

.NET App Dev Hands-On Workshop

API Lab 4 – Versioning, Swagger

This lab adds versioning support first and then builds on the default Swagger implementation to enhance the documentation and support versioning. Prior to starting this lab, you must have completed MVC Lab 3.

Part 1: Add Versioning

Step 1: Add the ApiVersion Attribute

- Add version 1.0 to each BaseCrudController action method:

```
[ApiVersion("1.0")]
[HttpGet]
public ActionResult<IEnumerable<TEntity>> GetAll()

[ApiVersion("1.0")]
[HttpGet("{id}")]
public ActionResult<TEntity> GetOne(int id)

[ApiVersion("1.0")]
[HttpPut("{id}")]
public IActionResult UpdateOne(int id, TEntity entity)

[ApiVersion("1.0")]
[HttpPost]
public ActionResult<TEntity> AddOne(TEntity entity)

[ApiVersion("1.0")]
[HttpDelete("{id}")]
public ActionResult<TEntity> DeleteOne(int id, TEntity entity)
```

- Add a deprecated version of GetAll named GetAllBad:

```
[ApiVersion("0.5", Deprecated = true)]
[HttpGet]
public ActionResult<IEnumerable<TEntity>> GetAllBad()
{
    throw new Exception("I said not to use this one");
}
```

- Add a beta version of GetAll named GetAllFuture

```
[ApiVersion("2.0-Beta")]
[HttpGet]
public ActionResult<IEnumerable<TEntity>> GetAllFuture()
{
    throw new NotImplementedException("I'm working on it");
}
```

- Add version 1.0 to the GetCarsByMake action method in the CarsController:

```
[ApiVersion("1.0")]
[HttpGet("bymake/{id?}")]
public ActionResult<IEnumerable<Car>> GetCarsByMake(int? id)
```

Step 2: Add an additional route to the base controller for URL segment versioning

```
[ApiController]
[Route("api/[controller]")]
[Route("api/v{version:apiVersion}/{controller}")]
public abstract class BaseCrudController<TEntity, TController> : ControllerBase
    where TEntity : BaseEntity, new()
    where TController : class
```

Step 3: Update the WeatherForecastController

- Update the original route, add an additional route, and add the Version Neutral attribute:

```
[ApiVersionNeutral]
[ApiController]
[Route("api/[controller]")]
[Route("api/v{version:apiVersion}/{controller}")]
public class WeatherForecastController : ControllerBase
```

- Remove the named route for the Get action method:

```
[HttpGet]
public IEnumerable<WeatherForecast> Get()
```

Step 4: Update the Global Using Statements

- Add the following to GlobalUsings.cs:

```
global using Asp.Versioning.ApplicationModels;
global using Microsoft.Extensions.DependencyInjection.Extensions;
```

Step 5: Add and Use the Versioning Extension Method

- Create a new directory named `ApiVersionSupport` in the `AutoLot.Api` root directory. Add a new public static class named `ApiVersionConfiguration` and update the code to the following:

```
namespace AutoLot.Api.ApiVersionSupport;

public static class ApiVersionConfiguration
{
    public static IServiceCollection AddAutoLotApiVersionConfiguration(
        this IServiceCollection services, ApiVersion defaultVersion = null)
    {
        services.AddProblemDetails();
        defaultVersion ??= ApiVersion.Default;
        services.AddApiVersioning(options =>
        {
            //Set Default version
            options.DefaultApiVersion = defaultVersion;
            options.AssumeDefaultVersionWhenUnspecified = true;
            // reporting api versions will return the headers "api-supported-versions"
            // and "api-deprecated-versions"
            options.ReportApiVersions = true;
            //This combines all available option and adds "v" and "api-version"
            // for query string, header, or media type versioning
            // NOTE: In a real application, pick one method, not all of them
            options.ApiVersionReader = ApiVersionReader.Combine(
                new UrlSegmentApiVersionReader(),
                new QueryStringApiVersionReader(), //defaults to "api-version"
                new QueryStringApiVersionReader("v"),
                new HeaderApiVersionReader("api-version"),
                new HeaderApiVersionReader("v"),
                new MediaTypeApiVersionReader(), //defaults to "v"
                new MediaTypeApiVersionReader("api-version")
            );
        })
        .AddMvc()
        .AddApiExplorer(options =>
        {
            options.DefaultApiVersion = defaultVersion;
            options.AssumeDefaultVersionWhenUnspecified = true;
            // note: the specified format code will format the version as "'v'major[.minor][-status]"
            options.GroupNameFormat = "'v'VVV";
            // note: this option is only necessary when versioning by url segment.
            // the SubstitutionFormat can also be used to control the format of the
            // API version in route templates
            options.SubstituteApiVersionInUrl = true;
        });
        //Only apply versioning to [ApiController] controllers
        services.TryAddEnumerable(
            ServiceDescriptor.Transient<IApiControllerSpecification, ApiBehaviorSpecification>());
        return services;
    }
}
```

