.NET 9 App Dev Hands-On Lab

MVC Lab 9b – Data Services Part 2

This lab swaps out the repos for the data service. Before starting this lab, you must have completed Razor Pages/MVC Lab 9a.

All work in this lab takes place in the `AutoLot.Mvc` project.

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 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.Mvc 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.Services.DataServices.Dal;
global using AutoLot.Services.DataServices.Interfaces;
global using AutoLot.Services.DataServices.Interfaces.Base;

Prompt: Add the following to the Program.cs file, after the existing service registrations:
builder.Services.AddScoped<ICarDataService, CarDalDataService>();
builder.Services.AddScoped<IMakeDataService, MakeDalDataService>();

Prompt: Update the BaseCrudController to take an instance of IDataServiceBase<TEntity> (parameter
name of mainDataService) instead of IBaseRepo<TEntity>.
Remove the BaseRepoInstance property and add protected readonly MainDataService property, updated
from the constructor parameter. Update all methods to be async, change the return type to be
Task<T> instead of T, add the Async suffix to the names, replace the calls to
RepoBaseInstance with the calls on the MainDataService class, await all async calls. Use
RemoveAsyncSuffix instead of Replace("Async", string.Empty).
```

```
Prompt: Update the CarsController to take an instance of ICarDataService instead of ICarRepo and
IMakeDataService instead of IMakeRepo. Don't assign IMakeDataService to a class level variable.
Update all methods to be async, change the return type to be Task<T> instead of T, add the Async
suffix to the names, and await the calls to MainDataService. Replace usages of RepoBaseInstance to
use MainDataService instead. Use RemoveAsyncSuffix instead of Replace("Async", string.Empty).
Only override the GetLookupValues (order by Name) and ByMake endpoints. Remember to set the
ViewBag.MakeName property. Keep the BadEndPoint commented out.
Use nameof instead of magic strings.

Prompt: Update the MakesController to take an instance of IMakeDataService instead of IMakeRepo.
Update all methods to be async, change the return type to be Task<T> instead of T, add the Async
suffix to the names, and await the calls to MainDataService.
usages of RepoBaseInstance to use MainDataService instead. Use RemoveAsyncSuffix instead of
Replace("Async", string.Empty).

Prompt: Update the MenuViewComponent to take in IMakeDataService instead of IMakeRepo. Remove the
Task.Run and replace the call to the repo with the call to the data service.

Prompt: Update all of the custom taghelpers by updating the actionname assignment to use the Async
suffix and RemoveAsyncSuffix.
```

Manual

Part 1: Update AutoLot.Mvc to use the DAL Data Service

Step 1: Change from Repos to Data Services

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

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

- Add the following to the `Program.cs` file:

```
builder.Services.AddScoped<ICarDataService, CarDalDataService>();
builder.Services.AddScoped<IMakeDataService, MakeDalDataService>();
```

Step 2: Update the BaseCrudController

- Update the primary constructor and field to declare and initialize `IDataServiceBase` instead of the `IBaseRepo`:

```
public abstract class BaseCrudController<TEntity>(
  IAppLogging appLogging,
  IDataServiceBase<TEntity> mainDataService) : Controller
  where TEntity : BaseEntity, new()
{
  protected readonly IAppLogging AppLoggingInstance = appLogging;
  protected readonly IBaseRepo<TEntity> BaseRepoInstance = baseRepo;
  protected readonly IDataServiceBase<TEntity> MainDataService = mainDataService;
  //omitted for brevity
}
```

- Update the `GetLookupValues` method to be async:

```
protected abstract Task<SelectList> GetLookupValuesAsync();
```

- Anywhere the `BaseRepoInstance` was called, change the method to async and update the code to call the `MainDataService`:

```
protected async Task<TEntity> GetOneEntityAsync(int? id)
  => id == null ? null : await MainDataService.FindAsync(id.Value);

