MICROSOFT ARCHITECT
By Joydip Kanjilal, Columnist, InfoWorld
NOV 12, 2018 3:00 AM PST

How to use session storage in ASP.Net Core

Take advantage of session storage middleware in ASP.Net Core to store user-specific data and session state on the server

To store user-specific data in ASP.Net Core web applications, we use the session state. However, using session state in ASP.Net core is not straightforward—at least, session state in ASP.Net Core doesn't work the way it used to work in legacy ASP.Net applications. This article examines how we can work with session state in ASP.Net Core.

[Go deeper: How to get started with WebAssembly. • What's next for WebAssembly. • 8 projects that give WebAssembly a lift • So, What's next for WebAssembly, exactly? | Keep up with hot topics in programming with InfoWorld's App Dev Report newsletter.]

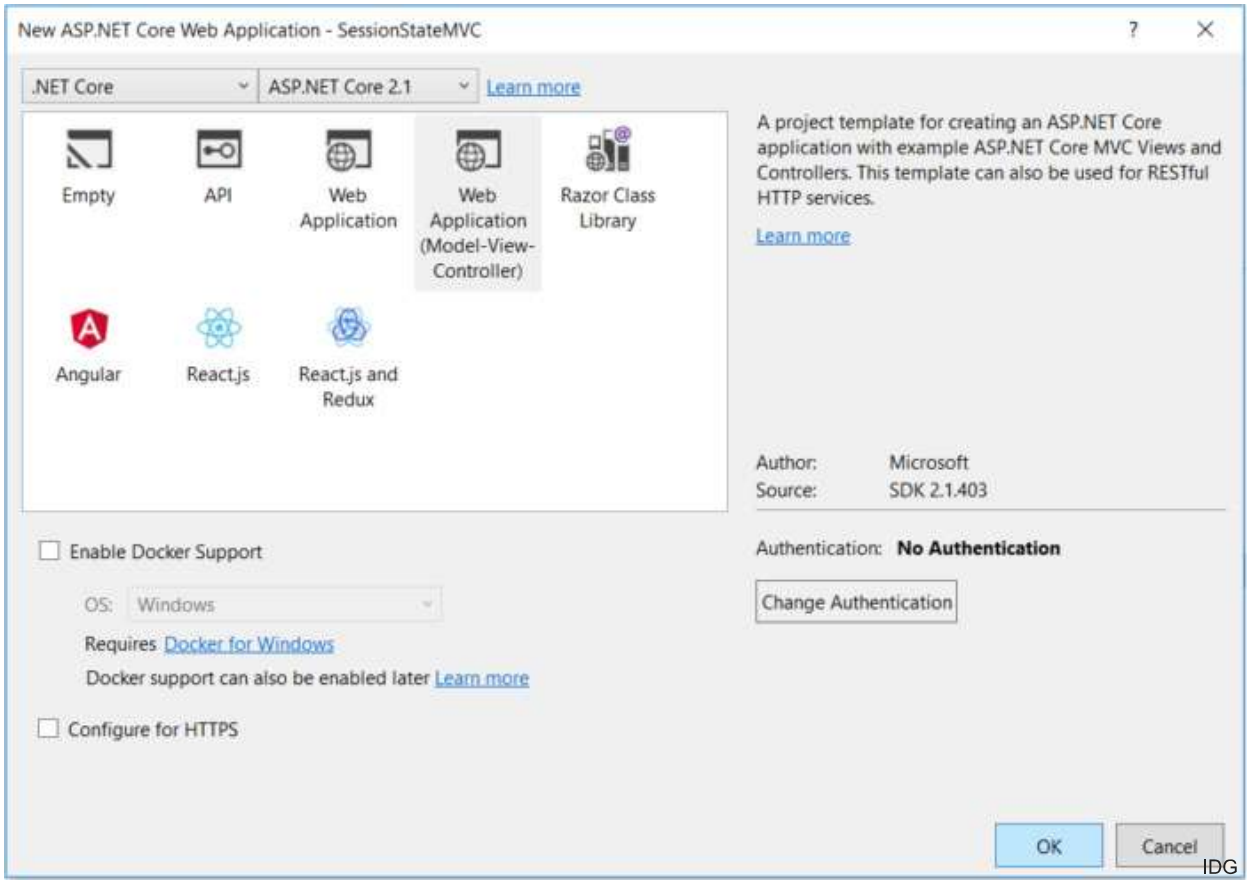
Create an ASP.Net Core Web API project

First off, let's create an ASP.Net Core project and install the necessary packages. If Visual Studio 2017 is up and running in your system, follow the steps outlined below to create an ASP.Net Core Web API project.

