

# .NET App Dev Hands-On Workshop

## Blazor Lab 7 – State Management and JS Interop

This lab uses JavaScript Interop to save data to browser storage. Before starting this lab, you must have completed Blazor Lab 6.

### Part 1: Add the ViewModel

- Create a new class named `DataInfo.cs` in the `ViewModels` folder of the `AutoLot.Blazor.Models` project. Update the code to the following:

```
namespace AutoLot.Blazor.Models.ViewModels;
public class DataInfo
{
    public string Value { get; set; }
    public int Length { get; set; }
    public DateTime Timestamp { get; set; }
}
```

### Part 2: Add the Services

#### Step 1: Add the storage service interface

- Create a new folder named `Storage` in the `Services` folder of the `AutoLot.Blazor` project. In this folder, add a new folder named `Interfaces`. In this folder, add an interface file named `IBrowserStorageService.cs`. Update the code to the following:

```
namespace AutoLot.Blazor.Services.Storage.Interfaces;

public interface IBrowserStorageService
{
    Task SetItemAsync<T>(string key, T item);
    Task<T> GetItemAsync<T>(string key);
}
```

- Add the following to the `GlobalUsings.cs` file in the `AutoLot.Blazor` project:

```
global using AutoLot.Blazor.Services.Storage;
global using AutoLot.Blazor.Services.Storage.Interfaces;
global using Microsoft.JSInterop;
global using System.Text.Json;
```

## Step 2: Add the Local Storage service

- Add a new class named `LocalStorageService.cs` in the `Services\Storage` directory. Update the code to the following:

```
namespace AutoLot.Blazor.Services.Storage;

public class LocalStorageService(IJSRuntime jsRuntime) : IBrowserStorageService
{
    public async Task SetItemAsync<T>(string key, T item)
    {
        await jsRuntime.InvokeVoidAsync(
            "skimedicInterop.setLocalStorage", key, JsonSerializer.Serialize(item));
    }
    public async Task<T> GetItemAsync<T>(string key)
    {
        var json = await jsRuntime.InvokeAsync<string>("skimedicInterop.getLocalStorage", key);
        return json == null ? default : JsonSerializer.Deserialize<T>(json);
    }
}
```

## Step 3: Add the Session Storage service.

- Add a new class named `SessionStorageService.cs` in the `Services\Storage` directory. Update the code to the following:

```
namespace AutoLot.Blazor.Services.Storage;

public class SessionStorageService(IJSRuntime jsRuntime) : IBrowserStorageService
{
    public async Task SetItemAsync<T>(string key, T item)
    {
        await jsRuntime.InvokeVoidAsync(
            "skimedicInterop.setSessionStorage", key, JsonSerializer.Serialize(item));
    }
    public async Task<T> GetItemAsync<T>(string key)
    {
        var json = await jsRuntime.InvokeAsync<string>("skimedicInterop.getSessionStorage", key);
        return json == null ? default : JsonSerializer.Deserialize<T>(json);
    }
}
```

## Step 4: Add the services to the DI container.

- Add the following two lines to `Program.cs` after the other interface injections:

```
builder.Services
    .AddKeyedScoped<IBrowserStorageService, LocalStorageService>(nameof(LocalStorageService));
builder.Services
    .AddKeyedScoped<IBrowserStorageService, SessionStorageService>(nameof(SessionStorageService));
```