- Add the following global using statement to GlobalUsings.cs:

```
global using AutoLot.Api.ApiVersionSupport;
```

- Add the call to AddAutoLotApiVersionSupport to Program.cs:

```
builder.Services.AddAutoLotApiVersionConfiguration(new ApiVersion(1, 0));
```

Part 2: Create the Swagger Infrastructure

Step 1: Add the Swagger Application Settings class

- Create a new directory named Swagger in the AutoLot.Api project. In that directory, add a new directory named Models. Add a new class named SwaggerApplicationSettings in that directory. Update the code to the following:

```
namespace AutoLot.Api.Swagger.Models;
```

```
public class SwaggerApplicationSettings
{
    public string Title { get; set; }
    public List<SwaggerVersionDescription> Descriptions { get; set; } = [];
    public string ContactName { get; set; }
    public string ContactEmail { get; set; }
    public class SwaggerVersionDescription
    {
        public int MajorVersion { get; set; }
        public int MinorVersion { get; set; }
        public string Status { get; set; }
        public string Description { get; set; }
    }
}
```

- Add the following global using statement to GlobalUsings.cs:

```
global using AutoLot.Api.Swagger;
global using AutoLot.Api.Swagger.Models;
```

Step 2: Update the Application Settings

- Update the appsettings.json to the following:

```
{
  "AllowedHosts": "*",
  "SwaggerApplicationSettings": {
    "Title": "AutoLot APIs",
    "Descriptions": [
      {
        "MajorVersion": 0,
        "MinorVersion": 0,
        "Status": "",
        "Description": "Unable to obtain version description."
      },
      {
        "MajorVersion": 0,
        "MinorVersion": 5,
        "Status": "",
        "Description": "Deprecated Version 0.5"
      },
      {
        "MajorVersion": 1,
        "MinorVersion": 0,
        "Status": "",
        "Description": "Version 1.0"
      },
      {
        "MajorVersion": 2,
        "MinorVersion": 0,
        "Status": "",
        "Description": "Version 2.0"
      },
      {
        "MajorVersion": 2,
        "MinorVersion": 0,
        "Status": "Beta",
        "Description": "Version 2.0-Beta"
      }
    ],
    "ContactName": "Phil Japikse",
    "ContactEmail": "blog@skimedic.com"
  }
}
```

