.NET 10 App Dev Hands-On Workshop

API Lab 5 – Versioning, Documenting

This lab introduces versioning support first, then builds on the default OpenAPI implementation to enhance documentation and support versioning. Before starting this lab, you must have completed MVC Lab 4. The lab works entirely on the `AutoLot.Api` project.

Part 1: Add Versioning

Copilot AgentMode

```
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. 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. All work is to be done in the Autolot.API project.

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

```
Prompt: Add the following to Program.cs after the existing call to AddCors() (don't add any
additional calls except what's listed here):
builder.Services.AddProblemDetails();

var defaultVersion = ApiVersion.Default;
builder.Services.AddApiVersioning(options =>
{
  options.DefaultApiVersion = defaultVersion;
  options.AssumeDefaultVersionWhenUnspecified = true;
  options.ReportApiVersions = true;
  options.ApiVersionReader = ApiVersionReader.Combine(
    new UrlSegmentApiVersionReader(),
    new QueryStringApiVersionReader(), //defaults to "api-version"
    new QueryStringApiVersionReader("v"),
    new HeaderApiVersionReader("x-ms-api-version"),
    new HeaderApiVersionReader("x-ms-v"),
    new MediaTypeApiVersionReader(), //defaults to "v"
    new MediaTypeApiVersionReader("api-version")
  );
  options.ApiVersionSelector = new DefaultApiVersionSelector(options);
  options.RouteConstraintName = "apiVersion";
  options.UnsupportedApiVersionStatusCode = StatusCodes.Status400BadRequest;
})
.AddMvc()
.AddApiExplorer(options =>
{
  options.GroupNameFormat = "'v'VVV";
  options.SubstituteApiVersionInUrl = true;
  options.AddApiVersionParametersWhenVersionNeutral = true;
});
builder.Services.TryAddEnumerable(
    ServiceDescriptor.Transient<IApiControllerSpecification, ApiBehaviorSpecification>());

Prompt: Add the URL Segment Route to the BaseCrudController, ContentNegotiationController,
DataShapingController, and ValuesController (don't remove the existing route attribute):
[Route("api/v{version:apiVersion}/[controller]")]

Prompt: Add the ApiVersionNeutral attribute to the ContentNegotiationController,
DataShapingController, and ValuesController.

Prompt: Update the Entities Controllers (CarDriversController, CarsController, DriversController,
MakesController, RadiosController)
by adding the ApiVersion(1.0) attribute to each of them.

Prompt: Add a new ActionMethod to the CarsController:
[ApiVersion(1.5, Deprecated = true)]
[HttpGet]
public ActionResult<IEnumerable<Car>> GetAllBad()
  => throw new Exception("I said not to use this one");
```

```
Prompt: In the Controllers folder, add a new a new class named CarsBetaController that inherits
from BaseCrudController<Car, ICarRepo>.
Don't add any route attributes to the class (they come from the base class).
Add the ApiVersion 2.5-Beta and the 3.0-Beta attributes to the class.

Add the following action methods to the CarsBetaController:
  [MapToApiVersion("2.5-Beta")]
  [HttpGet]
  public ActionResult<IQueryable<Car>> GetAllFuture()
    => throw new NotImplementedException("I'm working on it");

  [MapToApiVersion("3.0-Beta")]
  [HttpGet]
  public ActionResult<IQueryable<Car>> GetAllFutureBeta()
    => throw new NotImplementedException("I'm working on it");

Prompt: In the Controllers folder, add a new a new class named HealthCheckController that inherits
from ControllerBase.
Add the ApiController attribute, both route attributes ("api/[Controller]",
"api/v{version:apiVersion}/[controller]"),
and the ApiVersionVersionNeutral attribute to the class.

