

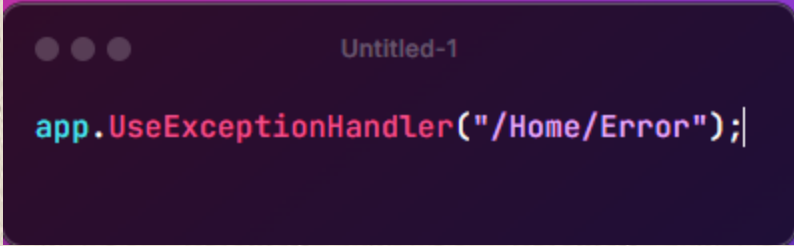
Order of middleware execution in asp.net

- ▣ In ASP.NET Core, middleware components are executed in the order they are added to the request pipeline. The order in which middleware components are added is critical because it determines how requests and responses are processed. Middleware components can handle requests, short-circuit the pipeline, and modify responses. Here's an overview of the typical order of middleware components in an ASP.NET Core application:

Typical Order of Middleware Components

▣ Exception Handling Middleware:

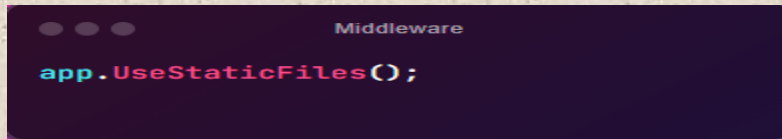
- Catches and handles exceptions globally.
- Added early to catch exceptions from subsequent middleware.

A screenshot of a code editor window titled "Untitled-1". The window has a dark background and shows a single line of code: `app.UseExceptionHandler("/Home/Error");`. The code is written in a light blue font, with the word "app" in a darker blue and "UseExceptionHandler" in a lighter blue. The path "/Home/Error" is in a light blue font, and the semicolon is in a light blue font.

```
app.UseExceptionHandler("/Home/Error");
```

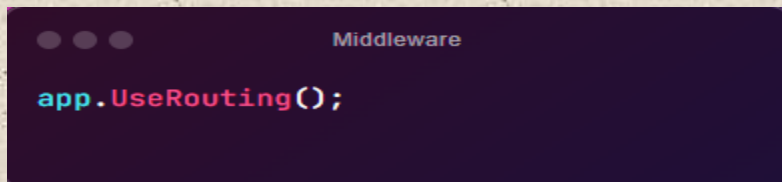
▣ Static Files Middleware:

- Serves static files such as HTML, CSS, JavaScript, and images.
- Should be placed early to serve static files efficiently.

A code editor window with a dark background. The title bar shows three colored dots (red, yellow, green) and the word "Middleware". The code is `app.UseStaticFiles();` in a light blue font.

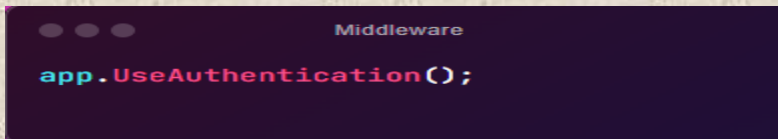
▣ Routing Middleware:

- Routes requests to the appropriate endpoint.
- Must be placed before any middleware that relies on routing (e.g., MVC, Razor Pages).

A code editor window with a dark background. The title bar shows three colored dots (red, yellow, green) and the word "Middleware". The code is `app.UseRouting();` in a light blue font.

▣ Authentication Middleware:

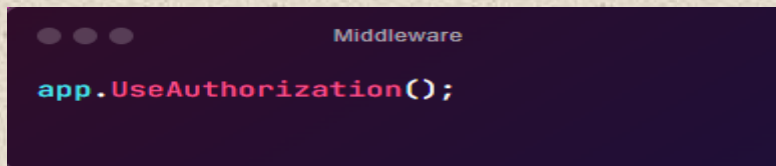
- Handles authentication.
- Must be added before authorization middleware.



A terminal window with a dark background and light gray window controls (three dots) in the top-left corner. The title bar of the window reads "Middleware". The terminal displays the command `app.UseAuthentication();` in a monospaced font, with "app" in blue, "UseAuthentication" in red, and the semicolon in white.

▣ Authorization Middleware:

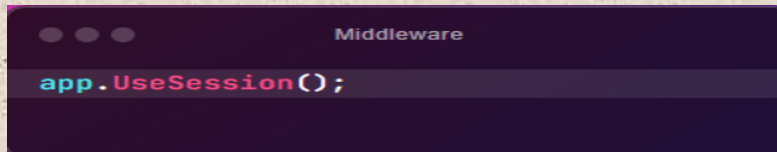
- Handles authorization.
- Ensures that users have the necessary permissions to access resources.



A terminal window with a dark background and light gray window controls (three dots) in the top-left corner. The title bar of the window reads "Middleware". The terminal displays the command `app.UseAuthorization();` in a monospaced font, with "app" in blue, "UseAuthorization" in red, and the semicolon in white.

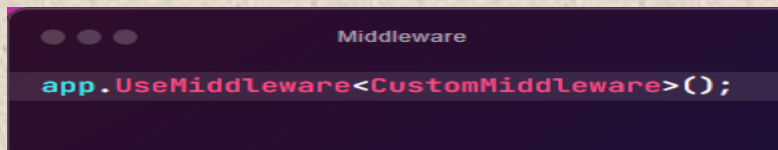
▣ Session Middleware:

- Manages user sessions.
- Should be added before any middleware that relies on session state.

A code editor window titled "Middleware" with three window control buttons in the top-left corner. It contains a single line of code: `app.UseSession();` in a syntax-highlighted font (blue for 'app', red for 'UseSession', and black for the rest).

▣ Custom Middleware:

- Custom middleware components can be added in any order as needed.
- The order of custom middleware should be carefully considered based on their functionality.

A code editor window titled "Middleware" with three window control buttons in the top-left corner. It contains a single line of code: `app.UseMiddleware<CustomMiddleware>();` in a syntax-highlighted font (blue for 'app', red for 'UseMiddleware', and black for the rest).

▣ Endpoint Middleware:

- Executes the endpoint associated with the request.
- Must be added last in the middleware pipeline.



```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
    endpoints.MapRazorPages();
});
```

▣ Example of Configuring Middleware in ASP.NET Core

```
Middleware

public class Startup
{
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();

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

        app.UseSession();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
            endpoints.MapRazorPages();
        });
    }
}
```

▣ **Important Points to Consider**

- ▣ **Order Matters:** The order in which middleware components are added using `app.Use...` methods is the order in which they are executed.
- ▣ **Short-Circuiting:** Middleware can short-circuit the pipeline, meaning that if a middleware component generates a response, it can prevent subsequent middleware from executing.
- ▣ **Custom Middleware:** When adding custom middleware, carefully consider its position in the pipeline to ensure it behaves as expected.
- ▣ By understanding the order of middleware and how to configure it properly, you can create efficient and maintainable ASP.NET Core applications.