ASP.NET Core,
Authentication & Authorization

Dominick Baier
https://leastprivilege.com
@leastprivilege

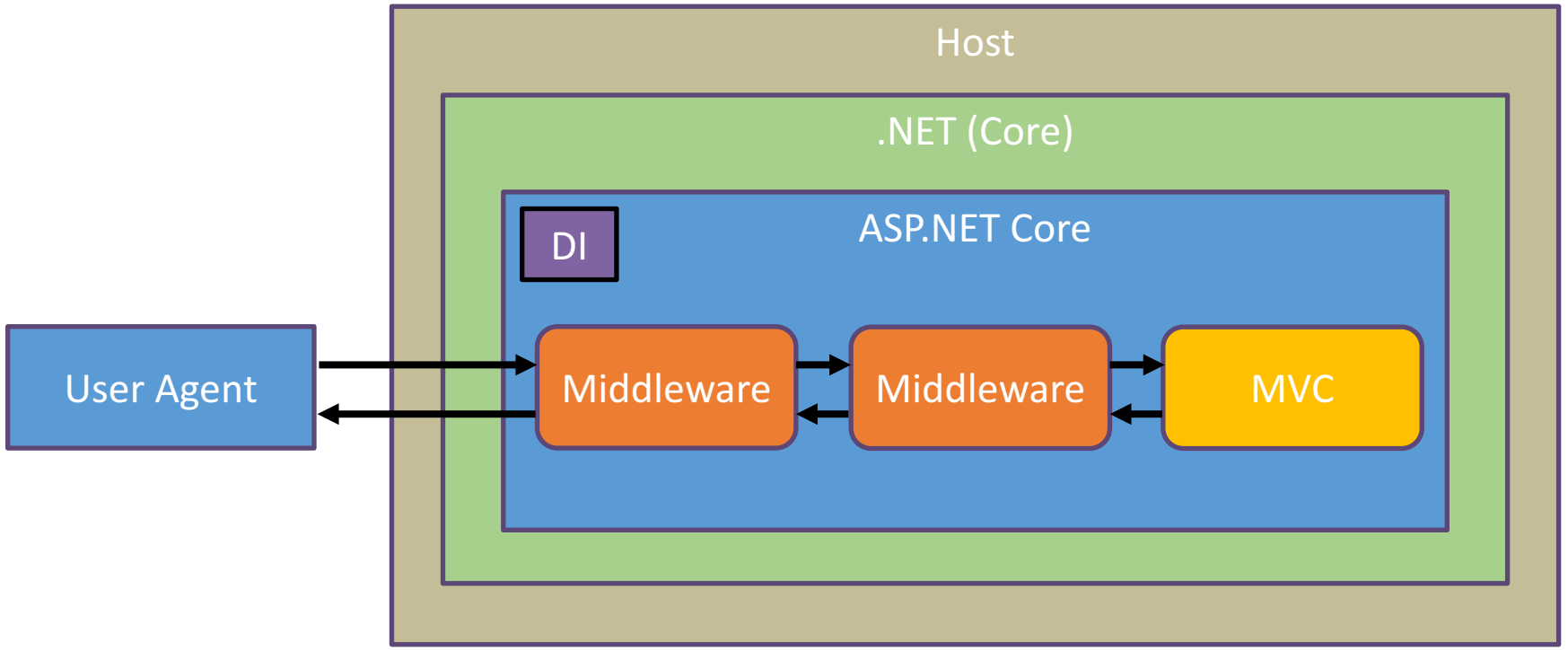
Brock Allen
https://brockallen.com
@brocklallen

What is ASP.NET Core?

- **Microsoft's new web framework**
  - Runs on .NET Core and the full .NET Framework
- **Middleware-based pipeline architecture**
  - Components that provide services for web applications
  - Many features packaged as middleware
- **Familiar HttpContext programming model**
  - But all new
- **Hosting is provided by Kestrel (by default)**
  - HTTP.SYS as a Windows-specific alternative

ASP.NET Core Architecture

- **ASP.NET Core is the runtime (hosted by .NET Core)**
- **MVC is Microsoft's primary application framework**
  - combines web UI & API



