.NET 10 App Dev Hands-On Lab

Razor Pages/MVC with API Lab 1a – API Data Services

This lab adds the HTTP Client factory for web applications to leverage the RESTful service. Before starting this lab, you must have completed Razor Pages/MVC Lab 9b and API Lab 4.

Part 1: Add the Api Service Wrapper

The API service wrapper handles the calls to the `AutoLot.Api` RESTful service.

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. 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.Services project.

Prompt: Add a directory named ApiWrapper to the AutoLot.Services project, and add a new directory
named Models to that directory. In the Models directory, add a new class named ApiServiceSettings
and update the code to the following:
namespace AutoLot.Services.ApiWrapper.Models;
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);
}

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.ApiWrapper;
global using AutoLot.Services.ApiWrapper.Models;
global using AutoLot.Services.ApiWrapper;
global using AutoLot.Services.ApiWrapper.Models;
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;
```

```
Prompt: Add a new directory named Interfaces under the ApiWrapper directory and a new directory
named Base into the Interfaces directory. Add a new interface named IApiServiceWrapperBase to the
Base directory. Update the code to the following:
namespace AutoLot.Services.ApiWrapper.Interfaces.Base;

Prompt: public interface IApiServiceWrapperBase<TEntity> 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);
}

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.ApiWrapper.Interfaces;
global using AutoLot.Services.ApiWrapper.Interfaces.Base;

Prompt: Add two new interfaces named ICarApiServiceWrapper and IMakeApiServiceWrapper to the
Interfaces directory in separate files. Both implement IApiServiceWrapper and should use the
appropriate entity class for the generic type (Car, Make). Add the following method to the
ICarApiServiceWrapper interface:
  Task<IList<Car>> GetCarsByMakeAsync(int id);

Prompt: Add a new directory named Base under the ApiWrapper directory. In that folder add a new
abstract class named ApiServiceWrapperBase, make it generic with TEntity constrained to
BaseEntity, new(). Implement the IApiServiceWrapperBase<TEntity> interface. Add the following
contructor and class level fields:
private readonly string _endPoint;
protected readonly HttpClient Client;
protected readonly ApiServiceSettings ApiSettings;
protected readonly string ApiVersion;
protected ApiServiceWrapperBase(
  HttpClient client,
  IOptionsSnapshot<ApiServiceSettings> apiSettingsSnapshot,
  string endPoint)
{
  Client = client;
  _endPoint = endPoint;
  ApiSettings = apiSettingsSnapshot.Value;
  client.BaseAddress = new Uri(ApiSettings.Uri);
  client.DefaultRequestHeaders.Accept.Add(
    new MediaTypeWithQualityHeaderValue("application/json"));
  ApiVersion = ApiSettings.ApiVersion;
}
```

```
Prompt: Add internal methods for Put, Post, and Delete operations:
internal async Task<HttpResponseMessage> PostAsJsonAsync(string uri, string json)
  => await Client.PostAsync(uri, new StringContent(json, Encoding.UTF8, "application/json"));
internal async Task<HttpResponseMessage> PutAsJsonAsync(string uri, string json)
  => 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);
}

Prompt: Implement the rest of the methods:
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($"{ApiSettings.Uri}{_endPoint}?v={ApiVersion}",
            JsonSerializer.Serialize(entity));
  if (response == null)
  {
    throw new Exception("Unable to communicate with the service");
  }
  var location = response.Headers?.Location?.OriginalString;
  return await response.Content.ReadFromJsonAsync<TEntity>() ?? await GetEntityAsync(entity.Id);
}
public async Task<TEntity> UpdateEntityAsync(TEntity entity)
{
  var response = await PutAsJsonAsync($"{ApiSettings.Uri}{_endPoint}/{entity.Id}?v={ApiVersion}",
  JsonSerializer.Serialize(entity));
  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));
  response.EnsureSuccessStatusCode();
}
```

```
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.ApiWrapper.Base;

Prompt: In the ApiWrapper directory, add a new class that is named CarApiServiceWrapper that
inherits from ApiServiceWrapperBase<Car> and implements ICarApiServiceWrapper.
Add the following action method:
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;
}

Prompt: In the ApiWrapper directory, add a new class that is named MakeApiServiceWrapper that
inherits from ApiServiceWrapperBase<Make> and implements IMakeApiServiceWrapper.

