Desarrollador de Aplicaciones Web
Programación Web III

Departamento de Ingeniería e Investigaciones Tecnológicas

Pasaje de Datos

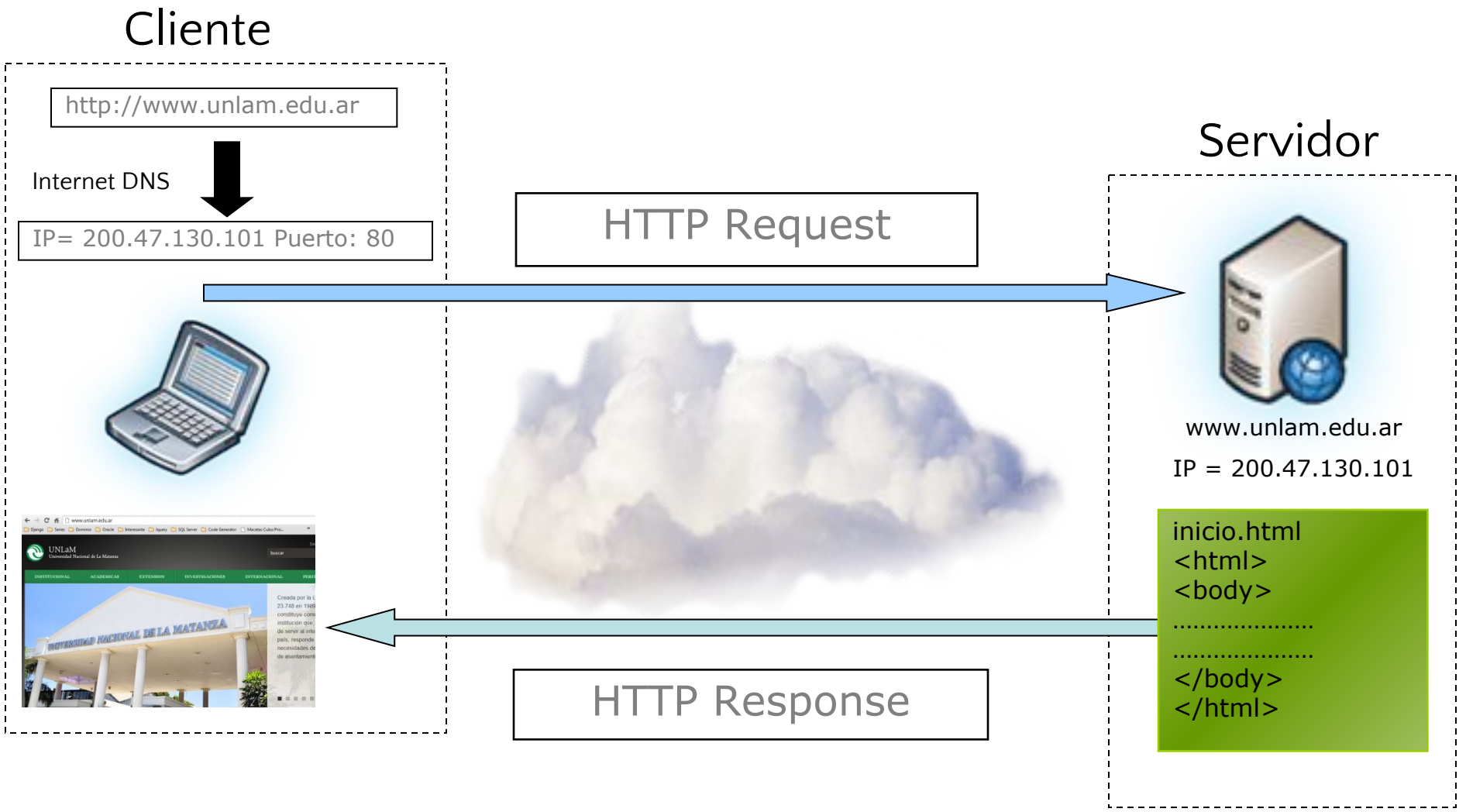
Ing. Mariano Juiz
Ing. Matias Paz Wasiuchnik
Ing. Pablo Nicolás Sanchez

Agenda

1. Introducción
2. Pasar Datos Controller => Vista
    1. ViewData
    2. ViewBag
    3. ViewModels
3. TempData
4. Variables de Sesión

Introducción : HTTP Request y Response

Páginas web



ViewData

Es utilizado para pasar datos desde el controlador a la vista.