Loading ASP.NET Core

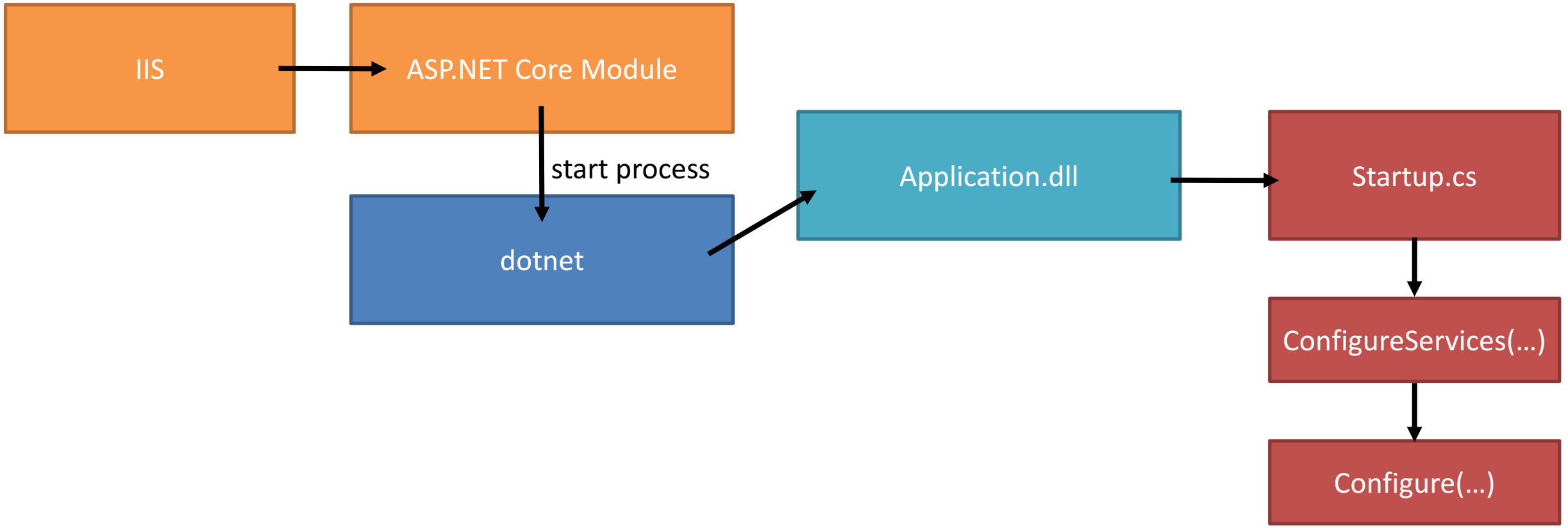
```
public class Program
{
    public static void Main(string[] args)
    {
        BuildWebHost(args).Run();
    }

    public static IWebHost BuildWebHost(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>()
            .Build();
}
```

