.NET 10 App Dev Hands-On Lab

Razor Pages 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.Web` project.

Copilot Agent Mode

```
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. Don't add a constructor unless instructed to
do so. Use primary constructors when possible and don't declare a class level variable if the
parameter from the constructor can be used. Don't initialize properties unless instructed to do
so. All work is to be done in the AutoLot.Web project unless otherwise specified.

Prompt:
```

Manual

Part 1: Update The Base Page

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 with the other calls to add interfaces to the DI container:

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

Step 2: Update the BasePageModel

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

```
public abstract class BasePageModel<TEntity>(
    IAppLogging appLoggingInstance,
    IDataServiceBase<TEntity> mainDataService,
    string pageTitle)
  : PageModel where TEntity : BaseEntity, new()
{
  protected readonly IAppLogging AppLoggingInstance = appLoggingInstance;
~~  protected readonly IBaseRepo<TEntity> BaseRepoInstance = baseRepoInstance;~~
  protected readonly IDataServiceBase<TEntity> MainDataService = mainDataService;
}
```

- Update the `GetLookupValues` method to be async:

```
protected virtual Task GetLookupValuesAsync() => Task.Run(() => LookupValues = null);
```

- Update the CRUD statements to use the new service instead of the repo:

```
Protected virtual async Task GetOneAsync(int? id)
{
  if (!id.HasValue)
  {
    Error = "Invalid request";
    Entity = null;
    return;
  }
  Entity = await MainDataService.FindAsync(id.Value);
  Error = Entity == null ? "Not found" : string.Empty;
}
protected virtual async Task<IActionResult> SaveOneAsync(
  Func<TEntity,bool,Task<TEntity>> saveFunction)
{
  if (!ModelState.IsValid)
  {
    return Page();
  }
  var savedEntity = await saveFunction(Entity, true);
  return RedirectToPage("Details", new { id = savedEntity.Id });
}
protected virtual async Task<IActionResult> SaveWithLookupAsync(
    Func<TEntity,bool,Task<TEntity>> saveFunction)
{
  if (!ModelState.IsValid)
  {
    await GetLookupValuesAsync();
    return Page();
  }
  var savedEntity = await saveFunction(Entity, true);
  return RedirectToPage("Details", new { id = savedEntity.Id });
}
protected virtual async Task<IActionResult> DeleteOneAsync(int id)
{
  await MainDataService.DeleteAsync(Entity);
  return RedirectToPage("./Index");
}
```

Part 2: Update the Cars Pages

Step 1: Update the Index Page

- Update the code behind to match the following:

```
namespace AutoLot.Web.Pages.Cars;
public class IndexModel(
  IAppLogging appLogging,
  ICarDataService carDataService)
  : BasePageModel<Car>(appLogging, carDataService, "Inventory")
{
  public string MakeName { get; set; }
  public int? MakeId { get; set; }
  public IEnumerable<Car> CarRecords { get; set; }
  public async Task OnGetAsync(int? makeId, string makeName)
  {
    if (!makeId.HasValue)
    {
      MakeName = "All Makes";
      CarRecords = await MainDataService.GetAllAsync();
      return;
    }
    MakeId = makeId;
    MakeName = makeName;
    CarRecords =  await dataService.GetAllByMakeIdAsync(makeId.Value);
  }
}
```

Step 2: Update the Create Page

- Update the code behind to match the following:

```
namespace AutoLot.Web.Pages.Cars;

public class CreateModel(IAppLogging appLogging, ICarDataService carDataService,
  IMakeDataService makeDataService): BasePageModel<Car>(appLogging, carDataService, "Create")
{
  public async Task OnGetAsync()
  {
    await GetLookupValuesAsync();
    Entity = new Car { IsDrivable = true };
  }
  public async Task<IActionResult> OnPostCreateNewCarAsync()
    => await SaveWithLookupAsync(MainDataService.AddAsync);
  protected override async Task GetLookupValuesAsync()
  {
    LookupValues = new SelectList(
      (await makeDataService.GetAllAsync()).OrderBy(m => m.Name).ToList(),
      nameof(Make.Id), nameof(Make.Name));
  }
}
```

Step 3: Update the Delete Page

- Update the code behind to match the following:

```
namespace AutoLot.Web.Pages.Cars;
public class DeleteModel(IAppLogging appLogging, ICarDataService carDataService)
  : BasePageModel<Car>(appLogging, carDataService, "Delete")
{
  public async Task OnGetAsync(int? id)
  {
    if (!id.HasValue)
    {
      Error = "Invalid request";
      Entity = null;
      return;
    }
    await GetOneAsync(id);
    Error = Entity == null ? "Not found" : string.Empty;
  }
  public async Task<IActionResult> OnPostAsync(int id)
  {
    if (Entity == null || id != Entity.Id)
    {
      Error = "Invalid Request";
      return Page();
    }
    var result = await DeleteOneAsync(id);
    Error = string.Empty;
    Entity = null;
    return result;
  }
}
```

