a useful message to the user of the program
31. }
33. **return** data;
34. }

37. **public** DataTable refresh\_OrderTable()
38. {
39. DataTable data = **new** System.Data.DataTable();
41. **try**
42. {
43. SqlDataAdapter dataAdapter = **new** System.Data.SqlClient.SqlDataAdapter("SELECT \* FROM OrderTable", Connection); //pass in the select command and the conncetion string
44. data.Locale = System.Globalization.CultureInfo.InvariantCulture;
45. dataAdapter.Fill(data); //fill the table
46. }
48. **catch** (System.Data.SqlClient.SqlException ex)
49. {
50. MessageBox.Show(ex.Message); //show a useful message to the user of the program
51. }
53. **return** data;
54. }
55. }
56. }

* **Add Item class which contains all the add item processes, which the main Mediator class calls: AddItem.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;
9. **namespace** \_SCM\_System
10. {
11. **class** AddItem
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";
16. **public** **void** ExecuteAddStocktable(**string** itemcode, **string** itemname, **string** itemprice, **string** itemquantity, **string** stockarrivaldate, **string** minimumrequired, **string** maximumrequired, **string** staffcheck) //for Stock form
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)";
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);
38. //Append values into StockTable
39. cmd.ExecuteNonQuery();
41. **int** i = cmd.ExecuteNonQuery();
43. **if** (i == 1)
44. {
45. MessageBox.Show("Item has been registered");
47. }
48. **else**
49. {
50. MessageBox.Show("Item couldn't be added, Please Try again");
51. }
52. }
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.Message);
58. }
60. }
61. }
63. **public** **void** ExecuteAddOrderTable(**string** itemcode, **string** itemname, **string** itemquantity, **string** stockarrivaldate, **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], [maximumrequired], [orderstatus])
67. VALUES ((@itemcode, @itemname, @itemquantity, @stockarrivaldate, @maximumrequired, @orderstatus)";
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);
83. //Append values into StockTable
84. cmd.ExecuteNonQuery();
86. **int** i = cmd.ExecuteNonQuery();
88. **if** (i == 1)
89. {
90. MessageBox.Show("Item has been registered");
92. }
93. **else**
94. {
95. MessageBox.Show("Item couldn't be added, Please Try again");
96. }
98. }
100. **catch** (Exception ex)
101. {
102. //gives error if there is a connection issue
103. MessageBox.Show("Unexpected Error has occured, Items couldn't be added into OrderTable:" + ex.Message);
104. }
106. }
107. }
109. }
110. }

* **Delete item class which contains all the Delete item processes, which the main Mediator class calls: DeleteItem.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;
9. **namespace** \_SCM\_System
10. {
11. **class** deletemediator
12. {
13. **public** **void** ExecuteStockTable(DataGridViewRow row)
14. {
15. SqlConnection con;
16. **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";
18. //this will grab the value from the item name of the selected record
19. **string** code = row.Cells["itemcode"].Value.ToString();
21. //This will grab the value from the item name field of the selected record
22. **string** name = row.Cells["itemname"].Value.ToString();
24. // This will grab the value from the item quantity field of the selected record
25. **string** quantity = row.Cells["itemquantity"].Value.ToString();
27. // This will grab the value from the stock arrival date field of the selected record
28. **string** arrival = row.Cells["stockarrivaldate"].Value.ToString();

31. // This will grab the value from the minimum required date field of the selected record
32. **string** Minimum = row.Cells["minimumrequired"].Value.ToString();
34. // This will grab the value from the maximum required field of the selected record
35. **string** Maximum = row.Cells["maximumrequired"].Value.ToString();
37. //string Staff = row.Cells["staffcheck"].Value.ToString();
38. **string** status = row.Cells["itemtstatus"].Value.ToString();
40. //Messahe box will pop up for user confirmation to delete
41. 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 the records from the sql table
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 connection
52. cmd.ExecuteNonQuery();//taking the command and connection, delete the selected records

55. }
56. **catch** (Exception ex)
57. {
58. MessageBox.Show(ex.Message);//runs if the code above fails(connection, other errors)
59. }
60. }
61. }
62. }
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";
69. //this will grab the value from the item name of the selected record
70. **string** code = row.Cells["itemcode"].Value.ToString();
72. //This will grab the value from the item name field of the selected record
73. **string** name = row.Cells["itemname"].Value.ToString();
75. // This will grab the value from the item quantity field of the selected record
76. **string** quantity = row.Cells["itemquantity"].Value.ToString();
78. // This will grab the value from the stock arrival date field of the selected record
79. **string** arrival = row.Cells["stockarrivaldate"].Value.ToString();
81. // This will grab the value from the maximum required field of the selected record
82. **string** Maximum = row.Cells["maximumrequired"].Value.ToString();
84. //string Staff = row.Cells["staffcheck"].Value.ToString();
85. **string** status = row.Cells["orderstatus"].Value.ToString();
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);
90. **string** deleteState = @"Delete from OrderTable where itemname = '" + name + "'";//this is the sql to delete the records from the sql table
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 connection
99. cmd.ExecuteNonQuery();//taking the command and connection, delete the selected records

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

* **Update Item class which contains all the Update item processes, which the main Mediator class calls: Update.cs**

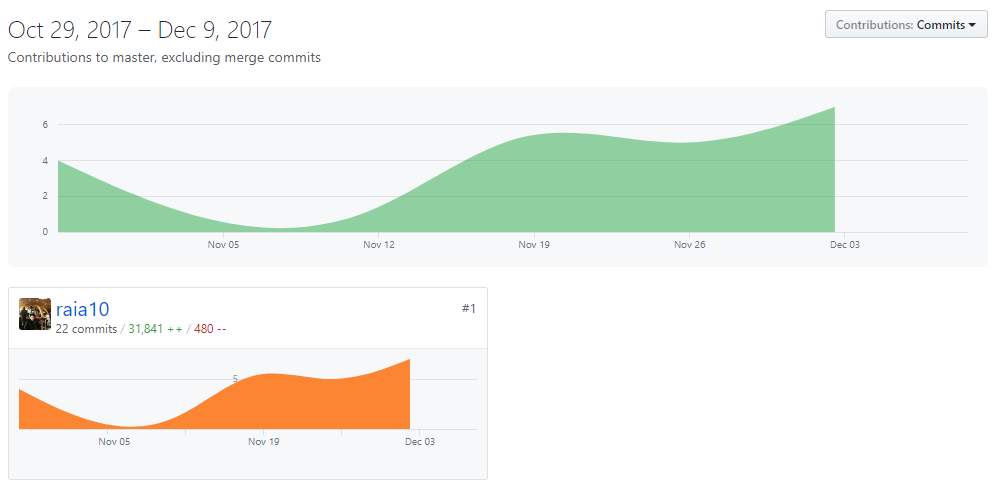
1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
6. **using** System.Data.SqlClient;
7. **using** System.Windows.Forms;
9. **namespace** \_SCM\_System
10. {
11. **class** updatemediator
12. {
13. SqlConnection 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";


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 private, 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** stockarrivaldate, **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";
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);
39. //Append values into StockTable
40. cmd.ExecuteNonQuery();
41. con.Close();
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. }
50. }
51. }
52. }
53. }

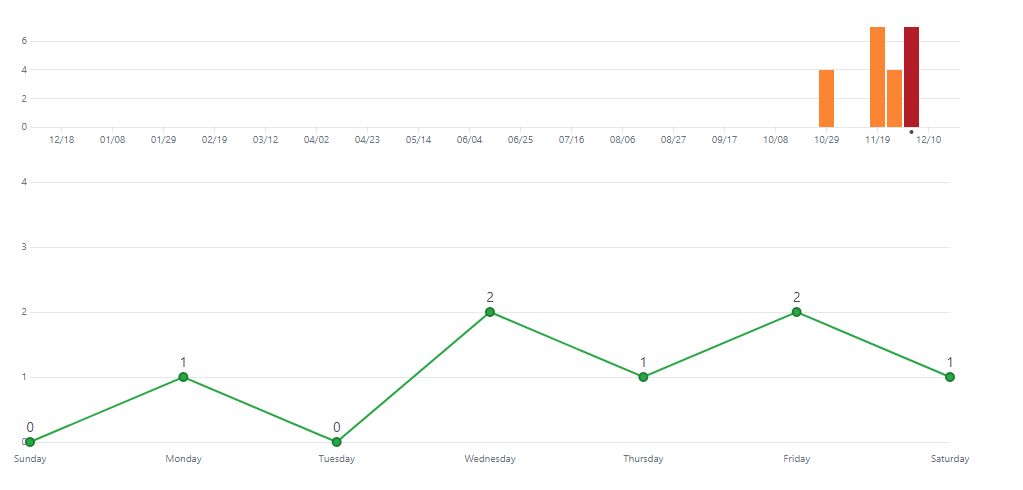
* **Notification class which contains all the notification processes, which the main Mediator class calls: Notification.cs**

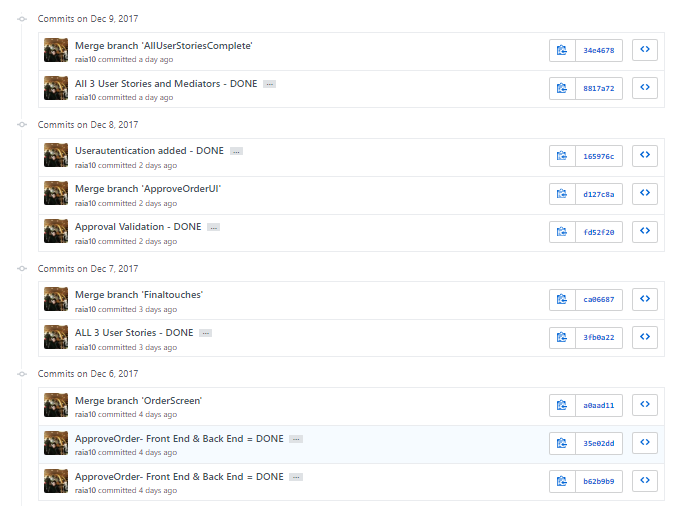
1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
6. **using** System.Windows.Forms;
7. **using** System.Data.SqlClient;
9. **namespace** \_SCM\_System
10. {
11. **class** Notification
12. {
13. SqlConnection 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";
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)";
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);
38. //Append values into StockTable
39. cmd.ExecuteNonQuery();
40. }
42. **catch** (Exception ex)
43. {
44. //gives error if there is a connection issue
45. MessageBox.Show("Unexpected Error has occured, Items couldn't be added into StockTable:" + ex.Message);
46. }
47. }
48. }
49. }
50. }

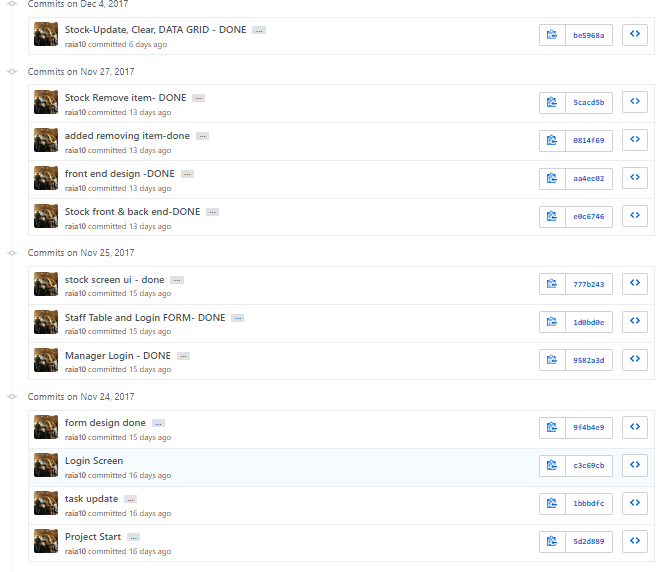
Evidenced of version control tool (GITHUB)

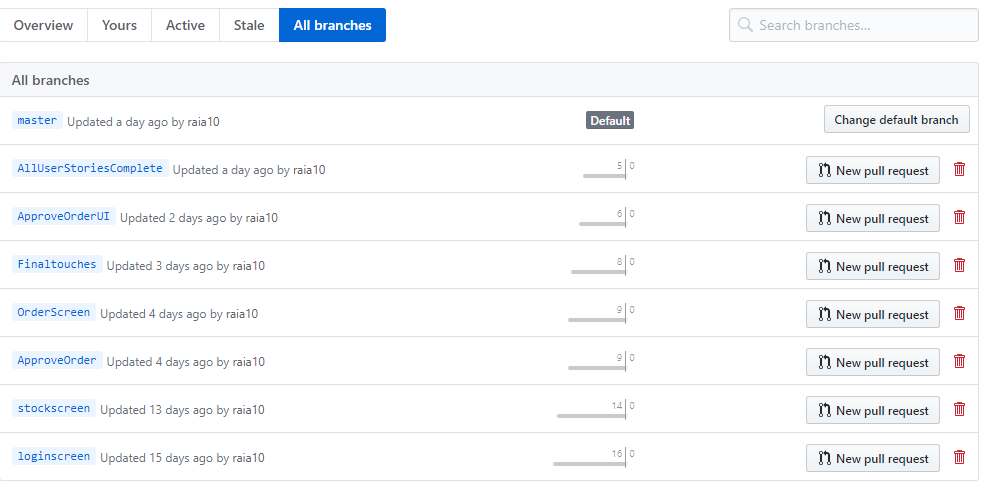


* **Regular commits showing the changes in code, Additions and deletions**







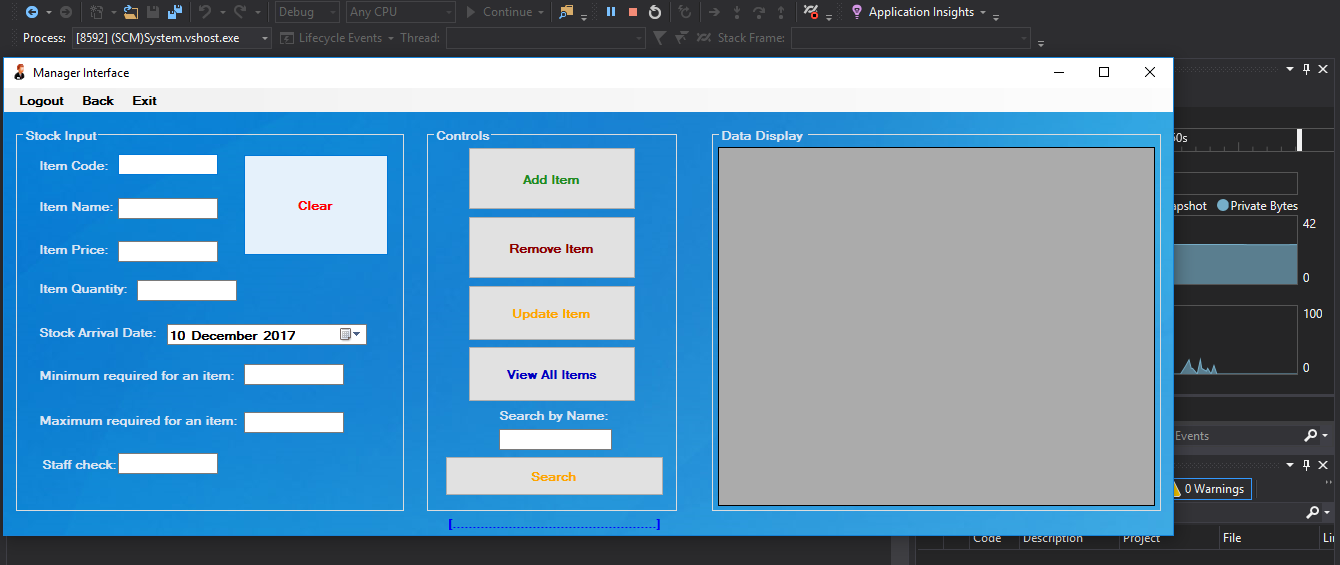


* **Different branches used to control different versions of the SCM system**

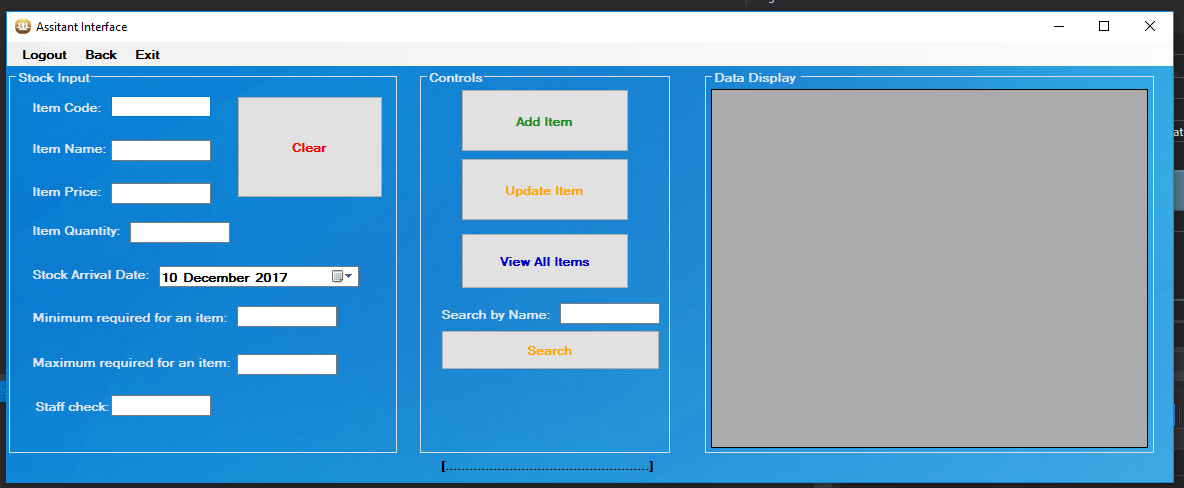
**Task 3**

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

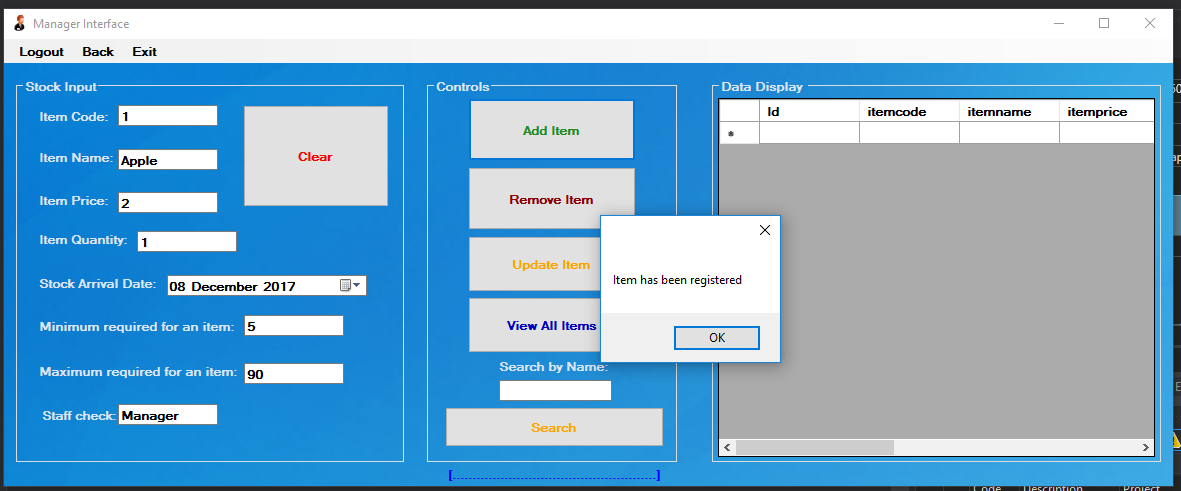
**User Story 1 & 3: Adding, removing (Only Manager), Updating and Searching stock items for Manager and Stock Assistant.**



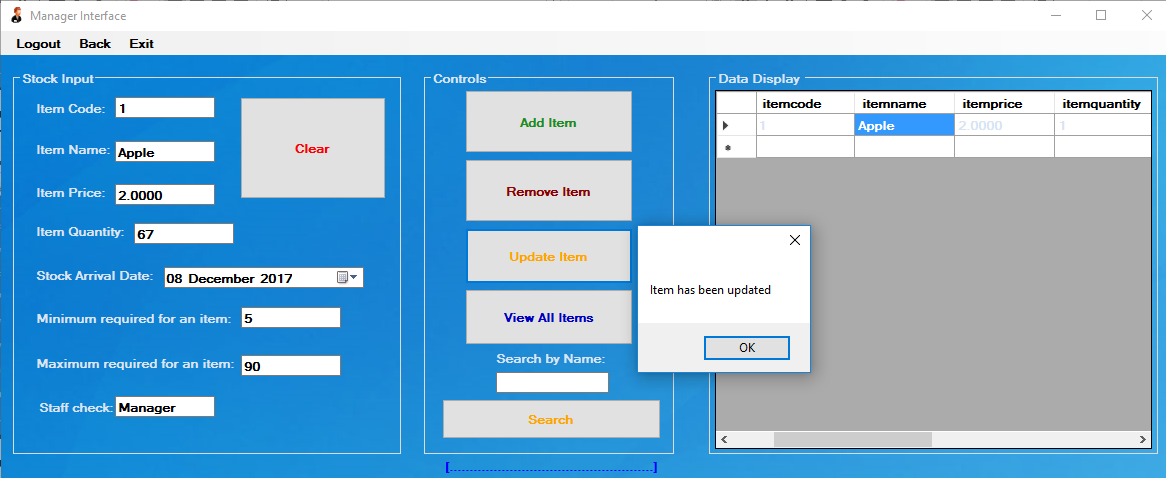
* **Manager can Add, Remove, Update and Search for items within the database. Only Manager can delete items in the database.**



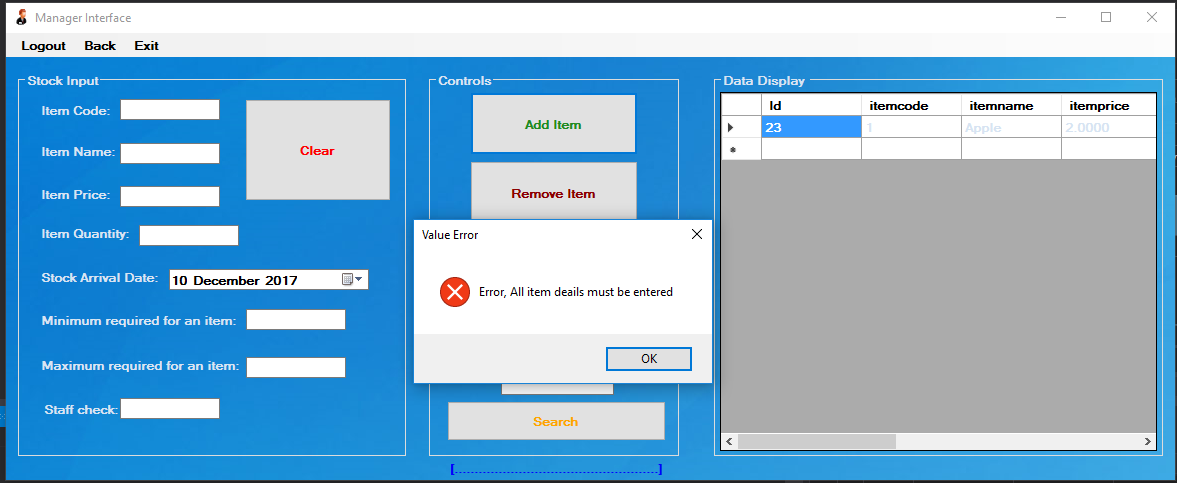
* **Stock Assistant Can Add, Update and Search for items within the database. Stock Assistant cannot delete an item from the database.**



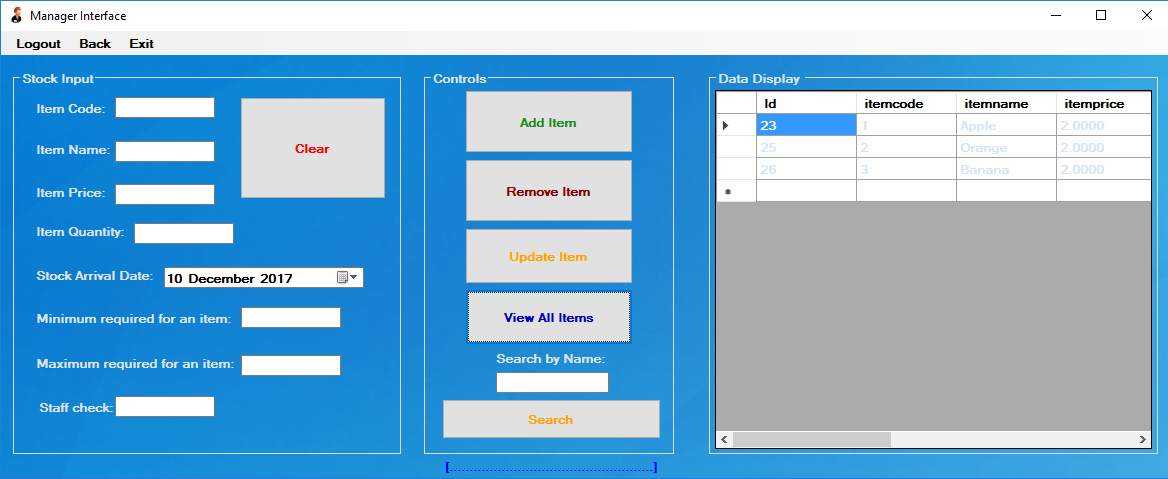
* **Both users can add new item into the database of their choice. Message box will pop up on the screen to confirm the item has been added to notify the user.**



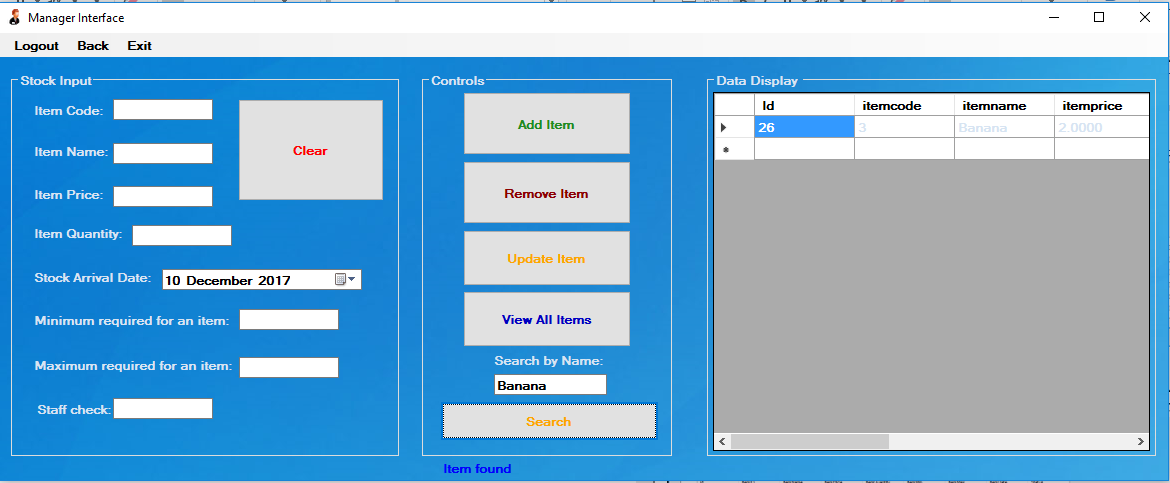
* **Both users can update values of existing items in the database. In the screenshot the item quantity of the item name ‘Apple’ has been changed from 1 to 67. A massage box has appeared to let the user know that the change was executed and was successful.**



