Chapter 6 Sample Question and Answer

1.Explain about State Management Strategies?

HTTP is a stateless protocol. So **HTTP requests** are independent messages that don't retain user values or app states. We need to take additional steps to manage state between the requests.

There two state management strategies
1. Server Side(Session State, TempData, HttpContext, Cache)

2. Client Side(Cookies, Query Strings, Hidden Fields)

Server Side State Management:

- Session State

Session state is an ASP.NET Core mechanism to store user data while the user browses the application. It uses a store maintained by the application to persist data across requests from a client. While working with the Session state, we should keep the following things in mind:

- A Session cookie is specific to the browser session
- When a browser session ends, it deletes the session cookie
- An Application doesn't retain empty sessions
- The application retains a session for a limited time after the last request. The app either sets the session timeout or uses the default value of 20 minutes
- Session state is ideal for storing user data that are specific to a particular session but doesn't require permanent storage across sessions
- An application deletes the data stored in session either when we call the `ISession.Clear` implementation or when the session expires

We need to configure the session state before using it in our application. This can be done in the `ConfigureServices()` method in the `Startup.cs` class:

```
services.AddSession();
```

Then, we need to enable session state in the `Configure()` method in the same class:

```
app.UseSession();
```

The order of configuration is important and we should invoke the `UseSession()` before invoking `UseMVC()`.

Let’s create a controller with endpoints to set and read a value from the session:

```
HttpContext.Session.SetString("Name", "Sunil Chaudhary");
HttpContext.Session.SetInt32("Age", 36);
```

- TempData

ASP.NET Core exposes the `TempData` property which can be used to store data until it is read. We can use the `Keep()` and `Peek()` methods to examine the data without deletion. `TempData` is particularly useful when we require the data for more than a single request. We can access them from controllers and views.

`TempData` is implemented by TempData providers using either cookies or session state.

Example:

Server side code:

```
public class TempDataController : Controller
{
	public IActionResult First()
	{
	TempData["UserId"] = 101;
	return View();
	}
}
```

Client side code:

```
@{
	ViewData["Title"] = "Second";
	var userId = TempData["UserId"]?.ToString();
}
<h1>Second</h1>
User Id : @userId
```

- **Cache:**

Caching is a technique of storing frequently used data in a temporary storage area. Caching improves performance and scalability. When we implement caching on data, the copy of data is stored in the temporary storage area. Hence, when the same data is requested next time, it is picked up from temporary storage area, loading it much faster than from the original source.

The InMemory cache which uses ImemoryCache interface requires NuGet package **"Microsoft.Extensions.Caching.Memory".** We have installed it in the application. The following code snippet is used for project.json file.
we register it in the ConfigureServices method of Startup class, as per the following code snippet.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddMemoryCache();
}
```

- Using HTTPContext:

A HttpContext object holds information about the current HTTP request. The important point is, whenever we make a new HTTP request or response then the Httpcontext object is created. Each time it is created it creates a server current state of a HTTP request and response.
 It can hold information like: Request, Response, Server, Session, Item, Cache, User's information like authentication and authorization and much more.

Client Side State Management:

- Cookies

Cookies store data in the user's browser. Browsers send cookies with every request Ideally, we should only store an identifier in the cookie and we should store the corresponding data using the application. Most browsers restrict cookie size to 4096 bytes.
Users can easily delete a cookie. Cookies can also expire on their own. Hence we should not use them to store sensitive information and their values should not be blindly trusted or used without proper validations.

```
        //read cookie from Request object
             string userName = Request.Cookies["UserName"];
```

```
            //set the key value in Cookie
            CookieOptions option = new CookieOptions();
            option.Expires = DateTime.Now.AddMinutes(10);
            Response.Cookies.Append("UserName", userName, option);

            //Delete the cookie

            Response.Cookies.Delete("UserName");
```

- ## Query strings

We can pass a limited amount of data from one request to another by adding it to the query string of the new request. This is useful for capturing the state in a persistent manner and allows the sharing of links with the embedded state.

Let's add a new method in our `WelcomeController`:

```
public IActionResult GetQueryString(string name, int age)
        {
            User newUser = new User()
            {
                Name = name,
                Age = age
            };
            return View(newUser);

        }
```

Now let's invoke this method by passing query string parameters:

`/welcome/getquerystring?name=Sunil&age=36`

- ## Hidden Fields

We can save data in hidden form fields and send back in the next request. Sometimes we require some data to be stored on the client side without displaying it on the page. Later when the user takes some action, we'll need that data to be passed on to the server side. This is a common scenario in many applications and hidden fields provide a good solution for this.

