Session and app state

ASP.NET Core

State management

Storage approach	**Storage mechanism**
Cookies	HTTP cookies (may include data stored using server-side app code)
Session state	HTTP cookies and server-side app code
TempData	HTTP cookies or session state
Query strings	HTTP query strings
Hidden fields	HTTP form fields
HttpContext.Items	Server-side app code
Cache	Server-side app code
Dependency Injection	Server-side app code

Approaches to preserve user data and app state between requests

Cookies

Cookies store data across requests. Because cookies are sent with every request, their size should be kept to a minimum. Ideally, only an identifier should be stored in a cookie with the data stored by the app. Most browsers restrict cookie size to 4096 bytes. Only a limited number of cookies are available for each domain.

Because cookies are subject to tampering, they must be validated by the app. Cookies can be deleted by users and expire on clients. However, cookies are generally the most durable form of data persistence on the client.

Cookies are often used for personalization, where content is customized for a known user. The user is only identified and not authenticated in most cases. The cookie can store the user's name, account name, or unique user ID (such as a GUID). You can then use the cookie to access the user's personalized settings, such as their preferred website background color.

Session state

Session state is an ASP.NET Core scenario for storage of user data while the user browses a web app. Session state uses a store maintained by the app to persist data across requests from a client. The session data is backed by a cache and considered ephemeral data—the site should continue to function without the session data. Critical application data should be stored in the user database and cached in session only as a performance optimization.

ASP.NET Core maintains session state by providing a cookie to the client that contains a session ID, which is sent to the app with each request. The app uses the session ID to fetch the session data.

Session state behaviors

- Because the session cookie is specific to the browser, sessions aren't shared across browsers.
- Session cookies are deleted when the browser session ends.
- If a cookie is received for an expired session, a new session is created that uses the same session cookie.
- Empty sessions aren't retained—the session must have at least one value set into it to persist the session across requests. When a session isn't retained, a new session ID is generated for each new request.
- The app retains a session for a limited time after the last request. The app either sets the session timeout or uses the default value of 20 minutes. Session state is ideal for storing user data that's specific to a particular session but where the data doesn't require permanent storage across sessions.
- Session data is deleted either when the ISession.Clear implementation is called or when the session expires.
- There's no default mechanism to inform app code that a client browser has been closed or when the session cookie is deleted or expired on the client.

Distributed apps

The in-memory cache provider stores session data in the memory of the server where the app resides. In a server farm scenario:

- Use *sticky sessions* to tie each session to a specific app instance on an individual server. Azure App Service uses Application Request Routing (ARR) to enforce sticky sessions by default. However, sticky sessions can affect scalability and complicate web app updates. A better approach is to use a Redis or SQL Server distributed cache, which doesn't require sticky sessions. For more information, see Distributed caching in ASP.NET Core.
- The session cookie is encrypted via IDataProtector. Data Protection must be properly configured to read session cookies on each machine. For more information, see ASP.NET Core Data Protection and Key storage providers.

Configure session state

The Microsoft.AspNetCore.Session package, which is included in the Microsoft.AspNetCore.App metapackage, provides middleware for managing session state. To enable the session middleware, `Startup` must contain:

