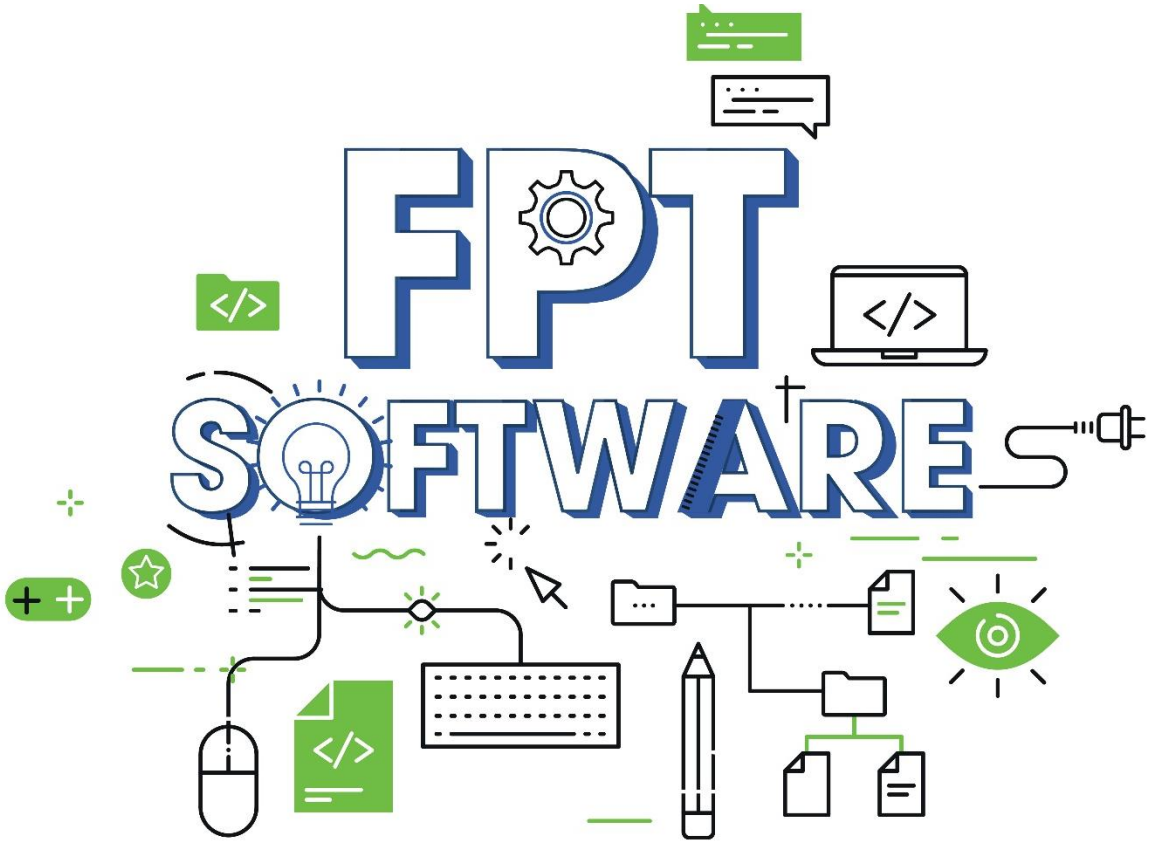
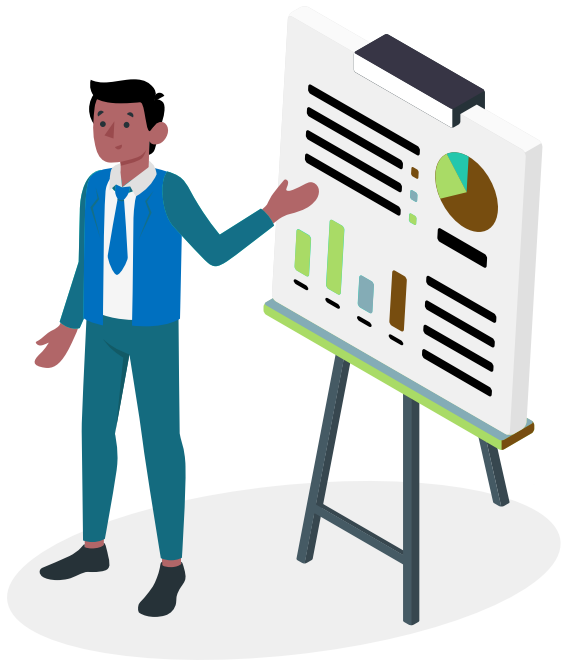
Error, exception and Web API Versioning