```
@Html.HiddenFor(model => model.Id)
```

Then we can use a submit button to submit the form:

```
<input type="submit" value="Submit" />|
```

2.What is Session State in Asp.net core? Explain with suitable examples.

Session State

Session state is an ASP.NET Core mechanism to store user data while the user browses the application. It uses a store maintained by the application to persist data across requests from a client. While working with the Session state, we should keep the following things in mind:

- A Session cookie is specific to the browser session
- When a browser session ends, it deletes the session cookie
- If the application receives a cookie for an expired session, it creates a new session that uses the same session cookie
- An Application doesn't retain empty sessions
- The application retains a session for a limited time after the last request. The app either sets the session timeout or uses the default value of 20 minutes
- Session state is ideal for storing user data that are specific to a particular session but doesn't require permanent storage across sessions
- An application deletes the data stored in session either when we call the `ISession.Clear` implementation or when the session expires

A Session State Example

We need to configure the session state before using it in our application. This can be done in the `ConfigureServices()` method in the `Startup.cs` class:

```
services.AddSession();
```

Then, we need to enable session state in the `Configure()` method in the same class:

```
app.UseSession();
```

The order of configuration is important and we should invoke the `UseSession()` before invoking `UseMVC()`.

Let's create a controller with endpoints to set and read a value from the session:

```
public class WelcomeController : Controller
    {
        public IActionResult Index()
        {

            HttpContext.Session.SetString("Name", "Sunil Chaudhary");
            HttpContext.Session.SetInt32("Age", 36);
            User newUser = new User()
            {
                Name = HttpContext.Session.GetString("Name"),
                Age = HttpContext.Session.GetInt32("Age").Value
            };
            return View(newUser);
        }

    }
```

Create a class inside model

User.cs

```
public class User
    {
        public string Name { get; set; }
        public int Age { get; set; }

    }
```

Index.cshtml

```
@model WebApplication18.Models.User

@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

<div>
    <h4>User</h4>
    <hr />
    <dl class="row">
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Age)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Age)
        </dd>
    </dl>
</div>
<div>
    @Html.ActionLink("Edit", "Edit", new { /* id = Model.PrimaryKey */ }) |
    <a asp-action="Index">Back to List</a>
</div>
```

3. What is Cookies? Explain with Examples.

Cookies

Cookies store data in the user's browser. Browsers send cookies with every request Ideally, we should only store an identifier in the cookie and we should store the corresponding data using the application. Most browsers restrict cookie size to 4096 bytes.

Users can easily delete a cookie. Cookies can also expire on their own. Hence we should not use them to store sensitive information and their values should not be blindly trusted or used without proper validations.

A Cookie Example

Let's create a new project and add a controller with endpoints to read and write values into cookies.

```
public class HomeController : Controller
    {
        public IActionResult Index()
        {   //read cookie from Request object
            string userName = Request.Cookies["UserName"];
            return View("Index", userName);
        }
        [HttpPost]
        public IActionResult Index(IFormCollection form)
        {
            string userName = form["userName"].ToString();

            //set the key value in Cookie
            CookieOptions option = new CookieOptions();
            option.Expires = DateTime.Now.AddMinutes(10);
            Response.Cookies.Append("UserName", userName, option);
            return RedirectToAction(nameof(Index));
        }
        public IActionResult RemoveCookie()
        {
            //Delete the cookie

            Response.Cookies.Delete("UserName");
            return View("Index");
        }


    }
```

In Index.cshtml

```
@model string
@{
    ViewData["Title"] = "Home Page";
}
@if (!string.IsNullOrWhiteSpace(Model))
{
    <div>Welcome back, @Model</div>
    @Html.ActionLink("Forget Me", "RemoveCookie")
}
```

```
else
{

    <form asp-action="Index">
        <span>Hey, seems like it's your first time here!</span><br />
        <label>Please provide your name:</label>
        @Html.TextBox("userName")
        <div class="form-group">
            <input type="submit" value="Update" class="btn btn-primary" />
        </div>
    </form>
}
```

4. What is the need of State Management? Difference Between Cookies and Session?

Let us assume that someone is trying to access a banking website and he has to fill in a form.

So the person fills in the form and submits it. After submission of the form, the person realizes he has made a mistake. So he goes back to the form page and he sees that the information he submitted is lost. So he again fills in the entire form and submits it again. This is quite annoying for any user. So to avoid such problems "STATE MANAGEMENT"

Cookie is a client side storage of your variables. It stored on client machine by browser physically. It's scope is machine wide. Different users at same machine can read same cookie.

Because of this :

1. You should not store sensitive data on cookie.
2. You should not store data that belongs to one user account.
3. Cookie has no effect on server resources.
4. Cookie expires at specified date by you.

Session is a server side storage of your variables. Default, it stored on server's memory. But you can configure it to store at SqlServer. It's scope is browser wide. Same user can run two or more browsers and each browser has it's own session.

Because of this :

1. You can save sensitive data in session.
2. You should not save everything in session. it's waste of server resources.
3. After user closes browser, session timeout clears all information. (default is 20 minutes)