.NET 10 App Dev Hands-On Lab

Razor Pages/MVC Lab 9a – Data Services

This lab builds the data services for the ASP.NET Core applications. Before starting this lab, you must have completed Razor Pages/MVC Lab 8.

Copilot Agent Mode

The following prompts will complete this lab. Please verify that the generated code matches the lab document.

```
Setup Prompt: Always use file scoped namespaces. Always combine attributes on a single line when
possible. The project does not use nullable reference types. There is a GlobalUsings.cs file that
includes common usings, don't include using statements in new files if they are already in the
globalusings.cs file. I prefer expression bodied members when possible. Single line if statements
should still use braces. Use ternary operators when appropriate. Use internal over private. All
classes and methods are public unless told otherwise. Use default primary constructors when
possible and don't declare a class level variable if the parameter from the constructor can be
used. All work is to be done in the AutoLot.Services project unless otherwise specified.

Prompt: Add the following global usings to the GlobalUsings.cs file if they do not already exist
(sorted alphabetically. Don't remove any existing global using statements).
global using AutoLot.Models.Entities;
global using AutoLot.Models.Entities.Base;

Prompt: Add a new folder named DataServices at the root of the project, and in that folder, create
a new folder named Interfaces. In that folder, create a new folder named Base. In that folder,
create a new interface named IDataServiceBase with TEntity generic type parameter constrained to
BaseEntity and new(). Add the following method signatures:
  Task<IQueryable<TEntity>> GetAllAsync();
  Task<TEntity> FindAsync(int id);
  Task<TEntity> UpdateAsync(TEntity entity, bool persist = true);
  Task DeleteAsync(TEntity entity, bool persist = true);
  Task<TEntity> AddAsync(TEntity entity, bool persist = true);
  Add this comment before the ResetChangeTracker interface method: //implemented ghost method
since it won’t be used by the API data service
  void ResetChangeTracker() { }

Prompt: Add the following global usings to the GlobalUsings.cs file if they do not already exist
(sorted alphabetically. Don't remove any existing global using statements).
global using AutoLot.Services.DataServices;
global using AutoLot.Services.DataServices.Interfaces;
global using AutoLot.Services.DataServices.Interfaces.Base;

Prompt: In the Interfaces folder, create a new interface named ICarDataService that inherits from
IDataServiceBase with TEntity as Car. Add the following method signature:
  Task<IQueryable<Car>> GetAllByMakeIdAsync(int? makeId);
In the interfaces folder, create a new interface named IMakeDataService that inherits from
IDataServiceBase with TEntity as Make. No additional methods are needed.
```

```
Prompt: Add a new folder named Dal under the DataServices folder. In that folder, create a new
folder named Base. In that folder, create a new public abstract class named DalDataServiceBase
with TEntity generic type parameter constrained to BaseEntity and new(). Implement the
IDataServiceBase interface. In the default constructor, take an instance of IAppLogging and
IBaseRepo<TEntity>. Add protected fields for those, adding Instance after the typename for the
field named. Implement all interface methods using await Task.Run and repo methods. The Update
method should be multi statement in the Task.Run, first saving, then returning the entity. Add
method should be multi statement in the Task.Run, saving the entity, then returning it. Neither
needs to call find after saving, they can just return the entity passed into the method.

Prompt: Add the following global usings to the GlobalUsings.cs file if they do not already exist
(sorted alphabetically. Don't remove any existing global using statements).
global using AutoLot.Services.DataServices.Dal;
global using AutoLot.Services.DataServices.Dal.Base;

Prompt: In the DataServices\Dal folder, create a class named CarDalDataService that inherits from
DalDataServiceBase with TEntity as Car. Implement the ICarDataService interface. In the
constructor, take in IAppLogging and ICarRepo parameters and pass them to the base class. Add the
GetAllByMakeIdAsync method implementation that calls GetAllBy on the ICarRepo instance using
makeId.Value. If the makeId does not have a value, return GetAllIgnoreQueryFilters().

Prompt: In the DataServices\Dal folder, create a class named MakeDalDataService that inherits from
DalDataServiceBase with TEntity as Make. Implement the IMakeDataService interface. In the
constructor, take in IAppLogging and IMakeRepo parameters and pass them to the base class.

Prompt: In the extension block in the StringExtensions class, add a new method named
RemoveAsyncSuffix that removes "Async" if it's the end of the string passed in. Use the existing
extension block, do not create a new one.
```

NOTE: At the time of this writing, Copilot is struggling with C# 14 extension blocks. Please verify the code matches what's in the listing at the end of this lab.

Manual

- Add the following to the `GlobalUsings.cs` in the `AutoLot.Services` project:

```
global using AutoLot.Models.Entities;
global using AutoLot.Models.Entities.Base;
```

- Add a directory named `DataServices` in the root of the `AutoLot.Services` project

Part 1: Add the Data Services Interface and DAL Classes

The data services will encapsulate the calls for CRUD operations.

