

.NET App Dev Hands-On Workshop

Blazor Lab 9 – API Services

This lab adds the API services to the AutoLot.Blazor project. Before starting this lab, you must have completed Blazor Lab 8 and the AutoLot API project.

Part 1: Add the ApiWrapper

Step 1: Add the API service settings view model

- Create a new class named `ApiServiceSettings.cs` in the `AutoLot.Blazor.Models.ViewModels` folder. Update the code to the following:

```
namespace AutoLot.Blazor.Models.ViewModels;

public class ApiServiceSettings
{
    public ApiServiceSettings() { }
    public string Uri { get; set; }
    public string CarBaseUri { get; set; }
    public string MakeBaseUri { get; set; }
    public int MajorVersion { get; set; }
    public int MinorVersion { get; set; }
    public string Status { get; set; }
    public string ApiVersion => $"{MajorVersion}.{MinorVersion}"
        + (!string.IsNullOrEmpty(Status) ? $"-{Status}" : string.Empty);
}
```

Step 1: Add the API service interfaces

- Create a new folder named `ApiWrapper` in the `Services` folder of the `AutoLot.Blazor` project. In this folder, add a new folder named `Interfaces`. Add another folder named `Base` in the `Interfaces` folder, and in that folder, add a new interface named `IApiServiceWrapperBase.cs`. Update the code to the following:

```
namespace AutoLot.Blazor.Services.ApiWrapper.Interfaces.Base;

public interface IApiServiceWrapperBase where TEntity : BaseEntity, new()
{
    Task<IList<TEntity>> GetAllEntitiesAsync();
    Task<TEntity> GetEntityAsync(int id);
    Task<TEntity> AddEntityAsync(TEntity entity);
    Task<TEntity> UpdateEntityAsync(TEntity entity);
    Task DeleteEntityAsync(TEntity entity);
}
```

- Add the following to the GlobalUsings.cs file in the AutoLot.Blazor project:

```
global using AutoLot.Blazor.Services.ApiWrapper;  
global using AutoLot.Blazor.Services.ApiWrapper.Interfaces;  
global using AutoLot.Blazor.Services.ApiWrapper.Interfaces.Base;  
global using Microsoft.Extensions.Options;  
global using System.Net.Http.Headers;  
global using System.Net.Http.Json;  
global using System.Text;  
global using System.Text.Json.Serialization;
```

- In the Interfaces folder, add two interface files: ICarApiServiceWrapper.cs, and IMakeApiServiceWrapper.cs. Update the code to the following listings:

```
//ICarApiServiceWrapper.cs  
namespace AutoLot.Blazor.Services.ApiWrapper.Interfaces;  
public interface ICarApiServiceWrapper : IApiServiceWrapperBase<Car>  
{  
    Task<IList<Car>> GetCarsByMakeAsync(int id);  
}
```

```
// IMakeApiServiceWrapper.cs  
namespace AutoLot.Blazor.Services.ApiWrapper.Interfaces;  
public interface IMakeApiServiceWrapper : IApiServiceWrapperBase<Make>  
{  
}
```

