ASP.NET Core 2.0 Authentication and Authorization System

- There is a component that exists in ASP.NET Core that conjures up an enchanted shield that protects portions (or all) of your website from unauthorized access.
- They can be broken down into identity, verbs, authentication handlers, and middleware.
- Since ASP.NET Core's most common authentication handler is the Cookies auth handler, these examples will use cookie authentication.

Identity

- Key to understanding how authentication works is to first understand what an identity is in ASP.NET Core 2.0.
- There are three classes which represent the identity of a user: **Claim**, **ClaimsIdentity**, and **ClaimsPrincipal**

Claims

- A ***Claim*** represents a single fact about the user.
- It could be the user's first name, last name, age, employer, birth date, or anything else that is true about the user.
- A single claim will contain only a single piece of information.
- A claim representing something about a user Binod Thapa could be his first name: Binod. A second claim would be his last name: Thapa.
- Claims are represented by the Claim class in ASP.Net Core. It's most common constructor accepts two strings: **type** and **value**.

CONTD…

- The 'type' parameter is the name of the claim, while the value is the information the claim is representing about the user.

```
//This claim uses a standard string. Claim type is 'FullName
new Claim("FullName",“Binod Thapa");


//This claim type expands to
'http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress'
new Claim(ClaimTypes.Email,“binod.thapa@gmail.com");
```

Claimsidentity

- An Identity represents a form of identification or, in other words, a single way of proving who you are.
- In real life this could be a driver's license. In ASP.Net Core, it is a ClaimsIdentity.
- This class represents a single form of digital identification.
- A single instance of a ClaimsIdentity can be authenticated or not authenticated.
- Simply setting the AuthenticationType will automatically ensure the IsAuthenticated property is true.

CONTD…

- This is because if you have authenticated the identity in any way, then it must, by definition, be authenticated.
- A driver's license contains many claims about its subject: first and last names, birthdate, hair and eye colors, height, and others.
- Similarly, a ClaimsIdentity can contain numerous claims about a user.

CONTD…

- A Principal represents the actual user.
- It can contain one or more instances of ClaimsIdentity, just like in life a person may have a driver's license, citizenship, voter card, and a passport.
- Each of the identities is used for a different purpose and may contain a unique set of claims, but they all identify the same user in some form or another.
- A ClaimsPrincipal represents a user and contains one or more instances of ClaimsIdentity, which in turn represent a single form of identification and contain one or more instances of Claim, which represents a single piece of information about a user.

CONTD…

The ClaimsPrincipal is what the HttpContext.SignInAsync method accepts and passes to the specified AuthenticationHandler.

Verbs

- There are 5 verbs (commands or behaviors) that are invoked by the auth system, and are not necessarily called in order.
- These are all independent actions that do not communicate among themselves, however, when used together allow users to sign in and access pages otherwise denied.

Authenticate

- Gets the user's information if any exists (e.g. decoding the user's cookie, if one exists)

Challenge

- Requests authentication by the user (e.g. showing a login page)

CONTD…

SignIn

- Persists the user’s information somewhere (e.g. writes a cookies)

SignOut

- Removes the user’s persisted information (e.g. deletes the cookies)

Forbid

- Denies access to a resource for unauthenticated users or authenticated but unauthorized users (e.g. displaying a “not authorized” page)

Authentication Handlers

- Authentication handlers are components that actually implement the behavior of the 5 verbs above.
- The default auth handler provided by ASP.NET Core is the Cookies authentication handler which implements all 5 of the verbs.
- Oauth handlers, for instance, do not implement the SignIn verb, but rather pass off that responsibility to another auth handler, such as the Cookies auth handler.

CONTD…

- Authentication handlers must be registered with the auth system in order to be used and are associated with schemes.
- A scheme is just a string that identifies a unique auth handler in a dictionary of auth handlers.
- The default scheme for the Cookies auth handler is "Cookies", but it can be changed to anything.