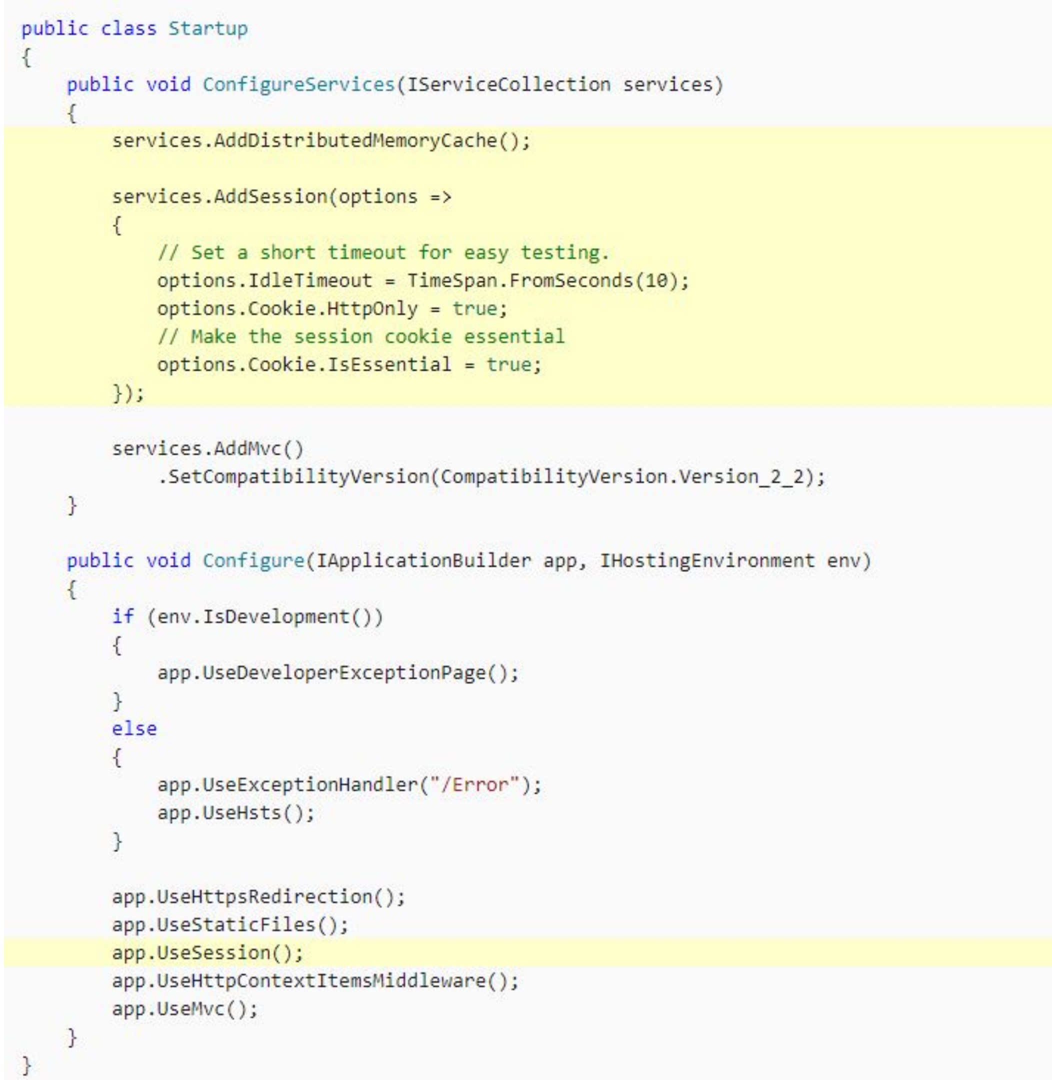
- Any of the IDistributedCache memory caches. The `IDistributedCache` implementation is used as a backing store for session.
- A call to AddSession in `ConfigureServices`.
- A call to UseSession in `Configure`.

The order of middleware is important

HttpContext.Session is available after session state is configured.

`HttpContext.Session` can't be accessed before `UseSession` has been called.

A new session with a new session cookie can't be created after the app has begun writing to the response stream. The exception is recorded in the web server log and not displayed in the browser.



```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDistributedMemoryCache();

        services.AddSession(options =>
        {
            // Set a short timeout for easy testing.
            options.IdleTimeout = TimeSpan.FromSeconds(10);
            options.Cookie.HttpOnly = true;
            // Make the session cookie essential
            options.Cookie.IsEssential = true;
        });

        services.AddMvc()
            .SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
    }

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

        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseSession();
        app.UseHttpContextItemsMiddleware();
        app.UseMvc();
    }
}
```

Session options

Option	Description
Cookie	Determines the settings used to create the cookie. Name defaults to SessionDefaults.CookieName (`.AspNetCore.Session`). Path defaults to SessionDefaults.CookiePath (`/`). SameSite defaults to SameSiteMode.Lax (`1`). HttpOnly defaults to `true`. IsEssential defaults to `false`.
IdleTimeout	The `IdleTimeout` indicates how long the session can be idle before its contents are abandoned. Each session access resets the timeout. This setting only applies to the content of the session, not the cookie. The default is 20 minutes.
IOTimeout	The maximum amount of time allowed to load a session from the store or to commit it back to the store. This setting may only apply to asynchronous operations. This timeout can be disabled using InfiniteTimeSpan. The default is 1 minute.