1. Launch the Visual Studio 2017 IDE.
2. Click on File > New > Project.
3. Select "ASP.Net Core Web Application (.Net Core)" from the list of templates displayed.
4. Specify a name for the project.
5. Click OK to save the project.
6. Select "API" in the "New .Net Core Web Application..." window.

7. Select ".Net Core" as the runtime and ASP.Net Core 2.1 (or later) from the drop-down list of controls at the top.
8. Select "Web Application (Model-View-Controller)" as the project template.
9. Uncheck the "Enable Docker Support" box.
10. Ensure that "No Authentication" is selected as we won't be using authentication here.
11. Ensure that "Configure for HTTPS" is unchecked as we won't need HTTPS either. (See the figure below.)
12. Click OK.

This will create a new ASP.Net Core 2.1 MVC project in Visual Studio 2017.



Creating a new ASP.Net Core MVC project in Visual Studio 2017.

Install the ASP.Net Core session state middleware

To get up and running with sessions, we must add the Microsoft.AspNetCore.Session NuGet package to our project. To do this, select the newly created project in the Solution Explorer window, then right-click and select the "Manage NuGet Packages..." option to open the NuGet Package Manager window. Next, search for the Microsoft.AspNetCore.Session NuGet package and install it.

Configure the ASP.Net Core session state middleware

The next step is to add the services for working with session state and configuring the HTTP request pipeline. Here is how you can add the services in the ConfigureServices method of the Startup.cs file.

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddSession(options => {
                options.IdleTimeout = TimeSpan.FromMinutes(1);
            });
            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1)
        }
```

Next, you can enable session state for the application in the Configure method of the Startup.cs file as shown in the code snippet below.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            app.UseSession();
            app.UseMvc();
        }
```

And that's it for configuration. In the section that follows, we will examine how we can use session state in the controller methods.

Store and retrieve session data in ASP.Net Core

User session data is stored in key-value pairs. You can get the session object by using HttpContext.Session, where HttpContext represents the current context. You can use the HttpContext.Session.SetString() method to store a string and the HttpContext.Session.GetString() method to retrieve the string based on a key from the session. The following code snippet illustrates how these methods can be used.

```
public class HomeController : Controller
    {
        public IActionResult Index()
        {
            HttpContext.Session.SetString("Key", "This is a test message.");
            return View();
        }
        public IActionResult About()
        {
            string message = HttpContext.Session.GetString("Key");
            ViewData["Message"] = message;
            return View();
        }
    }
```

Check if session has expired in ASP.Net Core

Note that we set the IdleTimeout for session storage to one minute in the ConfigureServices method of the Startup.cs file. So, after one minute of inactivity, the session data will be lost. We can update the code shown earlier to check if the session is still alive, i.e., that the session object is not null. Here is the updated code for the HomeController class for reference.

```
public class HomeController : Controller
    {
        public IActionResult Index()
        {
            HttpContext.Session.SetString("Key", "This is a test message.");
            return View();
        }
public IActionResult About()
        {
            string message = null;
            if (HttpContext.Session != null)
            {
                message = HttpContext.Session.GetString("Key");
                if(string.IsNullOrEmpty(message))
                    message ="Session timed out.";
            }
            ViewData["Message"] = message;
            return View();
        }
    }
```

Store complex objects in a session in ASP.Net Core

If you want to store complex data in the session, you can create an extension class that serializes and de-serializes the objects appropriately as shown in the code snippet below.

```
public static class SessionCoreExtensions
    {
        public static void Add<T>(this ISession iSession, string key, T data)
        {
            string serializedData = JsonConvert.SerializeObject(data);
            iSession.SetString(key, serializedData);
        }
        public static T Get<T>(this ISession iSession, string key)
        {
            var data = iSession.GetString(key);
            if(null != data)
                return JsonConvert.DeserializeObject<T>(data);
            return default(T);
        }
    }
```

Assuming you have a POCO (plain old class object) called Author, you can store a complex object as shown in the code snippet below.

```
var author = new Author();
HttpContext.Session.Add<author>("Key", author);
```

To retrieve the complex object from the session, you can use the following code.