Prompt: Add a new directory named Configuration in the ApiWrapper directory, and in that
directory, add a class named ServiceConfiguration and update the code to the following:
namespace AutoLot.Services.ApiWrapper.Configuration;
public static class ServiceConfiguration
{
  public static IServiceCollection ConfigureApiServiceWrapper(
    this IServiceCollection services, IConfiguration config)
  {
    services.Configure<ApiServiceSettings>(config.GetSection(nameof(ApiServiceSettings)));
    services.AddHttpClient<ICarApiServiceWrapper, CarApiServiceWrapper>();
    services.AddHttpClient<IMakeApiServiceWrapper, MakeApiServiceWrapper>();
    return services;
    }
}

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.ApiWrapper.Configuration;
```

Manual

Step 1: Add the ApiServiceSettings Class

- Add a directory named `ApiWrapper` to the `AutoLot.Services` project, and add a new directory named Models to that directory. In the `Models` directory, add a new class named `ApiServiceSettings` and update the code to the following:

```
namespace AutoLot.Services.ApiWrapper.Models;

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);
}
```

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

```
global using AutoLot.Services.ApiWrapper;
global using AutoLot.Services.ApiWrapper.Models;
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;
```

Step 2: Add the Interfaces

- Add a new directory named `Interfaces` under the `ApiWrapper` directory and a new directory named `Base` into the `Interfaces` directory. Add a new interface named `IApiServiceWrapperBase` to the `Base` directory. Update the code to the following:

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

public interface IApiServiceWrapperBase<TEntity> 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` class:

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

- Add two new interfaces named `ICarApiServiceWrapper` and `IMakeApiServiceWrapper` to the `Interfaces` directory and update the code to the following:

```
// ICarApiServiceWrapper.cs
namespace AutoLot.Services.ApiWrapper.Interfaces;

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

public interface IMakeApiServiceWrapper : IApiServiceWrapperBase<Make> { }
```

Step 3: Add the ApiServiceWrapperBase Class

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

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

public abstract class ApiServiceWrapperBase<TEntity> : IApiServiceWrapperBase<TEntity>
  where TEntity : BaseEntity, new()
{
  //implementation goes here
}
```

- Add a constructor that takes an instance of `HttpClient`, `IOptionsSnapshot<ApiServiceSettings>`, and a `string` for the derived class's endpoint and assign them to class-level fields. In the constructor, configure the `HttpClient` and get the `API` version from the settings:

```
private readonly string _endPoint;
protected readonly HttpClient Client;
protected readonly ApiServiceSettings ApiSettings;
protected readonly string ApiVersion;
protected ApiServiceWrapperBase(
  HttpClient client,
  IOptionsSnaphot<ApiServiceSettings> apiSettingsSnapshot,
  string endPoint)
{
  Client = client;
  _endPoint = endPoint;
  ApiSettings = apiSettingsSnapshot.Value;
  client.BaseAddress = new Uri(ApiSettings.Uri);
  client.DefaultRequestHeaders.Accept.Add(
    new MediaTypeWithQualityHeaderValue("application/json"));
  ApiVersion = ApiSettings.ApiVersion;
}
```

- Implement three internal helper methods for Put, Post, and Delete operations:

```
internal async Task<HttpResponseMessage> PostAsJsonAsync(string uri, string json)
  => await Client.PostAsync(uri, new StringContent(json, Encoding.UTF8, "application/json"));
internal async Task<HttpResponseMessage> PutAsJsonAsync(string uri, string json)
  => 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);
}
```

- Implement the Get methods:

```
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;
}
```

