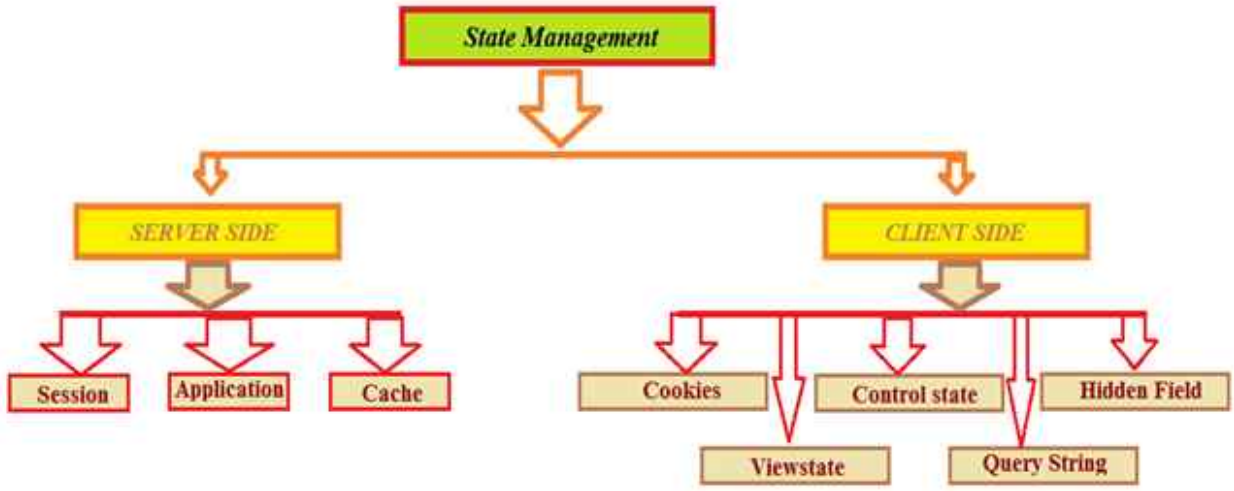
State Management in .NET

https://medium.com/@prashantramnyc/difference-between-session-cookies-vs-jwt-json-web-tokens-for-session-management-4be67d2f066e



Client Side State Management:

1. **Hidden field:** <input type=”hidden” id=”myid” runat=”server”/>
   The html element will be hidden, but developers can see it in the code and use the value of the field.
2. **ViewState:** present in asp.net web forms its an object
3. **QueryStrings:** we can pass data using query string by specifying key value pairs in url
4. **ControlState:** present in asp.net web forms
5. **Cookies:** Cookies are small pieces of data stored on the client-side (usually in the user's web browser) by websites to remember certain information about the user or their interactions with the website. They are commonly used for session management, user authentication, tracking user preferences, and personalizing user experiences
   Cookie is included in the header.The Set-Cookie header is used to specify cookie attributes.

   Session Cookies: These cookies are temporary and exist only for the duration of a user's session. They are stored in memory and are typically deleted when the browser is closed or the session ends.

Persistent Cookies: Persistent cookies have an expiration date set in the future, allowing them to persist even after the browser is closed. They are useful for remembering user preferences or maintaining long-term session information.

Advantage: can be used for session management, remembering user preferences
Disadvantage: Size limitation(few kbs), privacy concern as they track user interaction and data

Cookies in Practice:

a. Add service in program.cs

```
//service for cookie authentication
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
.AddCookie(CookieAuthenticationDefaults.AuthenticationScheme, (option) =>
    {
        option.SlidingExpiration = true;
        option.ExpireTimeSpan = new TimeSpan(0, 10, 0);//0hr 10min 0sec
    });
//to access httpcontext class outside controller
builder.Services.AddHttpContextAccessor();
```

app.UseAuthentication();

b. Controller:

```
public class AuthController : ControllerBase
{
    private readonly IAuthService _authService;
    public AuthController(IAuthService authService)
    {
        _authService = authService;
```

```
    }
    [HttpPost("Login")]
    [AllowAnonymous]
    public async Task<IActionResult> Login()
    {
        bool res=await _authService.Login();
        return Ok(new {Message="login success"});
    }
    [Authorize(Roles = "Admin")]
    [HttpGet("GetData")]
    public IActionResult GetData()
    {
        return Ok(new { Message = "Your data" });
    }
}
```

C. Service class

```
public class AuthService:IAuthService
{
    private readonly IHttpContextAccessor _httpContextAccessor;
    public AuthService(IHttpContextAccessor httpContextAccessor)
    {
        _httpContextAccessor = httpContextAccessor;
    }
    public async Task<bool> Login()
    {
        List<Claim> claims = new List<Claim>()
        {
            new Claim(ClaimTypes.Email,"amangaur@gmail.com"),
            new Claim(ClaimTypes.Role,"Admin"),
        };

        ClaimsIdentity claimsIdentity = new
ClaimsIdentity(claims,CookieAuthenticationDefaults.AuthenticationScheme);
        //store cookie in browser
```

```
        await
_httpContextAccessor.HttpContext.SignInAsync(CookieAuthenticationDefaults.
AuthenticationScheme,
            new ClaimsPrincipal(claimsIdentity));

        return true;
    }
}
```

HttpContext

- represents the context of an individual HTTP request being processed by the server
- HttpContext encapsulates the HTTP request (HttpRequest) and response (HttpResponse) associated with the current HTTP transaction.
- It provides properties and methods to access request headers, query parameters, form data, route data, cookies, and other request-specific information.
- **IHttpContextAccessor:** It provides access to HttpContext, when we need to use HttpContext outside controller class we use this.

Sessions in Asp.Net Core:

1. Add service for session:

```
//service for session
builder.Services.AddDistributedMemoryCache();

builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromMinutes(30);//by default it
is 20 min
});
```

2. Add session middleware:

```
app.UseSession();
app.UseHttpsRedirection();

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

3. Controller class:

```
public async Task<IActionResult> Login()
{
    HttpContext.Session.SetString("Mykey", "Aman Gaur");
    string data = HttpContext.Session.GetString("MyKey");
    Console.WriteLine(data);
    bool res=await _authService.Login();
    return Ok(new {Message="login success"});
}
```

- Session are more secure as they store user info on server side.
- After successful authentication, a session is created for the user. A session represents a logical connection between the client (web browser) and the server and is used to store user-specific data during the user's interaction with the application.
- The server generates a unique session identifier (session ID) for the user's session. This session ID is typically stored in a cookie on the client-side (browser) and sent back to the server with each subsequent request.