Add the following action method to the HealthCheckController:
[HttpOptions]
public IActionResult Options([FromServices] IApiVersionDescriptionProvider provider)
{
  Response.Headers["Allow"] = "GET, POST, PUT, DELETE, OPTIONS";
  var supportedVersions = provider.ApiVersionDescriptions
    .Where(d => !d.IsDeprecated)
    .Select(d => d.ApiVersion.ToString())
    .Distinct()
    .OrderBy(v => v)
    .ToArray();
  var deprecatedVersions = provider.ApiVersionDescriptions
    .Where(d => d.IsDeprecated)
    .Select(d => d.ApiVersion.ToString())
    .Distinct()
    .OrderBy(v => v)
    .ToArray();
  Response.Headers["api-supported-versions"] = string.Join(", ", supportedVersions);
  if (deprecatedVersions.Length > 0)
  {
    Response.Headers["api-deprecated-versions"] = string.Join(", ", deprecatedVersions);
  }
  return Ok();
}
```

```
Prompt: In the Controllers folder, add a new a new class named Version1Controller that inherits
from ControllerBase.
Add the ApiController attribute, both route attributes ("api/[Controller]",
"api/v{version:apiVersion}/[controller]"),
and the ApiVersion(1.0) attribute to the class. Add the following action methods to the
Version1Controller:
  [HttpGet]
  public virtual string Get(ApiVersion apiVersion)
    => $"Controller = {GetType().Name}{Environment.NewLine}Version = {apiVersion}";
  [HttpGet("{id}")]
  public virtual string Get(int id)
  {
    ApiVersion version = HttpContext.GetRequestedApiVersion();
    var newLine = Environment.NewLine;
    return $"Controller = {GetType().Name}{newLine}Version = {version}{newLine}id = {id}";
  }

Prompt: In the Controllers folder, add a new a new class named Version2Controller that inherits
from Version1Controller.
Add the ApiVersion(2.0) attribute to the class.
Add the following action method to the Version2Controller:
  [HttpGet]
  public override string Get(ApiVersion apiVersion)
    => $"Controller = {GetType().Name}{Environment.NewLine}Version = {apiVersion}";
  [HttpGet("{id}")]
  public override string Get(int id)
  {
    ApiVersion version = HttpContext.GetRequestedApiVersion();
    var newLine = Environment.NewLine;
    return $"Controller = {GetType().Name}{newLine}Version = {version}{newLine}id = {id}";
  }
```

Manual

Step 1: Add the following to the GlobalUsings.cs file

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

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

Step 2: Update Program.cs

- Add the following to the `Program.cs` file, after the call to `AddCors()`:

```
builder.Services.AddProblemDetails();

var defaultVersion = ApiVersion.Default;
builder.Services.AddApiVersioning(options =>
{
  options.DefaultApiVersion = defaultVersion;
  options.AssumeDefaultVersionWhenUnspecified = true;
  options.ReportApiVersions = true;
  options.ApiVersionReader = ApiVersionReader.Combine(
    new UrlSegmentApiVersionReader(),
    new QueryStringApiVersionReader(), //defaults to "api-version"
    new QueryStringApiVersionReader("v"),
    new HeaderApiVersionReader("x-ms-api-version"),
    new HeaderApiVersionReader("x-ms-v"),
    new MediaTypeApiVersionReader(), //defaults to "v"
    new MediaTypeApiVersionReader("api-version")
  );
  options.ApiVersionSelector = new DefaultApiVersionSelector(options);
  options.RouteConstraintName = "apiVersion";
  options.UnsupportedApiVersionStatusCode = StatusCodes.Status400BadRequest;
})
.AddMvc()
.AddApiExplorer(options =>
{
  options.GroupNameFormat = "'v'VVV";
  options.SubstituteApiVersionInUrl = true;
  options.AddApiVersionParametersWhenVersionNeutral = true;
});
builder.Services.TryAddEnumerable(
    ServiceDescriptor.Transient<IApiControllerSpecification, ApiBehaviorSpecification>());
```

Step 3: Add the additional Route to the BaseCrudController

- Add the URL Segment route attributes to the `BaseCrudController`:

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

Step 4: Update the Entity Controllers

- Add the `ApiVersion` attribute to the `CarDriversController`, `CarsController`, `DriversController`, `MakesController`, and `RadiosController` classes:

```
[ApiVersion(1.0)]
public class <EntityName>sController(IAppLogging<CarsController> logger, ICarRepo repo)
  : BaseCrudController<<EntityName>, <EntityName>sController>(logger, repo)
```

- Add a new deprecated method to the `CarsController`:

```
[ApiVersion(1.5, Deprecated = true)]
[HttpGet]
public ActionResult<IEnumerable<Car>> GetAllBad()
  => throw new Exception("I said not to use this one");
```

Step 5: Update the Remaining Controllers

- Add the `ApiVersionNeutral` attribute and the URL segment route attributes to the `ContentNegotiationController`, `DataShapingController`, and `ValuesController`:

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

Step 6: Add the CarsBetaController

- Add a new class named `CarsBetaController` to the Controllers folder and update the code to the following:

