> Ficha Prática Nº 8

Objetivos/Temas:

- Gestão de estados em aplicações web
- Gestão de estados em aplicações ASP.NET Core
- Variáveis de Sessão
- TempData
- Gráficos

> Parte I – Conceitos

>> Gestão de estados em aplicações web

HTTPS://LEARN.MICROSOFT.COM/PT-PT/ASPNET/CORE/FUNDAMENTALS/APP-STATE?VIEW=ASPNETCORE-6.0
HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/SECURITY/GDPR?VIEW=ASPNETCORE-6.0#TEMPDATA-PROVIDER-AND-SESSION-STATE-COOKIES-ARENT-ESSENTIAL

O HTTP é um protocolo sem estado. Por padrão, as solicitações HTTP são mensagens independentes que não retêm valores do utilizador.

O estado pode ser armazenado utilizando diferentes abordagens:

Abordagem	
Cookies	HTTP cookies. Pode incluir dados utilizando o código da aplicação (servidor)
Session state	HTTP cookies + código da aplicação (servidor)
TempData	HTTP cookies ou estado de sessão
Query strings	HTTP Query strings
Campos ocultos	HTTP form fields
HttpContext.Items	Código da aplicação (servidor)
Cache	Código da aplicação (servidor)

Cookies	• Um cookie é um pequeno ficheiro de texto que é guardado no dispositivo que está a consultar o sítio web; • Normalmente são usados para personalização do sítio web (guardar preferências do utilizador, idioma, moeda, tema gráfico, etc.) • São enviados em cada solicitação HTTP; • Podem ser manipulados pelo cliente, pelo que não devem ser usados para guardar informação sensível.;

	• Como tal, devem ser validados pela aplicação;
	• Dois tipos:
	  ○ Cookies persistentes;
	  ○ Cookies de sessão;
	• Deve ter-se especial atenção ao seu uso (ver RGPD https://commission.europa.eu/law/law-topic/data-protection_pt)

Session state	• Session State / estado da sessão é uma funcionalidade do ASP.NET Core para armazenamento de dados do utilizador enquanto o utilizador navega numa aplicação web;
	• Armazenamento mantido pela aplicação para persistir dados através de pedidos de um cliente;
	• Apoiados por uma cache e considerados dados efémeros;
	• A aplicação web deve continuar a funcionar sem os dados da sessão.
	• Os dados críticos da aplicação devem ser armazenados em base de dados e armazenados em sessão apenas como uma otimização do desempenho.
	• Não suportado pelo SginalR.
	• O ASP.NET Core mantém o estado da sessão fornecendo um cookie ao cliente que contém um ID de sessão. O ID da sessão cookie:
	  ○ é enviado para a aplicação com cada pedido;
	  ○ É utilizado pela aplicação para ir buscar os dados da sessão;
	• O estado da sessão apresenta os seguintes comportamentos (entre outros):
	  ○ O cookie de sessão é específico para o browser;
	  ○ As sessões não são partilhadas entre browsers;
	  ○ Os cookies de sessão são apagados quando a sessão do programa de navegação termina;
	  ○ Se um cookie é recebido para uma sessão expirada, é criada uma nova sessão que utiliza o mesmo cookie de sessão;
	  ○ As sessões vazias não são retidas. A sessão deve ter pelo menos um valor definido para persistir a sessão através dos pedidos;

TempData	• Propriedade (Páginas Razor e/ou controllers) que armazena dados até que estes sejam lidos no próximo pedido;
Query strings	• Dados embebidos na URL; • As URL são públicas – não utilizar para armazenar dados sensíveis. • Limite de tamanho • Vetor de ataques do tipo CSRF;
Campos ocultos	• Campos de formulário escondidos/ocultos - <input type=”hidden” /> • Podem ser manipulados pelo cliente e por este motivo devem ser sempre validados pelo código da aplicação (servidor);
HttpContext.Items	• Uma coleção usada para armazenar dados durante o processamento de um pedido; • O conteúdo da coleção é descartado após o pedido ser processado; • A coleção Items costuma ser usada para permitir que componentes ou middleware comuniquem entre si quando operam em diferentes momentos do pedido e não têm nenhuma maneira direta de passar parâmetros;

Cache	• O cache é uma forma eficiente de armazenar e obter dados. A aplicação pode controlar a vida útil dos itens armazenados em cache. • Os dados em cache não estão associados a um pedido específico, utilizador ou sessão; • Não se deve armazenar em cache dados específicos de utilizadores - que possam ser obtidos por outros pedidos de outros utilizadores;

