



**Populating Vendors, Selecting Vendors, Display vendor information**

|  |  |  |
| --- | --- | --- |
| **Controls** | | **Events** |
| VendorsDDL | | None, Display from ODS |
| **BBL Class(es) and Method (s)** | | |
| VendorController | List<SelectionList> List\_VendorNames()  A list of VendorName and its associated VendorId | |
| **SQL Table(s): (C,R,U,D)** | | **Entities/DTOs/POCOs** |
| Vendor(R) | | DDLSelectionList(P) |

When the fetch button is clicked,

Fetch button event #1**: Order details**: Creating the Current “Active” Order View

|  |  |
| --- | --- |
| **Controls** | **Events** |
| Fetch(VendorsDDL) #1 | Fetch OnClick / wired via ODS   1. Set int fetchIDValue to VendorDDL.SelectedValue. 2. Set int stockID, purchaseOrderID and string username 3. PurchaseOrderDetailsController sysmgr = new PurchaseOrderDetailsController(); 4. Call the controller method, TryRun(()=> ActiveOrderList.DataSource = results; 5. ActiveOrderList.DataBind(); |
| **BBL Class(es) and Method (s)** | |
| PurchaseOrderDetailsController | 1. From the Selected Vendor value, Display information via a POCO   Something like this:  from x in PurchaseOrderDetails  where x.PurchaseOrder.Vendor.VendorID == VendorDDL.SelectedValue  where x.PurchaseOrder.PurchaseOrderNumber==null && x.PurchaseOrder.OrderDate == null  select new {  SID = x.StockItem.StockItemID,  Description = x.StockItem.Description,  QOH = x.StockItem.QuantityOnHand,  QOO = x.StockItem.QuantityOnOrder,  ROL = x.StockItem.ReOrderLevel,  QTO = x.Quantity,  }  1.1 Check if the selected Vendor exist or not,  To determine if exist or not, under the “using (var context = new eToolsContext()){}”  1.2 Check if the VendorID equals to the VendorDDL.SelectedValue  If VendorDDL.SelectedValue does not match, then error message “Nothing matches the data base”  1.3 Afterwards check  If (PurchaseOrder.PurchaseOrderNumber==null && x.PurchaseOrder.OrderDate == null) { Display the current active Order }  Else { return null }   1. Add StockItems to the “Active Order View”   public void Add\_ItemToActiveOrderlist(int orderID, string username, int stockID)  {  //this list of strings will be used with the BusinessRuleException. Every item in the list will be an error to displayed.  List<string> reasons = new List<string>();  //Part one -- add of the new PurchaseOrderDetail item to validate the item is NOT on the existing list.  using (var context = new eToolsContext())  {   * 1. Determine if the PurchaseOrder already exists on the database   2. if (exists == null), then add the parent record (PurchaseOrder)   2.3 else, The Order exists on the database, the PurchaseOrderDetails may or maynot have any StockItems  StockItems can only appear once in the PurchaseOrderDetail view. To see if this is a duplicate StockItem? look upon the StockItems of the PurchaseOrderDetails, testing for the incoming SID, something like this:  newSItem = exists.StockItems.SingleOrDefault(x => x. StockItemID == StockItemID);  2.4 Adjust the order of display to be the next stockitem displayed on this list.  2.5 if (newSItem!= null), then it means it is a duplicate, add to a list of reasons and let the user know.  2.6 if (reasons.Count() > 0), issue BusinessRuleException  3. Add new PurchaseOrderDetails list item  newSItem = new PurchaseOrderDetails();  newSItem.StockItemID = stockID;  3.1 Set the QTO to 1;  3.2 Check or setup the parent: PurchaseOrders if it has been setup with a Hashset, then add the new child (PurchaseOrderDetails) using the Navigational property, something like this:  exists.PurchaseOrderDetails.Add(newSItem);  context.SaveChanges(); |
| **SQL Table(s): (C,R,U,D)** | **Entities/DTOs/POCOs** |
| Vendor(R)  StockItem (R)  PurchaseOrders(R)(C opt)  PurchaseOrdersDetals (R)(C) | ActiveOrderDetailList(P)  PurchaseOrders (E)  PurchaseOrderDetails(E)  StockItem (E)  Vendor(E) |
|  | |

Fetch button event #2: Creating the Vendor StockItems view, **Vendor products not on order**

|  |  |
| --- | --- |
| **Controls** | **Events** |
| Fetch(VendorsDDL) #2  (continue from #1) | Fetch OnClick / wired via ODS   1. Set int fetchIDValue to VendorDDL.SelectedValue. 2. Set int purchaseOrderID, stockID, string username, List< ActiveOrderDetailList > 3. VendorItemList.DataBind(). |
| **BBL Class(es) and Method (s)** | |
| StockItemController | 1. From the Selected Vendor value, Display information via a POCO   Something like this:  List< VendorStockItemList > List\_TotalRowsVendorItemPOCO  from x in StockItems  where x.Vendor.VendorID == fetchIDValue  where (x.QuantityOnHand + x.QuantityOnOrder - x.ReOrderLevel)>=0  select new {  SID = x.StockItemID,  ItemDescription = x.Description,  QOH = x.QuantityOnHand,  QOO = x.QuantityOnOrder,  ROL = x.ReOrderLevel,  Buffer = x.QuantityOnHand + x.QuantityOnOrder - x.ReOrderLevel,  Price = x.SellingPrice,  }   1. Vendor Products not in PurchaseOrderDetails   looking for an item in VendorStockItemList is inside ActiveOrderDetailList  something like this:  var exists = (from x in context. VendorStockItemList  where x.StockItemID.Equals(SID) && x.UserName.Equals(username)  select x).FirstOrDefault();  if exist is not null, it means we found our item, remove it from the VendorStockItemList; exists.PlaylistTracks.Remove(item);   1. Generate the new suggested order   ROL-(QOH+QOO) = buffer;  When buffer <0, call the PurchaseOrderDetailsController, use the method #2:  public void Add\_ItemToActiveOrderlist(int orderID, string username, int stockID)  to add the Stock item to the PurchaseOrderDetails |
| **SQL Table(s): (C,R,U,D)** | **Entities/DTOs/POCOs** |
| Vendor(R)  StockItem (R)  PurchaseOrdersDetals (R) | VendorStockItemList(P)  PurchaseOrderDetails(E)  StockItem (E) |
|  | |