Agenda

1. • **Error and Exception Handling in ASP.NET Core API**
2. • **Web API Versioning**



Lesson Objectives

- Understand the importance of error and exception handling in ASP.NET Core API development.
- Learn the different types of errors and exceptions that can occur in an API and their impact on the application.
- Implement a centralized error handling mechanism to handle unhandled exceptions and provide consistent error responses to clients.
- Understand the importance of versioning in API development and its benefits for maintaining backward compatibility and evolving the API over time.
- Learn the different approaches to versioning in ASP.NET Core API, such as URL-based versioning, query parameter versioning, header-based versioning, or media type versioning.
- Implement a versioning strategy that allows clients to choose the desired API version and supports multiple versions simultaneously.
- Handle versioning in a consistent and predictable manner across different API endpoints and controllers.

Section 1

Error and Exception Handling in ASP.NET Core API

HTTP Response Status Codes in ASP.NET Core API

- In this presentation, we will explore the different HTTP response status codes and their usage in ASP.NET Core API.
- Understanding the appropriate status code for each API response is crucial for building robust and user-friendly APIs.
- Let's delve into the details of HTTP response status codes and their significance.

Overview of HTTP Status Codes

- HTTP status codes are three-digit numbers that indicate the outcome of an HTTP request.
- They provide information about whether the request was successful, encountered an error, or requires further action.
- Status codes are grouped into different categories based on their first digit.

Common HTTP Status Code Categories

There are several common categories of HTTP status codes:

- *1xx*: **Informational** - Request received, continuing process.
- *2xx*: **Success** - The request was successfully received, understood, and accepted.
- *3xx*: **Redirection** - Further action is needed to complete the request.
- *4xx*: **Client Errors** - The request contains bad syntax or cannot be fulfilled.
- *5xx*: **Server Errors** - The server failed to fulfill a valid request.

Examples of Common HTTP Status Codes

Some examples of commonly used HTTP status codes include:

- **200 OK**: The request was successful.
- **201 Created**: The request has been fulfilled, resulting in the creation of a new resource.
- **400 Bad Request**: The server cannot process the request due to a client error.
- **401 Unauthorized**: The request requires user authentication.
- **404 Not Found**: The requested resource could not be found on the server.
- **500 Internal Server Error**: The server encountered an unexpected condition that prevented it from fulfilling the request.

Using status code in Controller

- Update method **GetBook()**:

```
public IActionResult GetBook(int id)
{
    var book = _bookService.GetBook(id);

    if (book != null)
        return Ok(book);

    return NotFound();
}
```

Handling exception using try catch

- Update method **AddBook** in controller

```
 [HttpPost("add-book-with-authors")]
public IActionResult AddBook([FromBody] BookVm book)
{
    try
    {
        _bookService.AddBookWithAuthors(book);

        return Ok();
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

Using throw exception

- Update method **DeleteBook()** in **BookService**

```
public void DeleteBook(int id)
{
    var book = _context.Books.Find(id);

    if (book != null)
    {
        _context.Books.Remove(book);
        _context.SaveChanges();
    }
    else
    {
        throw new Exception($"The book with id: {id} does not exist!");
    }
}
```

- Using try..catch in the DeleteBookById method.

Global error handling - 1

- Add **ErrorVm**:

```
using Newtonsoft.Json;
public class ErrorVm
{
    public int StatusCode { get; set; }
    public string Message { get; set; }
    public string Path { get; set; }
    public override string ToString()
    {
        return JsonConvert.SerializeObject(this);
    }
}
```

Global error handling - 2

- Add class **ExceptionMiddlewareExtensions** in folder **Exceptions**:

```
public static class ExceptionMiddlewareExtensions
    {
        public static void ConfigureBuildInExceptionHandler(this IApplicationBuilder app)
        {
            app.UseExceptionHandler(appError =>
            {
                appError.Run(async context =>
                {
                    context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;
                    context.Response.ContentType = "application/json";

                    var contextFeature = context.Features.Get<IExceptionHandlerFeature>();
                    var contextRequest = context.Features.Get<IHttpRequestFeature>();

                    if (contextFeature != null)
                    {
                        await context.Response.WriteAsync(new ErrorVm()
                        {
                            StatusCode = context.Response.StatusCode,
                            Message = contextFeature.Error.Message,
                            Path = contextRequest.Path
                        }.ToString());
                    }
                });
            });
        }
    }
```

Global error handling - 3

- Add method in **BookController()**:

```
[HttpGet("get-book-throw-exception/{id}")]
        public IActionResult GetBookThrowException(int id)
        {
            throw new Exception("Global exception");
        }
```

- Register in **Program.cs** file:

```
app.ConfigureBuildInExceptionHandler();
```

Lesson Summary

- Understand the significance of HTTP response status codes in the context of ASP.NET Core API.
- Learn about the common HTTP status codes and their meanings, such as 200, 201, 400, 401, 404, and 500.
- Using try catch to exception handling.
- Global error handling with built-in middleware.

Section 2

Web API Versioning

What will you learn

- Setting up development environment
- Web API versioning
  - Query String
  - URL Path
  - HTTP Header
  - Content/Media Type

Need for API versioning

- As APIs evolve, changes can break existing client applications.
- Versioning allows introducing changes while maintaining backward compatibility.
- It enables different clients to use different versions of the API simultaneously.

Different approaches to versioning

- **URL-based versioning**: Include the version number in the URL path.

  Example: /api/v1/products

- **Query string versioning**: Add the version number as a parameter in the query string.

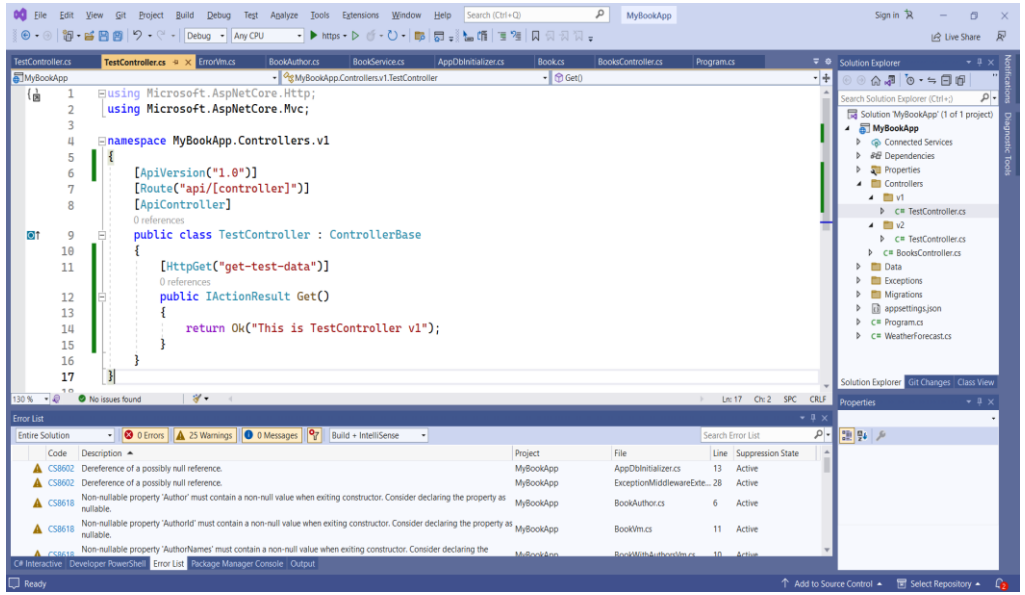
  Example: /api/products?version=1

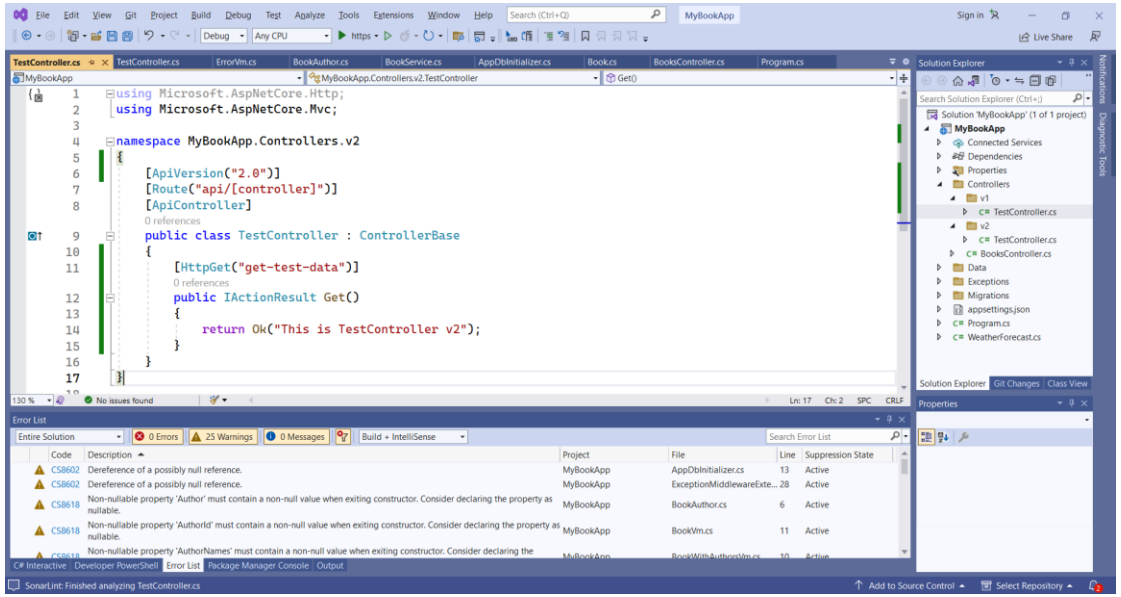
- **Header-based versioning**: Use custom headers to specify the API version.

  Example: X-API-Version: 1.

Setting up Versioning in .NET Web API

- Add two folders in **Controller** and two class **TestController**:
- Install package: **Asp.Versioning.Mvc.ApiExplorer**





Add Services