> Parte II – Exercícios

>> Criar e Ler um Cookie que permita alterar a cor de fundo da página de perfil do utilizador

1. Adicione o seguinte código C# ao método `OnGetAsync()` na página Razor de perfil do utilizador:

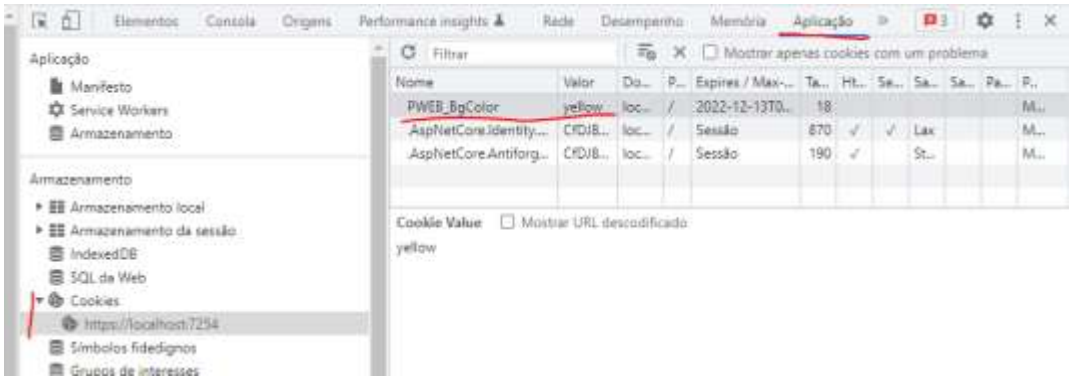
```
var cookieOptions = new CookieOptions();
cookieOptions.Expires = DateTime.Now.AddDays(10);
cookieOptions.Path = "/";
var cookie = Request.Cookies["PWEB_BgColor"];
if (cookie == null)
        cookie = "yellow";

// adiciona / modifica o cookie com o nome PWEB_BgColor
Response.Cookies.Append("PWEB_BgColor", cookie, cookieOptions);

// elimina o cookie com o nome PWEB_BgColor
// Response.Cookies.Delete("PWEB_BgColor");
```

2. Execute a aplicação, faça login e navegue até à página de perfil do utilizador.

3. Abra as ferramentas de programação do browser e verifique os cookies existentes.



4. Navegue por outras “páginas” da aplicação e veja se o cookie se mantém.

5. Adicione o seguinte código HTML ao formulário da página Razor de perfil do utilizador:

```
<div class="form-floating">
        <label for="pageBgColor">Cor de fundo da página</label>
        <select name="pageBgColor" id="pageBgColor"
                onchange="setCookie('PWEB_BgColor',this.value,1)"
                class="form-control form-select form-select-sm">
                <option value="white">White</option>
                <option value="red">Red</option>
                <option value="orange">Orange</option>
                <option value="blue">Blue</option>
        </select>
</div>
```

6. Adicione o seguinte código Javascript à página Razor de perfil do Utilizador (dentro da secção Scripts "`@section Scripts {…}`"):

```
<script>
        checkCookie();
        function setCookie(cname, cvalue, exdays) {
            console.log(cvalue);
            const d = new Date();
            d.setTime(d.getTime() + (exdays * 24 * 60 * 60 * 1000));
            let expires = "expires=" + d.toUTCString();
            document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
            document.body.style.setProperty("background-color", cvalue, "important");
        }

        function getCookie(cname) {
            let name = cname + "=";
            let ca = document.cookie.split(';');
            for (let i = 0; i < ca.length; i++) {
                let c = ca[i];
                while (c.charAt(0) == ' ') {
                    c = c.substring(1);
                }
                if (c.indexOf(name) == 0) {
                    return c.substring(name.length, c.length);
                }
            }
            return "";
        }

        function checkCookie() {
            let bgColor = getCookie("PWEB_BgColor");
            let select = document.getElementById('pageBgColor');
            if (bgColor != "") {
                document.body.style.setProperty("background-color", bgColor,
"important");
                select.value = bgColor;
            } else {
                alert("Please choose a Background color:");
            }
        }
    </script>
```

7. Teste o funcionamento da aplicação – verifique se é possível mudar a cor de fundo da página.

8. Faça debug à aplicação e verifique se no método `OnGetAsync()` é possível ler o valor do cookie em cada pedido que é efetuado. Analise o código com o docente.

