

# .NET 10 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

#### 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. If there is a GlobalUsings.cs file, 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. The @code block in razor components should be at the bottom of the file. All work is to be done in the AutoLot.Blazor project unless otherwise specified.

Prompt: Create a new public class named ApiServiceSettings.cs in the AutoLot.Blazor.Models.ViewModels folder. Add the following public properties:

```
Uri (string)
CarBaseUri (string)
MakeBaseUri (string)
MajorVersion (int)
MinorVersion (int)
Status (string)
ApiVersion (string) - read only, returns a string in the format "MajorVersion.MinorVersion-
Status" (omit the -Status if Status is null or empty)
```

Prompt: Create a new ApiWrapper folder 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 generic interface named IApiServiceWrapperBase.cs typed to TEntity and constrained to BaseEntity, new(). Add the following method signatures to the interface:

```
Task<IList< TEntity >> GetAllEntitiesAsync(input: none ; return: Task< List< TEntity >>);
Task< TEntity > GetEntityAsync(input: int id; return: Task< TEntity >);
Task< TEntity > AddEntityAsync(input: TEntity entity; return: Task< TEntity >);
Task< TEntity > UpdateEntityAsync(input: TEntity entity; return: Task< TEntity >);
Task DeleteEntityAsync(input: TEntity entity; return: Task);
```

Prompt: Add the following global usings to the GlobalUsings.cs file if it does not already exist (sorted alphabetically. Don't remove any existing global using statements).

```
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;
```

Prompt: In the Interfaces folder, add two interface files: ICar ApiServiceWrapper.cs, and IMake ApiServiceWrapper.cs.

Implement IApiServiceWrapperBase, passing in Make and Car as the type. Add the following method to the ICar ApiServiceWrapper interface:

```
GetCarsByMakeAsync(input: int id; return: Task<List<Car>>);
```

Prompt: Create a new folder named Base in the ApiWrapper folder, and in that folder, create a new abstract class file named ApiServiceWrapperBase.cs

that implements IApiServiceWrapperBase where TEntity is constrained to BaseEntity, new().

Add the following fields:

```
Client (HttpClient, protected, readonly)
_endpoint (string, private, readonly)
ApiSettings ( ApiServiceSettings, protected, readonly)
ApiVersion (string, protected, readonly)
```

Prompt: Also create a protected readonly field named JsonOptions (JsonSerializerOptions) initialized to a new instance with the following values:

```
AllowTrailingCommas = true,
PropertyNameCaseInsensitive = true,
PropertyNamingPolicy = null,
ReferenceHandler = ReferenceHandler.IgnoreCycles
```

Prompt: Create a constructor that takes an HttpClient (client),  
IOptionsSnapshot< ApiServiceSettings> (apiSettingsSnapshot) and a string (endpoint)  
and initializes the fields like this:

```
_endPoint = endPoint;
ApiSettings = apiSettingsSnapshot.Value;
Client = client;
client.BaseAddress = new Uri(ApiSettings.Uri);
client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
ApiVersion = ApiSettings.ApiVersion;
```

Prompt: Add the following internal methods:

```
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)
{
    HttpResponseMessage request = new HttpResponseMessage
    {
        Content = new StringContent(json, Encoding.UTF8, "application/json"),
        Method = HttpMethod.Delete,
        RequestUri = new Uri(uri)
    };
    return await Client.SendAsync(request);
}
```

Prompt: Add the public methods to Get, Add, Update, and Delete entities:

```
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);
}
```

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

Prompt: Add the following global usings to the GlobalUsings.cs file if it does not already exist (sorted alphabetically. Don't remove any existing global using statements).

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

Prompt: Create a new file named Car ApiServiceWrapper.cs in the ApiWrapper folder, inherit from ApiServiceWrapperBase with type equal Car,

and implement ICar ApiServiceWrapper. For the GetCarsByMakeAsync, use the following:

```
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: Create a new file named Make ApiServiceWrapper.cs in the ApiWrapper folder, inherit from ApiServiceWrapperBase with type equal Make,

and implement ICar ApiServiceWrapper.

Prompt: Create a new file named Car ApiServiceData Service.cs in the Services folder that takes an instance of ICar ApiServiceWrapper (serviceWrapper)

in the primary constructor and implements ICar DataService.

Create an internal method named CreateCleanCar that takes a Car (entity) as a parameter and returns a new Car instance that removes the navigation properties.

Implement all of the interface methods by calling the corresponding method on the serviceWrapper instance making sure to call CreateCleanCar where appropriate to avoid sending navigation properties to the API.

Prompt: Create a new file named Make ApiServiceData Service.cs in the Services folder that takes an instance of IMake ApiServiceWrapper (serviceWrapper)

in the primary constructor and implements IMake DataService.

Create an internal method named CreateCleanMake that takes a Make (entity) as a parameter and returns a new Make instance that removes the navigation properties.

Implement all of the interface methods by calling the corresponding method on the serviceWrapper instance making sure to call CreateCleanMake where appropriate to avoid sending navigation properties to the API.

Prompt: Remove the following from Program.cs if it exists:

```
builder.Services.AddScoped(
    sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) });
builder.Services.AddScoped<ICarDataService, CarDataService>();
builder.Services.AddScoped<IMakeDataService, MakeDataService>();
```

Prompt: Add the following to Program.cs to register the new services (don't remove any existing entries):

```
builder.Services.Configure<ApiServiceSettings>(
    builder.Configuration.GetSection(nameof(ApiServiceSettings)));
builder.Services.AddHttpClient<ICar ApiServiceWrapper, Car ApiServiceWrapper>();
builder.Services.AddHttpClient<IMake ApiServiceWrapper, Make ApiServiceWrapper>();
if (builder.Configuration.GetValue<bool>("UseApi"))
{
    builder.Services.AddScoped<ICar DataService, Car ApiServiceWrapper>();
    builder.Services.AddScoped<IMake DataService, Make ApiServiceWrapper>();
}
else
{
    builder.Services.AddScoped<ICar DataService, Car DataService>();
    builder.Services.AddScoped<IMake DataService, Make DataService>();
}
```

Prompt: Add the following to appsettings.Development.json and appsettings.Staging.json files:

```
"UseApi": true,
"ApiServiceSettings": {
    "Uri": "https://localhost:5011/",
    "CarBaseUri": "api/Cars",
    "MakeBaseUri": "api/Makes",
    "MajorVersion": 1,
    "MinorVersion": 0,
    "Status": ""}
```

# Manual

## 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 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 => string.IsNullOrWhiteSpace(Status)
        ? $"{MajorVersion}.{MinorVersion}"
        : $"{MajorVersion}.{MinorVersion}-{Status}";
}
```

## Step 2: Add the API service interfaces

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

```
namespace AutoLot.Blazor.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` 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: **ICar ApiServiceWrapper.cs**, and **IMake ApiServiceWrapper.cs**. Update the code to the following listings:

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

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

### Step 3: Add the API service base implementation

- Create a new folder named **Base** in the **ApiWrapper** folder, and in that folder, create a new abstract class file named **ApiServiceWrapperBase.cs**. Add fields to hold the **HttpClient**, **endpoint**, **ApiServiceSettings** instance, the **ApiVersion**, the **JsonOptions**, and the protected constructor:

```
namespace AutoLot.Blazor.Services.ApiWrapper.Base;
public abstract class ApiServiceWrapperBase<TEntity> : IApiServiceWrapperBase<TEntity>
    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,
        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;
    }
    //remaining implementation goes here
}
```

- Add three internal methods to execute Post, Put, and Delete calls to the endpoint:

```
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);
}
```

- Add the public methods to Get, Add, Update, and Delete entities:

```
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 4: Add the Car and Make API service implementations

- Create two new files named Car ApiServiceWrapper.cs and Make ApiServiceWrapper.cs in the ApiWrapper folder and update the code to the following listings:

```
//Car ApiServiceWrapper.cs
namespace AutoLot.Blazor.Services.ApiWrapper;

public class Car ApiServiceWrapper(
    HttpClient client, IOptionsMonitor<ApiServiceSettings> apiSettingsMonitor)
    : ApiServiceWrapperBase<Car>(
        client, apiSettingsMonitor, apiSettingsMonitor.CurrentValue.CarBaseUri),
    ICar ApiServiceWrapper
{
    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;
    }
}

//Make ApiServiceWrrapper
namespace AutoLot.Blazor.Services.ApiWrapper;

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

## Step 5: 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
        {
            Id = entity.Id,
            Color = entity.Color,
            DateBuilt = entity.DateBuilt,
            IsDrivable = entity.IsDrivable,
            MakeId = entity.MakeId,
            PetName = entity.PetName,
            Price = entity.Price,
            TimeStamp = entity.TimeStamp
        };
    }
    public async Task<Car> GetEntityAsync(int id) => await serviceWrapper.GetEntityAsync(id);
    public async Task<List<Car>> GetAllEntitiesAsync()
        => await serviceWrapper.GetAllEntitiesAsync();
    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>> GetByMakeAsync(int makeId)
        => (await serviceWrapper.GetCarsByMakeAsync(makeId)).ToList();
}
// MakeApiDataService.cs
namespace AutoLot.Blazor.Services;
public class MakeApiDataService(IMakeApiServiceWrapper serviceWrapper) : IMakeDataService
{
    internal Make CreateCleanMake(Make entity) => new()
    {
        Id = entity.Id,
        Name = entity.Name,
        TimeStamp = entity.TimeStamp
    };
    public async Task<Make> GetEntityAsync(int id) => await serviceWrapper.GetEntityAsync(id);
    public async Task<IEnumerable<Make>> GetAllEntitiesAsync()
        => await serviceWrapper.GetAllEntitiesAsync();
    public async Task<Make> AddEntityAsync(Make entity)
        => await serviceWrapper.AddEntityAsync(CreateCleanMake(entity));
    public async Task<Make> UpdateEntityAsync(Make entity)
        => await serviceWrapper.UpdateEntityAsync(CreateCleanMake(entity));
    public async Task DeleteEntityAsync(Make entity)
        => await serviceWrapper.DeleteEntityAsync(CreateCleanMake(entity));
}
```

## Step 6: 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 (or delete) 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<ICar ApiServiceWrapper, Car ApiServiceWrapper>();
builder.Services.AddHttpClient<IMake ApiServiceWrapper, Make ApiServiceWrapper>();
```

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

```
builder.Services.AddScoped<ICar DataService, Car DataService>();
builder.Services.AddScoped<IMake DataService, Make DataService>();
if (builder.Configuration.GetValue<bool>("UseApi"))
{
  builder.Services.AddScoped<ICar DataService, Car Api DataService>();
  builder.Services.AddScoped<IMake DataService, Make Api DataService>();
}
else
{
  builder.Services.AddScoped<ICar DataService, Car DataService>();
  builder.Services.AddScoped<IMake DataService, Make DataService>();
}
```

## Part 2: Error Handling

### Step 1: Add the ErrorBoundary to the NavMenu

- Replace the MakesSubMenu component with the following ErrorBoundary component, placing the MakesSubMenu as the ChildContent, and an error message in the ErrorContent:

```
<ErrorBoundary>
  <ChildContent>
    <MakesSubMenu></MakesSubMenu>
  </ChildContent>
  <ErrorContent>
    <div class="text-danger px-3">
      <span class="fa-solid fa-bomb pe-2" aria-hidden="true"></span>Unable to load Makes menu
    </div>
  </ErrorContent>
</ErrorBoundary>
```

### Step 2: Conditional Error Messages based on Environment

- Add the following to the \_Imports.razor:

```
@using Microsoft.AspNetCore.Components.WebAssembly.Hosting
```

- Add the following to the @code block in the Index.razor page in the Pages\Cars folder:

```
[Inject] IWebAssemblyHostEnvironment HostEnv { get; set; }
private string _errorMessage = string.Empty;
```

- Surround the "Loading" block with the following if ... else block:

```
if (string.IsNullOrEmpty(_errorMessage))
{
  <div>
    <em>Loading...</em>
  </div>
}
else
{
  <div class="text-warning"> @_errorMessage </div>
}
```

- Surround the call to the service with a `try...catch` and update the error message based on the environment:

```
try
{
    _cars = MakeId is > 0
        ? await CarDataServiceInstance.GetByMakeAsync(MakeId.Value)
        : await CarDataServiceInstance.GetAllEntitiesAsync();
}
catch (Exception ex)
{
    _errorMessage = HostEnv.IsDevelopment()
        ? $"{ex.Message}<br/>{ex.StackTrace}"
        : "An error occurred loading the inventory.";
    Console.WriteLine(ex);
}
```

## Part 3: Testing the Application Updates

- To test the application using Visual Studio, set both the `AutoLot.API` and the `AutoLot.Blazor` projects as start-up projects. In VS Code, type `dotnet run` for each project. To test error handling, start the Blazor app only.

## Summary

This lab completed the `AutoLot.Blazor` application.