



Module 3 Day 8

MVC Controllers - Session

What makes an application?

- Program Data

- ✓ Variables & .NET Data Types
- ✓ Arrays
- ✓ More Collections (list, dictionary, stack, queue)
- ✓ Classes and objects (OOP)

- Program Logic

- ✓ Statements and expressions
- ✓ Conditional logic (if)
- ✓ Repeating logic (for, foreach, do, while)
- ✓ Methods (functions / procedures)
- ✓ Classes and objects (OOP)
- ❖ Frameworks (MVC)

- Input / Output

- User
 - ✓ Console read / write
 - ✓ HTML / CSS
 - ❑ Front-end frameworks (HTML / CSS / JavaScript)
- Storage
 - ✓ File I/O
 - ✓ Relational database
 - ❑ APIs

Cross-Site Request Forgery

- A malicious site can use a previously created authentication cookie to get work done and do damage
- ASP.Net sends an additional token with each POST request to prevent this type of attack (Anti-forgery token)
- You mark your code to “validate” the token on posts
 - On the controller class or method (action): `[ValidateAntiForgeryToken]`
 - On the controller class: `[AutoValidateAntiforgeryToken]`
 - Globally: `services.AddMvc(options => options.Filters.Add(new AutoValidateAntiforgeryTokenAttribute()));`
- <https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-2.2>



Demo

Session – What it's for

- Http is inherently stateless
 - Every request is independent from the last one
 - This helps make sites highly scalable
- Sometimes we need to maintain state between requests
- Session is one way of managing state (anonymously)
- Examples
 - Shopping experience. Browse, add to cart, browse, add, view cart, check out
 - Multi-page job application
 - Multi-page tax form submission
- <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-2.2#state-management>

Session – How it works

- On the user's first request
 - Server creates a "Session" object (like a dictionary)
 - Server creates a Session Id and stores the object with the Id
 - Server sends the Session Id to the client in a "cookie"
- On subsequent user requests
 - Browser sends the cookie information automatically
 - Server uses the Session Id to find the Session object
 - Then our code has access to the Session object
- The Session object
 - Store Key => Value
 - Key is string
 - Value is either string or Int32 (you can mix)

Session – Setup (Startup.cs)

- In ConfigureServices:

```
// Add MVC services to the services container.  
services.AddMvc();  
services.AddDistributedMemoryCache(); // Adds a default in-memory implementation of IDistributedCache  
services.AddSession();
```

- In Configure:

```
// IMPORTANT: This session call MUST go before UseMvc()  
app.UseSession();  
  
// Add MVC to the request pipeline.  
app.UseMvc(routes =>  
{  
    routes.MapRoute(  
        name: "default",  
        template: "{controller=Home}/{action=Index}/{id?}");  
});
```



Demo

Using Session

- `HttpContext.Session.GetString / SetString`
- `HttpContext.Session.GetInt32 / SetInt32`
- Complex objects must be *Serialized* to strings when Setting
- Then *de-serialized* back to objects when Getting
- We use JSON (JavaScript Object Notation)



Let's
Code

TempData

- Similar to Session data, but stores data for this and next request
- Access it like you access ViewData
- Store:

```
TempData["myKey"] = "myValue";
```

- Retrieve:

```
string theData = (string)TempData["myKey"];
```
- Often used in conjunction with Redirects to “pass” data
- Example: Add City



Let's
Code