>> Criar um carrinho de compras (comprar cursos)

9. Configurar Sessões - Adicione o seguinte código (fundo a amarelo) C# ao ***Program.cs***:

```
builder.Services.AddControllersWithViews();

// session state
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
    options.Cookie.Name = ".PWEbApp.Session";
    options.IdleTimeout = TimeSpan.FromSeconds(10);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});

var app = builder.Build();
/*
 …
 …

*/
app.UseAuthentication();
app.UseAuthorization();

// session state
app.UseSession();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

10. Uma vez que por omissão apenas podemos guardar nas variáveis de sessão *Strings* e *Int32* temos de criar uma extensão para podermos guardar e obter objetos complexos (no formato JSON).

    Para isso crie uma classe nova chamada de **SessionExtensions** com o seguinte código:

```
public static class SessionExtensions
    {
        public static void SetJson<T>(this ISession session, string key, T value)
        {
            session.SetString(key, JsonSerializer.Serialize(value));
        }

        public static T? GetJson<T>(this ISession session, string key)
        {
            var value = session.GetString(key);
            return value == null ? default : JsonSerializer.Deserialize<T>(value);
        }
    }
```

    *Sugestão: crie uma pasta com o nome **Helpers** e crie a classe dentro desta pasta*

11. De seguida crie as seguintes classes que vão auxiliar na gestão do carrinho de compras.

CarrinhoItem

```
public class CarrinhoItem
{
        public int  CursoId { get; set; }
        public string CursoNome { get; set; }
        public int Quantidade { get; set; }
        public decimal PrecoUnit { get; set;}
}
```

Carrinho

```
public class Carrinho
{
        public List<CarrinhoItem> items { get; set; } = new List<CarrinhoItem>();

        public void AddItem(CarrinhoItem curso, int qtd){}
        public void RemoveItem(CarrinhoItem curso) {}
        public decimal Total(){}
        public void Clear() => items.Clear();
}
```

12. Implemente os métodos ***AddItem()***, ***RemoveItem()*** e ***Total()***

13. Crie uma *view* vazia com o nome ***_carrinho.cshtml***, dentro da pasta **Shared**.

14. Copie o seguinte código para a *view* que criou no ponto anterior:

```
@using PWEB_AulasP_2223.Helpers
@{
    var CarrinhoDeCompras = Context.Session.GetJson<Carrinho>("CarrinhoDeCompras") ??
new Carrinho();
    int qtd = CarrinhoDeCompras == null ? 0 : CarrinhoDeCompras.items.Count();
}
<div class="form-inline d-flex ">
    <span class="badge badge-dark fs-6 ">@qtd</span>
    <strong>
        <svg bootstrap-icon="Cart"
             class="text-white me-2" width="24" height="24"
             aria-label="Search"></svg>
    </strong>
</div>
```

15. Inclua esta *view* parcial nos ficheiros de layout por forma a ficar com

```
<partial name="_carrinho" />
<partial name="~/Views/Cursos/QuickSearchPartial.cshtml" />
<partial name="_LoginPartial" />
```

16. Adicione um “botão” “comprar curso” na *view* ***search*** do controller ***Cursos*** *e na view* ***Index*** do controller **Home**, conforme imagem seguinte:

Este botão deve enviar um pedido HTTP GET para o método ***comprar***, passando como parâmetro o Id do curso a comprar.
Exemplo do URL de um pedido: https://localhost:7254/Cursos/Comprar/1

17. Implemente o método ***Comprar*** no *controller Cursos*.

- O método deve:
  1. Adicionar o curso escolhido (recebido como parâmetro) ao carrinho de compras.
  2. Redirecionar o utilizador para a acção "Carrinho" – a criar nos pontos seguintes

18. Crie uma view vazia com o nome ***Carrinho*** e copie o seguinte código para a View:

```
@model PWEB_AulasP_2223.Models.Carrinho
<h1>Carrinho de compras</h1>
Items no carrinho: <span class="badge bg-secondary">@Model.items.Count</span>
<table class="table table-hover table-striped">
    <thead>
        <tr>
            <th>Item</th>
            <th>Preço Unitário</th>
            <th>Quantidade</th>
            <th>Sub-total</th>
        </tr>
    </thead>
    <tbody>
    </tbody>
    <tfoot>
    </tfoot>