[HttpGet, Route("/[controller]"), Route("/[controller]/[action]")]
public virtual async Task<IActionResult> IndexAsync()
  => View(await MainDataService.GetAllAsync());
[HttpGet("{id?}")]
public virtual async Task<IActionResult> DetailsAsync(int? id)
{
  if (!id.HasValue) { return BadRequest();}
  var entity = await GetOneEntityAsync(id);
  return entity == null ? NotFound() : View(entity);
}
[HttpGet]
public virtual async Task<IActionResult> CreateAsync()
{
  ViewData["LookupValues"] = await GetLookupValuesAsync();
  return View();
}
[HttpPost]
public virtual async Task<IActionResult> CreateAsync(TEntity entity)
{
  if (!ModelState.IsValid)
  {
    ViewData["LookupValues"] = await GetLookupValuesAsync();
    return View(entity);
  }
  var savedEntity = await MainDataService.AddAsync(entity);
  return RedirectToAction(nameof(DetailsAsync).RemoveAsyncSuffix(), new { id = savedEntity.Id });
}
[HttpGet("{id?}")]
public virtual async Task<IActionResult> EditAsync(int? id)
{
  if (!id.HasValue) { return BadRequest();}
  var entity = await GetOneEntityAsync(id);
  if (entity == null) { return NotFound();}
  ViewData["LookupValues"] = await GetLookupValuesAsync();
  return View(entity);
}
[HttpPost("{id}")]
public virtual async Task<IActionResult> EditAsync(int id, TEntity entity)
{
  if (id != entity.Id) { return BadRequest();}
  if (!ModelState.IsValid)
  {
    ViewData["LookupValues"] = await GetLookupValuesAsync();
    return View(entity);
  }
  await MainDataService.UpdateAsync(entity);
  return RedirectToAction(nameof(DetailsAsync).RemoveAsyncSuffix(), new {id = entity.Id});
```

```
}
[HttpGet("{id?}")]
public virtual async Task<IActionResult> DeleteAsync(int? id)
{
  if (!id.HasValue) { return BadRequest();}
  var entity = await GetOneEntityAsync(id);
  return entity == null ? NotFound() : View(entity);
}
```

```
[HttpPost("{id}")]
public virtual async Task<IActionResult> DeleteAsync(int id, TEntity entity)
{
  if (id != entity.Id) { return BadRequest();}
  await MainDataService.DeleteAsync(entity);
  return RedirectToAction(nameof(IndexAsync).RemoveAsyncSuffix());
}
```

Step 3: Update the CarsController

- Update the class to the following:

```
public class CarsController(
  IAppLogging logging,
  ICarDataService carDataService,
  IMakeDataService makeDataService)
    : BaseCrudController<Car>(logging, carDataService)
{
  protected override async Task<SelectList> GetLookupValuesAsync()
    => new((await makeDataService.GetAllAsync()).OrderBy(m => m.Name),
            nameof(Make.Id), nameof(Make.Name));
  [HttpGet("{makeId}/{makeName}")]
  public async Task<IActionResult> ByMakeAsync(int makeId, string makeName)
  {
    ViewBag.MakeName = makeName;
    return View(await carDataService.GetAllByMakeIdAsync(makeId));
  }
  //public IActionResult BadEndPoint() => new OkObjectResult(5);
}
```

Step 4: Update the MakesController (in the Admin Area)

- Update the class to the following:

```
[Area("Admin")]
[Route("Admin/[controller]/[action]")]
public class MakesController(
  IAppLogging appLogging,
  IMakeDataService mainDataService)
    : BaseCrudController<Make>(appLogging, mainDataService)
{
  protected override Task<SelectList> GetLookupValuesAsync()
    => Task.FromResult<SelectList>(null);
  [Route("/Admin")]
  [Route("/Admin/[controller]")]
  [Route("/Admin/[controller]/[action]")]
  public override async Task<IActionResult> IndexAsync() => await base.IndexAsync();
}
```


Step 5: Update the MenuViewComponent

- Update the class to the following:

```
public class MenuViewComponent(IMakeDataService dataService) : ViewComponent
{
  public async Task<IViewComponentResult> InvokeAsync()
  {
    var makes = (await dataService.GetAllAsync()).ToList();
    if (!makes.Any())
    {
      return new ContentViewComponentResult("Unable to get the makes");
    }
    return View("MenuView", makes);
  }
}
```

Step 6: Update the Custom TagHelpers

- Update the code that sets the `ActionName` in each of the custom `TagHelpers`:

```
//ItemDetailsTagHelper.cs
  ActionName = nameof(CarsController.DetailsAsync).RemoveAsyncSuffix();
//ItemEditTagHelper.cs
  ActionName = nameof(CarsController.EditAsync).RemoveAsyncSuffix();
//ItemListTagHelper.cs
  ActionName = nameof(CarsController.IndexAsync).RemoveAsyncSuffix();
//ItemCreateTagHelper.cs
  ActionName = nameof(CarsController.CreateAsync).RemoveAsyncSuffix();
//ItemDeleteTagHelper.cs
  ActionName = nameof(CarsController.DeleteAsync).RemoveAsyncSuffix();
```

Summary

This lab updated the ASP.NET Core web application using the MVC pattern to use the Data Services and completed the web application.