```
namespace AutoLot.Api.Controllers;
[ApiVersion("2.5-Beta")]
[ApiVersion(3.0, "Beta")]
public class CarsBetaController(IAppLogging appLogging, ICarRepo repo)
  : BaseCrudController<Car>(appLogging, repo)
{
  [MapToApiVersion("2.5-Beta")]
  [HttpGet]
  public ActionResult<IQueryable<Car>> GetAllFuture()
    => throw new NotImplementedException("I'm working on it");
  [MapToApiVersion("3.0-Beta")]
  [HttpGet]
  public ActionResult<IQueryable<Car>> GetAllFutureBeta()
    => throw new NotImplementedException("I'm working on it");
}
```

Step 7: Add the HealthCheckController

- Create a new controller named `HealthCheckController.cs` in the `Controllers` folder and update the file to match the following:

```
namespace AutoLot.Api.Controllers;
[ApiVersionNeutral]
[ApiController]
[Route("api/[controller]")]
[Route("api/v{version:apiVersion}/[controller]")]
public class HealthCheckController : Controller
{
  //action method goes here
}
```

- The `Options` action method gets the requested API version from the `HttpContext` and returns additional information for the API service:

```
[HttpOptions]
public IActionResult Options([FromServices] IApiVersionDescriptionProvider provider)
{
  Response.Headers["Allow"] = "GET, POST, PUT, DELETE, OPTIONS";
  var supportedVersions = provider.ApiVersionDescriptions
    .Where(d => !d.IsDeprecated)
    .Select(d => d.ApiVersion.ToString())
    .Distinct()
    .OrderBy(v => v)
    .ToArray();
  var deprecatedVersions = provider.ApiVersionDescriptions
    .Where(d => d.IsDeprecated)
    .Select(d => d.ApiVersion.ToString())
    .Distinct()
    .OrderBy(v => v)
    .ToArray();
  Response.Headers["api-supported-versions"] = string.Join(", ", supportedVersions);
  if (deprecatedVersions.Length > 0)
  {
    Response.Headers["api-deprecated-versions"] = string.Join(", ", deprecatedVersions);
  }
  return Ok();
}
```

Step 8: Add the Version1Controller

- Add a new class named `Version1Controller.cs` into the `Controllers` folder and update the code to the following:

```
namespace AutoLot.Api.Controllers;
[ApiController]
[ApiVersion(1.0)]
[Route("api/[controller]")]
[Route("api/v{version:apiVersion}/[controller]")]
public class Version1Controller : ControllerBase
{
  [HttpGet]
  public virtual string Get(ApiVersion apiVersion)
    => $"Controller = {GetType().Name}{Environment.NewLine}Version = {apiVersion}";
  [HttpGet("{id}")]
  public virtual string Get(int id)
  {
    ApiVersion version = HttpContext.GetRequestedApiVersion();
    var newLine = Environment.NewLine;
    return $"Controller = {GetType().Name}{newLine}Version = {version}{newLine}id = {id}";
  }
}
```

Step 9: Add the Version2Controller

- Add a new class named `Version2Controller.cs` into the `Controllers` folder and update the code to the following:

```
namespace AutoLot.Api.Controllers;
[ApiVersion(2.0)]
public class Version2Controller : Version1Controller
{
  [HttpGet]
  public override string Get(ApiVersion apiVersion)
    => $"Controller = {GetType().Name}{Environment.NewLine}Version = {apiVersion}";
  [HttpGet("{id}")]
  public override string Get(int id)
  {
    ApiVersion version = HttpContext.GetRequestedApiVersion();
    var newLine = Environment.NewLine;
    return $"Controller = {GetType().Name}{newLine}Version = {version}{newLine}id = {id}";
  }
}
```

Copilot Agent Mode and Manual

Test with Bruno or CURL

- Try the following URLs to test the versioning of the app:

```
https://localhost:5011/api/HealthCheck (return version 1 information)
https://localhost:5011/api/HealthCheck?v=2 (return version 2 information)
https://localhost:5011/api/HealthCheck?v=99 (return a problem report with unsupported version)

https://localhost:5011/api/Version?v=1
https://localhost:5011/api/Version?v=2
```

Part 2: OpenAPI Documentation - Scalar

Microsoft's new OpenAPI package replaces the majority of the functionality of Swashbuckle's SwaggerGen. Since it does not provide a UI, we will use Scalar to present the OpenAPI documentation.

Copilot Agent Mode

```
Setup Prompt: 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. 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. All work is to be done in the Autolot.API project unless
otherwise specified.

