ASP.NET Core Entity Framework Core Session Management

FSMVÜ Computer Eng. Dr. Öğr Üyesi Ali NİZAM

State management

- State can be stored using several approaches. Each approach is described later in this topic.

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

State management

- A common requirement of web applications is the need to store temporary state data.
- One of the simplest methods is to use **querystring** parameters or post data to send state to subsequent requests. However doing so requires sending that data to the user's browser, which may not be desirable, especially for sensitive data. For that reason, extra care must be taken when using this approach.
- **Cookies** can also be used to store small bits of data, though again, these make a roundtrip to the user's browser, so must be kept small, and if sensitive, must be secured.
- For each request there exists a property Items on **HttpContext**. This is an IDictionary<string, object> which can be used to store arbitrary objects against a string key. The data stored here lasts for just a single request.
- **Files** and **database storage** can obviously be used to store state data, whether related to a particular user or the application in general. However they are typically slower to store and retrieve data than other available options.
- **Session state** relies on a cookie identifier to identify a particular browser session, and stores data related to the session on the server.

https://andrewlock.net/an-introduction-to-session-storage-in-asp-net-core/

What Are Cookies?

- Cookies are small files which are stored on a user's computer. They are designed to hold a modest amount of data specific to a particular client and website, and can be accessed either by the web server or the client computer.
- This allows the server to deliver a page tailored to a particular user, or the page itself can contain some script which is aware of the data in the cookie and so is able to carry information from one visit to the website (or related site) to the next.

http://www.whatarecookies.com/

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.

Session state

- 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.
- The ASP.NET Core MVC and Razor pages templates include support for General Data Protection Regulation (GDPR). Session state cookies aren't marked essential by default, so session state isn't functional unless tracking is permitted by the site visitor.

Session support

- The HttpContext object contains a Session property of type ISession.
- The get and set portion of the interface is shown below (see the full interface here):

```
public interface ISession
{
    bool TryGetValue(string key, out byte[] value);
    void Set(string key, byte[] value);
    void Remove(string key);
}
```

Sensitive data

- Don't store sensitive data in session state. The user might not close the browser and clear the session cookie. Some browsers maintain valid session cookies across browser windows. A session might not be restricted to a single user—the next user might continue to browse the app with the same session cookie.
- 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

- Any of the IDistributedCache memory caches. The IDistributedCache implementation is used as a backing store for session. For more information, see Dağıtılmış önbelleğe alma ASP.NET Core.
- A call to AddSession in ConfigureServices.
- A call to UseSession in Configure.

Configure session state

```
public void ConfigureServices(IServiceCollection services)
{
     …
    services.AddSession(options =>
    {
        options.Cookie.Name = ".AdventureWorks.Session";
        options.IdleTimeout = TimeSpan.FromSeconds(10);
        options.Cookie.IsEssential = true;
    });
}
```

Use session

- Finally, configure the session middleware in the Startup.Configure method. As with all middleware, order is important in this method, so you will need to enable the session before you try and access it, e.g. in your MVC middleware:

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();

    //enable session before MVC
    app.UseSession();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

- With all this in place, the Session object can be used to store our data.

SessionOptions.

- **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

```
public void OnGet()
  {
      // 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);
```

Serialization

- 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 (see the methods and extension methods of ISession). Complex types must be serialized by the user using another mechanism, such as JSON.
- Add the following extension methods to set and get serializable objects:

```
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

- MVC also exposes a TempData property on a Controller which is an additional wrapper around Session. This can be used for storing transient data that only needs to be available for a single request after the current one.
- Keys inside TempData will be removed if you read them once!

TempData

- TempData keeps data for the time of HTTP Request, which means that it holds data between two consecutive requests.
- TempData helps to transfer data between controllers or between actions.
- TempData internally use Session variables.
- Note that TempData is only work during the current and subsequent request. It is generally used to store one time messages.
- With the help of the TempData.Keep() method we can keep value in TempData object after request completion.

https://www.c-sharpcorner.com/blogs/viewdata-vs-viewbag-vs-tempdata-in-mvc1

Configure Temp Data Provider

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

Temp Data Example

```
//Controller Code
public ActionResult Index()
{
    List<string> Student = new List<string>();
    Student.Add("Jignesh");
    Student.Add("Tejas");
    Student.Add("Rakesh");

    TempData["Student"] = Student;
    return View();
}
//page code
<ul>
    <% foreach (var student in TempData["Student"] as List<string>)
        { %>
    <li><%: student%></li>
    <% } %>
</ul>
```

Dependency Injection

- Use Dependency Injection to make data available to all users

DI : Define a service containing the data

- public class MyAppData
- {
-     // Declare properties and methods
- }

DI: Add Services

- public void ConfigureServices(IServiceCollection services)
- {
-     services.AddSingleton<MyAppData>();
- }

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
    }
}
```

Basic Example

- https://www.youtube.com/watch?v=ZBMy24rZX9U

Questions