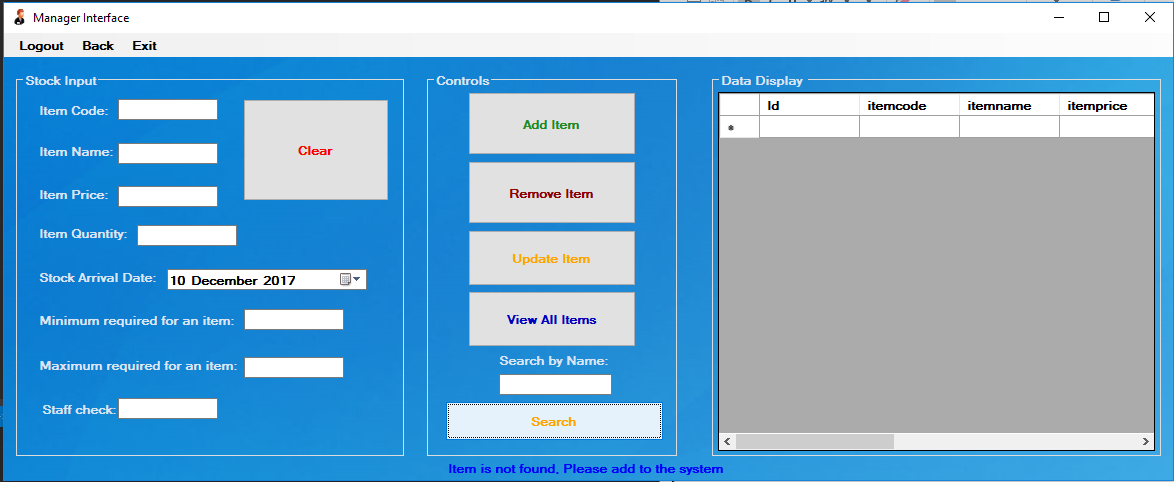
* **Validation process, which makes sure that all the fields are filled when the user tries to add a new item.**



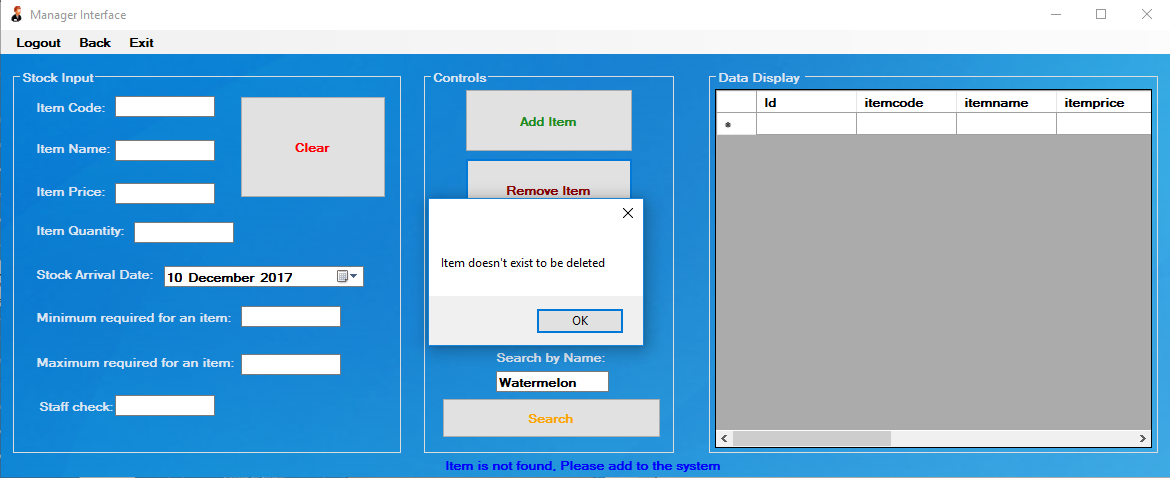
* **This feature is when the database is loaded with massive number of items. The user types an ‘item name’ or ‘item code’. For example, in this context ‘Banana is searched’. Below is the result.**



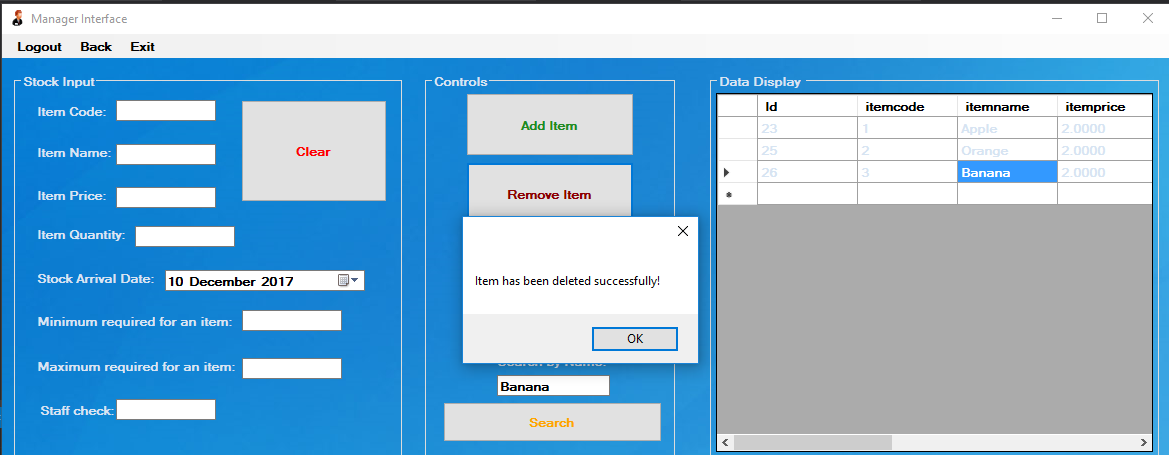
* **As expected, only banana is found in the database and displayed on the Data Grid View. The label underneath is also changed to ‘Item found’, because the item was found in the database to notify the user.**



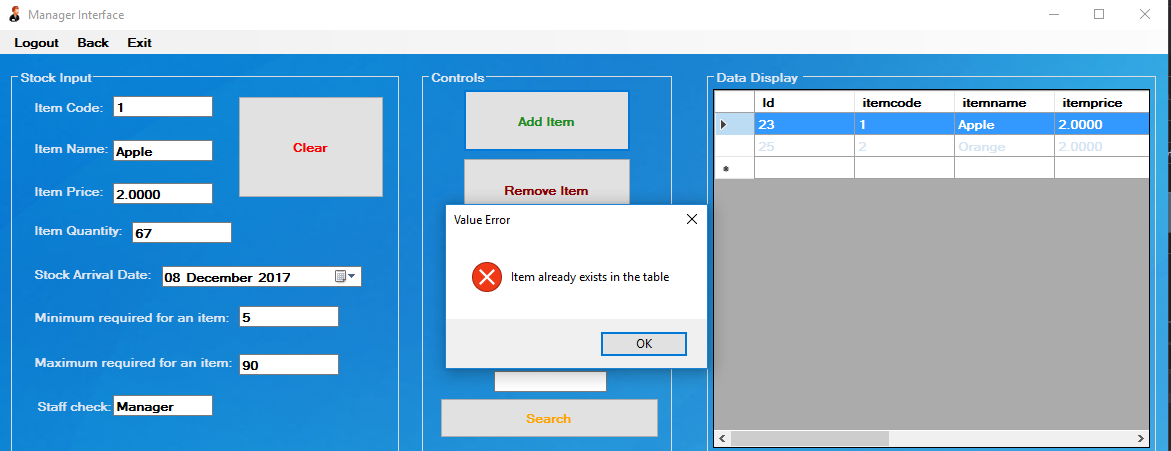
* **If the item is not found it would say ‘Item not found. Please add to the system’ to notify the user.**



* **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.**

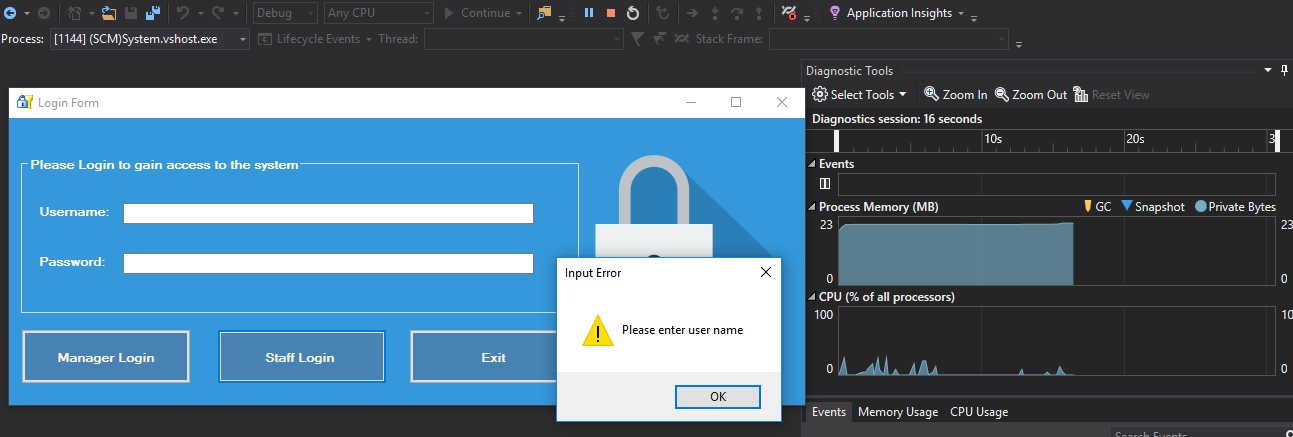


* **If the user enters an item to delete, which exists in the system then the item is deleted. In this context, ‘Banana’ item is deleted as it exists in the system.**

