Designing and Securing Middleware Components

Optimizing and Securing Middleware in ASP.NET Core Web API

Objective: design, implement, and secure middleware components within an ASP.NET Core Web API, ensuring they meet performance and security requirements.

Step 1: Set Up a New ASP.NET Core Web API Project

1. Open Visual Studio Code and create a new folder for your project.
2. Open the terminal and run the following commands to create a new ASP.NET Core Web API project: dotnet new webapi -o MiddlewareOptimizationApp cd MiddlewareOptimizationApp
3. Open the Program.cs file. You'll be modifying this file to implement middleware components.
4. Delete any controller files in the Controllers folder to focus solely on middleware implementation.

Step 2: Configure HTTP only

For simplicity, configure the application to listen on HTTP only by removing any HTTPS-specific code in Program.cs. This will allow learners to test the middleware without requiring a secure HTTPS connection.

```
var builder = WebApplication.CreateBuilder(args);

// Configure to listen on HTTP only for simplicity
builder.WebHost.ConfigureKestrel(options =>
{
    options.ListenLocalhost(5294);
});

var app = builder.Build();
```

This setup allows the app to respond only to HTTP requests on http://localhost:5294.

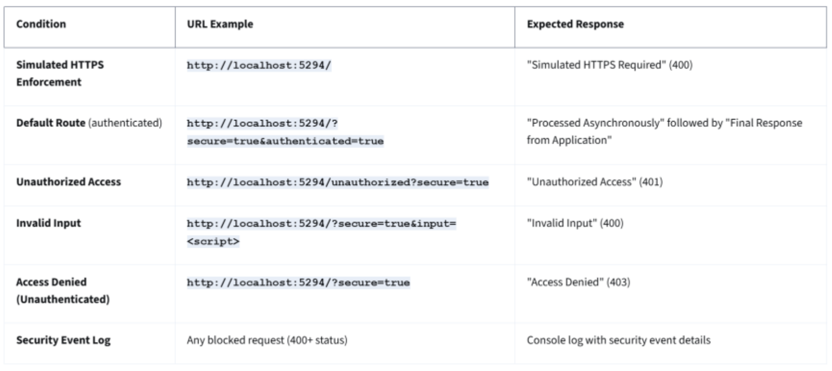
Step 3: Design Middleware for Performance Optimization and Security

In this step, learners will write middleware components to handle performance optimization and security. Specifically:

- Simulated HTTPS Enforcement: Use a query parameter to simulate HTTPS enforcement. If the secure=true parameter is missing, the middleware should block the request as if it were non-HTTPS.
- Short-Circuit Unauthorized Access: Stop further processing for unauthorized requests.
- Asynchronous Processing: Implement asynchronous methods to handle I/O operations without blocking other requests.
- Input Validation: Validate incoming request data and sanitize any unsafe input.
- Authentication Checks: Add early authentication checks to restrict access for unauthenticated users.
- Security Event Logging: Log security events for any blocked or failed requests.

Step 4: Testing Middleware Performance and Security

After writing the middleware components, follow these testing steps. You can use a tool like Postman or curl for testing, or adjust URLs directly in your browser.



Condition	URL Example	Expected Response
Simulated HTTPS Enforcement	`http://localhost:5294/`	"Simulated HTTPS Required" (400)
Default Route (authenticated)	`http://localhost:5294/?secure=true&authenticated=true`	"Processed Asynchronously" followed by "Final Response from Application"
Unauthorized Access	`http://localhost:5294/unauthorized?secure=true`	"Unauthorized Access" (401)
Invalid Input	`http://localhost:5294/?secure=true&input=<script>`	"Invalid Input" (400)
Access Denied (Unauthenticated)	`http://localhost:5294/?secure=true`	"Access Denied" (403)
Security Event Log	Any blocked request (400+ status)	Console log with security event details

Testing Steps

1. Default Route (Asynchronous Processing Test):
   a. URL: http://localhost:5294/?secure=true
   b. Expected Output: "Processed Asynchronously" followed by "Final Response from Application."
   c. Explanation: Confirms that asynchronous middleware is functioning as expected.
2. Simulated HTTPS Enforcement Test:
   a. URL: http://localhost:5294/?secure=true&authenticated=true

   b. Expected Output: "Simulated HTTPS Required" with a 400 status code.
   c. Explanation: Ensures the middleware blocks requests that don't include ?secure=true, simulating HTTPS enforcement.

3. Unauthorized Access Test:
   a. URL: http://localhost:5294/unauthorized?secure=true
   b. Expected Output: "Unauthorized Access" with a 401 status code
   c. Explanation: Tests that unauthorized requests are blocked early in the pipeline.
4. Invalid Input Test:
   a. URL: http://localhost:5294/?secure=true&input=<script>
   b. Expected Output: "Invalid Input" with a 400 status code.
   c. Explanation: This tests input validation by blocking unsafe input, such as JavaScript or HTML.
5. Access Denied Test:
   a. URL: Any URL without authentication setup, such as http://localhost:5294/?secure=true
   b. Expected Output: "Access Denied" with a 403 status code.
   c. Explanation: This middleware simulates access control, blocking unauthenticated requests by default.
6. Security Event Log Test:
   a. Trigger: Any request that results in a 400 or higher status code.
   b. Expected Output: Check the console in Visual Studio Code for log messages like: Security Event: /unauthorized - Status Code: 401
   c. Explanation: This middleware logs security-related events, providing feedback on blocked or failed requests.

Program.cs:

```csharp
using System.Text.RegularExpressions;

var builder = WebApplication.CreateBuilder(args);

builder.WebHost.ConfigureKestrel(options =>
{
    options.ListenLocalhost(5294);
});

var app = builder.Build();

app.Use(async (context, next) =>
{
    if (!context.Request.Query.TryGetValue("secure", out var secure) ||
secure != "true")
    {
        context.Response.StatusCode = 400;
        await context.Response.WriteAsync("Simulated HTTPS Required");
        Console.WriteLine($"Security Event: {context.Request.Path} -
Status Code: 400");
        return;
    }
    await next();
});

app.Use(async (context, next) =>
{
    if (context.Request.Path.StartsWithSegments("/unauthorized"))
    {
        context.Response.StatusCode = 401;
        await context.Response.WriteAsync("Unauthorized Access");
        Console.WriteLine($"Security Event: {context.Request.Path} -
Status Code: 401");
        return;
    }
    await next();
});

app.Use(async (context, next) =>
{
    if (context.Request.Query.TryGetValue("input", out var input))
    {
        if (Regex.IsMatch(input.ToString(), "<.*?>"))
        {
            context.Response.StatusCode = 400;
            await context.Response.WriteAsync("Invalid Input");
            Console.WriteLine($"Security Event: {context.Request.Path} -
Status Code: 400");
            return;
        }
    }
    await next();
});

app.Use(async (context, next) =>
{
    if (!context.Request.Query.TryGetValue("authenticated", out var 
authenticated) || authenticated != "true")
    {
        context.Response.StatusCode = 403;
        await context.Response.WriteAsync("Access Denied");
        Console.WriteLine($"Security Event: {context.Request.Path} - 
Status Code: 403");
        return;
    }
    await next();
});

app.Use(async (context, next) =>
{
    await Task.Delay(100);
    await context.Response.WriteAsync("Processed Asynchronously\n");
    await next();
});

app.Map("/", async context =>
{
    await context.Response.WriteAsync("Final Response from 
Application");
});

app.Run();
```