Default Web Host

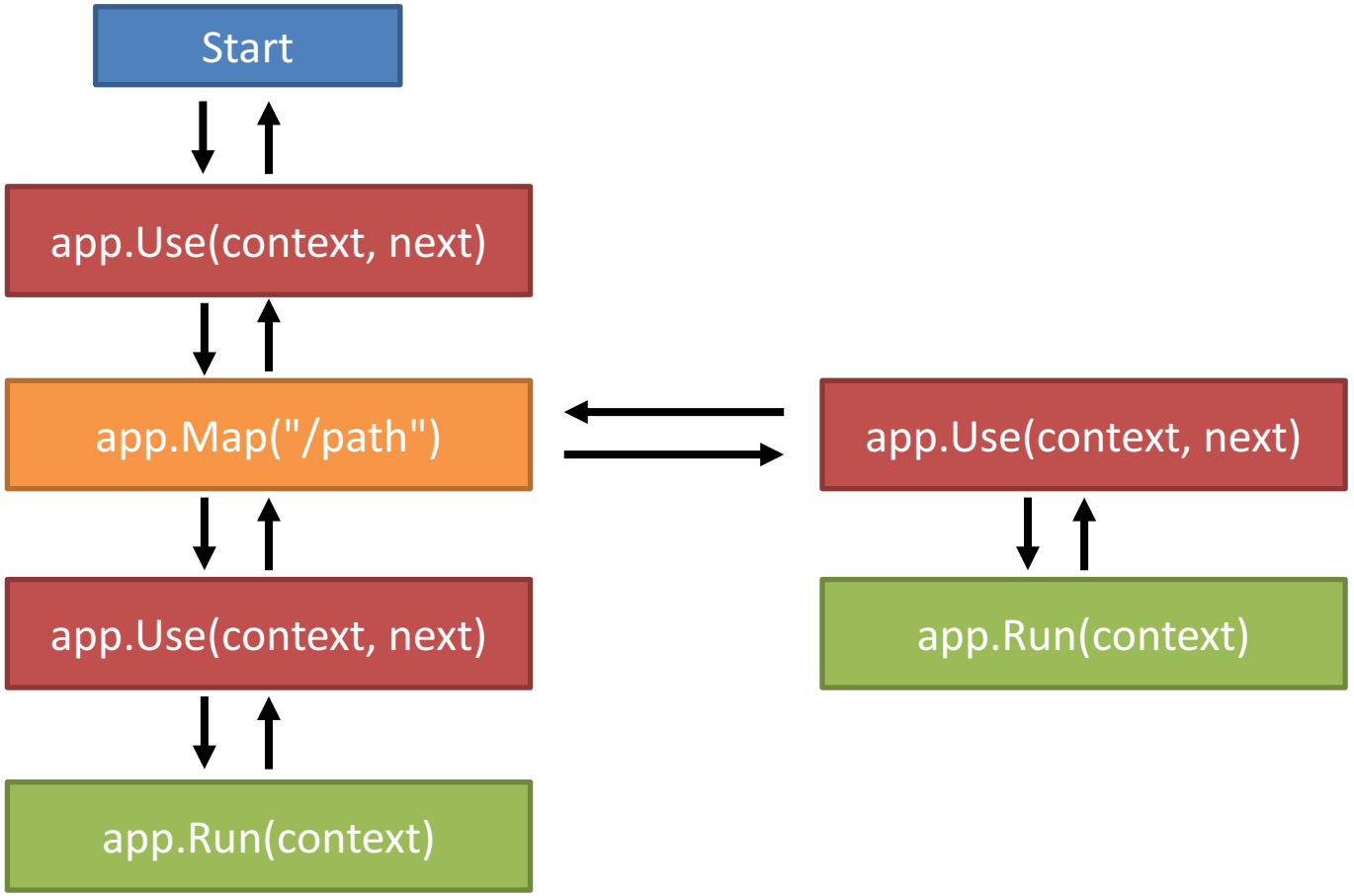
- **Convenience method for setting up a default host**
  - Reads hosting environment and URLs from environment variables
  - Sets up Kestrel with IIS integration
  - Set up configuration infrastructure
    - appsettings.json / appsettings.{environment}.json / environment variables
  - Sets up default logging
    - debug and console
  - Sets up user secrets
  - Sets up a developer exception page when environment is set to 'Development'

- **Can be customized**

How ASP.NET Core Applications start



Pipeline primitives



Run

```
namespace Microsoft.AspNetCore.Builder
{
    public delegate Task RequestDelegate(HttpContext context);
}
```

```
app.Run(async context =>
{
    await context.Response.WriteAsync("Hello ASP.NET Core");
});
```

Map

```
app.Map("/hello", helloApp =>
{
    helloApp.Run(async (HttpContext context) =>
    {
        await context.Response.WriteAsync("Hello ASP.NET Core");
    });
});
```

Use

```
app.Use(async (context, next) =>
{
    if (!context.Request.Path.Value.EndsWith("/favicon.ico"))
    {
        Console.WriteLine("pre");
        Console.WriteLine(context.Request.Path);

        await next();

        Console.WriteLine("post");
        Console.WriteLine(context.Response.StatusCode);
    }
    else
    {
        await next();
    }
});
```

Middleware classes

```
app.UseMiddleware<InspectionMiddleware>();
```

```
public class InspectionMiddleware
{
    private readonly RequestDelegate _next;

    public InspectionMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task Invoke(HttpContext context)
    {
        Console.WriteLine($"request: {context.Request.Path}");
        await _next(context);
    }
}
```

Authentication in ASP.NET Core

- **Combination of middleware and authentication handlers in DI**
  - middleware invokes handlers for request related processing
  - handlers can be also invoked manually