Deriva de la clase ViewDataDictionary.

Su contenido está solo disponible para el request actual.

Si utilizamos tipos de datos complejos (por ejemplo, una clase) debemos realizar un cast para poder utilizar el objeto almacenado y validar que el objeto no sea nulo.

Si una redirección (302) ocurre, el valor se convierte en null.

ViewData

```
//Controller Code
public ActionResult Index()
{
    List<string> empleado = new List<string>();
    empleado.Add("Juana");
    empleado.Add("Pedro");

    ViewData["Empleados"] = empleado;
    return View();
}
//page code
<ul>
  @foreach (var empleado in ViewData["Empleados"] as List<string>)
    {
  <li>@empleado</li>
    }
</ul>
```

ViewBag

También es utilizado para pasar datos desde el controlador a la vista.

Deriva del tipo dynamic. Es un tipo de dato dinámico que es interpretado en momento de ejecución.

Su contenido está solo disponible para el request actual.

Si utilizamos tipos de datos complejos no es necesario hacer un cast

Si una redirección (302) ocurre, el valor se convierte en null.

ViewBag

```
//Controller Code
public ActionResult Index()
{
    List<string> empleado = new List<string>();
    empleado.Add("Juana");
    empleado.Add("Pedro");

    ViewBag.Empleados = empleado;
    return View();
}
//page code
<ul>
   @foreach (var empleado in ViewBag.Empleados)
      {
   <li>@empleado</li>
      }
</ul>
```

View Models

Un View Model representa los datos que se desea visualizar en la vista. Es el **modelo de la vista**. Es un **patrón de visualización.**.

Puede ser diferente al modelo de dominio de tu sistema.

Difiere del modelo de dominio ya que sólo contiene los datos que queres mostrar y ningún comportamiento de negocio. Permite separar responsabilidades.

No es requerido que siempre tengas un view model para cada vista. En vistas simples puedes utilizar tipos de datos simples (string, int, bool), objetos de dominio o viewData/viewBag.

View Models

```
//Controller Code
public ActionResult Index()
{
    EmpleadoView empleado = new EmpleadoView();
    empleado.Nombre = “Juana";
    empleado.Apeliido =“Pedro";

    return View(empleado);
}
//page code

@model EmpleadoView

<p>@Model.Nombre</p>
<p>@Model.Apellido</`p>
```

TempData

- TempData deriva de la clase TempDataDictionary.

- Permite pasar datos entre el request actual y el siguiente.

- Ayuda a mantener la información cuando nos movemos de un controlador a otro controlador o de una acción a otra acción.