Set and get Session values

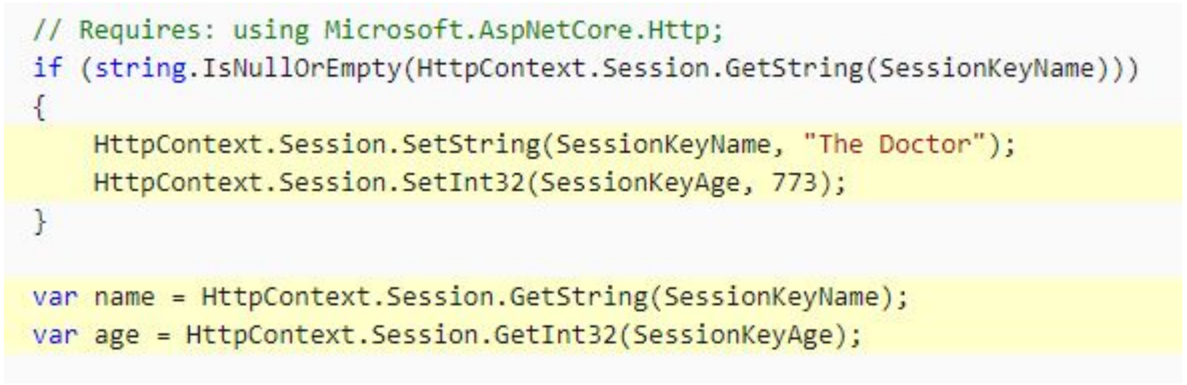
Session state is accessed from a Razor Pages PageModel class or MVC Controller class with HttpContext.Session. This property is an ISession implementation.

The `ISession` implementation provides several extension methods to set and retrieve integer and string values. The extension methods are in the Microsoft.AspNetCore.Http namespace (add a `using Microsoft.AspNetCore.Http`;statement to gain access to the extension methods) when the Microsoft.AspNetCore.Http.Extensions package is referenced by the project. Both packages are included in the Microsoft.AspNetCore.App metapackage.

`ISession` extension methods:

- Get(ISession, String)
- GetInt32(ISession, String)
- GetString(ISession, String)
- SetInt32(ISession, String, Int32)
- SetString(ISession, String, String)

how to set and get an integer and a string



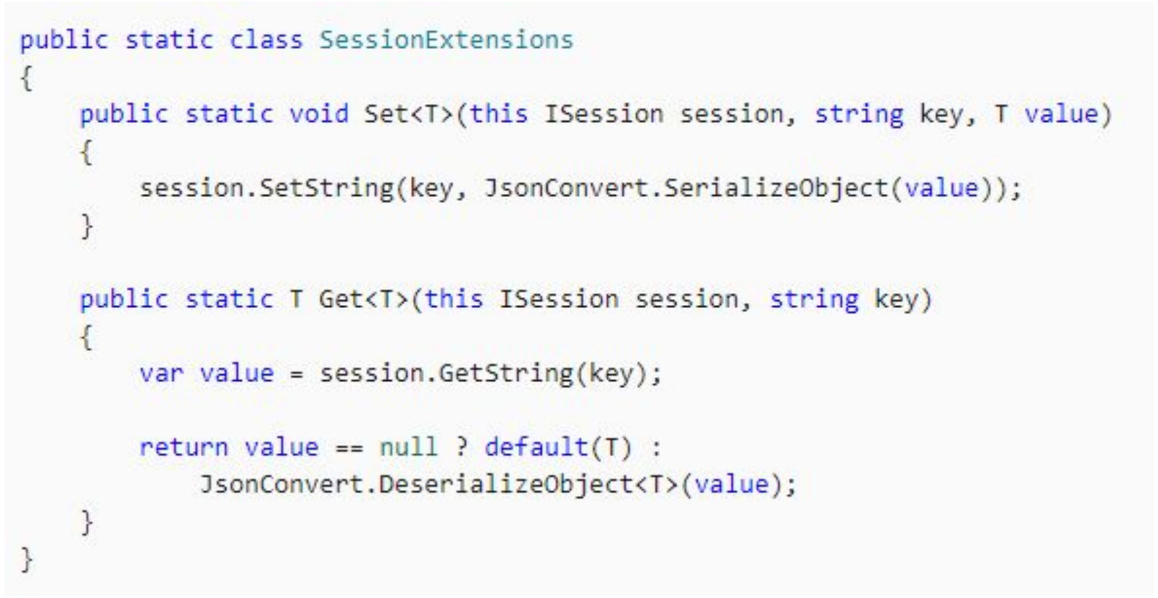
```
// Requires: using Microsoft.AspNetCore.Http;
if (string.IsNullOrEmpty(HttpContext.Session.GetString(SessionKeyName)))
{
    HttpContext.Session.SetString(SessionKeyName, "The Doctor");
    HttpContext.Session.SetInt32(SessionKeyAge, 773);
}

var name = HttpContext.Session.GetString(SessionKeyName);
var age = HttpContext.Session.GetInt32(SessionKeyAge);
```

Complex types

All session data must be serialized to enable a distributed cache scenario, even when using the in-memory cache. Minimal string and number serializers are provided .