- **Handlers implement specific authentication methods**
  - Cookies for browser based authentication
  - Google, Facebook, and other social authentication
  - OpenId Connect for external authentication
  - JSON web token (JWT) for token-based authentication

Interacting with the authentication system

- **Extension methods on *HttpContext* call the *IAuthenticationService* in DI**

```
public static class AuthenticationHttpContextExtensions
{
    public static Task SignInAsync(this HttpContext context, ClaimsPrincipal principal) { }
    public static Task SignInAsync(this HttpContext context, string scheme, ClaimsPrincipal principal) { }

    public static Task SignOutAsync(this HttpContext context) { }
    public static Task SignOutAsync(this HttpContext context, string scheme) { }

    public static Task ChallengeAsync(this HttpContext context) { }
    public static Task ChallengeAsync(this HttpContext context, string scheme) { }

    public static Task ForbidAsync(this HttpContext context) { }
    public static Task ForbidAsync(this HttpContext context, string scheme) { }

    public static Task<AuthenticateResult> AuthenticateAsync(this HttpContext context) { }
    public static Task<AuthenticateResult> AuthenticateAsync(this HttpContext context, string scheme) { }
}
```

Setting up authentication

- **Global settings go into DI**
  - e.g. default schemes
- **Authentication middleware invokes handlers**

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication(options =>
    {
        options.DefaultScheme = "Cookies";
    });
}

public void Configure(IApplicationBuilder app)
{
    app.UseAuthentication();
}
```

Setting up authentication (2)

- **Scheme settings can be more fine-grained**

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = "...";

        options.DefaultSignInScheme = "...";
        options.DefaultSignOutScheme = "...";

        options.DefaultChallengeScheme = "...";
        options.DefaultForbidScheme = "...";

    });
}
```

Cookie Authentication

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication(defaultScheme: "Cookies")
        .AddCookie("Cookies", options =>
        {
            options.LoginPath = "/account/login";
            options.AccessDeniedPath = "/account/denied";

            options.Cookie.Name = "myapp";
            options.Cookie.Expiration = TimeSpan.FromHours(8);
            options.SlidingExpiration = false;
        });
}
```

Cookies: Logging in

- **SignInAsync issues cookie**
  - either using a named scheme, or default

```
var claims = new Claim[]
{
    new Claim("sub", "37734"),
    new Claim("name", "Brock Allen")
};

var ci = new ClaimsIdentity(claims, "password", "name", "role");
var cp = new ClaimsPrincipal(ci);

await HttpContext.SignInAsync(cp);
```

Cookies: Logging out

- **SignOutAsync removes cookie**

```
await HttpContext.SignOutAsync();
```

Claims Transformation

- **Per-request manipulation of principal & claims**
  - register an instance of *IClaimsTransformation* in DI
  - gets called from the handler's *AuthenticateAsync* method

```
public class ClaimsTransformer : IClaimsTransformation
{
    public async Task<ClaimsPrincipal> TransformAsync(ClaimsPrincipal principal)
    {
        return await CreateApplicationPrincipalAsync(principal);
    }
}
```

```
services.AddTransient<IClaimsTransformation, ClaimsTransformer>();
```

Data Protection

- **Used to protect cookies and other secrets**
  - *IDataProtection*Provider in DI
- **Uses a key container file**
  - stored outside of application directory*
  - uses a key ring with automatic rotation
  - keys *should* be protected
- **Needs to be synchronized between nodes in a farm**

* https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/configuration/default-settings

Authorization

- **Complete re-write**
  - better separation of business code and authorization logic
  - policy based authorization
  - resource/action based authorization
  - DI enabled

[Authorize]

- **Similar syntax**
  - roles still supported

```
[Authorize]
public class HomeController : Controller
{
    [AllowAnonymous]
    public IActionResult Index()
    {
        return View();
    }

    [Authorize(Roles = "Sales")]
    public IActionResult About()
    {
        return View(User);
    }
}
```

Authorization policies

Startup

```
services.AddAuthorization(options =>
{
    options.AddPolicy("ManageCustomers", policy =>
    {
        policy.RequireAuthenticatedUser();
        policy.RequireClaim("department", "sales");
        policy.RequireClaim("status", "senior");
    });
});
```

Controller

