Build an ASP.NET Core Service and App with .NET (Core) 5.0 Two-Day Hands-On Lab

Lab 11

This lab builds on the default Swagger implementation in the RESTful service. Prior to starting this lab, you must have completed Lab 10. This entire lab works on the `AutoLot.Api` project.

Part 1: Update the Default Swagger Configuration

Step 1: Update the call in Configure Services

- In the `ConfigureServices` method of the `Startup.cs` file, locate the call to `AddSwaggerGen` and update the code to the following:

```
services.AddSwaggerGen(c =>
{
  c.SwaggerDoc("v1",
    new OpenApiInfo
    {
      Title = "AutoLot Service",
      Version = "v1",
      Description = "Service to support the AutoLot sample eCommerce site",
      License = new OpenApiLicense
      {
        Name = "Skimedic Inc",
        Url = new Uri("http://www.skimedic.com")
      }
    });
  var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
  var xmlPath = Path.Combine(AppContext.BaseDirectory,xmlFile);
  c.IncludeXmlComments(xmlPath);
});
```

Step 2: Configure the application to use Swagger

- In the Configure method, move the calls to swagger out of the development section and into the main section:

```
if (env.IsDevelopment())
{
...
  //app.UseSwagger();
  //app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "AutoLot.Api v1"));
...
}
app.UseSwagger();
app.UseSwaggerUI(c => { c.SwaggerEndpoint("/swagger/v1/swagger.json", "AutoLot Service v1"); });
```

Step 3: Configure the project to generate the XML documentation

- Edit the `AutoLot.Api.csproj` file to add the following node (1591 and 1573 removes warnings for no /// comments or missing parameters):

```
<PropertyGroup>
  <DocumentationFile>AutoLot.Api.xml</DocumentationFile>
  <NoWarn>1701;1702;1591;1573</NoWarn>
</PropertyGroup>
<Target Name="PrepublishScript" BeforeTargets="PrepareForPublish">
  <ItemGroup>
    <DocFile Include="*.xml" />
  </ItemGroup>
  <Copy SourceFiles="@(DocFile)" DestinationFolder="$(PublishDir)" SkipUnchangedFiles="false" />
</Target>

<ItemGroup>
  <None Update="AutoLot.Api.xml">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  </None>
</ItemGroup>
```

Part 2: Update the Controllers with Documentation

Step 1: Update the Base Crud Controller

- Header for `GetAll`:

```
/// <summary>
/// Gets all records
/// </summary>
/// <returns>All records</returns>
/// <response code="200">Returns all items</response>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[HttpGet]
public ActionResult<IEnumerable<T>> GetAll()
```

- Header for `GetOne`:

```
/// <summary>
/// Gets a single record
/// </summary>
/// <param name="id">Primary key of the record</param>
/// <returns>Single record</returns>
/// <response code="200">Found the record</response>
/// <response code="204">No content</response>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[HttpGet("{id}")]
public ActionResult<T> GetOne(int id)
```


- Header for `UpdateOne`:

```
/// <summary>
/// Updates a single record
/// </summary>
/// <remarks>
/// Sample body:
/// <pre>
/// {
///   "Id": 1,
///   "TimeStamp": "AAAAAAAAB+E="
///   "MakeId": 1,
///   "Color": "Black",
///   "PetName": "Zippy",
///   "MakeColor": "VW (Black)",
/// }
/// </pre>
/// </remarks>
/// <param name="id">Primary key of the record to update</param>
/// <returns>Single record</returns>
/// <response code="200">Found and updated the record</response>
/// <response code="400">Bad request</response>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[HttpPut("{id}")]
public IActionResult UpdateOne(int id,T entity)
```

- Header for `AddOne`:

```
/// <summary>
/// Adds a single record
/// </summary>
/// <remarks>
/// Sample body:
/// <pre>
/// {
///   "Id": 1,
///   "TimeStamp": "AAAAAAAAB+E="
///   "MakeId": 1,
///   "Color": "Black",
///   "PetName": "Zippy",
///   "MakeColor": "VW (Black)",
/// }
/// </pre>
/// </remarks>
/// <returns>Added record</returns>
/// <response code="201">Found and updated the record</response>
/// <response code="400">Bad request</response>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[HttpPost]
public ActionResult<T> AddOne(T entity)
```


- Header for `DeleteOne`:

```
/// <summary>
/// Deletes a single record
/// </summary>
/// <remarks>
/// Sample body:
/// <pre>
/// {
///   "Id": 1,
///   "TimeStamp": "AAAAAAAAB+E="
/// }
/// </pre>
/// </remarks>
/// <returns>Nothing</returns>
/// <response code="200">Found and deleted the record</response>
/// <response code="400">Bad request</response>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[HttpDelete("{id}")]
public ActionResult<T> DeleteOne(int id, T entity)
```

Step 2: Update the CarsController

- Header for `GetCarsByMake`:

```
/// <summary>
/// Gets all cars by make
/// </summary>
/// <returns>All cars for a make</returns>
/// <param name="id">Primary key of the make</param>
/// <response code="200">Returns all cars by make</response>
/// <response code="200">Returns all cars by make</response>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[HttpGet("bymake/{id?}")]
public ActionResult<IEnumerable<Car>> GetCarsByMake(int? id)
```

Summary

This lab configured `Swagger` and `SwaggerUI` for the service.

Next steps

In the next part of this tutorial series, you will add Docker support and a Docker-Compose orchestration.