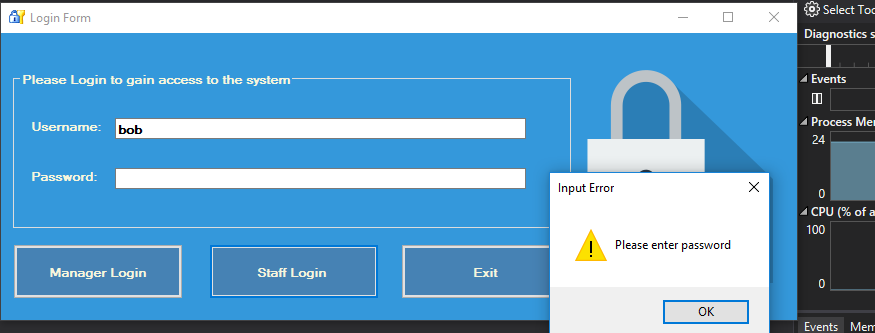


* **This validation processor makes sure that the item with same name and code is not entered twice, to prevent duplication of items. In this context, user is trying to add the same item ‘Apple’ and code ‘1’ into the system which already exists, and the processor will throw an error message to notify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

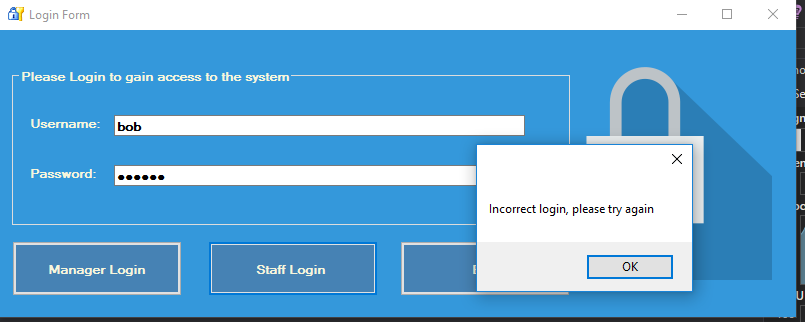
**User Story 4: All staff be able to Login using their credentials and view stock status**



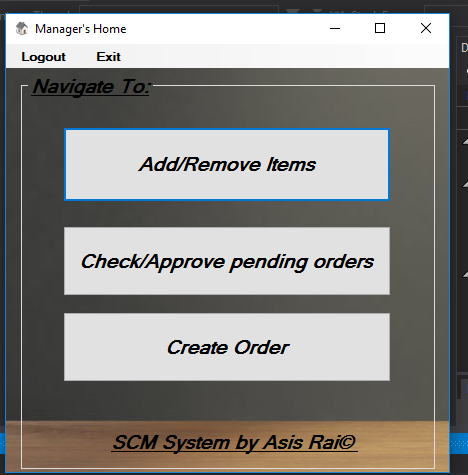
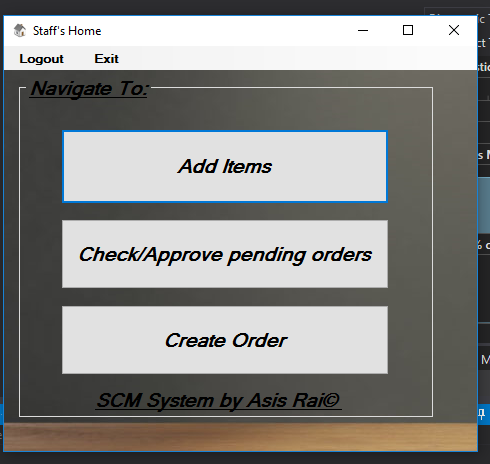
* **This is the main Login form when the system starts, If the manager wants to login then he/she should just enter their credentials and press ‘Manager Login’ and if other staff wants to Login then they should enter their credentials and press ‘Staff Login’. This is a validation processor to make sure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



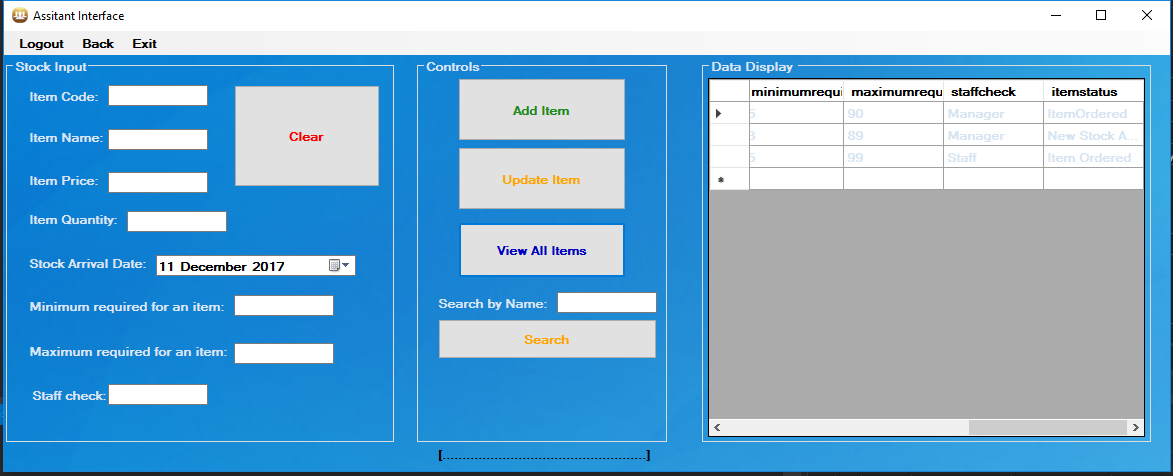
* **Similarly, if only the username is entered without entering the password then the validation processor will throw an error message to notify the user that password also must be entered to Login.**



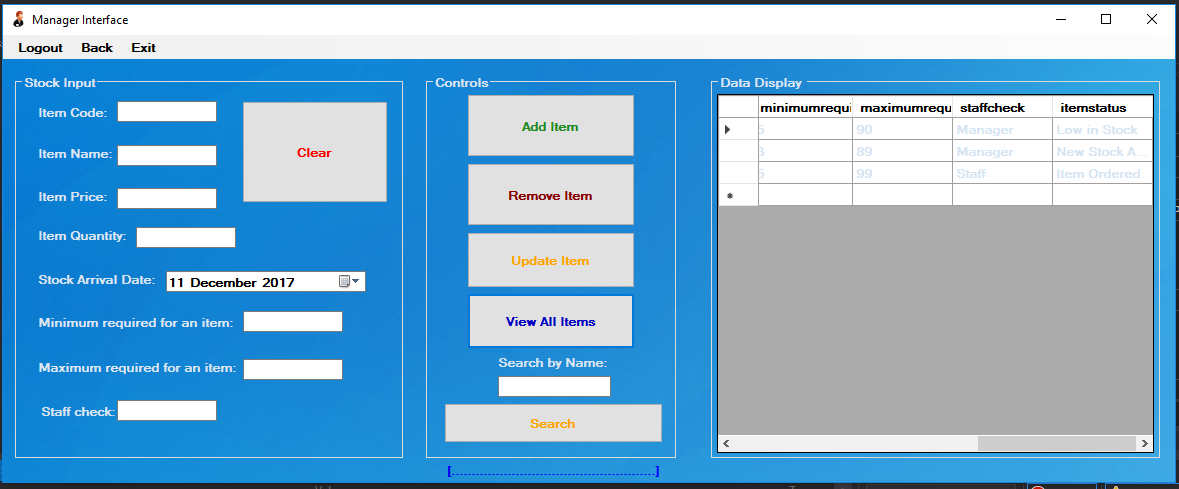
* **If the entered credentials are wrong then the validation processor will make sure that user checks their credentials again and entered the correct credentials, the processor will throw an error message to notify the user.**