Authentication Middleware

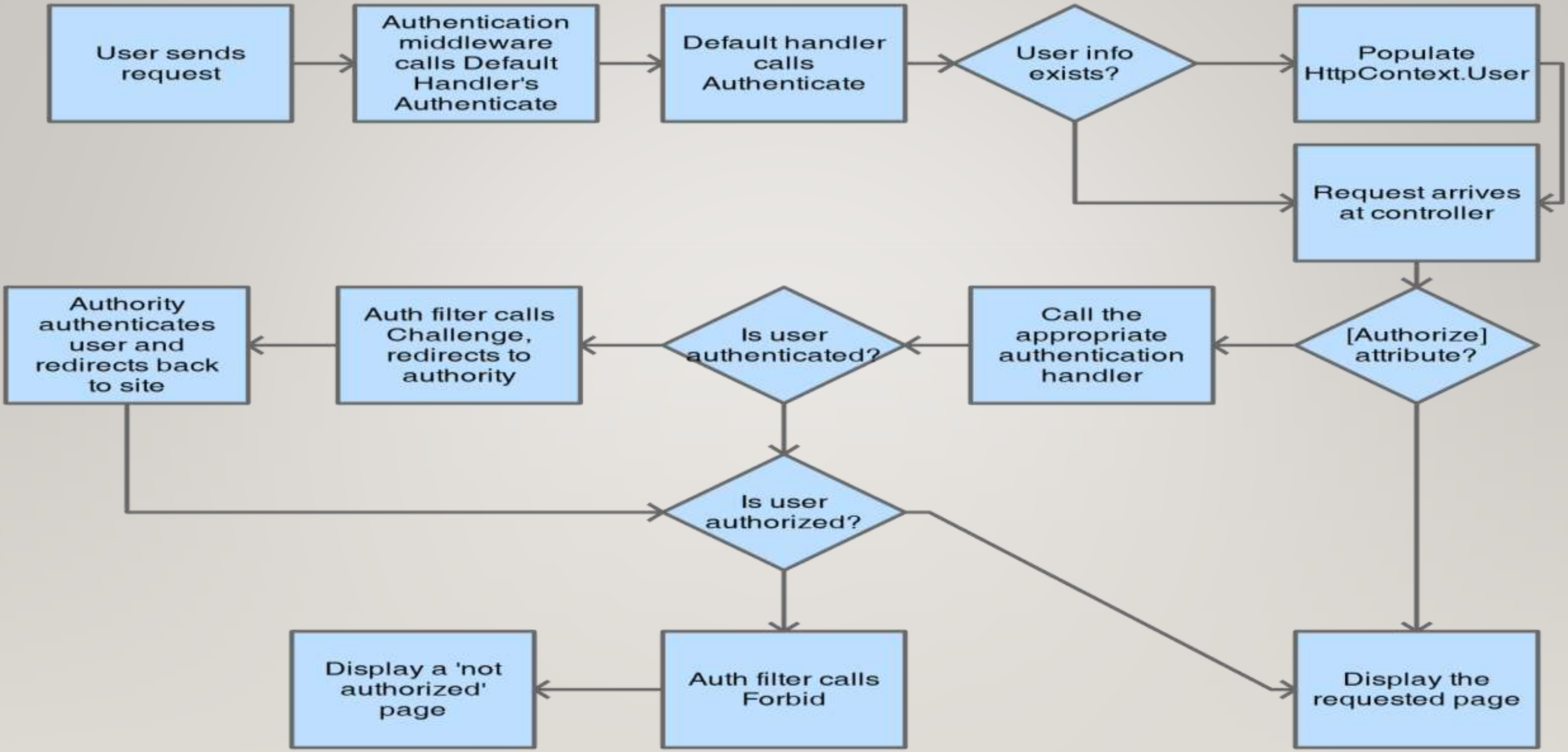
- A middleware is a module that can be inserted into the startup sequence and is run on every request.
- This code checks if a user is authenticated (or not) on every request.
- Recall that the Authenticate verb gets the user info, but only if it exists.
- When the request is run, the authentication middleware asks the default scheme auth handler to run its authentication code.
- The auth handler returns the information to the authentication middleware which then populates the HttpContext.User object with the returned information.