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 Microsoft.AspNetCore.OpenApi;
global using Microsoft.OpenApi;
global using Scalar.AspNetCore;

Prompt: Add a new folder named ScalarUtilities, and in that folder create a new class named
AddHeaderParameterDocumentFilter.cs with the following code:
namespace AutoLot.Api.ScalarUtilities;
public class AddHeaderParameterDocumentFilter : IOpenApiOperationTransformer
{
  public Task TransformAsync(
    OpenApiOperation operation,
    OpenApiOperationTransformerContext context,
    CancellationToken cancellationToken)
  {
    operation.Parameters ??= [];
    operation.Parameters.Add(new OpenApiParameter
    {
      Name = "Accept",
      In = ParameterLocation.Header,
      Required = false,
      Description = "Specifies the desired response format.",
      AllowEmptyValue = true,
      Schema = new OpenApiSchema
      {
        Type = JsonSchemaType.String
      }
    });
    return Task.CompletedTask;
  }
}
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.Api.ScalarUtilities;
```

```
Prompt: Update the project file for AutoLot.Models by adding the two nodes to the TargetFramework
PropertyGroup to enable XML comments generation:
  <GenerateDocumentationFile>true</GenerateDocumentationFile>
  <NoWarn>1591</NoWarn>

Prompt: Update the project file for AutoLot.Api by adding the three nodes to the TargetFramework
PropertyGroup to enable XML comments generation,
set the NoWarnd, and get OpenAPI specification version:
  <GenerateDocumentationFile>true</GenerateDocumentationFile>
  <NoWarn>1591</NoWarn>
  <OpenApiGenerateDocumentsOptions>--openapi-version OpenApi3_1</OpenApiGenerateDocumentsOptions>

Prompt: Add the following to the top of the Program.cs file before and code and after any existing
using statements:
using JsonOptions = Microsoft.AspNetCore.Http.Json.JsonOptions;

Prompt: Add the following to Program.cs before builder.Services.AddScoped<ICarDriverRepo,
CarDriverRepo>(); (don't add any additional calls except what's listed here):
builder.Services.Configure<JsonOptions>(options =>
{
  options.SerializerOptions.PropertyNamingPolicy = JsonNamingPolicy.CamelCase;
  options.SerializerOptions.DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull;
  options.SerializerOptions.WriteIndented = true;
  options.SerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles;
});

Prompt: Add three local functions to Program.cs after all other code:
OpenApiInfo BuildDocumentInfo(string version, bool isDeprecated)
{
  var description = "API for AutoLot Car Dealership.";
  if (isDeprecated)
  {
    description += "<p><font color='red'>This API version has been deprecated.</font></p>";
  }
  return new()
  {
    Title = "AutoLot API",
    Version = version,
    Description = description,
    Contact = new OpenApiContact() { Name = "Phil Japikse", Email = "skimedic@outlook.com" },
    TermsOfService = new System.Uri("https://www.linktotermsofservice.com"),
    License = new OpenApiLicense()
    {
      Name = "MIT",
      Url = new Uri("https://opensource.org/licenses/MIT")
    }
  };
}
```

```
void AddDeprecated(OpenApiDocument openApiDocument, List<string> controllerNames)
{
  foreach (var path in openApiDocument.Paths.Where(x => x.Value?.Operations != null))
  {
    foreach (var op in path.Value.Operations!)
    {
      if (controllerNames is { Count: > 0 })
      {
        controllerNames
          .Where(controllerName =>
             path.Key.Contains($"/{controllerName}", StringComparison.OrdinalIgnoreCase))
          .ToList()
          .ForEach(controllerName => op.Value.Deprecated = true);
      }
      else
      {
        op.Value.Deprecated = true;
      }
    }
  }
}

void ConfigureOptions(
  OpenApiOptions openApiOptions,
  string documentName,
  bool includeBlankGroups = false,
  bool isDeprecated = false,
  List<string> controllerNamesToDeprecate = null)
{
  openApiOptions.AddDocumentTransformer((document, context, cancellationToken)
    => {
         document.Info = BuildDocumentInfo("v1.5", true);
         if (isDeprecated)
         {
           AddDeprecated(document, controllerNamesToDeprecate);
         }
         return Task.CompletedTask;
  });