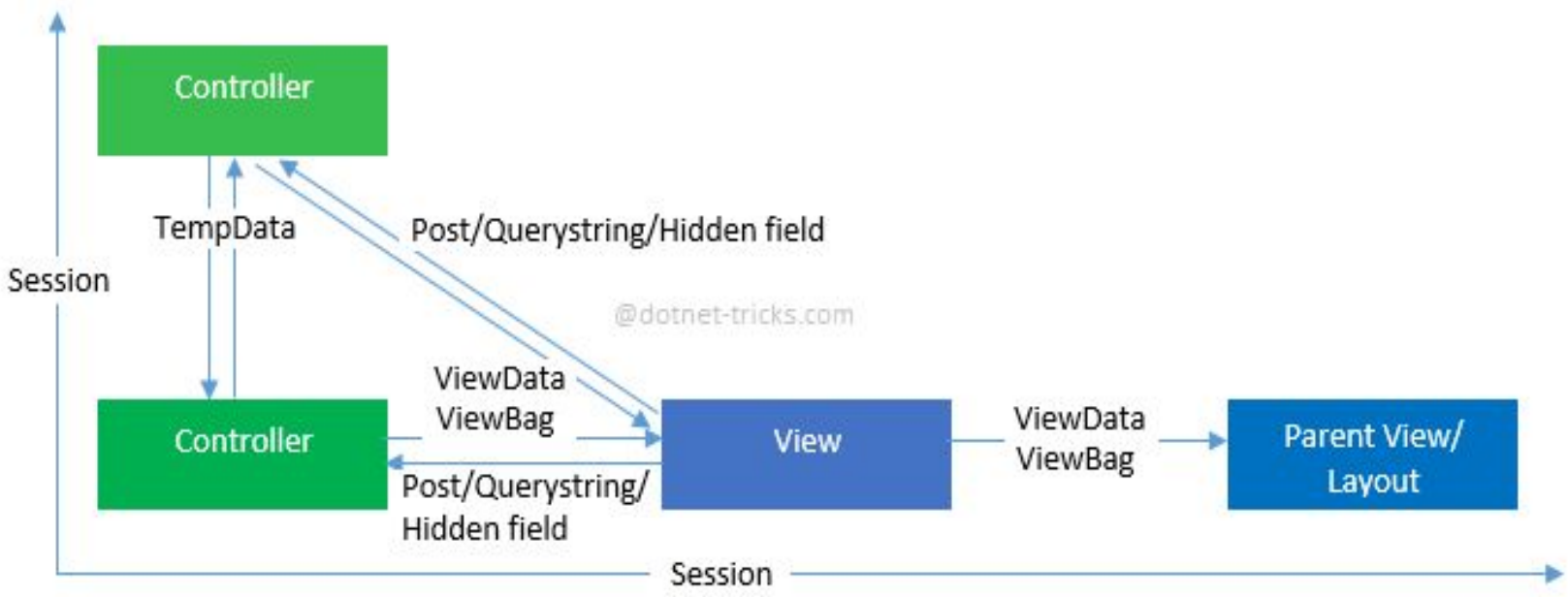
- Requiere cast de tipo de datos para datos complejos (similar al ViewData).

- En general, es utilizado para almacenar un mensaje una sola vez. Ejemplo: mensajes de error o de validación.

TempData

```
//Controller Code
public ActionResult Save(string nombre)
{
    Business.Save(nombre);
    TempData["Resultado"] = $“Se ha grabado correctamente: {nombre}”;

     return RedirectToAction(“Index”);
}

//Index Code

<h1>@TempData["Resultado"]</h1>
```

ViewBag / ViewData / TempData



Variables de Sesión (ASP.NET 4.x)

- Son objetos que se inician cuando el usuario ingresa en la página y finalizan cuando deja el sitio o por timeout.

- Son datos que solo visualiza el usuario en cuestión.

En Global.asax (se inicializan los objetos)

```
protected void Session_Start(Object sender, EventArgs e)
{
        Session[“usuarioID"] = String.Empty;
        Session[“sexo"] = String.Empty;
}
```

En cualquier parte de la aplicación web (se setean los objetos)

```
Session[“usuarioId“] = “lquiroga”;
Session[“sexo“] = “M”;
```

Variables de Sesión (ASP.NET 4.x)

Modos

- InProc
- State Server
- SqlServer
- Custom
- Off

Variables de Sesión (ASP.NET 4.x)

Modo InProc

- Es el modo por defecto (y el más óptimo)
- El estado de la sesión se almacena en la memoria del servidor web
- Ofrece el mejor rendimiento y buena seguridad
- No se persiste si se reinicia la aplicación web o a través varios servidores (Web Farm)

```
<sessionState mode="InProc" cookieless="false" timeout="20" />
```

- Cookieless define si almacena el "session Id" en el usuario o es ingresado en la querystring de la url

Recomendaciones

- No guardar muchos datos porque ocupan memoria de servidor.

- Encriptar los datos antes de guardarlos.