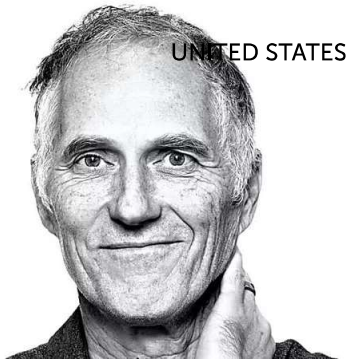
```
var obj = HttpContext.Session.Get<Author>("Key");
```

HTTP is a stateless protocol. Every request in the HTTP world occurs in absolute isolation. Whereas session state management was built into legacy ASP.Net, the leaner ASP.Net Core requires us to add the Microsoft.AspNetCore.Session middleware to support it. Albeit the fact that ASP.Net Core works a bit differently, storing session state is still easy to do when we need it.

Joydip Kanjilal is a Microsoft MVP in ASP.Net, as well as a speaker and author of several books and articles. He has more than 20 years of experience in IT including more than 16 years in Microsoft .Net and related technologies.

Follow

UNITED STATES

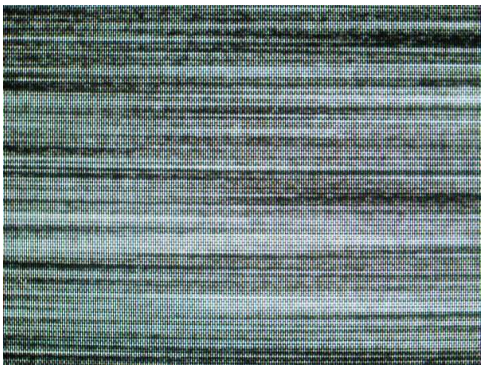


Tim O'Reilly: the golden age of the programmer is over

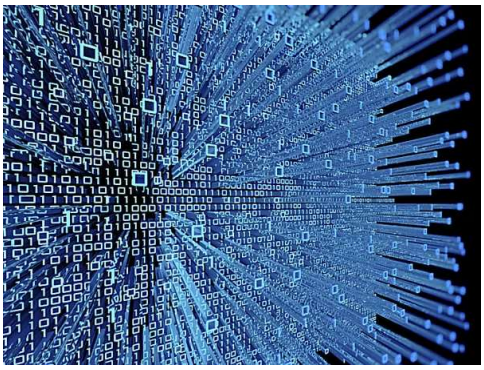
Build more secure software with Rust for Windows

The best open source software of 2020

When to use Task.WaitAll vs. Task.WhenAll in .NET

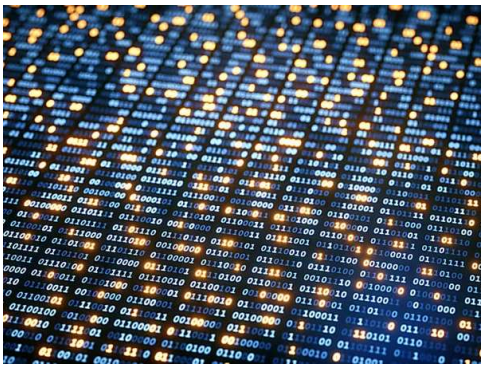


Python developers want static typing



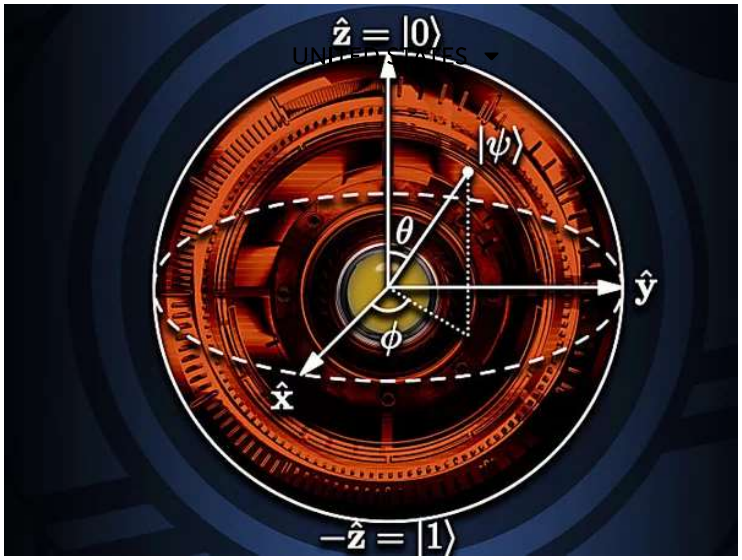
Mozilla spins out Pyodide Python-in-the-browser project

Are industry clouds an opportunity or a



What is a computational storage drive? Much-

Red Hat Enterprise Linux takes aim at edge



Amazon Braket: Get started with quantum computing

The shifting market for PostgreSQL