Prompt: Remove the empty call to builder.Service.AddOpenApi();

Prompt: Add the following to Program.cs before var app = builder.Build() (don't add any additional
calls except what's listed here):
builder.Services.AddOpenApi("v1", options =>
  ConfigureOptions(options, "v1", includeBlankGroups: true));
builder.Services.AddOpenApi("v1.5", options =>
  ConfigureOptions(options, "v1.5", isDeprecated: true, controllerNamesToDeprecate: ["Cars"]));
builder.Services.AddOpenApi("v2", options => ConfigureOptions(options, "v2"));
builder.Services.AddOpenApi("v2.5-Beta", options => ConfigureOptions(options, "v2.5-Beta"));
builder.Services.AddOpenApi("v3-Beta", options => ConfigureOptions(options, "v3-Beta"));
```

```
Add the scalar code after the call to app.MapOpenApi();
var provider = app.Services.GetRequiredService<IApiVersionDescriptionProvider>();
app.MapScalarApiReference(options =>
{
  var docs = new List<ScalarDocument>();
  foreach (var versionDescription in provider.ApiVersionDescriptions)
  {
    var isDeprecated = versionDescription.IsDeprecated ? " (Deprecated)":"";
    var name = $"AutoLot API Version {versionDescription.ApiVersion.ToString()}{isDeprecated}";
    docs.Add(new ScalarDocument(versionDescription.GroupName, name));
  }
  options.AddDocuments(docs);
});

Update the launchSetting.json file replace "launchUrl": "openapi/v1.json" with "launchUrl":
"scalar"
```

Manual

Step 1: Add the following to the GlobalUsings.cs file

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

```
global using Microsoft.AspNetCore.OpenApi;
global using Microsoft.OpenApi;
global using Scalar.AspNetCore;
```

Step 2: Generate the XML Documentation Files and OpenAPI version

- Edit the `AutoLot.Models.csproj` file to add the following nodes to the main `PropertyGroup` to create the documentation file from the triple-slash comments and remove warnings for 1591 (no /// comments):

```
<PropertyGroup>
  <TargetFramework>netXX.0</TargetFramework>
  <Nullable>disable</Nullable>
  <ImplicitUsings>enable</ImplicitUsings>
  <GenerateDocumentationFile>true</GenerateDocumentationFile>
  <NoWarn>1591</NoWarn>
</PropertyGroup>
```

- Edit the `AutoLot.Api.csproj` file to add the following nodes to the main `PropertyGroup` to create the documentation file from the triple-slash comments, remove warnings for 1591 (no /// comments), and set the version of OpenApi to 3.1:

```
<PropertyGroup>
  <TargetFramework>netXX.0</TargetFramework>
  <Nullable>disable</Nullable>
  <ImplicitUsings>enable</ImplicitUsings>
  <GenerateDocumentationFile>true</GenerateDocumentationFile>
  <NoWarn>1591</NoWarn>
  <OpenApiGenerateDocumentsOptions>--openapi-version OpenApi3_1</OpenApiGenerateDocumentsOptions>
</PropertyGroup>
```

Step 3: Add the HeaderParameter Document Filter

- Create a new folder called `ScalarUtilities` in the root of the project and add a new class named `AddHeaderParameterDocumentFilter`. Update the code to the following:

```
namespace AutoLot.Api.ScalarUtilities;

public class AddHeaderParameterDocumentFilter : IOpenApiOperationTransformer
{
  public Task TransformAsync(
    OpenApiOperation operation,
    OpenApiOperationTransformerContext context,
    CancellationToken cancellationToken)
  {
    operation.Parameters ??= [];
    operation.Parameters.Add(new OpenApiParameter
    {
      Name = "Accept",
      In = ParameterLocation.Header,
      Required = false,
      Description = "Specifies the desired response format.",
      AllowEmptyValue = true,
      Schema = new OpenApiSchema
      {
        Type = JsonSchemaType.String
      }
    });
    return Task.CompletedTask;
  }
}
```

Step 4: Update Program.cs

- Add the Global JSON Options (used by minimal APIs and OpenAPI docs):