## Part 3: Add the StateManagement Component and Page

### Step 1: Add the StorageExample component

- Add a new component named `StorageExample.razor` into the Shared folder. Update the code to the following:

```
<h3>@Title</h3>
@FieldDisplayName:
<input type="text" @bind-value="_data" size="25" /><hr />
<button @onclick="SaveToLocalStorageSync">Save to @StorageType Storage</button>
<button @onclick="ReadFromLocalStorageAsync">Read from @StorageType Storage</button>
<div class="mt-4">@_message</div>
@code {
    private string _data;
    private string _message;
    [Parameter][EditorRequired]public string Title { get; set; }
    [Parameter][EditorRequired]public string FieldDisplayName { get; set; }
    [Parameter][EditorRequired]public string StorageType { get; set; }
    [Parameter][EditorRequired]public EventCallback<string> OnDataReturnedCallback { get; set; }
    [Parameter][EditorRequired]
    public IBrowserStorageService StorageService { get; set; }
    async Task SaveToLocalStorageSync()
    {
        var dataInfo = new DataInfo()
        {
            Value = _data,
            Length = _data!.Length,
            Timestamp = DateTime.Now
        };
        await StorageService!.SetItemAsync<DataInfo>("localStorageData", dataInfo);
        _message = "Saved";
    }
    async Task ReadFromLocalStorageAsync()
    {
        DataInfo savedData = await StorageService!.GetItemAsync<DataInfo>("localStorageData");
        string result = $"localStorageData = {savedData?.Value ?? "Missing"}";
        await OnDataReturnedCallback.InvokeAsync(result);
        _message = "";
    }
}
```

## Step 2: Add the StateManagement Blazor Page

- Add a new page named `StateManagement.razor` into the Pages folder. Update the code to the following:

```
@page "/state-management"
<h1>State Management</h1>
<StorageExample Title="Local Storage"
  FieldDisplayName="localStorageData"
  StorageType="local"
  StorageService="LocalService"
  OnDataReturnedCallback="@{(async (value) => { await Task.Yield(); await ShowData(value); })}">
</StorageExample>
<StorageExample Title="Session Storage"
  FieldDisplayName="sessionStorageData"
  StorageType="session"
  StorageService="SessionService"
  OnDataReturnedCallback="@{(async (value) => { await Task.Yield(); await ShowData(value); })}">
</StorageExample>
@code {
    [Inject] private IJSRuntime JsRuntime { get; set; }
    private IJSObjectReference _module;
    //scoped to browser window
    [Inject(Key = nameof(LocalStorageService))]
    private IBrowserStorageService LocalService { get; set; }
    //scoped to browser tab
    [Inject(Key = nameof(SessionStorageService))]
    private IBrowserStorageService SessionService { get; set; }
    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        if (firstRender)
        {
            _module = await JsRuntime.InvokeAsync<IJSObjectReference>
                ("import", "../Pages/StateManagement.razor.js");
        }
    }
    async Task ShowData(string message)
    {
        if (_module is not null)
        {
            await _module.InvokeVoidAsync("showStorageData", message);
        }
    }
}
```

## Part 4: Add the JavaScript Files

### Step 1: Add the shared JavaScript file

- Add a new folder named `js` under the `wwwroot` folder, and in that folder, add a new JavaScript file named `interop.js`. Update the code to the following:

```
var skimedictInterop = {};
skimedictInterop.setLocalStorage = function (key, data) {
    //scoped to browser window
    localStorage.setItem(key, data);
}
skimedictInterop.getLocalStorage = function (key) {
    return localStorage.getItem(key);
}
skimedictInterop.setSessionStorage = function (key, data) {
    //scoped to browser tab
    sessionStorage.setItem(key, data);
}
skimedictInterop.getSessionStorage = function (key) {
    return sessionStorage.getItem(key);
}
```

- Update the `index.html` page to include the JavaScript file (just before the closing body tag):

```
<script src="js/interop.js"></script>
```

### Step 2: Add the isolated JavaScript file

- Add a new JavaScript file named `StateManagement.razor.js` into the `Pages` directory. Update the code to the following:

```
export function showStorageData(data) {
    alert(data);
}
```

## Part 5: Update the Navigation

- Add the following as the last navigation entry in the `NavMenu.razor` component:

```
<NavLink class="nav-link" href="state-management">
    Session Storage <span class="fas fa-archive ps-2" aria-hidden="true"></span>
</NavLink>
```

## Summary

This lab added JavaScript interop to save state into the browser.

## Next Steps

The next lab will add in calls to `AutoLot.API` instead of using the local hard-coded data service classes.