```
[Authorize("ManageCustomers")]
public IActionResult Manage()
{
    // stuff
}
```

Programmatically using policies

```
public class CustomerController : Controller
{
    private readonly IAuthorizationService _authz;

    public CustomerController(IAuthorizationService authz)
    {
        _authz = authz;
    }

    public async Task<IActionResult> Manage()
    {
        var result = await _authz.AuthorizeAsync(User, "ManageCustomers");
        if (result.Succeeded) return View();

        return Forbid();
    }
}
```

…or from a View

```
@using Microsoft.AspNetCore.Authorization
@inject IAuthorizationService _authz

@if ((await _authz.AuthorizeAsync(User, "ManageCustomers")).Succeeded)
{
    <div>
        <a href="/customers/test">Manage</a>
    </div>
}
```

Custom Requirements

```
public class JobLevelRequirement : IAuthorizationRequirement
{
    public JobLevel Level { get; }

    public JobLevelRequirement(JobLevel level)
    {
        Level = level;
    }
}

public static class StatusPolicyBuilderExtensions
{
    public static AuthorizationPolicyBuilder RequireJobLevel(
      this AuthorizationPolicyBuilder builder, JobLevel level)
    {
        builder.AddRequirements(new JobLevelRequirement(level));
        return builder;
    }
}
```

Handling Requirements

```
public class JobLevelRequirementHandler : AuthorizationHandler<JobLevelRequirement>
{
    private readonly IOrganizationService _service;

    public JobLevelRequirementHandler(IOrganizationService service)
    {
        _service = service;
    }

    protected override void Handle(
        AuthorizationContext context, JobLevelRequirement requirement)
    {
        var currentLevel = _service.GetJobLevel(context.User);

        if (currentLevel == requirement.Level)
        {
            context.Succeed(requirement);
        }
    }
}
```

Resource-based Authorization

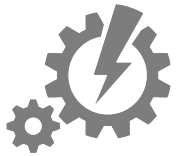
Subject

- client ID
- subject ID
- scopes

- more claims

+ DI

Operation



- read
- write
- send via email
- ...

Object

- ID
- owner

- more properties

+ DI

Example: Document resource