```
builder.Services.Configure<JsonOptions>(options =>
{
  options.SerializerOptions.PropertyNamingPolicy = JsonNamingPolicy.CamelCase;
  options.SerializerOptions.DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull;
  options.SerializerOptions.WriteIndented = true;
  options.SerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles;
});
```

- Add three local functions at the end of the top-level statements:

```
OpenApiInfo BuildDocumentInfo(string version, bool isDeprecated)
{
  var description = "API for AutoLot Car Dealership.";
  if (isDeprecated)
  {
    description += "<p><font color='red'>This API version has been deprecated.</font></p>";
  }
  return new()
  {
    Title = "AutoLot API",
    Version = version,
    Description = description,
    Contact = new OpenApiContact() { Name = "Phil Japikse", Email = "skimedic@outlook.com" },
    TermsOfService = new System.Uri("https://www.linktotermsofservice.com"),
    License = new OpenApiLicense()
    {
      Name = "MIT",
      Url = new Uri("https://opensource.org/licenses/MIT")
    }
  };
}

void AddDeprecated(OpenApiDocument openApiDocument, List<string> controllerNames)
{
  foreach (var path in openApiDocument.Paths.Where(x => x.Value?.Operations != null))
  {
    foreach (var op in path.Value.Operations!)
    {
      if (controllerNames is { Count: > 0 })
      {
        controllerNames
          .Where(controllerName =>
             path.Key.Contains($"/{controllerName}", StringComparison.OrdinalIgnoreCase))
          .ToList()
          .ForEach(controllerName => op.Value.Deprecated = true);
      }
      else
      {
        op.Value.Deprecated = true;
      }
    }
  }
}
```

```
void ConfigureOptions(
  OpenApiOptions openApiOptions,
  string documentName,
  bool includeBlankGroups = false,
  bool isDeprecated = false,
  List<string> controllerNamesToDeprecate = null)
{
  openApiOptions.AddDocumentTransformer((document, context, cancellationToken)
    => {
         document.Info = BuildDocumentInfo("v1.5", true);
         if (isDeprecated)
         {
           AddDeprecated(document, controllerNamesToDeprecate);
         }
         return Task.CompletedTask;
  });
```

- Remove the call to `AddOpenApi()` and replace it with the following:

```
builder.Services.AddOpenApi("v1", options =>
  ConfigureOptions(options, "v1", includeBlankGroups: true));
builder.Services.AddOpenApi("v1.5", options =>
  ConfigureOptions(options, "v1.5", isDeprecated: true, controllerNamesToDeprecate: ["Cars"]));
builder.Services.AddOpenApi("v2", options => ConfigureOptions(options, "v2"));
builder.Services.AddOpenApi("v2.5-Beta", options => ConfigureOptions(options, "v2.5-Beta"));
builder.Services.AddOpenApi("v3-Beta", options => ConfigureOptions(options, "v3-Beta"));
```

- Add `Scalar` after the call to `app.MapOpenApi()`:

```
app.MapOpenApi();
var provider = app.Services.GetRequiredService<IApiVersionDescriptionProvider>();
app.MapScalarApiReference(options =>
{
  var docs = new List<ScalarDocument>();
  foreach (var versionDescription in provider.ApiVersionDescriptions)
  {
    var isDeprecated = versionDescription.IsDeprecated ? " (Deprecated)":"";
    var name = $"AutoLot API Version {versionDescription.ApiVersion.ToString()}{isDeprecated}";
    docs.Add(new ScalarDocument(versionDescription.GroupName, name));
  }
  options.AddDocuments(docs);
});
```

Step 5: Update the LaunchSettings.json File

- Update the `Properties\launchSettings.json` file to change the launch URL to scalar (changes in bold):

```
"launchUrl": "scalar",
```

Part 3: OpenAPI Documentation - SwaggerUI

Another option to Scalar is to use Swagger UI to present the OpenAPI documentation.

- Add `Scalar` after the call to `app.MapOpenApi()`:

```
app.UseSwaggerUI(options =>
{
  foreach (var versionDescription in provider.ApiVersionDescriptions)
  {
    var groupName = versionDescription.GroupName;
    var url = $"/openapi/{groupName}.json";
    var name = $"AutoLot API: {groupName}";
    options.SwaggerEndpoint(url, name);
  }
});
```

- Update the `Properties\launchSettings.json` file to change the launch URL to swagger (changes in bold):

```
"launchUrl": "swagger",
```

Part 4: Document the Controllers

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

Copilot Agent Mode

```
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. 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. All work is to be done in the Autolot.API project unless otherwise instructed. Prompt: When
adding attributes, keep the ApiVersion attribute at the top, following by the HTTP Verb attribute,
then followed by route attributes (if any), then documentation attributes.
Add each attribute on it's own line.