Step 3: Create the custom operation filter

- Create a new class named `SwaggerDefaultValues` in the Swagger directory and update the code to match the following:

```
namespace AutoLot.Api.Swagger;

public class SwaggerDefaultValues : IOperationFilter
{
    public void Apply(OpenApiOperation operation, OperationFilterContext context)
    {
        var apiDescription = context.ApiDescription;
        operation.Deprecated |= apiDescription.IsDeprecated();
        foreach (var responseType in context.ApiDescription.SupportedResponseTypes)
        {
            var responseKey = responseType.IsDefaultResponse
                ? "default"
                : responseType.StatusCode.ToString();
            var response = operation.Responses[responseKey];
            foreach (var contentType in response.Content.Keys)
            {
                if (responseType.ApiResponseFormats.All(x => x.MediaType != contentType))
                {
                    response.Content.Remove(contentType);
                }
            }
        }
        if (operation.Parameters == null)
        {
            return;
        }
        foreach (var parameter in operation.Parameters)
        {
            var description = apiDescription.ParameterDescriptions.First(p => p.Name == parameter.Name);
            parameter.Description ??= description.ModelMetadata?.Description;
            if (parameter.Schema.Default == null && description.DefaultValue != null)
            {
                var json = JsonSerializer.Serialize(description.DefaultValue,
                    description.ModelMetadata.ModelType);
                parameter.Schema.Default = OpenApiAnyFactory.CreateFromJson(json);
            }
            parameter.Required |= description.IsRequired;
        }
    }
}
```

Step 4: Add the IConfigurationOptions<SwaggerGenOptions> Implementation

- Add a new class named ConfigureSwaggerOptions into the Swagger directory and update it to the following:

```
namespace AutoLot.Api.Swagger;

public class ConfigureSwaggerOptions(
    IApiVersionDescriptionProvider provider,
    IOptionsMonitor<SwaggerApplicationSettings> settingsMonitor)
    : IConfigurationOptions<SwaggerGenOptions>
{
    private readonly SwaggerApplicationSettings _settings = settingsMonitor.CurrentValue;
    public void Configure(SwaggerGenOptions options)
    {
        foreach (var description in provider.ApiVersionDescriptions)
        {
            options.SwaggerDoc(description.GroupName, CreateInfoForApiVersion(description, _settings));
        }
    }
    internal static OpenApiInfo CreateInfoForApiVersion(
        ApiVersionDescription description,
        SwaggerApplicationSettings settings)
    {
        var versionDesc =
            settings.Descriptions.FirstOrDefault(x =>
                x.MajorVersion == (description.ApiVersion.MajorVersion ?? 0)
                && x.MinorVersion == (description.ApiVersion.MinorVersion ?? 0)
                && (string.IsNullOrEmpty(description.ApiVersion.Status) ||
                    x.Status == description.ApiVersion.Status));
        var info = new OpenApiInfo()
        {
            Title = settings.Title,
            Version = description.ApiVersion.ToString(),
            Description = $"{versionDesc?.Description}",
            Contact = new OpenApiContact() { Name=settings.ContactName, Email = settings.ContactEmail },
            TermsOfService = new System.Uri("https://www.linktotermsofservice.com"),
            License=new OpenApiLicense() {
                Name="MIT",
                Url=new Uri("https://opensource.org/licenses/MIT")
            }
        };
        if (description.IsDeprecated)
        {
            info.Description += "<p><font color='red'>This API version has been deprecated.</font></p>";
        }
        return info;
    }
}
```

Step 5: Add the Configuration Extension Method

- Add a new public static class named `SwaggerConfiguration` in the `Swagger` folder and update the code to the following:

```
namespace AutoLot.Api.Swagger;

public static class SwaggerConfiguration
{
    public static void AddAndConfigureSwagger(
        this IServiceCollection services, IConfiguration config, string xmlPathAndFile)
    {
        services.Configure<SwaggerApplicationSettings>(
            config.GetSection(nameof(SwaggerApplicationSettings)));
        services.AddTransient<IConfigureOptions<SwaggerGenOptions>, ConfigureSwaggerOptions>();
        services.AddSwaggerGen(c =>
        {
            c.EnableAnnotations();
            c.OperationFilter<SwaggerDefaultValues>();
            c.IncludeXmlComments(xmlPathAndFile);
        });
    }
}
```