Variables de Sesión (ASP.NET 4.x)

StateServer

- El estado de la sesión se almacena en un servicio llamado ASP.NET State Service (aspnet_state.exe)
- Se envían datos por el protocolo HTTP sobre un puerto TCP
- Persiste aunque se reinicie la aplicación o a través de varios servidores (Web Farm)
- Ofrece menor rendimiento que el modo InProc, pero mayor fiabilidad y escalabilidad

```
<sessionState mode="StateServer" cookieless="false"
stateConnectionString="tcpip=myserver:42424" timeout="20" />
```

Recomendaciones

- No detener el servicio (se pierden los datos de la sesión)

- Encriptar los datos antes de guardarlos

Variables de Sesión (ASP.NET 4.x)

SqlServer

- El estado de la sesión se almacena en una base de datos de SQL Server, brindando mayor estabilidad y escalabilidad
- Persiste aunque se reinicie la aplicación o a través de varios servidores (Web Farm)
- En las mismas condiciones de Hardware, ofrece menor rendimiento que **State Server** pero ofrece una mejor integridad de los datos y reporting.

```
<sessionState mode="SqlServer"
sqlConnectionString="data source=127.0.0.1;user
id=sa; password=" cookieless="false" timeout="20" />
```

Recomendaciones

- Crear la base de datos “ASPState” usando el script “InstallState.sql” (ubicado en la carpeta *WinDir\Microsoft.Net\Framework\Version)*

- Encriptar los datos antes de guardarlos

Variables de Sesión (ASP.NET 4.x)

Custom

- Permite especificar un proveedor de almacenamiento de la sesión customizado
  - Ej: http://www.codeproject.com/KB/session/sessiontool.aspx

- Es necesario implementarlo

```
<sessionState mode="StateServer" cookieless="false"
stateConnectionString="tcpip=myserver:42424" timeout="20" />
```

Recomendaciones

- Encriptar los datos antes de guardarlos

Off

- Deshabilita el estado de la sesión.
- Si la aplicación web no usa sesión, se mejora el rendimiento.

Variables de Sesión (ASP.NET 4.x)

Custom

- Permite especificar un proveedor de almacenamiento de la sesión customizado
  - Ej: http://www.codeproject.com/KB/session/sessiontool.aspx

- Es necesario implementarlo

```
<sessionState mode="StateServer" cookieless="false"
stateConnectionString="tcpip=myserver:42424" timeout="20" />
```

Recomendaciones

- Encriptar los datos antes de guardarlos

Off

- Deshabilita el estado de la sesión.
- Si la aplicación web no usa sesión, se mejora el rendimiento.

Variables de Sesión (ASP.NET Core)

El estado de sesión es una técnica / escenario de ASP.NET Core utiilizada para el almacenamiento de datos de usuario mientras el usuario navega por una aplicación web.

El estado de la sesión utiliza un “almacén” (store) mantenido por la aplicación para conservar los datos en las solicitudes de un cliente.

Los datos de la sesión están respaldados por una caché y se consideran datos efímeros o temporales.

Como buena práctica, los datos críticos de la aplicación deben almacenarse en la base de datos o repositorio permanente del usuario y almacenarse en la sesión solo como una optimización del rendimiento

Variables de Sesión (ASP.NET Core)

ASP.NET Core mantiene el estado de la sesión mediante una cookie en el cliente la cual contiene un identificador de sesión.

Algunas Consideraciones:

- La cookie de sesión es específica del navegador. Las sesiones no se comparten entre navegadores.
- La aplicación retiene una sesión por un tiempo limitado (timeout) después de la última solicitud.
- Se puede establecer ese tiempo limitado (timeout) de la sesión. El valor predeterminado es de 20 minutos.
- El estado de la sesión es ideal para almacenar datos de usuario:
- Que son específicos para una sesión en particular.
- Donde los datos no requieren almacenamiento permanente entre sesiones.
- Los datos de la sesión se eliminan cuando se llama a la implementación de ISession.Clear o cuando la sesión expira (timeout).
- No existe un mecanismo predeterminado para informar al código de la aplicación (servidor) que el navegador de un cliente se ha cerrado o cuando la cookie de sesión se elimina.

Variables de Sesión (ASP.NET Core)

En asp.net Core, es necesario habilitar el uso de variables de sesión de la siguiente manera:

- Se necesita el package Microsoft.AspNetCore.Session el cual está incuído implícitamente en el framework .

- Proveer middleware para administrar el estado de sesión.

