**State management**

State can be stored using several approaches.

| **Storage approach** | **Storage mechanism** |
| --- | --- |
| [Cookies](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-2.1#cookies) | HTTP cookies (may include data stored using server-side app code) |
| [Session state](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-2.1#session-state) | HTTP cookies and server-side app code |
| [TempData](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-2.1#tempdata) | HTTP cookies or session state |
| [Query strings](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-2.1#query-strings) | HTTP query strings |
| [Hidden fields](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-2.1#hidden-fields) | HTTP form fields |
| [HttpContext.Items](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-2.1#httpcontextitems) | Server-side app code |
| [Cache](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-2.1#cache) | Server-side app code |
| [Dependency Injection](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-2.1#dependency-injection) | Server-side app code |

**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.

**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.

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 exhibits the following 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](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.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.

The [Microsoft.AspNetCore.Session](https://www.nuget.org/packages/Microsoft.AspNetCore.Session/) package, which is included in the [Microsoft.AspNetCore.App metapackage](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/metapackage-app?view=aspnetcore-2.1), provides middleware for managing session state. To enable the session middleware, Startup must contain:

* Any of the [IDistributedCache](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.caching.distributed.idistributedcache) memory caches. The IDistributedCacheimplementation is used as a backing store for session.
* A call to [AddSession](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.sessionservicecollectionextensions.addsession) in ConfigureServices.
* A call to [UseSession](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.builder.sessionmiddlewareextensions" \l "methods_) in Configure.

### Set and get Session values

Session state is accessed from a Razor Pages [PageModel](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.razorpages.pagemodel) class or MVC [Controller](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.controller) class with [HttpContext.Session](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.httpcontext.session). This property is an [ISession](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.isession) implementation.

The ISession implementation provides several extension methods to set and retreive integer and string values. The extension methods are in the [Microsoft.AspNetCore.Http](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http)namespace (add a using Microsoft.AspNetCore.Http; statement to gain access to the extension methods) when the [Microsoft.AspNetCore.Http.Extensions](https://www.nuget.org/packages/Microsoft.AspNetCore.Http.Extensions/) package is referenced by the project. Both packages are included in the [Microsoft.AspNetCore.App metapackage](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/metapackage-app?view=aspnetcore-2.1).

**TempData**

ASP.NET Core exposes the [TempData property of a Razor Pages page model](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.razorpages.pagemodel.tempdata) or [TempData of an MVC controller](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.controller.tempdata). This property stores data until it's read. The [Keep](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.viewfeatures.itempdatadictionary.keep) and [Peek](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.viewfeatures.itempdatadictionary.peek) methods can be used to examine the data without deletion. 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**

In ASP.NET Core 2.0 or later, the cookie-based TempData provider is used by default to store TempData in cookies.

**Choose a TempData provider**

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

services.AddMvc()

.SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1)

.AddSessionStateTempDataProvider();

services.AddSession();

**HttpContext.Items**

The [HttpContext.Items](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.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.

In the following example, [middleware](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/index?view=aspnetcore-2.1) adds isVerified to the Items collection.

app.Use(async (context, next) =>

{

// perform some verification

context.Items["isVerified"] = true;

await next.Invoke();

});

**Dependency Injection**

Use Dependency Injection to make data available to all users:

1. Define a service containing the data. For example, a class named MyAppData is defined:

public class MyAppData

{

// Declare properties and methods

}

1. Add the service class to Startup.ConfigureServices:

public void ConfigureServices(IServiceCollection services)

{

services.AddSingleton<MyAppData>();

}

1. Consume the data service class:

public class IndexModel : PageModel

{

public IndexModel(MyAppData myService)

{

// Do something with the service

// Examples: Read data, store in a field or property

}

}

**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.**

Caching can significantly improve the performance and scalability of an app by reducing the work required to generate content. Caching works best with data that changes infrequently. Caching makes a copy of data that can be returned much faster than from the original source. You should write and test your app to never depend on cached data.

ASP.NET Core supports several different caches. The simplest cache is based on the IMemoryCache, which represents a cache stored in the memory of the web server. Apps which run on a server farm of multiple servers should ensure that sessions are sticky when using the in-memory cache. Sticky sessions ensure that subsequent requests from a client all go to the same server.

Microsoft.Extensions.Caching.Memory/IMemoryCache (described in this topic) is recommended over System.Runtime.Caching/MemoryCache because it's better integrated into ASP.NET Core. For example, IMemoryCache works natively with ASP.NET Core dependency injection.

**Using IMemoryCache**

In-memory caching is a service that's referenced from your app using [Dependency Injection](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-2.1). Call AddMemoryCache in ConfigureServices: