Table Paging Lab

Lab 1: Add Pager Classes

Locate the **\Labs-StartingCode** and copy the \PagerClasses folder into the **AdvWorks.Common** project

Lab 2: Modify View Model Base Class

In the **AdvWorks.Common** project open the **\BaseClasses\ViewModelBase.cs** file and add a few new properties

```
/// <summary>
/// Get/Set # of records per page
/// </summary>
public int PageSize { get; set; }

/// <summary>
/// Get/Set whether or not the pager is visible
/// </summary>
public bool IsPagerVisible { get; set; }

/// <summary>
/// Get/Set the Pager object
/// </summary>
public PDSCPager Pager { get; set; }

/// <summary>
/// Get/Set the page collection
/// </summary>
public PagerItemCollection Pages { get; set; }

/// <summary>
/// Get/Set a list of page sizes to display to the user
/// </summary>
public List<int> PageSizes { get; set; }

/// <summary>
/// Get/Set the current page command
/// such as 'first', 'next', 'last', 'previous', or a page number
/// </summary>
public string PageCommand { get; set; }
```

Modify the Init() method to initialize some of these properties

```
public virtual void Init() {
  SortExpression = string.Empty;
  SortExpressionPrevious = string.Empty;
  SortDirection = "asc";

  Pager = new PDSCPager();
  IsPagerVisible = true;
  PageSize = 5;
  // The following could be put into a drop-down for the user to
select the page size
  PageSizes = new List<int>
  {
    5,
    10,
    25,
    50,
    100
  };
}
```

Modify the SetSortProperties() method to only do the sorting if not doing the paging.

```
protected virtual string SetSortProperties() {
  string ret;

  // If not paging, do the sorting
  if (string.IsNullOrEmpty(PageCommand)) {
    // See if sort expression is same as previous one
    if (SortExpression == SortExpressionPrevious) {
      ret = (SortExpression + (SortDirection == "asc" ? "_desc" :
"_asc")).ToLower();
      SortDirection = SortDirection == "asc" ? "desc" : "asc";
    }
    else {
      ret = SortExpression.ToLower() + "_asc";
      SortDirection = "asc";
    }

    // Set Previous Expression
    SortExpressionPrevious = SortExpression;
  }
  else {
    // If paging, just return the current sort stuff
    ret = SortExpression.ToLower() + "_" + SortDirection;
  }

  return ret;
}
```

Add two new methods

```
#region SetPagerObject Method
protected virtual void SetPagerObject(int totalRecords) {
  // Set Pager Information
  Pager.TotalRecords = totalRecords;
  Pager.PageSize = PageSize;

  // Set Pager Properties
  Pager.SetPagerProperties(PageCommand);

  // Build paging collection
  Pages = new PagerItemCollection(Pager);

  // Reset PageCommand
  PageCommand = string.Empty;
}
#endregion

#region ResetPagerProperties Method
public virtual void ResetPagerProperties() {
  // Set Pager Information
  Pager = new PDSCPager();
  PageCommand = string.Empty;
}
#endregion
```

Lab 3: Modify Search Base Class

Open the **SearchBase.cs** file and add a couple of new properties

```
/// <summary>
/// Get/Set the Page number the user is on
/// </summary>
public int PageIndex { get; set; }
/// <summary>
/// Get/Set the Number of Record per page
/// </summary>
public int PageSize { get; set; }
```

In the constructor, initialize these properties

```
public SearchBase() {
  PageIndex = 0;
  PageSize = 5;
}
```