- Para habilitar el middleware la clase Startup debe contener:
  - Algún tipo de cache: IDistributedCache. Para más información ver: Distributed caching in ASP.NET Core.
  - Una llamada a AddSession en **ConfigureServices**.
  - Una llamada a UseSession en **Configure**

1) ConfigureServices:

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

    services.AddSession(options =>
    {
        options.Cookie.Name = ".MiAPP.Session";
        options.IdleTimeout = TimeSpan.FromSeconds(60);
    });
}
```

Variables de Sesión (ASP.NET Core)

2) Configure:

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

    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

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

    app.UseSession();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapDefaultControllerRoute();
        endpoints.MapRazorPages();
    });
}
```

Variables de Sesión (ASP.NET Core)

¿Cómo las Uso?

incluir el namespace: Microsoft.AspNetCore.Http;

```
using Microsoft.AspNetCore.Http;
```

Para guardar un dato en la sesión:

```
if (string.IsNullOrEmpty(HttpContext.Session.GetString(“Nombre”)))
  {
      HttpContext.Session.SetString(“Nombre”, "Ramiro Gimenez");
      HttpContext.Session.SetInt32(“Edad”, 27);
  }
```

Para recuperar información de la sesión

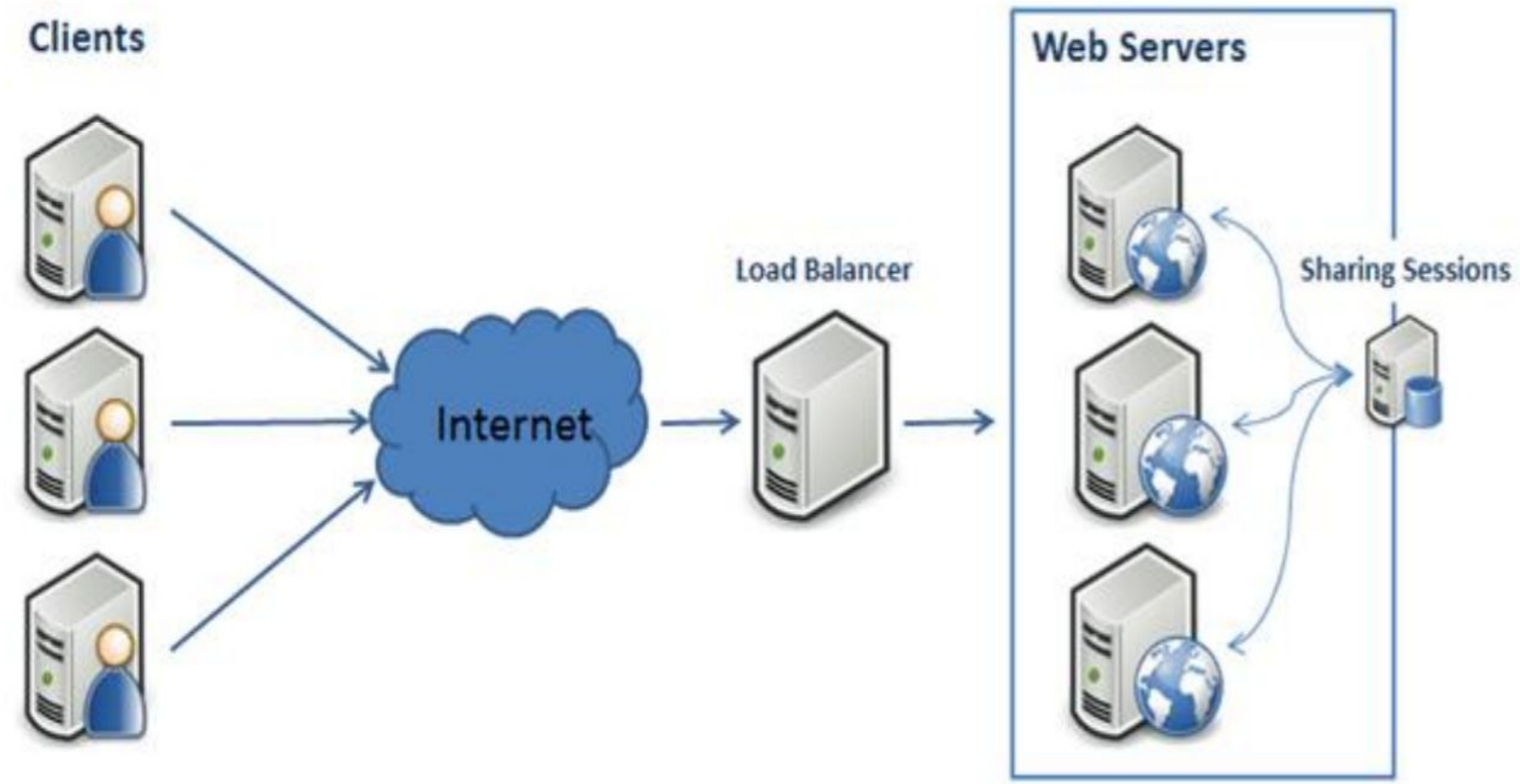
```
var nombre = HttpContext.Session.GetString(“Nombre”);
var edad = HttpContext.Session.GetInt32(“Edad”);
```