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 System.ComponentModel;

Prompt: Add the following attribute to all controller action methods (even if deprecated) that do
not currently have a Produces attribute:
[Produces("application/json")]

Prompt: Add the following attributes to all controller action methods (even if deprecated) that do
not currently have them:
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType<ProblemDetails>(StatusCodes.Status400BadRequest,
"application/problem+json")]

Prompt: Add the following attribute to all controller action methods (even if deprecated) that
return a 201 Created response:
[ProducesResponseType(StatusCodes.Status201Created)]

Prompt: Add the following attribute to all controller action methods (even if deprecated) that
return a 204 No Content response:
[ProducesResponseType(StatusCodes.Status204NoContent)]

Prompt: Add the following attributes to all action methods on all controllers:
[EndpointSummary(summary of the endpoint)]
[EndpointDescription(longer description of the endpoint, followed by an example, followed by the
JSON required in the request body (if applicable))]

Prompt: Add the following attribute to all parameters on all action methods on all controllers
(even if deprecated) unless the parameter is an interface:
Description - description of the parameter and if it's required or not (required == not nullable
or marked with required attribute)
```

```
Prompt: The following work is to be done in the AutoLot.Models project:
Add /// XML comments to the entity classes (BaseEntity, Car, CarDriver, Driver, Make, Model,
Radio) all public properties. Make sure the comments come before any existing attributes.
Include <summary>, <remarks>.
```

Manual

In this lab, I only show how to update the `BaseCrudController` and the `Cars` entity fully. Optionally, you can update all controllers' action methods and entity classes accordingly.

Step 1: Update the Base Crud Controller

- Header for `GetAll`:

```
[HttpGet]
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType<ProblemDetails>(StatusCodes.Status400BadRequest,
"application/problem+json")]
[EndpointSummary("Gets all entities.")]
[EndpointDescription("Returns a list of all entities. Example: GET /api/[controller]. No request
body required.")]
public ActionResult<IEnumerable<TEntity>> GetAll()
```

- Header and parameter for `GetOne`:

```
[HttpGet("{id}")]
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType<ProblemDetails>(StatusCodes.Status400BadRequest,
"application/problem+json")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[EndpointSummary("Gets a single entity by ID.")]
[EndpointDescription("Returns the entity with the specified ID. Example: GET /api/[controller]/1.
No request body required.")]
public virtual ActionResult<TEntity> GetOne(
  [Description("The unique identifier of the entity. Required.")] int id)
```

- Header and parameters for `UpdateOne`:

```
[HttpPut("{id}")]
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType<ProblemDetails>(StatusCodes.Status400BadRequest,
"application/problem+json")]
[EndpointSummary("Updates an entity by ID.")]
[EndpointDescription("Updates the entity with the specified ID. Example: PUT /api/[controller]/1.
JSON body required: { \"property\": \"value\" }")]
public virtual ActionResult<TEntity> UpdateOne(
  [Description("The unique identifier of the entity. Required.")] int id,
  [Description("The entity to update. Required.")] TEntity entity)
```

- Header and parameters for `AddOne`:

```
[HttpPost]
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType<ProblemDetails>(StatusCodes.Status400BadRequest,
"application/problem+json")]
[EndpointSummary("Adds a new entity.")]
[EndpointDescription("Creates a new entity. Example: POST /api/[controller]. JSON body required: {
\"property\": \"value\" }")]
public virtual ActionResult<TEntity> AddOne(
  [Description("The entity to add. Required.")] TEntity entity)
```

- Header for `DeleteOne`:

```
[HttpDelete("{id}")]
[Produces("application/json")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType<ProblemDetails>(StatusCodes.Status400BadRequest,
"application/problem+json")]
[EndpointSummary("Deletes an entity by ID.")]
[EndpointDescription("Deletes the entity with the specified ID. Example: DELETE
/api/[controller]/1. JSON body required: { \"id\": 1 }")]
public virtual IActionResult DeleteOne(
  [Description("The unique identifier of the entity. Required.")] int id,
  [Description("The entity to delete. Required.")] TEntity entity)
```

Step 2: Update the Cars Entity