Step 2: Add the API service base implementation

- Create a new folder named Base in the ApiWrapper folder, and in that folder, create a new class file named ApiServiceWrapperBase.cs. Update the code to the following:

```
namespace AutoLot.Blazor.Services.ApiWrapper.Base;
public abstract class ApiServiceWrapperBase : IApiServiceWrapperBase
    where TEntity : BaseEntity, new()
{
    protected readonly HttpClient Client;
    private readonly string _endPoint;
    protected readonly ApiServiceSettings ApiSettings;
    protected readonly string ApiVersion;
    protected readonly JsonSerializerOptions JsonOptions = new JsonSerializerOptions
    {
        AllowTrailingCommas = true,
        PropertyNameCaseInsensitive = true,
        PropertyNamingPolicy = null,
        ReferenceHandler = ReferenceHandler.IgnoreCycles
    };
    protected ApiServiceWrapperBase( HttpClient client,
        IOptionMonitor<ApiServiceSettings> apiSettingsMonitor, string endPoint)
    {
        Client = client;
        _endPoint = endPoint;
        ApiSettings = apiSettingsMonitor.CurrentValue;
        client.BaseAddress = new Uri(ApiSettings.Uri);
        client.DefaultRequestHeaders.Accept.Add(
            new MediaTypeWithQualityHeaderValue("application/json"));
        ApiVersion = ApiSettings.ApiVersion;
    }
    internal async Task<HttpResponseMessage> PostAsJsonAsync(string uri, string json)
    {
        return await Client.PostAsync(uri, new StringContent(json, Encoding.UTF8,
            "application/json"));
    }
    internal async Task<HttpResponseMessage> PutAsJsonAsync(string uri, string json)
    {
        return await Client.PutAsync(uri, new StringContent(json, Encoding.UTF8, "application/json"));
    }
    internal async Task<HttpResponseMessage> DeleteAsJsonAsync(string uri, string json)
    {
        HttpRequestMessage request = new HttpRequestMessage
        {
            Content = new StringContent(json, Encoding.UTF8, "application/json"),
            Method = HttpMethod.Delete,
            RequestUri = new Uri(uri)
        };
        return await Client.SendAsync(request);
    }
}
```

```

public async Task<IList<TEntity>> GetAllEntitiesAsync()
{
    var response = await Client.GetAsync($"{ApiSettings.Uri}{_endPoint}?v={ApiVersion}");
    response.EnsureSuccessStatusCode();
    var result = await response.Content.ReadFromJsonAsync<IList<TEntity>>();
    return result;
}
public async Task<TEntity> GetEntityAsync(int id)
{
    var response = await Client.GetAsync($"{ApiSettings.Uri}{_endPoint}/{id}?v={ApiVersion}");
    response.EnsureSuccessStatusCode();
    var result = await response.Content.ReadFromJsonAsync<TEntity>();
    return result;
}
public async Task<TEntity> AddEntityAsync(TEntity entity)
{
    var response = await PostAsJsonAsync($"{ApiSettings.Uri}{_endPoint}?v={ApiVersion}",
        JsonSerializer.Serialize(entity, JsonOptions));
    if (response == null)
    {
        throw new Exception("Unable to communicate with the service");
    }
    var location = response.Headers?.Location?.OriginalString;
    return await response.Content.ReadFromJsonAsync<TEntity>() ?? await GetEntityAsync(entity.Id);
}
public async Task<TEntity> UpdateEntityAsync(TEntity entity)
{
    var response =
        await PutAsJsonAsync($"{ApiSettings.Uri}{_endPoint}/{entity.Id}?v={ApiVersion}",
            JsonSerializer.Serialize(entity, JsonOptions));
    response.EnsureSuccessStatusCode();
    return await response.Content.ReadFromJsonAsync<TEntity>() ?? await GetEntityAsync(entity.Id);
}
public async Task DeleteEntityAsync(TEntity entity)
{
    var response =
        await DeleteAsJsonAsync($"{ApiSettings.Uri}{_endPoint}/{entity.Id}?v={ApiVersion}",
            JsonSerializer.Serialize(entity, JsonOptions));
    response.EnsureSuccessStatusCode();
}
}

```

- Add the following to the GlobalUsings.cs file:

```
global using AutoLot.Blazor.Services.ApiWrapper.Base;
```

Step 3: Add the Car and Make API service implementations

- Create two new files named `CarApiServiceWrapper.cs` and `MakeApiServiceWrapper.cs` in the `ApiWrapper` folder and update the code to the following listings:

```
//CarApiServiceWrapper.cs
namespace AutoLot.Blazor.Services.ApiWrapper;

public class CarApiServiceWrapper(
    HttpClient client, IOptionsMonitor<ApiServiceSettings> apiSettingsMonitor)
    : ApiServiceWrapperBase<Car>(
        client, apiSettingsMonitor, apiSettingsMonitor.CurrentValue.CarBaseUri),
    ICarApiServiceWrapper
{
    public async Task<IList<Car>> GetCarsByMakeAsync(int id)
    {
        var response = await Client.GetAsync(
            $"{ApiSettings.Uri}{ApiSettings.CarBaseUri}/bymake/{id}?v={ApiVersion}");
        response.EnsureSuccessStatusCode();
        var result = await response.Content.ReadFromJsonAsync<IList<Car>>();
        return result;
    }
}

//MakeApiServiceWrrapper
namespace AutoLot.Blazor.Services.ApiWrapper;

public class MakeApiServiceWrapper(
    HttpClient client, IOptionsMonitor<ApiServiceSettings> apiSettingsMonitor)
    : ApiServiceWrapperBase<Make>(
        client, apiSettingsMonitor, apiSettingsMonitor.CurrentValue.MakeBaseUri),
    IMakeApiServiceWrapper;
```