Variables de Sesión (ASP.NET Core)

Para almacenar en sesión otros tipos de datos mas complejos es necesario serializar y deserializar los mismos, por ejemplo mediante JSON:

```
//Asegurarse de tener los siguientes namespaces:
using System.Text.Json;
using Microsoft.AspNetCore.Http;

    public static class SessionExtensions
    {
        public static void Set<T>(this ISession session, string clave, T valor)
        {
            session.SetString(clave, JsonSerializer.Serialize(valor));
        }

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

            if (valor == null)
                return default(T);
            else
                return JsonSerializer.Deserialize<T>(valor);
        }
    }

//Luego podríamos usar Get y Set en alguna acción o método:
  if (HttpContext.Session.Get<Empleado>("EMPLEADODELMES") == default)
  {
         Empleado empleado = new Empleado { Nombre = "Joaquin", Edad = 20 };
         HttpContext.Session.Set<Empleado>("EMPLEADODELMES", empleado);
  }
```

Variables de Sesión (ASP.NET Core)



Variables de Sesión (ASP.NET Core) - Escalabilidad:

Para lograr **escalabilidad** y usar variables de session en escenarios donde mi aplicación requiera de varios servers (Cloud o WebFarm) se utiliza el concepto o técnica: **"Caché Distribuida":**

Una caché distribuida es una caché compartida por varios servidores de aplicaciones, que normalmente se mantiene como un servicio externo para los servidores de aplicaciones que acceden a ella. Una caché distribuida puede mejorar el rendimiento y la escalabilidad de una aplicación ASP.NET Core, especialmente cuando la aplicación está alojada en un servicio en la nube o una granja de servidores.

Los distintos modos o formas de caché distribuida son y se configuran en `Startup.ConfigureServices`:

- **Distributed Memory Cache: (`AddDistributedMemoryCache()`)**
  Este modo en realidad NO almacena la información de manera distribuida. Por el contrario, la información se almacena en la instancia de la aplicación en el mismo server de la aplicación.

- **Distributed SQL Server cache `(AddDistributedSqlServerCache())`**
  Utiliza un servidor SQL Server para almacenar la información.

- **Distributed NCache cache `(AddNCacheDistributedCache):`**
  Es un caché distribuido de código abierto desarrollado de forma nativa en .NET y .NET Core. NCache funciona tanto localmente como configurado como un clúster de caché distribuido para una aplicación ASP.NET Core que se ejecuta en Azure o en otras plataformas de hosting.

- **Distributed Redis cache `(AddStackExchangeRedisCache()):`**
  Es un almacén de datos en memoria de código abierto, que a menudo se utiliza como caché distribuida. Se puede utilizar con Azure.

Para mas información consultar:
https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed?view=aspnetcore-5.0

Desarrollador de Aplicaciones Web
Programación Web III

Departamento de Ingeniería e Investigaciones Tecnológicas

Muchas gracias

Ing. Mariano Juiz
Ing. Matias Paz Wasiuchnik
Ing. Pablo Nicolás Sanchez