Implementing Middleware Components

Configuring Middleware in ASP.NET Core Application

Objective: set up and configure middleware components in an ASP.NET Core project. This lab will guide you through setting up a new ASP.NET Core project, configuring built-in middleware components, developing custom middleware, and testing the middleware pipeline.

Step 1: Prepare the Application

You'll create a small ASP.NET Core application using the command line. This application will integrate several built-in middleware components, and you'll add custom middleware for logging requests and response details.

Steps:

1. Open a terminal and navigate to your desired directory.
2. Create a new ASP.NET Core empty project with the following command: dotnet new web -o MyAspNetCoreApp
3. Navigate to the project folder: cd MyAspNetCoreApp
4. Open the project in your code editor (e.g., Visual Studio Code): code .
5. In the root directory, locate the Program.cs file. This file will be used to configure all middleware components.

Step 2: Configure Built-ln Middleware Components

In this step, you will configure essential built-in middleware components: Exception Handling, Authentication, Authorization, and Logging. You don't have to fully implement authentication and authorization. Just include the middleware components.

Steps:

1. In Program.cs, configure the exception handling middleware for production and development environments.
2. Add the authentication middleware that can be used to verify user identities.
3. Add authorization middleware that can be used to control user permissions.
4. Configure HTTP logging to capture request and response details. Add the HTTP logging service in builder.Services and apply the middleware.

Save your changes after each configuration so you can verify that everything is set up correctly.

Step 3: Develop Custom Middleware

You will now create custom middleware that logs the request path and response status, as well as middleware that records the request duration.

Steps:

1. In Program.cs, add custom middleware to log the request path and response status.
2. Add another custom middleware to track the request duration.

Save the changes once you've completed all the configurations.

Step 4: Test the Middleware Pipeline

With all middleware components in place, test the application to observe how requests and responses are processed.

Steps:

1. Run the application: dotnet run
2. Open a browser and make requests to the application, such as accessing http://localhost:5000.
3. Observe the logs in the terminal to confirm the output of the custom middleware, request timing, and built-in middleware.
4. Verify that error handling correctly redirects to an error page in production and shows detailed errors in development.

Example output (shown in the terminal in VS Code)

Request Path: /
Start Time: 10/29/2024 PM
Response Time: 8.319 ms
Response Status Code: 200
Request Path: /favicon . ico
Start Time: 10/29/2024 PM
Response Time: O. 084 ms
Response Status Code: 404

Program.cs:

```
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddHttpLogging(o => { });

builder.Services.AddAuthentication(CookieAuthenticationDefaults.Authenti
cationScheme)
    .AddCookie(o =>
    {
        o.LoginPath = null;
        o.AccessDeniedPath = null;
        o.Events.OnRedirectToLogin = ctx =>
        {
            ctx.Response.StatusCode = 403;
            return Task.CompletedTask;
        };
        o.Events.OnRedirectToAccessDenied = ctx =>
        {
            ctx.Response.StatusCode = 403;
            return Task.CompletedTask;
        };
    });

builder.Services.AddAuthorization(o =>
{
    o.AddPolicy("AdminsOnly", p => p.RequireRole("Admin"));
});

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}
else
{
    app.UseExceptionHandler("/error");
}

app.UseHttpLogging();
app.UseAuthentication();
app.UseAuthorization();

app.Use(async (context, next) =>
{
    var path = context.Request.Path;
    var method = context.Request.Method;
    var time = DateTime.Now.ToString("HH:mm:ss.fff");

    await next.Invoke();

    var status = context.Response.StatusCode;

    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("————————————————————————————");
```

```
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine($"[{time}] {method} {path}");
    Console.ForegroundColor = status >= 400 ? ConsoleColor.Red :
ConsoleColor.Yellow;
    Console.WriteLine($"Status: {status}");
    Console.ResetColor();

    if (path == "/admin" && status == 200)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("Conclusion: Admin access granted.");
        Console.ResetColor();
    }
    else if ((path == "/admin" || path == "/logout") && status == 403)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Conclusion: Access denied. User not
authorized.");
        Console.ResetColor();
    }
});

app.Use(async (context, next) =>
{
    var watch = System.Diagnostics.Stopwatch.StartNew();
    await next.Invoke();
    watch.Stop();

    Console.ForegroundColor = ConsoleColor.Magenta;
    Console.WriteLine($"Duration: {watch.ElapsedMilliseconds} ms");
    Console.WriteLine("————————————————————————————\n");
    Console.ResetColor();
});

app.MapGet("/", () => "Hello World");

// In practice this should be POST (but GET is used here for easy
browser testing)
app.MapGet("/login", async (HttpContext ctx) =>
{
    var claims = new List<System.Security.Claims.Claim>
    {
        new(System.Security.Claims.ClaimTypes.Name, "TestUser"),
        new(System.Security.Claims.ClaimTypes.Role, "Admin")
    };
    var identity = new System.Security.Claims.ClaimsIdentity(claims,
CookieAuthenticationDefaults.AuthenticationScheme);
    var principal = new
System.Security.Claims.ClaimsPrincipal(identity);
    await
ctx.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme,
principal);
    return Results.Ok("You are logged in as Admin.");
});

// In practice this should be POST (but GET is used here for easy
browser testing)
app.MapGet("/logout", async (HttpContext ctx) =>
{
```

```
    await
ctx.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
    return Results.Ok("You are logged out.");
}).RequireAuthorization("AdminsOnly");

app.MapGet("/admin", () => "Welcome,
Admin!").RequireAuthorization("AdminsOnly");

app.MapGet("/error", () => Results.Problem("An error occurred"));

app.Run();
```