Authentication And Authorization Flow

- **[Authorize]** can be used in controller or action of controller
- The request arrives at the server.
- The authentication middleware calls the default handler's Authenticate method and populates the HttpContext.User object with any available information.
- The request arrives at the controller action.
- If the action is not decorated with the [Authorize] attribute, display the page and stop here.

CONTD…

- If the action is decorated with [Authorize], the auth filter checks if the user was authenticated.
- If the user was not, the auth filter calls Challenge, redirecting to the appropriate signin authority.
- Once the signin authority directs the user back to the app, the auth filter checks if the user is authorized to view the page.
- If the user is authorized, it displays the page, otherwise it calls Forbid, which displays a 'not authorized' page.


User sends request
Authentication middleware calls Default Handler's Authenticate
Default handler calls Authenticate
User info exists?
Populate HttpContext.User
Request arrives at controller
[Authorize] attribute?
Call the appropriate authentication handler
Is user authenticated?
Auth filter calls Challenge, redirects to authority
Authority authenticates user and redirects back to site
Is user authorized?
Auth filter calls Forbid
Display a 'not authorized' page
Display the requested page

Startup Class

```
public void ConfigureServices(IServiceCollection services) {
   //Adds cookie middleware to the services collection and configures it
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
        .AddCookie(options => options.LoginPath = new PathString("/account/login"));
}
public void Configure(IApplicationBuilder app, IHostingEnvironment env) {
     //Adds the authentication middleware to the pipeline
     app.UseAuthentication();
}
```

CONTD…

- In ConfigureServices the AddAuthentication method is called to add the authentication middleware to the services collection.
- There is a chained method call to AddCookie that adds a Cookies authentication handler with an option configured to the authentication middlware.
- In the Configure method, the method UseAuthentication is called to add the authentication middleware to the execution pipeline.
- This is what allows the authentication middleware to actually run on every request.

Applicationuser Class

- It is a representation of a user and stores a username and password for a user.

```
public class ApplicationUser
{
    public string UserName { get; set; }
    public string Password { get; set; }
    public ApplicationUser() { }
    public ApplicationUser(string username, string password) {
        this.UserName = username;
        this.Password = password;
    }
}
```

Class Accountcontroller

- In order to do anything meaningful with the authentication middleware and handler, some actions are needed.
- Below is an MVC Controller called AccountController which contains methods that do the work of signing in and out.
- This class handles the verbs SignIn and SignOut through the HttpContext's convenience methods, which in turn invoke the SignInAsync and SignOutAsync methods on the specified or default auth handler.

```
[HttpPost]
    public async Task<IActionResult> Login(ApplicationUser user, string returnUrl = null) {
        const string badUserNameOrPasswordMessage = "Username or password is incorrect.";
        if (user == null) {
            return BadRequest(badUserNameOrPasswordMessage);
        }
        var lookupUser = Users.FirstOrDefault(u => u.UserName == user.UserName);

        if (lookupUser?.Password != user.Password) {
            return BadRequest(badUserNameOrPasswordMessage);
        }
 var identity = new ClaimsIdentity(CookieAuthenticationDefaults.AuthenticationScheme);
        identity.AddClaim(new Claim(ClaimTypes.Name, lookupUser.UserName));
await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new
ClaimsPrincipal(identity));
 if(returnUrl == null) {
            returnUrl = TempData["returnUrl"]?.ToString();
        }
 if(returnUrl != null) {
            return Redirect(returnUrl);
        }

        return RedirectToAction(nameof(HomeController.Index), "Home");
    }
public async Task<IActionResult> Logout() {
        await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
        return RedirectToAction(nameof(HomeController.Index), "Home");
    }
}
```