```
/// <summary>
/// Represents a car in the inventory.
/// </summary>
/// <remarks>
/// Contains details about the car, its make, drivers, and radio.
/// </remarks>
[Table("Inventory", Schema = "dbo")]
[EntityTypeConfiguration(typeof(CarConfiguration))]
[Index(nameof(MakeId), Name = "IX_Inventory_MakeId")]
public class Car : BaseEntity
{
  /// <summary>
  /// The color of the car.
  /// </summary>
  /// <remarks>
  /// Required, maximum length 50.
  /// </remarks>
  [Required, MaxLength(50)]
  public string Color { get; set; }
```

```
/// <summary>
/// The price of the car.
/// </summary>
/// <remarks>
/// May be null or empty if not set.
/// </remarks>
public string Price { get; set; }

/// <summary>
/// Indicates if the car is drivable.
/// </summary>
/// <remarks>
/// Defaults to true.
/// </remarks>
[DisplayName("Is Drivable")]
public bool IsDrivable { get; set; } = true;

/// <summary>
/// The date the car was built.
/// </summary>
/// <remarks>
/// Nullable; may not be set for all cars.
/// </remarks>
public DateTime? DateBuilt { get; set; }

/// <summary>
/// The display string for the car.
/// </summary>
/// <remarks>
/// Computed by the database.
/// </remarks>
[DatabaseGenerated(DatabaseGeneratedOption.Computed)]
public string Display { get; set; }

/// <summary>
/// The pet name of the car.
/// </summary>
/// <remarks>
/// Required, maximum length 50.
/// </remarks>
[Required, MaxLength(50), DisplayName("Pet Name")]
public string PetName { get; set; }

/// <summary>
/// The foreign key to the make of the car.
/// </summary>
/// <remarks>
/// Required.
/// </remarks>
[Required, DisplayName("Make")]
public int MakeId { get; set; }
```

```
  /// <summary>
  /// Navigation property to the make entity.
  /// </summary>
  /// <remarks>
  /// References the make of the car.
  /// </remarks>
  [ForeignKey(nameof(MakeId))]
  [InverseProperty(nameof(Make.Cars))]
  public Make MakeNavigation { get; set; }

  /// <summary>
  /// Navigation property to the radio entity.
  /// </summary>
  /// <remarks>
  /// References the radio installed in the car.
  /// </remarks>
  [InverseProperty(nameof(Radio.CarNavigation))]
  public Radio RadioNavigation { get; set; }

  /// <summary>
  /// Navigation property to the drivers of the car.
  /// </summary>
  /// <remarks>
  /// Collection of drivers associated with the car.
  /// </remarks>
  [InverseProperty(nameof(Driver.Cars))]
  public ICollection<Driver> Drivers { get; set; } = new List<Driver>();

  /// <summary>
  /// Navigation property to the car-driver relationships.
  /// </summary>
  /// <remarks>
  /// Collection of car-driver entities.
  /// </remarks>
  [InverseProperty(nameof(CarDriver.CarNavigation))]
  public ICollection<CarDriver> CarDrivers { get; set; } = new List<CarDriver>();

  /// <summary>
  /// The name of the make for this car.
  /// </summary>
  /// <remarks>
  /// Returns "Unknown" if the make is not set.
  /// </remarks>
  [NotMapped]
  public string MakeName => MakeNavigation?.Name ?? "Unknown";

  /// <summary>
  /// Returns a string representation of the car.
  /// </summary>
  /// <remarks>
  /// Includes pet name, color, make, and ID.
  /// </remarks>
  public override string ToString()
    => $"{PetName ?? "**No Name**"} is a {Color} {MakeNavigation?.Name} with ID {Id}.";
}
```

Part 4: Exclude Actions from OpenAPI Documentation

Step 1: Update the ValuesController

- Add a new action method to the `ValuesController`:

```
[ApiExplorerSettings(IgnoreApi = true)]
[HttpGet("hidden/{id?}")]
public string HiddenEndPoint(int? id, ApiVersion apiVersion)
  => $"Controller = {GetType().Name}{Environment.NewLine}Version = {apiVersion}";
```

Summary

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