Complex types must be serialized by the user using another mechanism, such as JSON.



```
public static class SessionExtensions
{
    public static void Set<T>(this ISession session, string key, T value)
    {
        session.SetString(key, JsonConvert.SerializeObject(value));
    }

    public static T Get<T>(this ISession session, string key)
    {
        var value = session.GetString(key);

        return value == null ? default(T) :
            JsonConvert.DeserializeObject<T>(value);
    }
}
```

TempData

ASP.NET Core exposes the Razor Pages TempData or Controller TempData. This property stores data until it's read in another request. Keep(String) and Peek(string) methods can be used to examine the data without deletion at the end of the request. Keep() marks all items in the dictionary for retention. `TempData` is particularly useful for redirection when data is required for more than a single request. `TempData` is implemented by `TempData` providers using either cookies or session state.

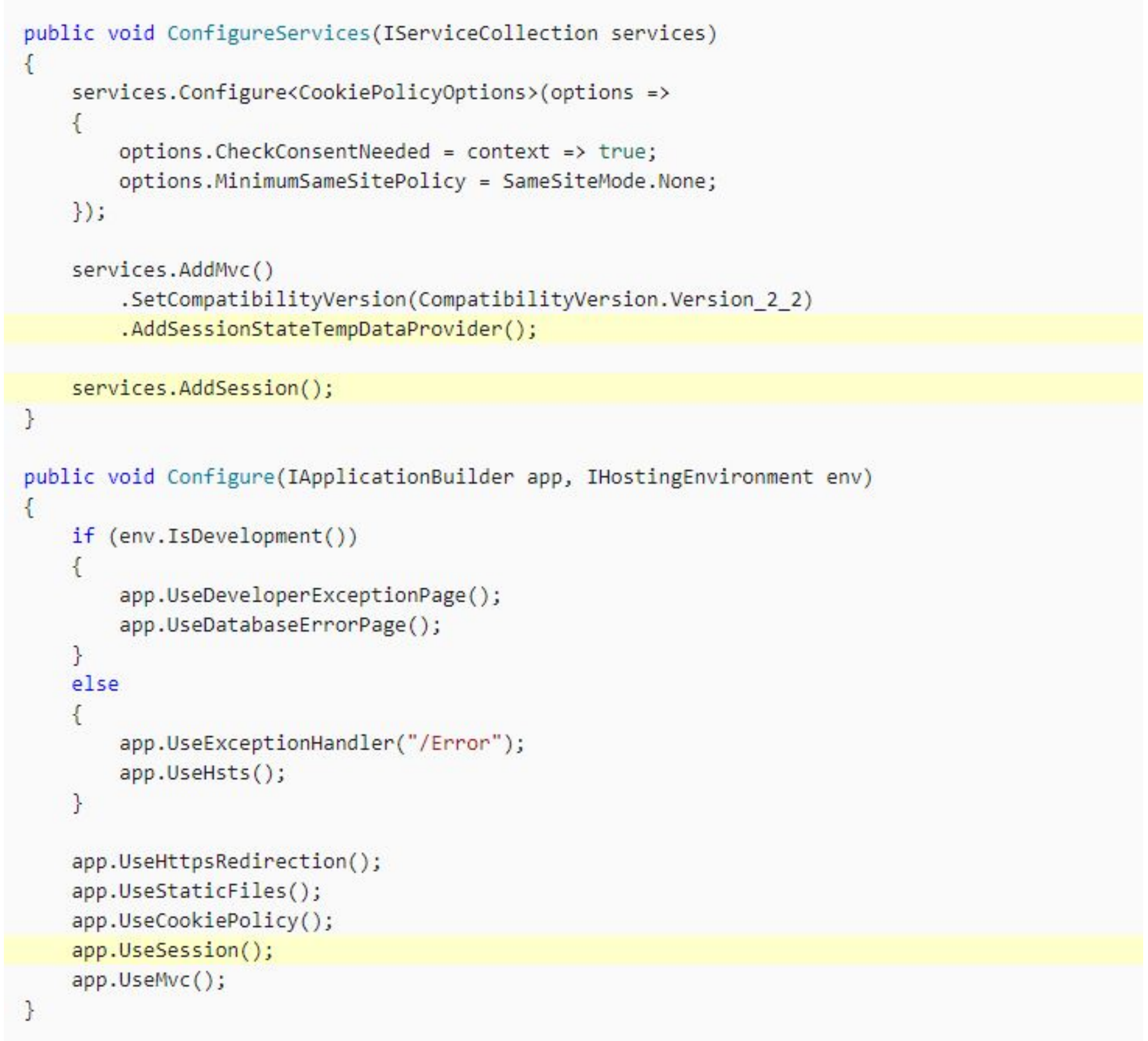
TempData providers

The cookie-based TempData provider is used by default to store TempData in cookies.

