Asynchronous Lab

Lab 1: Add GetAsync() Method

Open the **IRepository.cs** file and make the interface look like the following:

```
public interface IRepository<TEntity, TSearch>
{
  // Asynchronous Methods
  Task<List<TEntity>> GetAsync();


  // Synchronous Methods
  // REST OF THE CODE HERE
}
```

Add Async Repository Methods

Open the **CustomerRepository.cs** file and add **partial** keyword to the class definition.

```
/// <summary>
/// Class to work synchronously with Customer data
/// </summary>
public partial class CustomerRepository : IRepository<Customer,
CustomerSearch>
{
  // REST OF THE CODE HERE
}
```

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

```
using AdvWorksAPI.EntityLayer;
using Microsoft.EntityFrameworkCore;

namespace AdvWorksAPI.RepositoryLayer;

/// <summary>
/// Class to work asynchronously with Customer data
/// </summary>
public partial class CustomerRepository
{
  #region GetAsync Method
  /// <summary>
  /// Get all Customer objects asynchronously
  /// </summary>
  /// <returns>A list of Customer objects</returns>
  public async Task<List<Customer>> GetAsync()
  {
    return await _DbContext.Customers.OrderBy(row =>
row.LastName).ToListAsync();
  }
  #endregion
}
```

Create Async Router

Right mouse-click on the Routers folder and add a new class named **CustomerAsyncRouter.cs**.

Replace the entire contents of the file with the following code.

```
using AdvWorksAPI.BaseClasses;
using AdvWorksAPI.EntityLayer;
using AdvWorksAPI.Interfaces;
using AdvWorksAPI.SearchClasses;

namespace AdvWorksAPI.RouterClasses;

public class CustomerAsyncRouter : RouterBase
{
  private readonly IRepository<Customer, CustomerSearch> _Repo;
  private readonly AdvWorksAPIDefaults _Settings;

  public CustomerAsyncRouter(IRepository<Customer, CustomerSearch>
repo, ILogger<CustomerRouter> logger, AdvWorksAPIDefaults settings)
: base(logger)
  {
    UrlFragment = "api/CustomerAsync";
    TagName = "CustomerAsync";
    _Repo = repo;
    _Settings = settings;
  }

  /// <summary>
  /// Add asynchronous routes
  /// </summary>
  /// <param name="app">A WebApplication object</param>
  public override void AddRoutes(WebApplication app)
  {
    app.MapGet($"/{UrlFragment}", async () => await GetAsync())
       .WithTags(TagName)
       .Produces(200)
       .Produces<List<Customer>>()
       .Produces(404)
       .Produces(500);
  }

  protected virtual async Task<IResult> GetAsync()
  {
    IResult ret;
    List<Customer> list;
    InfoMessage = "No Customers Found.";

    try {
      list = await _Repo.GetAsync();

      if (list == null || list.Count == 0) {
        ret = Results.NotFound(InfoMessage);
      }
      else {
        ret = Results.Ok(list);
      }
    }
    catch (Exception ex) {
      // Return generic message for the user
      InfoMessage = _Settings.InfoMessageDefault
          .Replace("{Verb}", "GET")
```

```
            .Replace("{ClassName}", TagName);

        ErrorLogMessage = "Error in CustomerAsyncRouter.GetAsync()";

        ret = HandleException(ex);
      }

      return ret;
    }
}
```

Open the **ServiceExtensions.cs** file and in the AddRouterClasses() method add the CustomerAsyncRouter class as a new scoped object.

```
services.AddScoped<RouterBase, CustomerAsyncRouter>();
```

Try it Out

Run the application and click on the **GET /api/CustomerAsync** button to see all results returned.

Lab 2: Add GetAsync(id) Method

Open the **IRepository.cs** file and add a GetAsync(id) method.

```
Task<TEntity?> GetAsync(int id);
```

Open the **CustomerAsyncRepository.cs** file and add a new method

```
#region GetAsync(id) Method
public async Task<Customer?> GetAsync(int id)
{
  return await _DbContext.Customers.Where(row => row.CustomerID ==
id).FirstOrDefaultAsync();
}
#endregion
```

Open the **CustomerAsyncRouter.cs** file and add a new method.

```
protected virtual async Task<IResult> GetAsync(int id)
{
  Customer? entity;

  entity = await _Repo.GetAsync(id);
  if (entity == null) {
    return Results.NotFound($"Customer with CustomerID = '{id}' was
not found.");
  }
  else {
    return Results.Ok(entity);
  }
}
```

Add a new MapGet() method to the AddAsyncRoutes() method

```
app.MapGet($"/{UrlFragment}/{{id:int}}", async (int id) => await
GetAsync(id))
     .WithTags(TagName)
     .Produces(200)
     .Produces<Customer>()
     .Produces(404);
```

Try it Out

Run the application and click on the **GET /api/CustomerAsync/{id}** button

Enter **235** to see the result returned.

Lab 3: Add SearchAsync() Method

Open the **IRepository.cs** file and add a SearchAsync() method.

```
Task<List<TEntity>> SearchAsync(TSearch search);
```

Open the **CustomerAsyncRepository.cs** file and add a new method

```
#region SearchAsync Methods
public async Task<List<Customer>> SearchAsync(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 await query.ToListAsync();
}
#endregion
```

Open the **CustomerAsyncRouter.cs** file and add a new method

```
protected virtual async Task<IResult> SearchAsync(CustomerSearch
search)
{
  IResult ret;
  List<Customer> list;

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

  try {
    // Search for Data
    list = await _Repo.SearchAsync(search);

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

    ret = HandleException(ex);
  }

  return ret;
}
```

Add a new app.MapGet() method to the AddAsyncRoutes() method.

```
app.MapGet($"/{UrlFragment}/Search", async (CustomerSearch search)
=> await SearchAsync(search))
     .WithTags(TagName)
     .Produces(200)
     .Produces<List<Customer>>()
     .Produces(404);
```

Try it Out

Run the application.

NOTE: You **CAN'T** pass parameters to the Search method from Swagger.

Type the following into the browser.

```
http://localhost:5114/api/Customerasync/Search?firstname=A&lastname=
B&title=Mrs
```