- Implement the `Add` method:

```
public async Task<TEntity> AddEntityAsync(TEntity entity)
{
  var response = await PostAsJsonAsync($"{ApiSettings.Uri}{_endPoint}?v={ApiVersion}",
            JsonSerializer.Serialize(entity));
  if (response == null)
  {
    throw new Exception("Unable to communicate with the service");
  }
  return await response.Content.ReadFromJsonAsync<TEntity>() ?? await GetEntityAsync(entity.Id);
}
```

- Implement the Update method:

```
public async Task<TEntity> UpdateEntityAsync(TEntity entity)
{
  var response = await PutAsJsonAsync($"{ApiSettings.Uri}{_endPoint}/{entity.Id}?v={ApiVersion}",
  JsonSerializer.Serialize(entity));
  response.EnsureSuccessStatusCode();
  return await response.Content.ReadFromJsonAsync<TEntity>() ?? await GetEntityAsync(entity.Id);
}
```

- Implement the Delete method:

```
public async Task DeleteEntityAsync(TEntity entity)
{
  var response =
    await DeleteAsJsonAsync($"{ApiSettings.Uri}{_endPoint}/{entity.Id}?v={ApiVersion}",
  JsonSerializer.Serialize(entity));
  response.EnsureSuccessStatusCode();
}
```

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

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

Step 4: Add the CarApiServiceWrapper Class

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

```
namespace AutoLot.Services.ApiWrapper;

public class CarApiServiceWrapper(HttpClient client,
  IOptionsSnapshot<ApiServiceSettings> apiSettingsSnapshot)
  : ApiServiceWrapperBase<Car>(client, apiSettingsSnapshot,
      apiSettingsSnapshot.Value.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;
  }
}
```

Step 5: Add the MakeApiServiceWrapper Class

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

```
namespace AutoLot.Services.ApiWrapper;
public class MakeApiServiceWrapper(HttpClient client,
  IOptionsSnapshot<ApiServiceSettings> apiSettingsSnapshot)
    : ApiServiceWrapperBase<Make>(client, apiSettingsSnapshot,
        apiSettingsSnapshot.Value.MakeBaseUri), IMakeApiServiceWrapper;
```

Step 6: Add the Configuration Extension Method

- Add a new directory named `Configuration` in the `ApiWrapper` directory, and in that directory, add a class named `ServiceConfiguration` and update the code to the following:

```
namespace AutoLot.Services.ApiWrapper.Configuration;
public static class ServiceConfiguration
{
  public static IServiceCollection ConfigureApiServiceWrapper(
    this IServiceCollection services, IConfiguration config)
  {
    services.Configure<ApiServiceSettings>(config.GetSection(nameof(ApiServiceSettings)));
    services.AddHttpClient<ICarApiServiceWrapper, CarApiServiceWrapper>();
    services.AddHttpClient<IMakeApiServiceWrapper, MakeApiServiceWrapper>();
    return services;
    }
}
```

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

```
global using AutoLot.Services.ApiWrapper.Configuration;
```

Part 2: Add the API Data Service Classes

The API data services will encapsulate CRUD calls for `AutoLot.Api` service wrapper.

Copilot Agent Mode

```
Prompt: Add a directory named Api under the DataServices directory in the AutoLot.Services
project. Add a new directory named Base under the new Api directory. In that folder, add a new
class named ApiDataServiceBase and update the code to the following:
namespace AutoLot.Services.DataServices.Api.Base;
public abstract class ApiDataServiceBase<TEntity>(
  IAppLogging appLogging, IApiServiceWrapperBase<TEntity> serviceWrapperBase)
    : IDataServiceBase<TEntity> where TEntity : BaseEntity, new()
{
  protected readonly IApiServiceWrapperBase<TEntity> ServiceWrapper = serviceWrapperBase;
  protected readonly IAppLogging AppLoggingInstance = appLogging;
  public async Task<IQueryable<TEntity>> GetAllAsync()
    => await ServiceWrapper.GetAllEntitiesAsync();
  public async Task<TEntity> FindAsync(int id)
    => await ServiceWrapper.GetEntityAsync(id);
  public async Task<TEntity> UpdateAsync(TEntity entity, bool persist = true)
    => await ServiceWrapper.UpdateEntityAsync(entity);
  public async Task DeleteAsync(TEntity entity, bool persist = true)
    => await ServiceWrapper.DeleteEntityAsync(entity);
  public async Task<TEntity> AddAsync(TEntity entity, bool persist = true)
    => await ServiceWrapper.AddEntityAsync(entity);
}
```