The cookie data is encrypted using IDataProtector, encoded with Base64UrlTextEncoder, then chunked

Choosing a TempData provider involves several considerations, such as:

1. Does the app already use session state? If so, using the session state TempData provider has no additional cost to the app (aside from the size of the data).
2. Does the app use TempData only sparingly for relatively small amounts of data (up to 500 bytes)? If so, the cookie TempData provider adds a small cost to each request that carries TempData. If not, the session state TempData provider can be beneficial to avoid round-tripping a large amount of data in each request until the TempData is consumed.
3. Does the app run in a server farm on multiple servers? If so, there's no additional configuration required to use the cookie TempData provider outside of Data Protection



```csharp
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
        .AddSessionStateTempDataProvider();

    services.AddSession();
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();
    app.UseSession();
    app.UseMvc();
}
```

Query strings

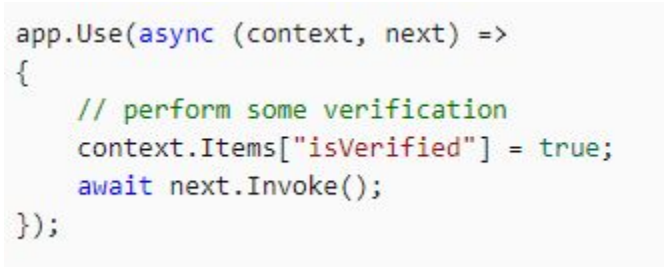
A limited amount of data can be passed from one request to another by adding it to the new request's query string. This is useful for capturing state in a persistent manner that allows links with embedded state to be shared through email or social networks. Because URL query strings are public, never use query strings for sensitive data.

Hidden fields

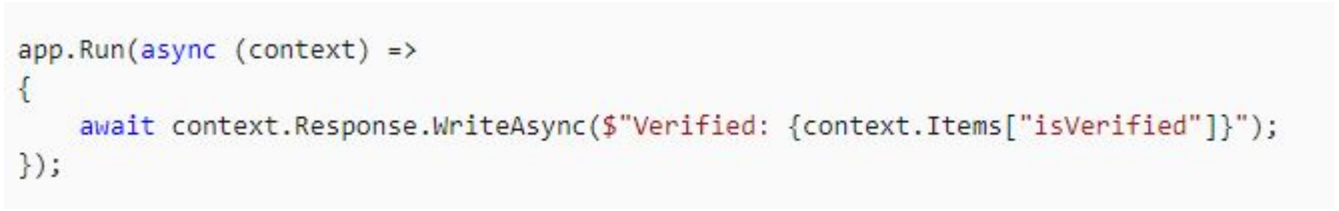
Data can be saved in hidden form fields and posted back on the next request. This is common in multi-page forms. Because the client can potentially tamper with the data, the app must always revalidate the data stored in hidden fields.

HttpContext.Items

The HttpContext.Items collection is used to store data while processing a single request. The collection's contents are discarded after a request is processed. The `Items` collection is often used to allow components or middleware to communicate when they operate at different points in time during a request and have no direct way to pass parameters.



```
app.Use(async (context, next) =>
{
    // perform some verification
    context.Items["isVerified"] = true;
    await next.Invoke();
});
```



```
app.Run(async (context) =>
{
    await context.Response.WriteAsync($"Verified: {context.Items["isVerified"]}");
});
```

Cache

Caching is an efficient way to store and retrieve data. The app can control the lifetime of cached items.

Cached data isn't associated with a specific request, user, or session. Be careful not to cache user-specific data that may be retrieved by other users' requests.

Common errors

- "Unable to resolve service for type 'Microsoft.Extensions.Caching.Distributed.IDistributedCache' while attempting to activate 'Microsoft.AspNetCore.Session.DistributedSessionStore'."
  This is usually caused by failing to configure at least one `IDistributedCache` implementation. For more information, see Distributed caching in ASP.NET Core and Cache in-memory in ASP.NET Core.
- In the event that the session middleware fails to persist a session (for example, if the backing store isn't available), the middleware logs the exception and the request continues normally. This leads to unpredictable behavior.
  For example, a user stores a shopping cart in session. The user adds an item to the cart but the commit fails. The app doesn't know about the failure so it reports to the user that the item was added to their cart, which isn't true.
  The recommended approach to check for errors is to call `await feature.Session.CommitAsync()`;from app code when the app is done writing to the session. `CommitAsync` throws an exception if the backing store is unavailable. If `CommitAsync` fails, the app can process the exception. `LoadAsync` throws under the same conditions where the data store is unavailable.

Resources

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-3.0