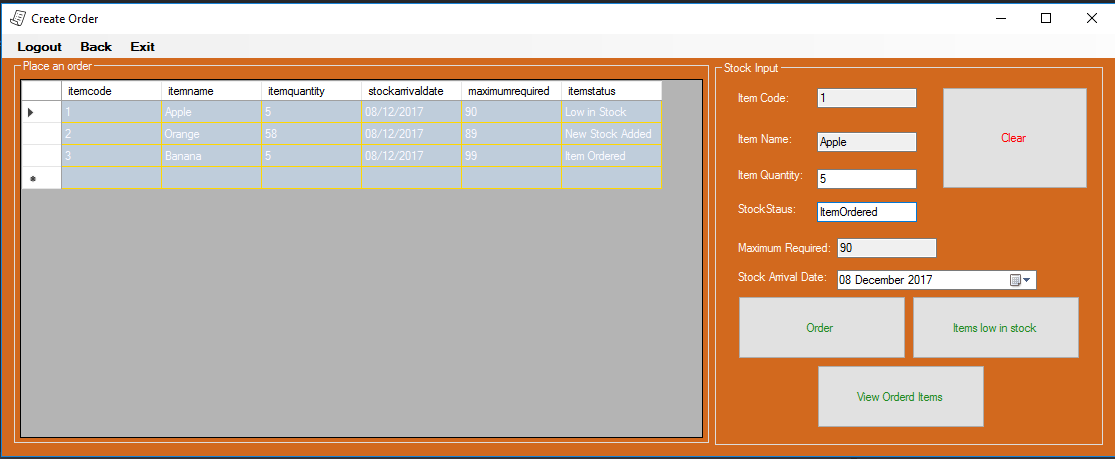
* **These are two different interfaces for different user types. If the manager logs into the system, then he/she will be directed to ‘Managers Home’ interface where he/she can only access the controls that is allowed. Similarly, when Staff Logs into the system then he/she will be directed to ‘Staff’s 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.**



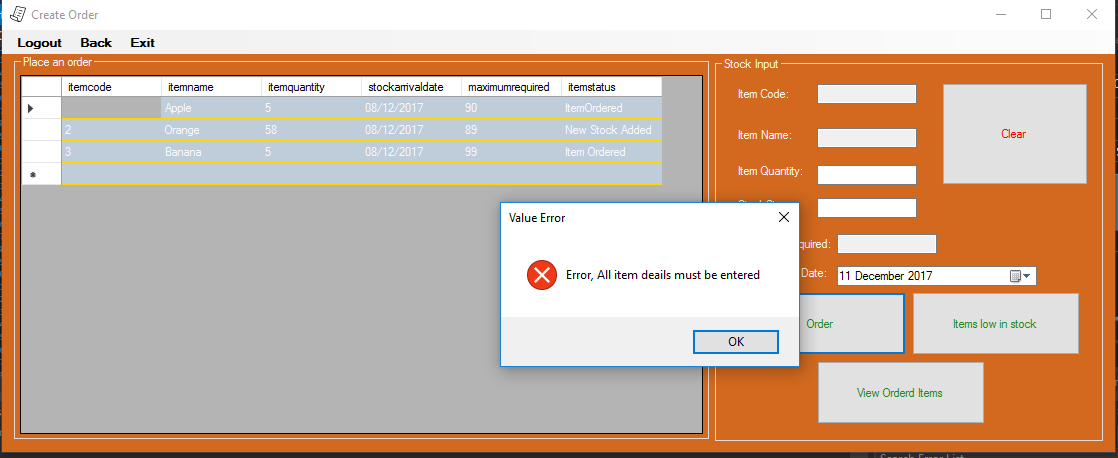
* **This is where other Staffs can view all the items and their status by clicking on ‘View All Items’. All item status is displayed on the Data Grid view.**



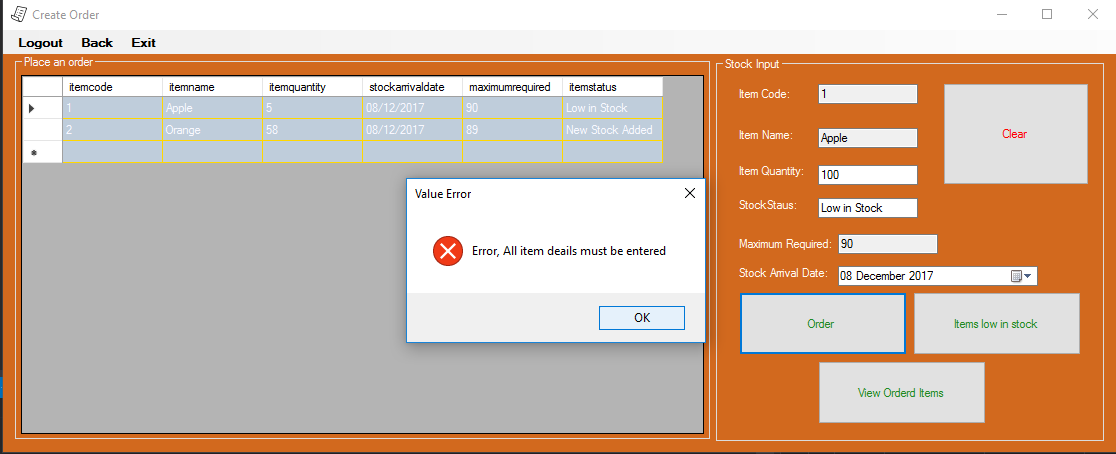
* **This is where Manager can view all the items and their status by clicking on ‘View All Items’. All item status is displayed on the Data Grid view. As sees on the screenshot, the manager has the option to delete any items from the database, however other staff do not.**

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

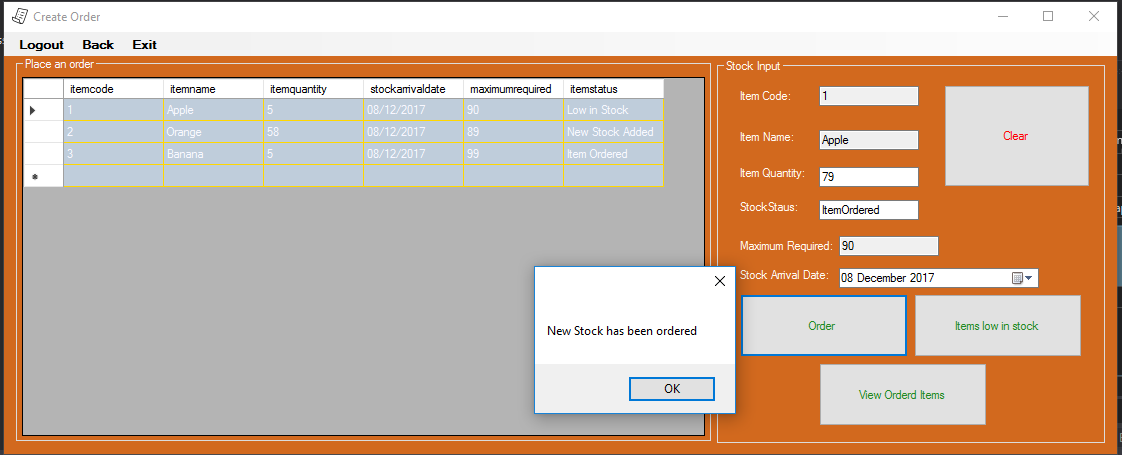
* **This is where the Stock Control Assistant can create new orders for the items that are ‘Low in Stock’.**
* **All the orders that are ‘Low in Stock’ will be displayed once the user presses ‘Items low in stock’, the data will be displayed on the Data Grid view.**



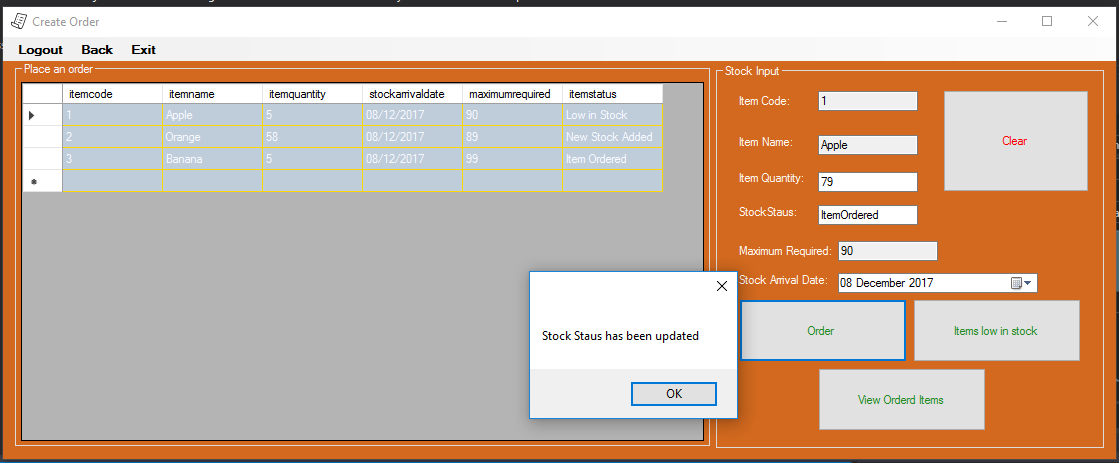
* **Form contains a Validation processor, which makes sure that every item detail is filled to Oder a new stock of an item.**



