Chapter 6: State Management on ASP.NET Core Application

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.

We're going to divide this article into the following sections:

- Cookies
- Session State
- Query strings
- Hidden Fields
- TempData
- Passing Data into Views

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");
        }


    }
```

The `Get` version of the `Index()` method reads the `UserName` from the cookie and pass it to the view.

We use the `Post` version of the `Index()` method to get the value for `userName` from the form collection and assign it to the cookie.

For removing the cookie value, we use the `RemoveCookie()` endpoint.

Now let's create the view: **index.cshtml**

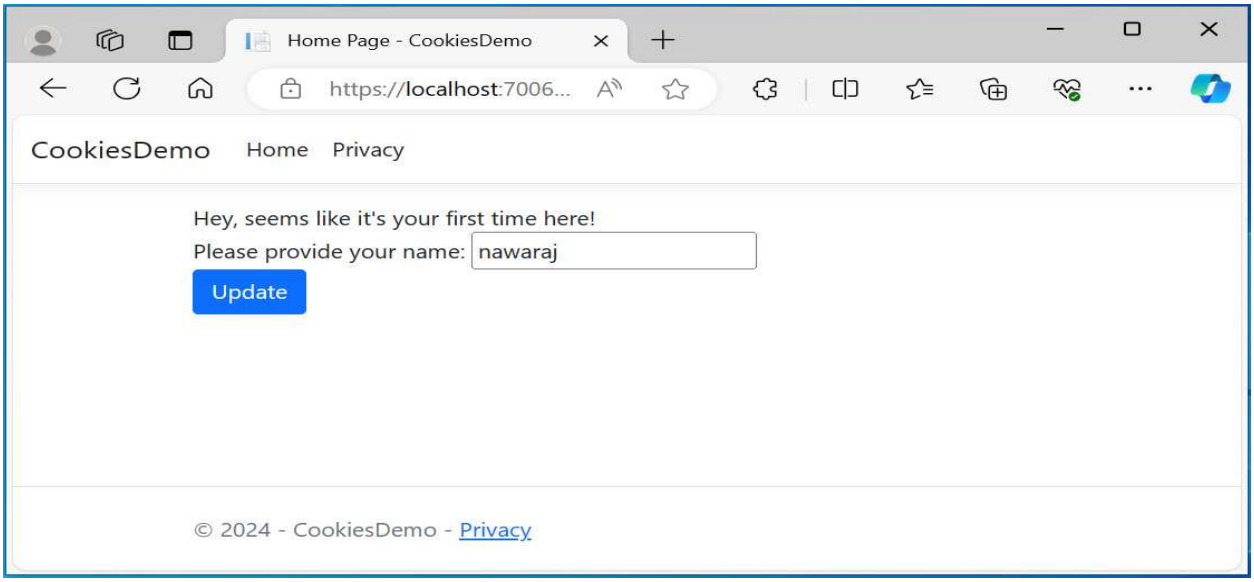
```
@model string
@{
    ViewData["Title"] = "Home Page";
}
@if (!string.IsNullOrWhiteSpace(Model))
{
    <div>Welcome back, @Model</div>
    @Html.ActionLink("Forget Me", "RemoveCookie")
}
else
{

    <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>
```

}Here, we pass the `UserName` value into the View as the model.

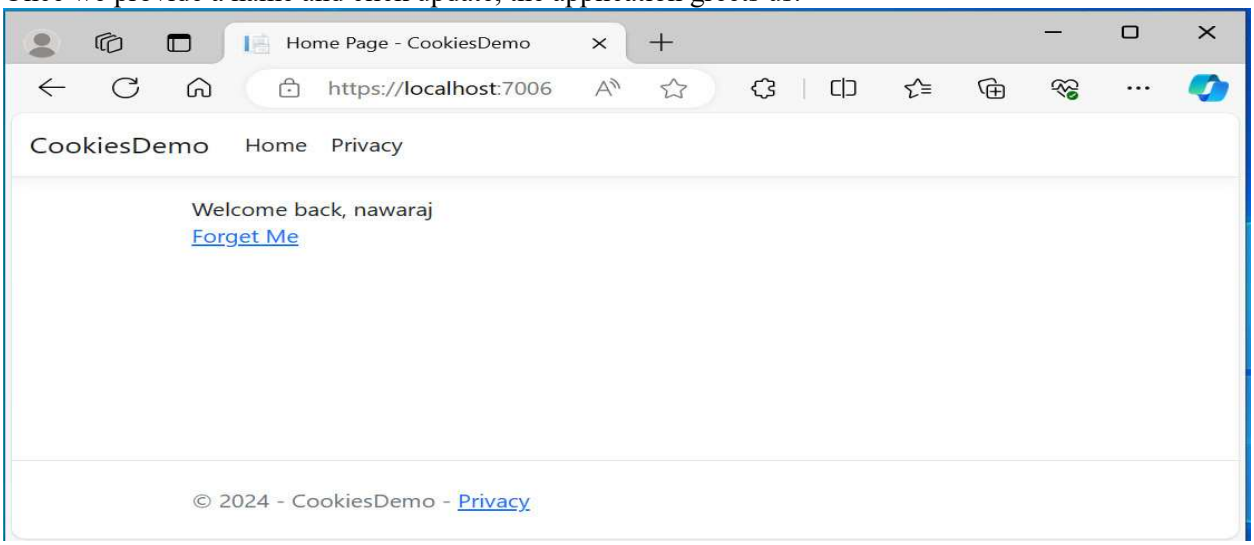
If the `UserName` has a value, we greet the user by that name and give the user an option to forget the value by removing it from the cookie.

In case the `UserName` is empty, we show an input field for the user to enter his name and a submit button to update this in the cookie.

Now let's run the application. Initially, the application asks the user to provide a name:



Once we provide a name and click update, the application greets us:



Even if we close and reopen the application, we can see that the cookie value persists. Once we click “Forget Me”, the application removes the cookie and asks for the name again.

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

program.cs file

```
//add before var app=builder.Build();
builder.Services.AddSession();

//var app=builder.Build();
app.UseSession();
```

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", "Nawaraj Prajapati");
      HttpContext.Session.SetInt32("Age", 43);
```

```
            User newUser = new User()
            {
                Name = HttpContext.Session.GetString("Name"),
                Age = HttpContext.Session.GetInt32("Age").Value
            };
            return View(newUser);
        }

    }
```

Create a class inside model

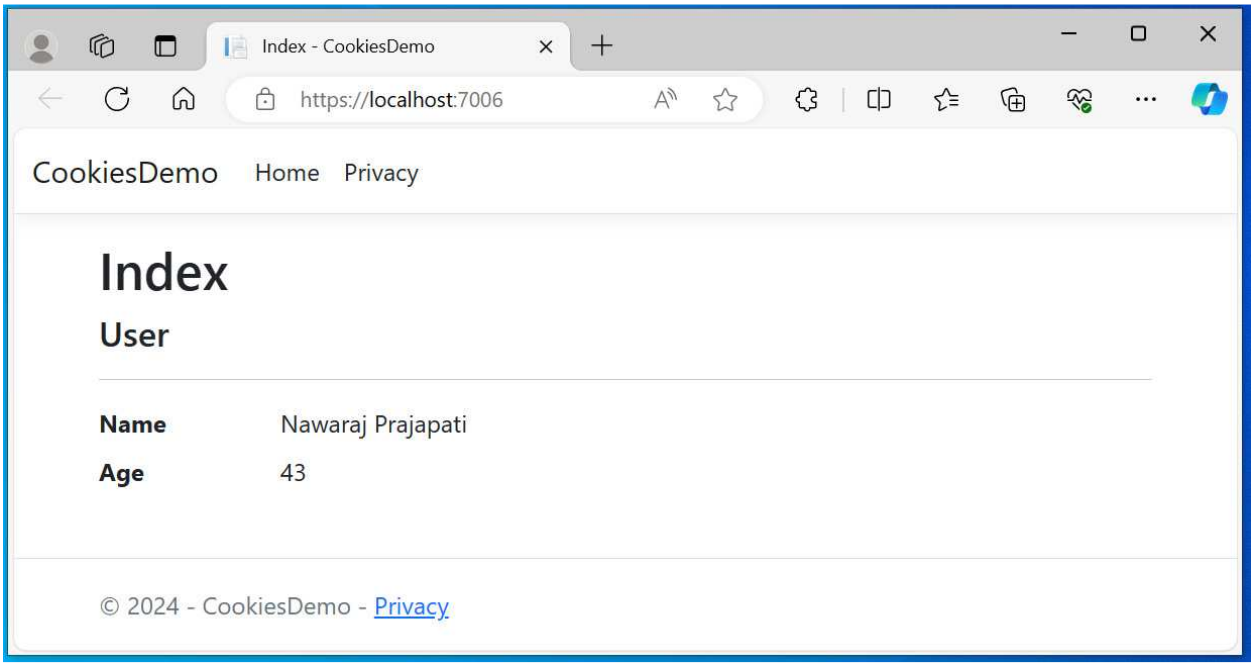
User.cs

```
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>
```