Lab 4: Modify Product View Model

Open the **ProductViewModel.cs** file and modify the Search method as shown below.

```
public virtual void Search() {
  IsDetailVisible = false;

  // Store Search Data
  base.StoreSearchAsJson<ProductSearch>(SearchEntity);

  // Set Sort Property
  SearchEntity.SortExpression = base.SetSortProperties();

  // Setup the Pager object
  base.SetPagerObject(Repository.Count(SearchEntity));
  SearchEntity.PageSize = base.Pager.PageSize;
  SearchEntity.PageIndex = base.Pager.PageIndex;

  if (Repository == null) {
    throw new ApplicationException("Must set the Repository
property.");
  }
  else {
    Products = Repository.Search(SearchEntity).ToList();
  }

  if (Products != null) {
    TotalRows = Products.Count;
  }
}
```

Lab 5: Modify Repository

Open the **ProductRepository.cs** file and add a new method called ApplyPaging().

```
#region ApplyPaging Method
public IQueryable<Product> ApplyPaging(IQueryable<Product> query,
ProductSearch search) {
  query = query.Skip(search.PageIndex * search.PageSize)
               .Take(search.PageSize);

  return query;
}
#endregion
```

Modify the Search() method to call this new method.

```
public IQueryable<Product> Search(ProductSearch search) {
  QueryObject = _DbContext.Products;

  // Add WHERE clause(s)
  QueryObject = AddWhereClause(QueryObject, search);

  // Add ORDER BY clause(s)
  QueryObject = AddOrderByClause(QueryObject, search);

  // Apply Paging
  QueryObject = ApplyPaging(QueryObject, search);

  return QueryObject;
}
```

Locate the Count() method and add the following code

```
public int Count(ProductSearch search) {
  QueryObject = _DbContext.Products;

  // Add WHERE clause(s)
  QueryObject = AddWhereClause(QueryObject, search);

  // Count using Search Criteria
  return QueryObject.Count();
}
```

Lab 6: Modify Product Maintenance Controller

Open the **ProductMaintenanceController.cs** file

Locate the ProductSearch() method and modify it as shown

```
[HttpGet]
public IActionResult ProductSearch(ProductViewModel vm) {
  vm.Repository = _repo;

  // Reset Pager
  vm.ResetPagerProperties();

  // Call method to search for products
  vm.Search();

  return View("ProductMaintenance", vm);
}
```

Lab 7: Modify HTML

Add New Function to Site.js

Open the **\wwwroot\js\site.js** file and add a new function.

```
function pageOnClick(command) {
  $("#PageCommand").val(command);

  $("form").submit();
}
```

Add New Partial View for Pager Hidden Fields

Add a new view named **_PageHiddenFields.cshtml** to the **\Views\Shared** folder.

```
@model AdvWorks.Common.ViewModelBase

<input type="hidden" asp-for="Pager.PageIndex" />
<input type="hidden" asp-for="Pager.StartingRow" />
<input type="hidden" asp-for="PageCommand" />
```

Add New Partial View for Pager

Add a new view named **_Pager.cshtml** to the **\Views\Shared** folder.

```
@if (Model.IsPagerVisible) {
  <div class="row">
    <div class="col-12">
      <ul class="pagination">
        @foreach (AdvWorks.Common.PagerItem item in Model.Pages) {
          <li class="page-item @item.CssClass">
            <a class="page-link" href="#"
               onclick="pageOnClick('@item.Argument')"
               title="@item.Tooltip">
              @Html.Raw(item.Text)
            </a>
          </li>
        }
      </ul>
    </div>
  </div>
}
```

Modify Product Maintenance Page

Open the **ProductMaintenance.cshtml** and add a new <partial> tag as shown in the code in bold below

```
@if (Model.IsDetailVisible) {
  <partial name="_Detail" />
}
else {
  <partial name="_Search" />
  <partial name="_List" />
  <partial name="_Pager" />
}
```

Modify _List Page

Open the **_List.cshtml** page and add the new _PageHiddenFields.cshtml partial page

```
<form method="get" asp-action="SortPage">
  <partial name="_SortHiddenFields" />
  <partial name="_PageHiddenFields" />

  // REST OF THE HTML HERE
</form>
```

Try it Out

Run the application

Try out paging, sorting, searching and all combinations to ensure everything is still working.