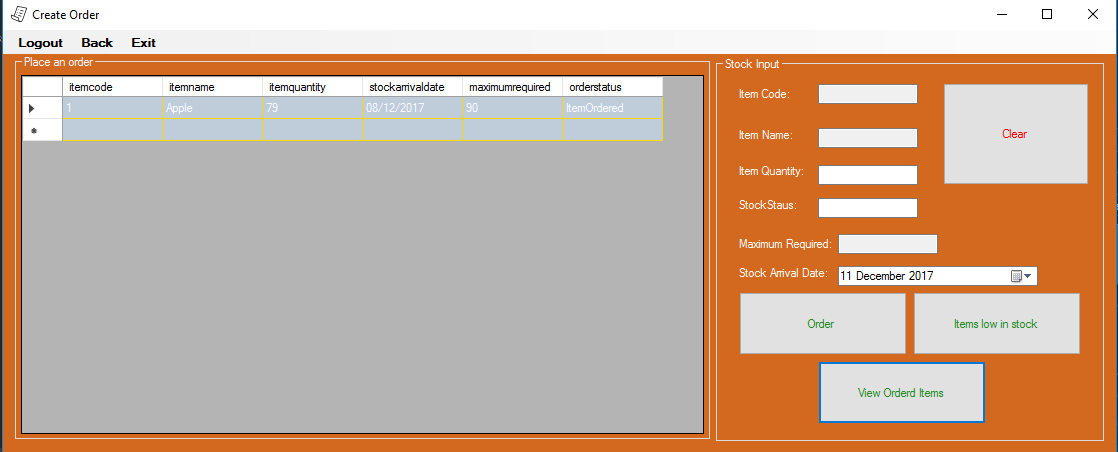
* **This is another Validation processor, where it makes sure that the item quantity the user is ordering, cannot be more than the ‘Maximum required’. This makes sure that the orders are placed rightly.**

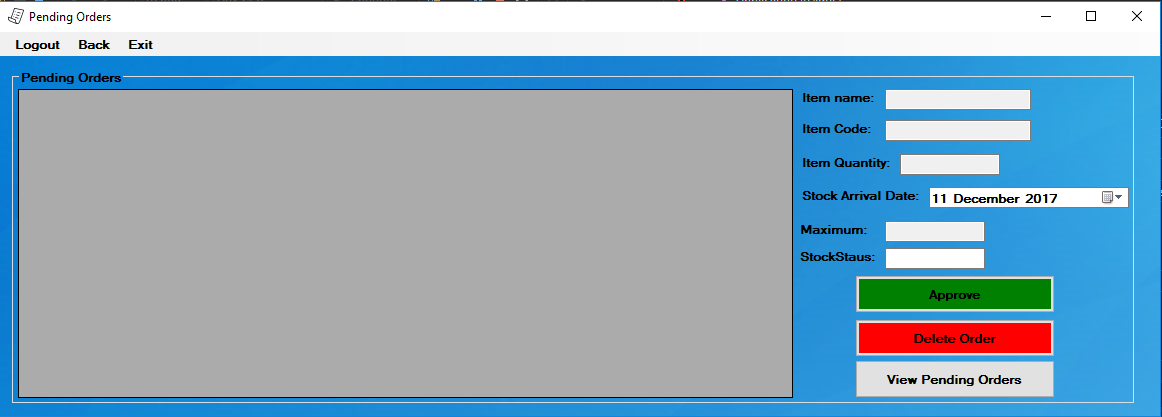


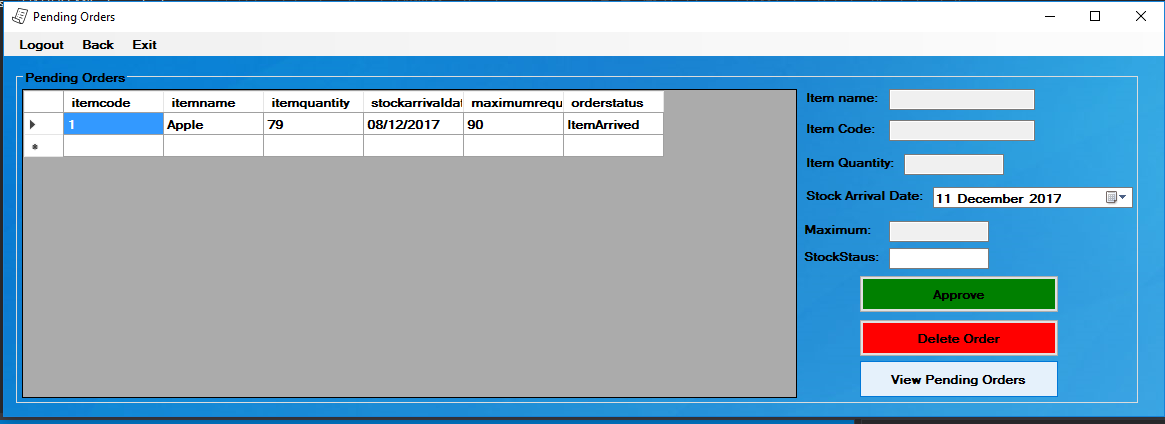
* **After the user creates a new order successfully, the order processor notifies the user of the success.**
* **Notification processor will also alert the user with a message box pop up, to notify the user of the success.**



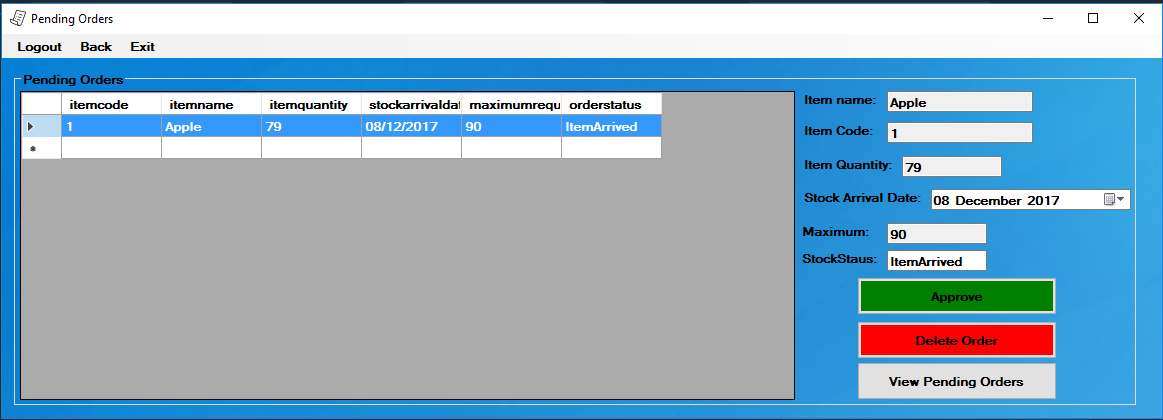
* **The order processor also notifies the user, that the Order Status has been updated in the main Stock table for all users to see.**
* **Notification processor will also alert the user with a message box pop up, to notify the user of the success.**



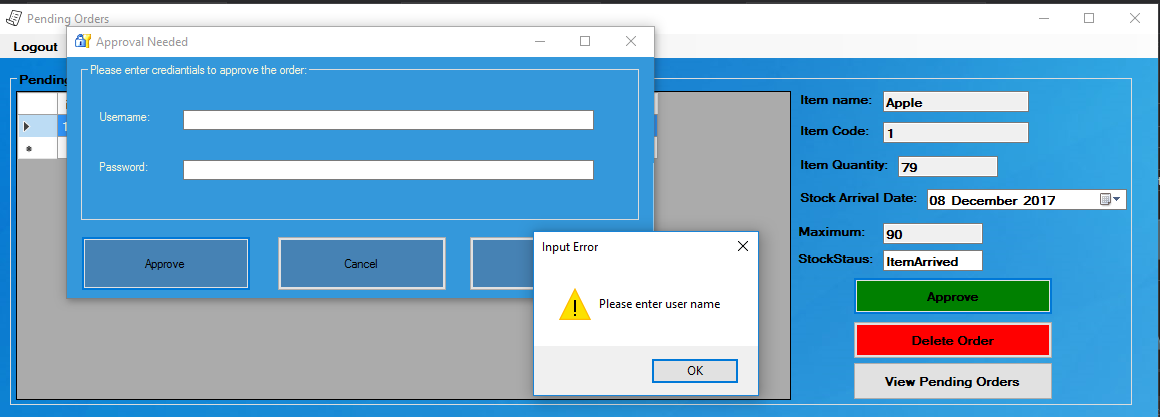
* **The order placed has been added into the Orders Table and with its status changed to ‘Item Ordered’. All users can see this by pressing ‘View Ordered Items’.**
* **The user goes back and selects ‘View/Approve Pending orders’. Pending orders form will be opened where the ‘Stock Control Assistant’ will be able to ‘View Pending Orders’, select it and Approve or delete the order.**