Output:



Query strings

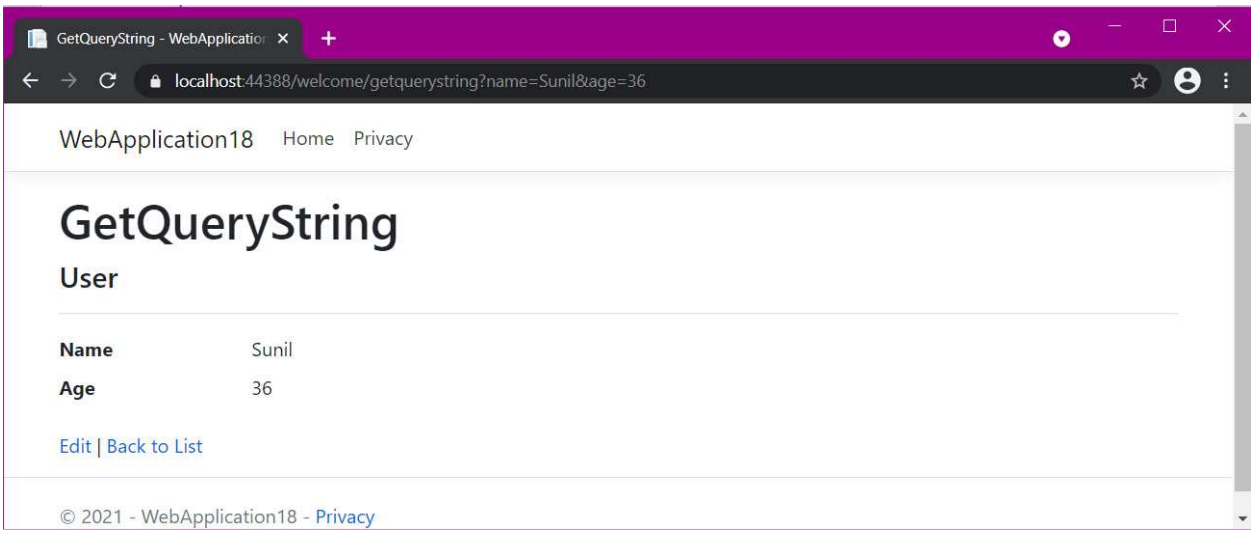
We can pass a limited amount of data from one request to another by adding it to the query string of the new request. This is useful for capturing the state in a persistent manner and allows the sharing of links with the embedded state.

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.

First Create a class inside model **User.cs**

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

   }
```

Let's add two methods in our `WelcomeController`:

```
[HttpGet]
      public IActionResult SetHiddenFieldValue()
      {
         User newUser = new User()
         {
            Id = 101,
            Name = "John",
            Age = 31
         };
         return View(newUser);
      }
      [HttpPost]
      public IActionResult SetHiddenFieldValue(User us)
      {
         var id = us.Id;
         ViewBag.ID = id;
```

