Searching Lab

Lab 1: Create Customer Search Class

Create a base class for all search classes to inherit from.

Right mouse-click on the BaseClasses folder and add a new class named **SearchBase**. Replace the entire contents of this new file with the following code.

```
namespace AdvWorksAPI.BaseClasses;

public class SearchBase
{
  public SearchBase()
  {
    OrderBy = string.Empty;
  }

  public SearchBase(string orderBy)
  {
    OrderBy = orderBy;
  }

  public string OrderBy { get; set; }
}
```

Create Customer Search Class

Right-mouse click on the AdvWorksAPI folder and add a new folder called **SearchClasses**.

Right-mouse click on the SearchClasses folder and add a new class named **CustomerSearch**. Replace the entire contents of this new file with the following code.

```
using AdvWorksAPI.BaseClasses;

namespace AdvWorksAPI.SearchClasses;

public class CustomerSearch : SearchBase
{
  public CustomerSearch()
  {
    OrderBy = "LastName";
    FirstName = string.Empty;
    LastName = string.Empty;
    Title = string.Empty;
  }

  public string? FirstName { get; set; }
  public string? LastName { get; set; }
  public string? Title { get; set; }

  /// <summary>
  /// The following allows us to bind the CustomerSearch
on the query line when using Minimal APIs
  /// </summary>
  /// <param name="httpContext"></param>
  /// <returns></returns>
  public static ValueTask<CustomerSearch>
BindAsync(HttpContext httpContext)
  {
    ValueTask<CustomerSearch> ret;

      ret = ValueTask.FromResult<CustomerSearch>(
        new CustomerSearch
        {
          FirstName =
httpContext.Request.Query["firstname"].ToString(),
          LastName =
httpContext.Request.Query["lastname"].ToString(),
          Title =
httpContext.Request.Query["title"].ToString(),
        });

    return ret;
  }
}
```

Lab 2: Add Search Methods to Customer Repository Class

Open the **CustomerRepository.cs** file and add some new Search methods.

```
#region Search Methods
public List<Customer> Search(CustomerSearch search)
{
  IQueryable<Customer> query = _DbContext.Customers;

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

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

  return query.ToList();
}

protected virtual IQueryable<Customer>
AddWhereClause(IQueryable<Customer> query,
CustomerSearch search)
{
  // Perform Searching
  if (!string.IsNullOrEmpty(search.FirstName)) {
    query = query.Where(row =>
row.FirstName.Contains(search.FirstName));
  }
  if (!string.IsNullOrEmpty(search.LastName)) {
    query = query.Where(row =>
row.LastName.Contains(search.LastName));
  }
  if (!string.IsNullOrEmpty(search.Title)) {
  // NOTE: Do NOT simplify this expression, or the query
will not work.
#pragma warning disable IDE0075 // Simplify conditional
expression
    query = query.Where(row =>
string.IsNullOrEmpty(row.Title) ? true :
row.Title.StartsWith(search.Title));
#pragma warning restore IDE0075 // Simplify conditional
expression
  }

  return query;
}

protected virtual IQueryable<Customer>
AddOrderByClause(IQueryable<Customer> query,
CustomerSearch search)
{
  switch (search.OrderBy.ToLower()) {
    case "":
    case "lastname":
      query = query.OrderBy(row => row.LastName);
      break;
    case "firstname":
      query = query.OrderBy(row => row.FirstName);
      break;
    case "title":
      query = query.OrderBy(row => row.Title);
      break;
  }

  return query;
}
#endregion
```

Lab 3: Add Search Methods to IRepository Interface

Open the **IRepository.cs** file and **replace** the entire contents of the file with the following code.

```
namespace AdvWorksAPI.Interfaces;

public interface IRepository<TEntity, TSearch>
{
  List<TEntity> Get();
  TEntity? Get(int id);
  List<TEntity> Search(TSearch search);

  TEntity Insert(TEntity entity);
  TEntity Update(TEntity entity);
  TEntity SetValues(TEntity current, TEntity changes);
  bool Delete(int id);
}
```

Lab 4: Update all Usages of IRepository Interface

You have now just broken everywhere that you were using IRepository<Customer>.

Open the **CustomerRepository.cs** file and modify the declaration

```
public class CustomerRepository : IRepository<Customer,
CustomerSearch>
```

Open the **ServiceExtensions.cs** file and modify the AddRepositoryClasses()

```
public static void AddRepositoryClasses(this
IServiceCollection services)
{
  // Add Repository Classes
  services.AddScoped<IRepository<Customer,
CustomerSearch>, CustomerRepository>();
}
```

Open the **CustomerRouter.cs** file and modify all occurrences of the ***IRepository<Customer>*** to ***IRepository<Customer, CustomerSearch>***.

Compile the code to ensure you fixed everything.

Lab 5: Retrieve Data Using the Search Method

Let's add a search method for data based on items filled into the Customer Search class.

Open the **CustomerRouter.cs** file and add a new method that looks like the following:

```
protected virtual IResult Search(CustomerSearch search,
IRepository<Customer, CustomerSearch> repo)
{
  IResult ret;
  List<Customer> list;

  InfoMessage = "Can't find customers matching the
criteria passed in.";

  try {
    // Search for Data
    list = repo.Search(search);

    if (list != null && list.Count > 0) {
      return Results.Ok(list);
    }
    else {
      return Results.NotFound(InfoMessage);
    }
  }
  catch (Exception ex) {
    ErrorLogMessage = "Error in
CustomerController.Search()";

    ret = HandleException(ex);
  }

  return ret;
}
```

Add a new MapGet() method to the AddRoutes() method.

```
app.MapGet($"/{UrlFragment}/Search", (CustomerSearch
entity, IRepository<Customer, CustomerSearch> repo) =>
Search(entity, repo))
   .WithTags(TagName)
   .Produces(200)
   .Produces<List<Customer>>()
   .Produces(404);
```

Try it Out

NOTE: You **CAN'T** call the Search method from Swagger

Type the following into the browser

```
http://localhost:5114/api/customer/Search?firstname=A&la
stname=B&title=Mrs
```