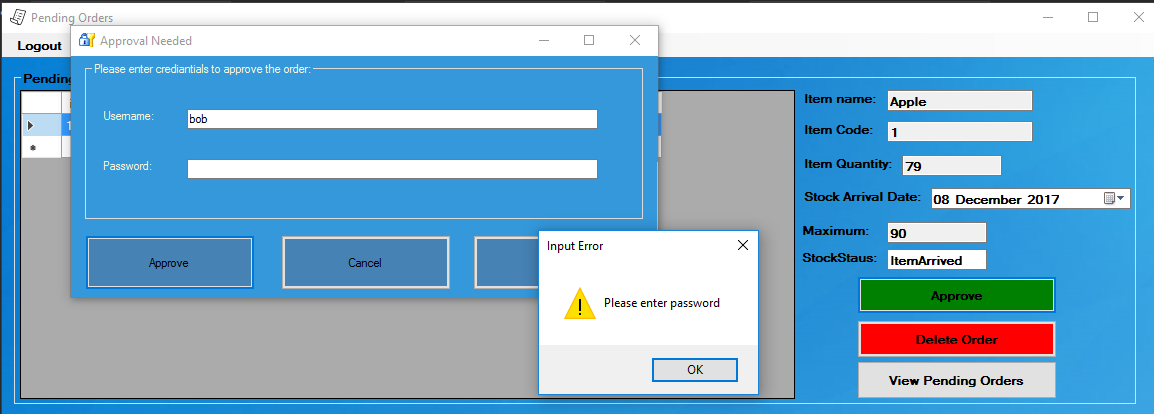
* **When the user clicks ‘View Pending Orders’, the Data Grid will display all orders that have arrived from the suppliers. In this context, the order placed earlier by the user has now arrived and awaiting approval.**



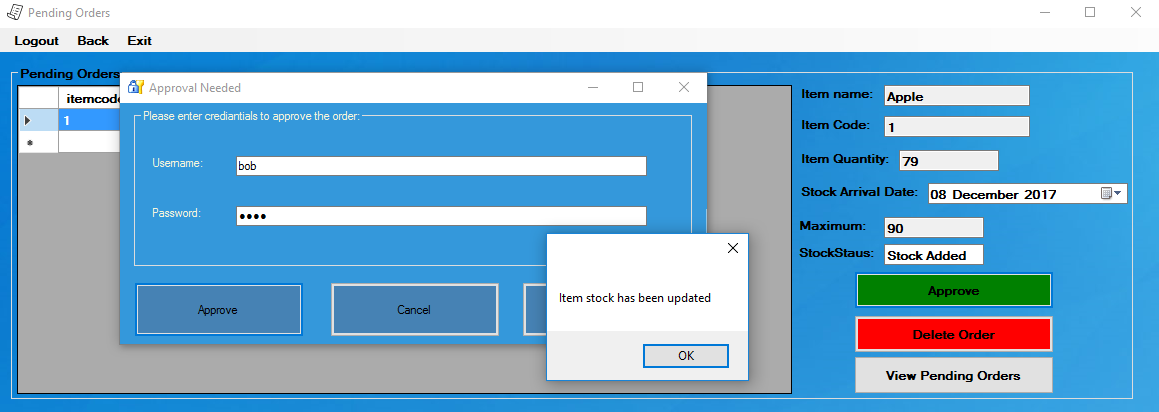
* **The assistant selects the he/she wants which will automatically fill the text boxes in the form and presses ‘Approve’ button, this will activate the ‘Authentication Processor’.**



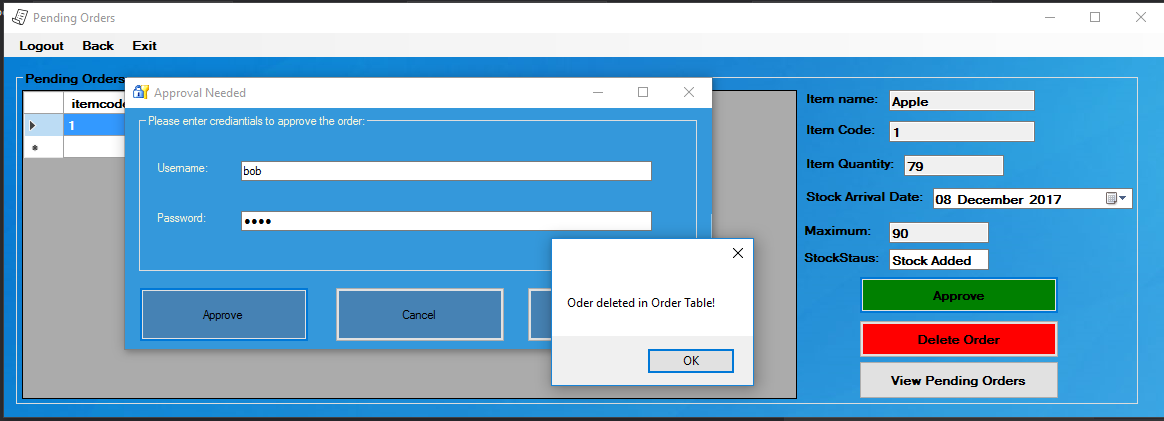
* **The Authentication processor will ask to verify the ‘Stock Control Assistant’s’ credentials again. If the field are left empty, validation processor will pop up a message box to notify the user that both fields must be filled to proceed.**



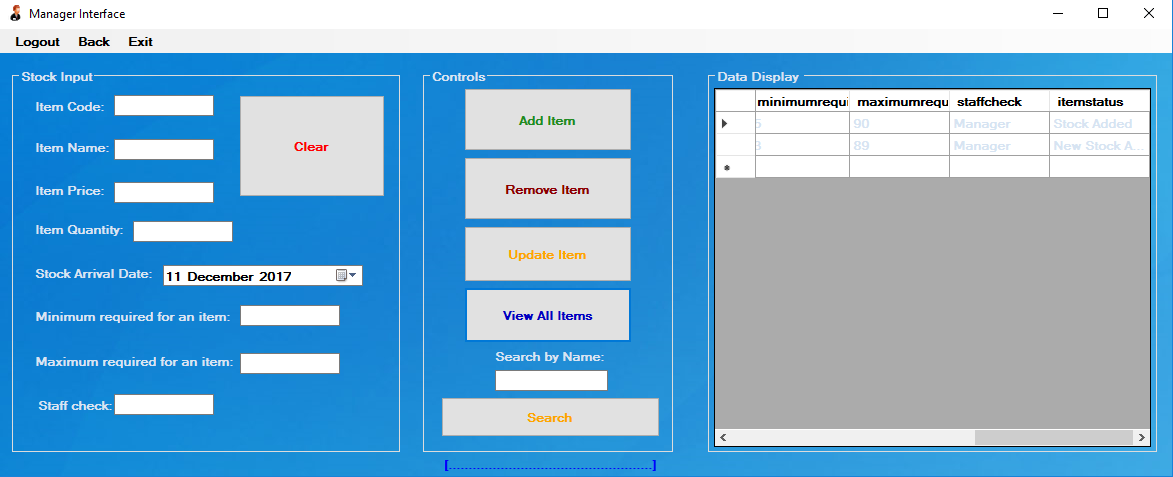
* **The validation processor, will also notify the user that password also must be entered to proceed if only username is entered.**



* **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.**

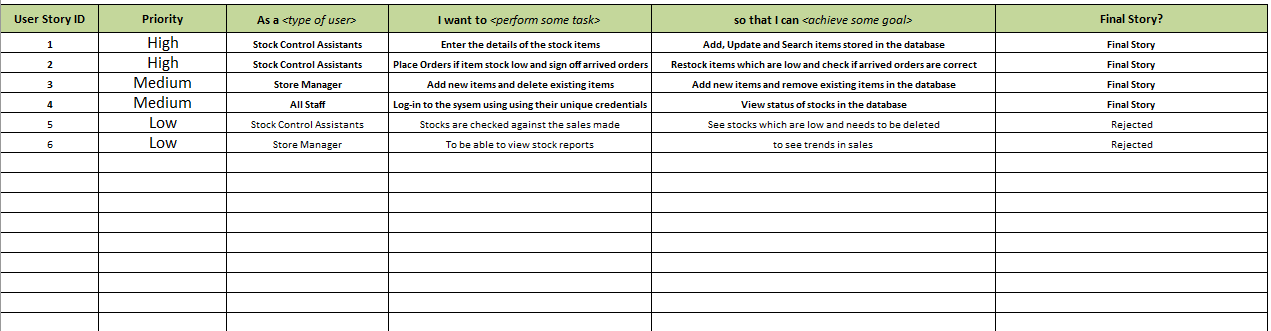


* **All staff are now able to see the most updated stock details in the system, including the new stock that was just added, ‘Stock Added’.**

**Task 4**

Critical Evaluation of user story and tasks breakdown:

BREAKDOWN OF TASKS:



**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 user story and tasks breakdown:

* 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 user story and tasks breakdown:

* User stories depend on the sprints, if the user stories change so does the sprints due to this, the client or the proposer cannot be given an agreement on the final user stories when the project is started, as they are likely to change at some point. This maybe a drawback for the client and his/her project.

Resolutions:

* To avoid these issues, constant communication with the end user is important. The end user could be a member of the team and who will keep monitoring you. This will help many developers who choose this technique to do their projects as the end user would be closely stuck with the project and requirements can be updated quicker. I used this initiative and worked closely with the lecturer to make sure my user stories were correct which helped me reduce time and I could focus on making the system more robust.

Limitations:

* Burndown chart and sprint backlogs is not part of this technique. I personally think it is better than user story and tasks breakdown because it will give a bigger picture of how the end-product would look like and it will also reduce the time for the overall project.

Lessons learnt/Suggestions:

* As predicted, I spent long time creating the user stories and it kept changing as the system was created therefore I think I would suggest, in the future of using sprint backlogs and burndown chart to help better organise tasks therefore the tasks be better organised and finished on time, not affecting or not having the need to keep changing user stories.

**Task 5**

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