```
// Add services Api versioning
builder.Services.AddApiVersioning(config =>
{
    config.DefaultApiVersion = new ApiVersion(1, 0);
config.AssumeDefaultVersionWhenUnspecified = true;

})
.AddApiExplorer(options =>
 {
     options.GroupNameFormat = "'v'VVV";
     options.SubstituteApiVersionInUrl = true;
 });

builder.Services.AddSwaggerGen(c =>
{
    var provider =
builder.Services.BuildServiceProvider().GetRequiredService<IApiVersionDescriptionProvider>();

    foreach (var description in provider.ApiVersionDescriptions)
    {
        c.SwaggerDoc(description.GroupName, new OpenApiInfo { Title = "My API", Version =
description.ApiVersion.ToString() });
    }

});
```

Configure the HTTP request pipeline

```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        var provider = app.Services.GetRequiredService<IApiVersionDescriptionProvider>();
        foreach (var description in provider.ApiVersionDescriptions)
        {
            c.SwaggerEndpoint($"/swagger/{description.GroupName}/swagger.json",
description.GroupName.ToUpperInvariant());
        }
    });
}
```

Query String-Based versioning in Web API

- Test api with url: localhost:1234/api/test/get-test-data
- Test api version 1.0 with url: localhost:1234/api/test/get-test-data?api-version=2.0
- Test api version 2.0 with url: localhost:1234/api/test/get-test-data?api-version=2.0

URL Based Versioning

URL Path based versioning:

- /api/v1/test
- /api/v2/test

```
[ApiVersion("1.0")]
//[Route("api/[controller]")]
[Route("api/v{version:apiVersion}/[controller]")]
[ApiController]
0 references
public class TestController : ControllerBase
{
    [HttpGet("get-test-data")]
    0 references
    public IActionResult Get()
    {
        return Ok("This is TestController v1");
    }
}
```

```
[ApiVersion("2.0")]
//[Route("api/[controller]")]
[Route("api/v{version:apiVersion}/[controller]")]
[ApiController]
0 references
public class TestController : ControllerBase
{
    [HttpGet("get-test-data")]
    0 references
    public IActionResult Get()
    {
        return Ok("This is TestController v2");
    }
}
```

HTTP Header-Based Versioning

- Change config in program.cs file:

```
builder.Services.AddApiVersioning(config =>
{
    config.DefaultApiVersion = new ApiVersion(1, 0);
    config.AssumeDefaultVersionWhenUnspecified = true;

    config.ApiVersionReader = new HeaderApiVersionReader("custom-
version-header");
});
```

- Change TestController:

```
[ApiVersion("1.0")]
[ApiVersion("1.1")]
[ApiVersion("1.2")]
[Route("api/[controller]")]
//[Route("api/v{version:apiVersion}/[controller]")]
[ApiController]
0 references
public class TestController : ControllerBase
{
    [HttpGet("get-test-data"), MapToApiVersion("1.0")]
    0 references
    public IActionResult Get()
    {
        return Ok("This is version v1.0");
    }

    [HttpGet("get-test-data"),MapToApiVersion("1.1")]
    0 references
    public IActionResult GetV11()
    {
        return Ok("This is version v1.1");
    }

    [HttpGet("get-test-data"), MapToApiVersion("1.2")]
    0 references
    public IActionResult GetV12()
    {
```

- Test with Postman add header value: custom-version-header=1.1

Lesson Summary

- *Microsoft.AspNetCore.Mvc.Versioning package*
- *Web Api versioning*
  - ✓*Query String - api/controller?api-version=1.1*
  - ✓*URL path – api/v1/controller*
  - ✓*http header – custom-verion-header in HTTP*

THANK YOU!