```
public class DocumentAuthorizationHandler :
    AuthorizationHandler<OperationAuthorizationRequirement, Document>
{
    public override Task HandleRequirementAsync(
      AuthorizationHandlerContext context,
      OperationAuthorizationRequirement operation,
      Document resource)
    {
        // authorization logic
    }
}
```

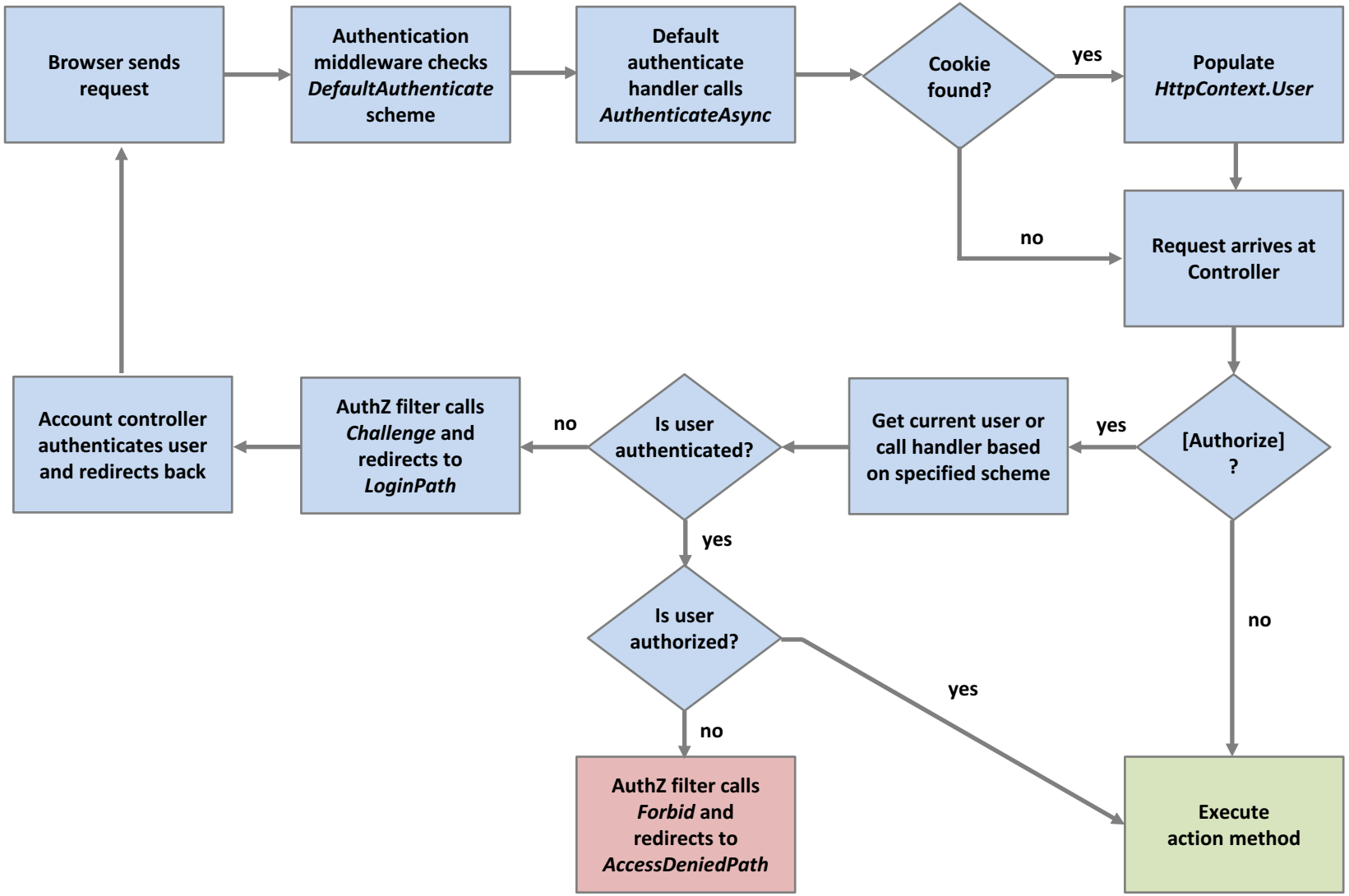
Add handler in DI:

```
services.AddTransient<IAuthorizationHandler, DocumentAuthorizationHandler>();
```

Invoking the authorization handler

```
public class DocumentController : Controller
{
    private readonly IAuthorizationService _authz;

    public DocumentController(IAuthorizationService authz)
    {
        _authz = authz;
    }

    public async Task<IActionResult> Update(Document doc)
    {
        if ((await _authz.AuthorizeAsync(User, doc, Operations.Update)).Failure)
        {
            return Forbid();
        }

        // do stuff
    }
}
```

Summary: Cookies & Authorization



External Authentication

- **ASP.NET Core supports**
  - Google, Twitter, Facebook, Microsoft Account
  - OpenID Connect & JSON Web Tokens

- **New generic OAuth 2.0 handler makes integration with other proprietary providers easier**
  - LinkedIn, Slack, Spotify, WordPress, Yahoo, Github, Instragram, BattleNet, Dropbox, Paypal, Vimeo…

https://github.com/aspnet-contrib/AspNet.Security.OAuth.Providers

Social Identity Providers

- **Enabled with *AddGoogle*, et al.**
  - Rely upon cookie authentication handler for sign-in

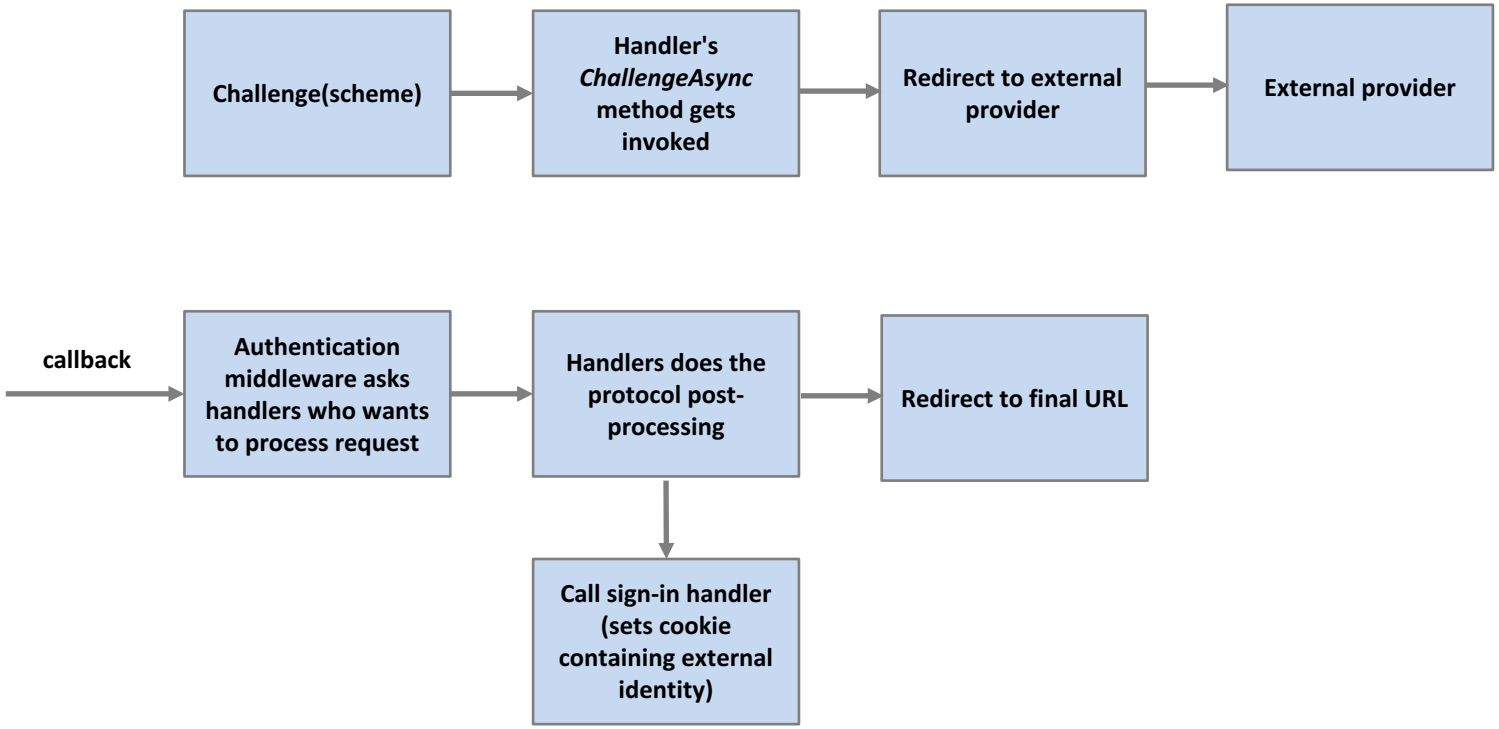
```
services.AddAuthentication("Cookies")
    .AddCookie("Cookies", options =>
    {
        options.LoginPath = "/account/login";
        options.AccessDeniedPath = "/account/denied";
    })
    .AddGoogle("Google", options =>
    {
        options.ClientId = "...";
        options.ClientSecret = "...";
    });
```

Social Identity Providers

- **Challenge triggers redirect for login**
  - Control URL user returns to and state with *AuthenticationProperties*
  - MVC *ChallengeResult* works with action result architecture

```
var props = new AuthenticationProperties
{
    RedirectUri = "/Home/Secure"
};
await HttpContext.ChallengeAsync("Google", props);

// or if using MVC:

return Challenge("Google", props);
```

Summary: External Authentication



External authentication with Callback

- **Add application level post-processing step**
  - provision logic, extra UI etc..
- **Second cookie handler to temporarily store external identity**

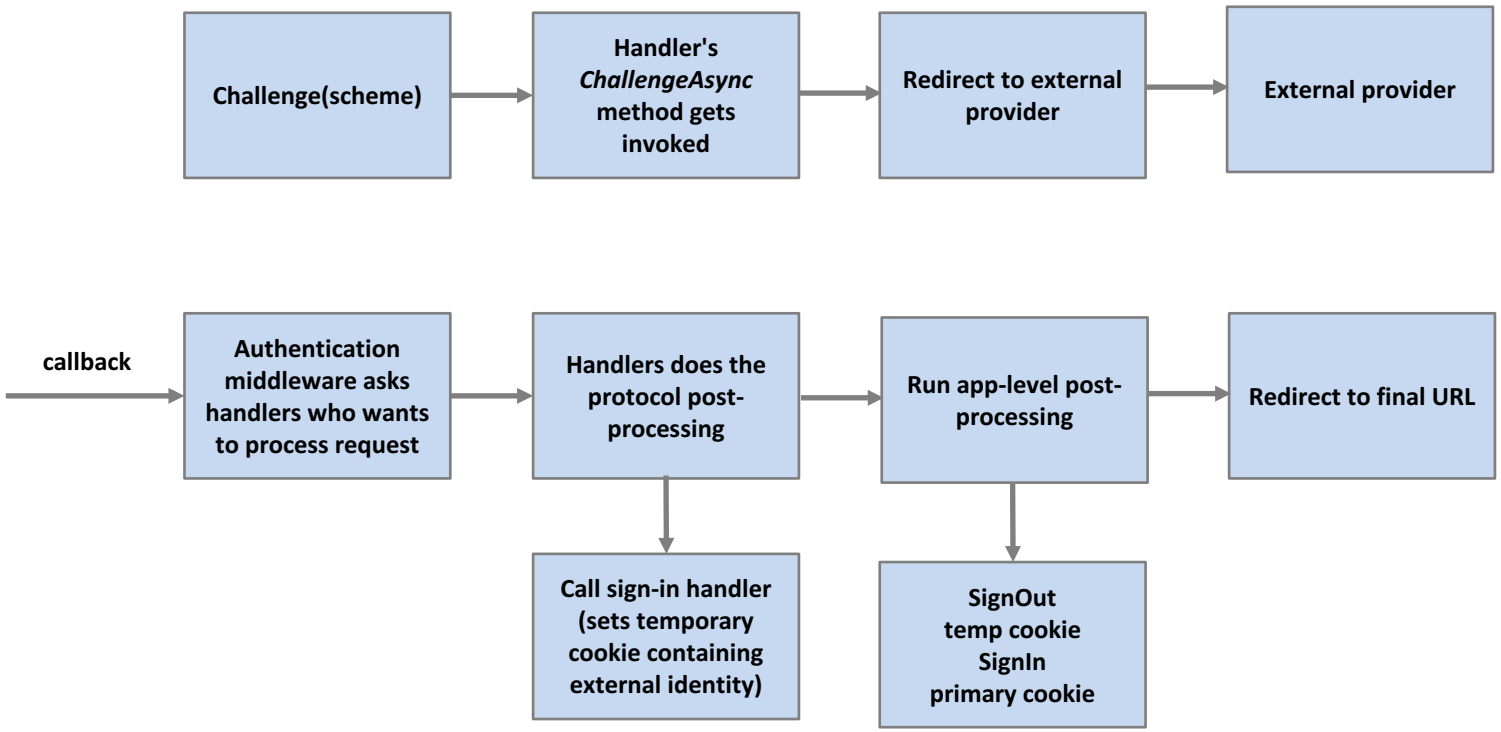
```
services.AddAuthentication("Cookies")
    .AddCookie("Cookies")
    .AddCookie("Temp")

    .AddGoogle("Google", options =>
    {
        options.SignInScheme = "Temp";

        options.ClientId = "...";
        options.ClientSecret = "...";
    });
```

Mixing local and external Authentication

- **Redirect page performs post-processing logic**
  - *AuthenticateAsync* triggers temp cookie handler
  - Run post-processing logic / flow
  - Use primary cookie handler to log user in (and remove temp cookie)

```
var result = await HttpContext.AuthenticateAsync("Temp");

var userId = result.Principal.FindFirst(ClaimTypes.NameIdentifier);
var extProvider = userId.Issuer;

// post-processing workflow

var user = new ClaimsPrincipal(...);
await HttpContext.SignInAsync(user);
await HttpContext.SignOutAsync("Temp");
```

Summary: External Authentication with Callback



Summary

- **ASP.NET Core is a new modular HTTP pipeline**
  - Middleware is central to the architecture
- **Authentication is implemented as combination of middleware and handlers**
- **IAuthenticationService coordinates authentication handlers**
- **Policy- and resource-based authorization improvements**