Step 4: Add the Car and Make API Data service implementations

- Create two new files named CarApiDataService.cs and MakeApiDataService.cs in the Services folder and update the code to the following listings:

```
//CarApiDataService.cs
namespace AutoLot.Blazor.Services;

public class CarApiDataService(ICarApiServiceWrapper serviceWrapper) : ICarDataService
{
    internal Car CreateCleanCar(Car entity)
    {
        return new Car
        {
            Color = entity.Color,
            DateBuilt = entity.DateBuilt,
            Id = entity.Id,
            TimeStamp = entity.TimeStamp,
            IsDrivable = entity.IsDrivable,
            MakeId = entity.MakeId,
            PetName = entity.PetName,
            Price = entity.Price
        };
    }
    public async Task<Car> GetEntityAsync(int id) => await serviceWrapper.GetEntityAsync(id);
    public async Task<Car> AddEntityAsync(Car entity)
        => await serviceWrapper.AddEntityAsync(CreateCleanCar(entity));
    public async Task<Car> UpdateEntityAsync(int id, Car entity)
        => await serviceWrapper.UpdateEntityAsync(CreateCleanCar(entity));
    public async Task DeleteEntityAsync(Car entity)
    {
        await serviceWrapper.DeleteEntityAsync(CreateCleanCar(entity));
    }
    public async Task<List<Car>> GetAllEntitiesAsync()
        => (await serviceWrapper.GetAllEntitiesAsync()).ToList();
    public async Task<List<Car>> GetByMakeAsync(int makeId)
        => (await serviceWrapper.GetCarsByMakeAsync(makeId)).ToList();
}

// MakeApiDataService.cs
namespace AutoLot.Blazor.Services;
public class MakeApiDataService(IMakeApiServiceWrapper serviceWrapper) : IMakeDataService
{
    public async Task<Make> GetEntityAsync(int id) => await serviceWrapper.GetEntityAsync(id);
    public async Task<Make> AddEntityAsync(Make entity)
        => await serviceWrapper.AddEntityAsync(entity);
    public async Task<Make> UpdateEntityAsync(int id, Make entity)
        => await serviceWrapper.UpdateEntityAsync(entity);
    public async Task DeleteEntityAsync(Make entity)
    {
        await serviceWrapper.DeleteEntityAsync(entity);
    }
    public async Task<List<Make>> GetAllEntitiesAsync()
        => (await serviceWrapper.GetAllEntitiesAsync()).ToList();
}
```

Step 5: Update the AppSettings files and Program.cs

- Add the following to appsettings.Development.json and appsettings.Staging.json files (don't forget to add the comma after the DealerInfo object and update the port to your local service):
Note: In a real application, the values for staging and development would differ.

```
{
  "DealerInfo": {
    "DealerName": "Skimedic's Used Cars Staging Site",
    "City": "West Chester",
    "State": "Ohio"
  },
  "UseApi": true,
  "ApiServiceSettings": {
    "Uri": "https://localhost:5011/",
    "CarBaseUri": "api/Cars",
    "MakeBaseUri": "api/Makes",
    "MajorVersion": 1,
    "MinorVersion": 0,
    "Status": ""
  }
}
```

- In the Program.cs file, comment out the call to add the HTTP Client:

```
//builder.Services.AddScoped(
//  sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) });
```

- Add the following to register the option pattern and the API wrappers:

```
builder.Services.Configure<ApiServiceSettings>(
  builder.Configuration.GetSection(nameof(ApiServiceSettings)));
builder.Services.AddHttpClient<ICarApiServiceWrapper, CarApiServiceWrapper>();
builder.Services.AddHttpClient<IMakeApiServiceWrapper, MakeApiServiceWrapper>();
```

- Replace the calls to add the ICarDataService and the IMakeService with the following:

```
if (builder.Configuration.GetValue<bool>("UseApi"))
{
  builder.Services.AddScoped<ICarDataService, CarApiDataService>();
  builder.Services.AddScoped<IMakeDataService, MakeApiDataService>();
}
else
{
  builder.Services.AddScoped<ICarDataService, CarDataService>();
  builder.Services.AddScoped<IMakeDataService, MakeDataService>();
}
```

Summary

This lab completed the AutoLot.Blazor application.