Step 6: Invoke the Configuration Extension

- Comment out the existing call to `add SwaggerGen` in `Program.cs` and replace it with the new extension method:

```
//builder.Services.AddSwaggerGen();
builder.Services.AddAndConfigureSwagger(
    builder.Configuration,
    Path.Combine(AppContext.BaseDirectory,
        $"{Assembly.GetExecutingAssembly().GetName().Name}.xml"));
```

- Update the call to `UseSwaggerUI` to the following (optionally move both calls outside of `IsDevelopment` if block):

```
app.UseSwagger();
app.UseSwaggerUI(
    options =>
    {
        foreach (var description in app.DescribeApiVersions())
        {
            var url = $"/swagger/{description.GroupName}/swagger.json";
            var name = $"AutoLot API: {description.GroupName.ToUpperInvariant()}";
            options.SwaggerEndpoint(url, name);
        }
    });
```


Step 7: Create the XML Documentation File

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

```
<PropertyGroup>
  <DocumentationFile>AutoLot.Api.xml</DocumentationFile>
  <NoWarn>1701;1702;1591;1573</NoWarn>
</PropertyGroup>
```

- Edit the `AutoLot.Api.csproj` file to add the following node to copy the documentation file to the output directory:

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

Part 3: Update the Controllers with Documentation

The triple-slash comments and attributes are used to provide additional information to the Swagger documentation.

Step 1: Update the Base Crud Controller

- Header for `GetAll`:

```
/// <summary>
/// Gets all records
/// </summary>
/// <returns>All records</returns>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[SwaggerResponse(200, "The execution was successful")]
[SwaggerResponse(400, "The request was invalid")]
[ApiVersion("1.0")]
[HttpGet]
public ActionResult<IEnumerable<TEntity>> GetAll()
```

- Header for GetAllBad:

```

/// <summary>
/// Gets all records
/// </summary>
/// <returns>All records</returns>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[SwaggerResponse(200, "The execution was successful")]
[SwaggerResponse(400, "The request was invalid")]
[SwaggerResponse(401, "Unauthorized access attempted")]
[ApiVersion("0.5", Deprecated = true)]
[HttpGet]
public ActionResult<IEnumerable<TEntity>> GetAllBad()

```

- Header for GetAllFuture:

```

/// <summary>
/// Gets all records really fast (when it's written)
/// </summary>
/// <returns>All records</returns>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[SwaggerResponse(200, "The execution was successful")]
[SwaggerResponse(400, "The request was invalid")]
[SwaggerResponse(401, "Unauthorized access attempted access attempted")]
[ApiVersion("2.0-Beta")]
[HttpGet]
public ActionResult<IEnumerable<TEntity>> GetAllFuture()

```

- Header for GetOne:

```

/// <summary>
/// Gets a single record
/// </summary>
/// <param name="id">Primary key of the record</param>
/// <returns>Single record</returns>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[SwaggerResponse(200, "The execution was successful")]
[SwaggerResponse(204, "No content")]
[SwaggerResponse(400, "The request was invalid")]
[ApiVersion("1.0")]
[HttpGet("{id}")]
public ActionResult<TEntity> 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>
/// <param name="entity">Entity to update</param>
/// <returns>Single record</returns>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[SwaggerResponse(200, "The execution was successful")]
[SwaggerResponse(400, "The request was invalid")]
[ApiVersion("1.0")]
[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>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[SwaggerResponse(201, "The execution was successful")]
[SwaggerResponse(400, "The request was invalid")]
[ApiVersion("1.0")]
[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>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[SwaggerResponse(200, "The execution was successful")]
[SwaggerResponse(400, "The request was invalid")]
[ApiVersion("1.0")]
[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>
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[SwaggerResponse(200, "The execution was successful")]
[SwaggerResponse(400, "The request was invalid")]
[ApiVersion("1.0")]
[HttpGet("bymake/{id?}")]
public ActionResult<IEnumerable<Car>> GetCarsByMake(int? id)

```

Summary

This lab added versioning and configured Swagger and SwaggerUI for the service and completed the AutoLot.Api project.