Step 4: Update the Details Page

- Update the code behind to match the following:

```
namespace AutoLot.Web.Pages.Cars;
public class DetailsModel(IAppLogging appLogging, ICarDataService carDataService)
  : BasePageModel<Car>(appLogging, carDataService, "Details")
{
  if (!id.HasValue)
  {
    Entity = null;
    Error = "Invalid Request";
    return;
  }
    await GetOneEntityAsync(id);
    Error = Entity == null ? "Not found" : string.Empty;
}
```

Step 5: Update the Edit Page

- Update the code behind to match the following:

```
namespace AutoLot.Web.Pages.Cars;
public class EditModel(IAppLogging appLogging,  ICarDataService carDataService,
  IMakeDataService makeDataService)
  : BasePageModel<Car>(appLogging, carDataService, "Edit")
{
  public async Task OnGetAsync(int id)
  {
    if (!id.HasValue)
    {
      Entity = null;
      Error = "Invalid Request";
      return;
    }
    await GetLookupValuesAsync();
    await GetOneAsync(id);
    Error = Entity == null ? "Not found" : string.Empty;
  }
  public async Task<IActionResult> OnPostAsync()
    => await SaveWithLookupAsync(MainDataService.UpdateAsync);
  protected override async Task GetLookupValuesAsync()
  {
    if (Entity == null || id != Entity.Id)
    {
      Error = "Invalid Request";
      return Page();
    }
    var result = await SaveOneWithLookupAsync(MainDataService.UpdateAsync);
    Error = string.Empty;
    return result;
  }
}
```

Part 3: Update the Makes Pages

Step 1: Update the Index Page

- Update the code behind to match the following:

```
namespace AutoLot.Web.Areas.Admin.Pages.Makes;
public class IndexModel(IAppLogging appLogging, IMakeDataService dataService) : PageModel
{
  [ViewData] public string Title => "Makes";
  public IEnumerable<Make> MakeRecords { get; set; }
  public async Task OnGetAsync() => MakeRecords = await dataService.GetAllAsync();
}
```

Step 2: Update the Create Page

- Update the code behind to match the following:

```
namespace AutoLot.Web.Areas.Admin.Pages.Makes;
public class CreateModel(IAppLogging appLogging,  IMakeDataService dataService)
  : BasePageModel<Make>(appLogging, dataService, "Create")
{
  public void OnGet() => Entity = new Make();
  public async Task<IActionResult> OnPostAsync() => await SaveOneAsync(MainDataService.AddAsync);
}
```

Step 3: Update the Delete Page

- Update the code behind to match the following:

```
namespace AutoLot.Web.Areas.Admin.Pages.Makes;
public class DeleteModel(IAppLogging appLogging, IMakeDataService dataService)
  : BasePageModel<Make>(appLogging, dataService, "Delete")
{
  public async Task OnGetAsync(int? id) => await GetOneAsync(id);
  public async Task<IActionResult> OnPostAsync(int id) => await DeleteOneAsync(id);
}
```

Step 4: Update the Details Page

- Update the code behind to match the following:

```
namespace AutoLot.Web.Areas.Admin.Pages.Makes;
public class DetailsModel(IAppLogging appLogging, IMakeDataService dataService)
  : BasePageModel<Make>(appLogging, dataService, "Details")
{
  public async Task OnGetAsync(int? id) => await GetOneAsync(id);
}
```

Step 5: Update the Edit Page

- Update the code behind to match the following:

```
namespace AutoLot.Web.Areas.Admin.Pages.Makes;
public class EditModel(IAppLogging appLogging, IMakeDataService dataService)
  : BasePageModel<Make>(appLogging, dataService, "Edit")
```

```
{
  public async Task OnGetAsync(int? id) => await GetOneAsync(id);
  public async Task<IActionResult> OnPostAsync()
     => await SaveOneAsync(MainDataService.UpdateAsync);
}
```

Part 4: Update the RazorSyntax Page

- Update the code behind to match the following:

```
public class RazorSyntaxModel(
  ICarDataService carDataService,
  IMakeDataService makeDataService) : PageModel
{
  [ViewData]
  public SelectList LookupValues { get; set; }
~~     = new(makeRepo.GetAll(), nameof(Make.Id), nameof(Make.Name));  [ViewData]~~
  public string Title => "Razor Syntax";
  [BindProperty]
  public Car Entity { get; set; }
  public async Task<IActionResult> OnGetAsync()
  {
    LookupValues = new(await makeDataService.GetAllAsync(), nameof(Make.Id), nameof(Make.Name));
    Entity = await dataService.FindAsync(6);
    return Page();
  }
}
```

Part 5: Update the MenuViewComponent

- Update the class to the following:

```
namespace AutoLot.Web.ViewComponents;

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);
  }
}
```

Summary

This lab updated the ASP.NET Core Razor Pages web application to use Data Services.