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

Prompt: Add a new class named CarApiDataService in the DataServices\Api directory and update the
code to the following:
namespace AutoLot.Services.DataServices.Api;
public class CarApiDataService(
  IAppLogging appLogging, ICarApiServiceWrapper serviceWrapper)
    : ApiDataServiceBase<Car>(appLogging, serviceWrapper), ICarDataService
{
  public async Task<IQueryable<Car>> GetAllByMakeIdAsync(int? makeId)
    => makeId.HasValue
       ? await ((ICarApiServiceWrapper)ServiceWrapper).GetCarsByMakeAsync(makeId.Value)
       : await GetAllAsync();
}

Prompt: Add a new class named MakeApiDataService in the DataServices\Api directory and update the
code to the following:
namespace AutoLot.Services.DataServices.Api;
public class MakeApiDataService(  IAppLogging appLogging, IMakeApiServiceWrapper serviceWrapper)
  : ApiDataServiceBase<Make>(appLogging, serviceWrapper), IMakeDataService;
```

Manual

Step 1: Add the ApiDataServiceBase Class

- Add a directory named `Api` under the `DataServices` directory in the `AutoLot.Services` project. Add a new directory named `Base` under the new `Api` directory. In that folder, add a new class named `ApiDataServiceBase` and update the code to the following:

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

public abstract class ApiDataServiceBase<TEntity>(
    IAppLogging appLogging, IApiServiceWrapperBase<TEntity> serviceWrapperBase)
    : IDataServiceBase<TEntity> where TEntity : BaseEntity, new()
{
  protected readonly IApiServiceWrapperBase<TEntity> ServiceWrapper = serviceWrapperBase;
  protected readonly IAppLogging AppLoggingInstance = appLogging;
  public async Task<IQueryable<TEntity>> GetAllAsync()
    => await ServiceWrapper.GetAllEntitiesAsync();
  public async Task<TEntity> FindAsync(int id)
    => await ServiceWrapper.GetEntityAsync(id);
  public async Task<TEntity> UpdateAsync(TEntity entity, bool persist = true)
    => await ServiceWrapper.UpdateEntityAsync(entity);
  public async Task DeleteAsync(TEntity entity, bool persist = true)
    => await ServiceWrapper.DeleteEntityAsync(entity);
  public async Task<TEntity> AddAsync(TEntity entity, bool persist = true)
    => await ServiceWrapper.AddEntityAsync(entity);
}
```

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

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

Step 2: Add the CarApiDataService Class

- Add a new class named `CarApiDataService` in the `DataServices\Api` directory and update the code to the following:

```
namespace AutoLot.Services.DataServices.Api;

public class CarApiDataService(
  IAppLogging appLogging, ICarApiServiceWrapper serviceWrapper)
    : ApiDataServiceBase<Car>(appLogging, serviceWrapper), ICarDataService
{
  public async Task<IQueryable<Car>> GetAllByMakeIdAsync(int? makeId)
    => makeId.HasValue
       ? await ((ICarApiServiceWrapper)ServiceWrapper).GetCarsByMakeAsync(makeId.Value)
       : await GetAllAsync();
}
```

Step 3: Add the MakeApiDataService Class

- Add a new class named `MakeApiDataService` in the `DataServices\Api` directory and update the code to the following:

```
namespace AutoLot.Services.DataServices.Api;

public class MakeApiDataService(
  IAppLogging appLogging, IMakeApiServiceWrapper serviceWrapper)
  : ApiDataServiceBase<Make>(appLogging, serviceWrapper), IMakeDataService;
```

Summary

This lab added `ApiDataServices`.

Next Steps

The following lab will update the web application to use the new data services.