**Order total summaries**

|  |  |  |
| --- | --- | --- |
| **Controls** | | **Events** |
| Total | | On Click  Calculating SubTotal,Tax and Total  double subtotal;tax;total;  use foreeach to cycle through the CurrentPurchaseOrderlistview, add prices together to get subtotal, something like this:  foreach(ListViewItem item in this.PurchaseOrderDetails.Item){  subtotal += double.Parse((item.FindControl(“PriceLabel”) as Label).Text);}  Then calculate Tax = subtotal \* 0.05;  Total = subtotal + Tax;  Display subtotal, Tax, Total via their associated labels. |
| **BBL Class(es) and Method (s)** | | |
|  |  | |
| **SQL Table(s): (C,R,U,D)** | | **Entities/DTOs/POCOs** |
|  | |  |

**Appropriate event buttons**

**Update order**

|  |  |
| --- | --- |
| **Controls** | **Events** |
| update | On Click  Find controls of the username, purchaseOrderID, stockID, QTO, Price.  Set string username, int purchaseOrderID, int stockID, int QTO, decimal price;  Use foreeach to cycle through the CurrentPurchaseOrderlistview, call the controller to “update” every single items in this list.  Call the Total event to get the updated total. |
| **BBL Class(es) and Method (s)** | |
| PurchaseOrderDetailsController | public void Update\_ItemToActiveOrderlist(int stockID, int purchaseOrderID, int quantitiesTO, string username, decimal price)  Find the existing item, then add the new values to the PurchaseOrderDetails  Snippet:   1. Find the purchase order   PurchaseOrders exists = context. PurchaseOrders.Where(x => x.PurchaseOrderID.Equals(purchaseOrderID  ).Select(x => x).FirstOrDefault();   1. Use the stockID to Find the item in that purchase order 2. Update the purchase order detail item.   updateSItem.StockItemID = stockID;  updateSItem.Quantities = quantitiesTO;  updateSItem.PurchasePrice = price;  exists.PurchaseOrderDetails.Update(updateSItem);  context.SaveChanges();  OR   1. Find the existing purchase order 2. Find the stock item 3. Update/Replace all of its information. |
| **SQL Table(s): (C,R,U,D)** | **Entities/DTOs/POCOs** |
| PurchaseOrders(R)  PurchaseOrderDetails(R,U) | PurchaseOrders(E)  PurchaseOrderDetails(E) |

**Add to Order**

|  |  |
| --- | --- |
| **Controls** | **Events** |
| Add | On Click  From the VendorStockItemList(P), Find controls of the username, purchaseOrderID, stockID.  Set string username, int purchaseOrderID, int stockID,  Call PurchaseOrderDetailsController to add the selected item to the PurchaseOrderDetails  Call the Total event to get the updated total. |
| **BBL Class(es) and Method (s)** | |
| PurchaseOrderDetailsController | public void Add\_ItemToActiveOrderlist(int orderID, string username, int stockID)  to add the Stock item to the PurchaseOrderDetails |
| **SQL Table(s): (C,R,U,D)** | **Entities/DTOs/POCOs** |
| PurchaseOrders(R)  PurchaseOrderDetails(R,U) | PurchaseOrders(E)  PurchaseOrderDetails(E)  VendorStockItemList(P) |

**Place Order**

|  |  |
| --- | --- |
| **Controls** | **Events** |
| Place Order | On Click  Set int orderID  Call PurchaseOrderController to generate OrderDate and OrderNumber  Refresh the page to clearout the Current Active Order field. |
| **BBL Class(es) and Method (s)** | |
| PurchaseOrderController | public void PlaceOrder(int orderID)   1. Find the existing order 2. Generate OrderNumber 3. Generate OrderDate,   context.SaveChanges(); |
| **SQL Table(s): (C,R,U,D)** | **Entities/DTOs/POCOs** |
| PurchaseOrders(R,U) | PurchaseOrders(E) |

**Delete Order**

|  |  |
| --- | --- |
| **Controls** | **Events** |
| Place Order | On Click  Foreach through the entire PurchaseOrderDetails, set the List<int> itemsToDelete,  Set int orderID; string username;  Call PurchaseOrderDetailsController to delete the PurchaseOrderDetails  Call PurchaseOrderController to delete the PurchaseOrderController  Refresh the page |
| **BBL Class(es) and Method (s)** | |
| PurchaseOrderDetailsController  PurchaseOrderController | PurchaseOrderDetailsController:  public void DeleteItem(string username, int orderID, List<int> int trackstodelete)  delete all Items  context.SaveChanges();  PurchaseOrderController  public void DeleteItem(string username, int orderID)  Delete the order that matches the orderID.  context.SaveChanges();  public void PlaceOrder(int orderID)   1. Find the existing order 2. Generate OrderNumber 3. Generate OrderDate, |
| **SQL Table(s): (C,R,U,D)** | **Entities/DTOs/POCOs** |
| PurchaseOrders(R,U) | PurchaseOrders(E) |

Comments

You are missing the Delete button event. You have gone into much more detail than expected (you documented code). You have a good understanding of what to do.