</table>
<div class="row">
    <div class="col-12 text-end ">
        <a href="#" class="btn btn-primary">continuar a comprar</a>
        <a href="#" class="btn btn-success">checkout</a>
    </div>

</div>
```

19. Implemente o método Carrinho e faça as alterações necessárias à vista por forma a obter um resultado semelhante a:

20. Faça as alterações necessárias na ***View*** e no ***Controller*** por forma a poder alterar as quantidades de um item do carrinho, bem como poder remover um item do carrinho e/ou limpar o carrinho.

>> Criar gráficos com recurso a uma biblioteca JavaScript

Existem várias bibliotecas JavaScript para criar gráficos em aplicações web.

Neste exercício / exemplo, vamos utilizar a biblioteca ***ChartJS*** (https://www.chartjs.org/docs/latest/).

O objetivo deste exercício é criar um gráfico que nos permita ver um resumo de vendas por mês e por produto.

Nota: Como ainda não temos as vendas na base de dados, vamos usar um repositório local como fonte de dados e não a base de dados. Mais tarde, quando existirem vendas na base de dados podem e devem fazer as alterações necessárias para ler os dados da base de dados.

21. Crie uma view vazia, com o nome ***GraficoVendas,*** no controller cursos e copie o seguinte código:

```
@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_layout2.cshtml";
}
<h1>Vendas mensais por curso</h1>
<h6>Gráfico com Chart.js</h6>
<div>
    <canvas id="chartGraficoVendas"></canvas>
</div>
@section Scripts {
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script>
        $.ajax({
            type: "POST",
            url: "/Cursos/GetDadosVendas",
            contentType: "application/json; charset=utf-8",
            dataType: "json",
            success: function (data) {
                let Labels = data[0];
                let Datasets1 = data[1];
                let dataT = {
                    labels: Labels,
                    datasets: [{
                        label: "Cursos",
                        data: Datasets1,
                        fill: false,
                        borderWidth: 1,
                        backgroundColor: ["red", "green", "blue", "cyan", "yellow"]
                    }]
                };
                let ctx = $("#chartGraficoVendas").get(0).getContext("2d");
                let myNewChart = new Chart(ctx, {
                    type: 'bar',
                    data: dataT,
                    options: {
                        responsive: true,
                        title: { display: true, text: 'Vendas de cursos ' },
                        legend: { position: 'bottom' },

                    }
                });



            }
        });
    </script>    }
```

22. Crie o método ***GraficoVendas*** e o método ***GetDadosVendas,*** no controller cursos e copie o seguinte código:

```
// GET: Cursos/GraficoVendas/5
        public async Task<IActionResult> GraficoVendas()
        {
            return View();
        }
        [HttpPost]
        // POST: Cursos/GraficoVendas/5
        public async Task<IActionResult> GetDadosVendas()
        {
            //dados de exemplo
            List<object> dados = new List<object>();

            DataTable dt = new DataTable();
            dt.Columns.Add("Cursos", System.Type.GetType("System.String"));
            dt.Columns.Add("Quantidade", System.Type.GetType("System.Int32"));
            DataRow dr = dt.NewRow();
            dr["Cursos"] = "CATEGORIA AM (Ciclomotor)";
            dr["Quantidade"] = 12;
            dt.Rows.Add(dr);
            dr = dt.NewRow();
            dr["Cursos"] = "CATEGORIA A1 (Motociclo - 11kw/125cc)";
            dr["Quantidade"] = 96;
            dt.Rows.Add(dr);
            dr = dt.NewRow();
            dr["Cursos"] = "CATEGORIA A2 (Motociclo - 35kw)";
            dr["Quantidade"] = 87;
            dt.Rows.Add(dr);
            dr = dt.NewRow();
            dr["Cursos"] = "CATEGORIA B1 (Quadriciclo)";
            dr["Quantidade"] = 67;
            dt.Rows.Add(dr);
            dr = dt.NewRow();
            dr["Cursos"] = "CATEGORIA B (Ligeiro Caixa Automática)";
            dr["Quantidade"] = 63;
            dt.Rows.Add(dr);

            foreach (DataColumn dc in dt.Columns)
            {
                List<object> x = new List<object>();
                x = (from DataRow drr in dt.Rows select drr[dc.ColumnName]).ToList();
                dados.Add(x);
            }
            return Json(dados);


        }
```