Step 1: Add the Interfaces

- Add a new directory named `Interfaces` under the `DataServices` directory. In that folder, add another folder named `Base`, and in that folder, add a new interface named `IDataServiceBase` and update the code to the following:

```
namespace AutoLot.Services.DataServices.Interfaces.Base;
public interface IDataServiceBase<TEntity> where TEntity : BaseEntity, new()
{
  Task<IQueryable<TEntity>> GetAllAsync();
  Task<TEntity> FindAsync(int id);
  Task<TEntity> UpdateAsync(TEntity entity, bool persist = true);
  Task DeleteAsync(TEntity entity, bool persist = true);
  Task<TEntity> AddAsync(TEntity entity, bool persist = true);
  //implemented ghost method since it won’t be used by the API data service
  void ResetChangeTracker() { }
}
```

- Add the following to the `GlobalUsings.cs` in the `AutoLot.Services` project:

```
global using AutoLot.Services.DataServices;
global using AutoLot.Services.DataServices.Interfaces;
global using AutoLot.Services.DataServices.Interfaces.Base;
```

- Add a new interface named `ICarDataService` to the Interfaces folder and update the code to the following:

```
namespace AutoLot.Services.DataServices.Interfaces;
public interface ICarDataService : IDataServiceBase<Car>
{
  Task<IQueryable<Car>> GetAllByMakeIdAsync(int? makeId);
}
```

- Add a new interface named `IMakeDataService` and update the code to the following:

```
namespace AutoLot.Services.DataServices.Interfaces;
public interface IMakeDataService : IDataServiceBase<Make> { }
```

Step 2: Add the DalDataServiceBase Class

- Add a new directory named `Dal` under the `DataServices` directory. In that folder, add a directory named `Base`. In that folder, add a new class named `DalDataServiceBase` and update the code to the following:

```
namespace AutoLot.Services.DataServices.Dal.Base;

public abstract class DalDataServiceBase<TEntity>(IAppLogging appLoggingInstance,
IBaseRepo<TEntity> baseRepoInstance)
  : IDataServiceBase<TEntity> where TEntity : BaseEntity, new()
{
  protected readonly IAppLogging AppLoggingInstance = appLoggingInstance;
  protected readonly IBaseRepo<TEntity> BaseRepoInstance = baseRepoInstance;
  public virtual async Task<IQueryable<TEntity>> GetAllAsync()
    => await Task.Run(() => BaseRepoInstance.GetAll());
  public virtual async Task<TEntity> FindAsync(int id)
    => await Task.Run(() => BaseRepoInstance.Find(id));
  public virtual async Task<TEntity> UpdateAsync(TEntity entity, bool persist = true)
  {
    return await Task.Run(() =>
    {
      BaseRepoInstance.Update(entity, persist);
      return entity;
    });
  }
  public virtual async Task DeleteAsync(TEntity entity, bool persist = true)
    => await Task.Run(() => BaseRepoInstance.Delete(entity, persist));
  public virtual async Task<TEntity> AddAsync(TEntity entity, bool persist = true)
  {
    return await Task.Run(() =>
    {
      BaseRepoInstance.Add(entity, persist);
      return entity;
    });
  }
  //implemented ghost method since it won't be used by the API data service
  public virtual void ResetChangeTracker() { }
}
```

- Add the following to the `GlobalUsings.cs` class:

```
global using AutoLot.Services.DataServices.Dal;
global using AutoLot.Services.DataServices.Dal.Base;
```

Step 3: Add the CarDalDataService Class

- Add a new class named `CarDalDataService` in the `Dal` directory and update the code to the following:

```
namespace AutoLot.Services.DataServices.Dal;

public class CarDalDataService(IAppLogging appLoggingInstance, ICarRepo carRepoInstance)
  : DalDataServiceBase<Car>(appLoggingInstance, carRepoInstance), ICarDataService
{
  public async Task<IQueryable<Car>> GetAllByMakeIdAsync(int? makeId)
    => await Task.Run(() => makeId.HasValue
      ? ((ICarRepo)BaseRepoInstance).GetAllBy(makeId.Value)
      : ((ICarRepo)BaseRepoInstance).GetAllIgnoreQueryFilters());
}
```

Step 4: Add the MakeDalDataService Class

- Add a new class named `MakeDalDataService` in the `Dal` directory and update the code to the following:

```
namespace AutoLot.Services.DataServices.Dal;

public class MakeDalDataService(IAppLogging appLoggingInstance, IMakeRepo makeRepoInstance)
  : DalDataServiceBase<Make>(appLoggingInstance, makeRepoInstance), IMakeDataService { }
```

Part 2: Add the RemoveAsync Extension Method

- Add the following method to the `extension` block in the `StringExtensions.cs` class (in the `Utilities` folder) in the `AutoLot.Services` project:

```
public string RemoveAsyncSuffix()
  => value != null && value.EndsWith("Async", StringComparison.Ordinal)
    ? value[..^5]
    : value;
```

Summary

This lab added the common code for the DAL Data Services to be used by the ASP.NET Core projects.

Next steps

In the next part of this tutorial series, you will use the data services in the ASP.NET Core project.