```
        return View(us);
    }
```

The `GET` version of the `SetHiddenValue()` method creates a user object and passes that into the view.

We use the `POST` version of the `SetHiddenValue()` method to read the value of a hidden field `Id` from `FormCollection`.

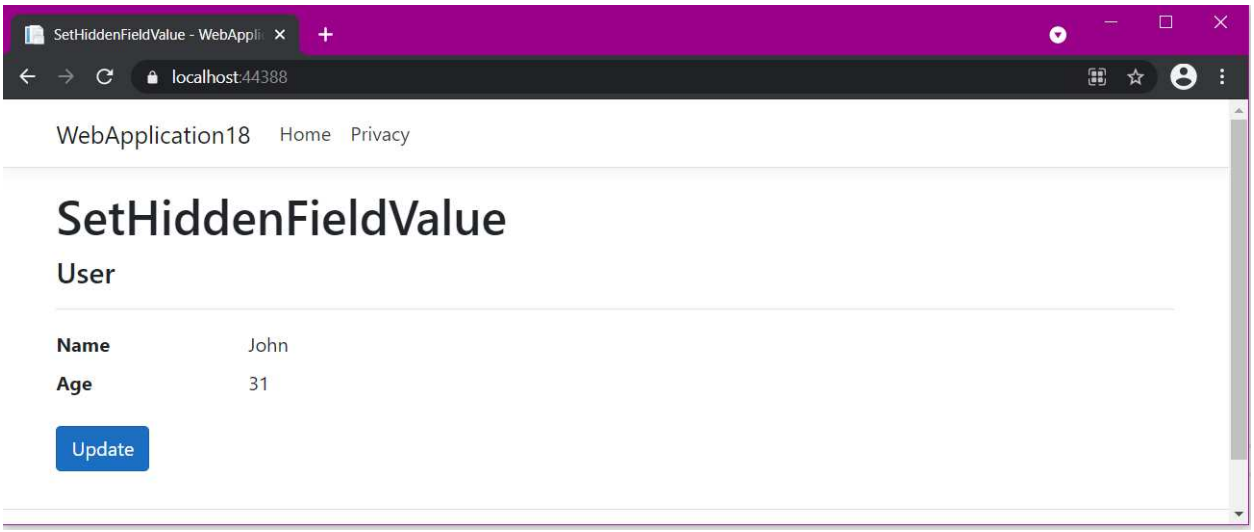
In the View, we can create a hidden field and bind the `Id` value from Model:

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

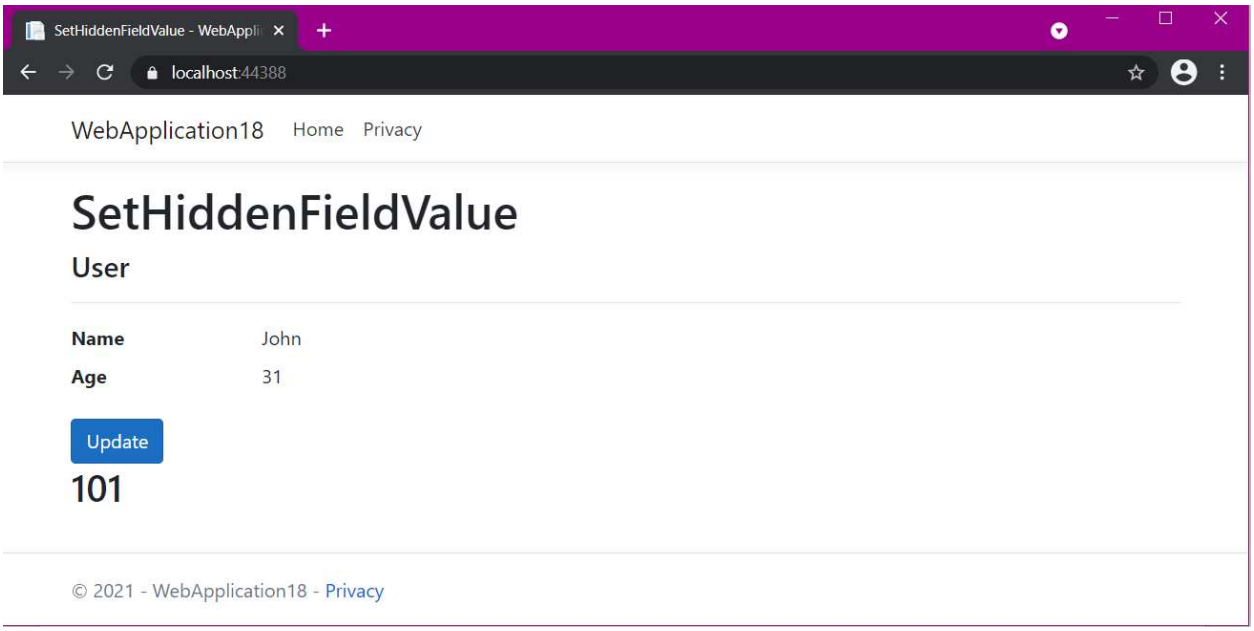
Then we can use a submit button to submit the form:

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

Now let's run the application and navigate to `/Welcome/SetHiddenFieldValue`:



After Update button click (output)


SetHiddenFieldValue - WebAppli
localhost:44388
WebApplication18
Home
Privacy
SetHiddenFieldValue
User
Name
John
Age
31
Update
101
© 2021 - WebApplication18 - Privacy