✅ STATE MANAGEMENT IN ASP.NET MVC

🔷 What is State Management?

In web development, HTTP is stateless, which means data is not stored between two requests.

📌 State management is the process of preserving user data, page information, or application state across:

Multiple pages

Postbacks

Requests

🎯 Goal: Keep user or page-specific information alive even if the request or page changes.

🔀 TYPES OF STATE MANAGEMENT:

There are two broad categories:

Category

- Client-Side

  Where data is stored→ On the user's browser/device

  Examples → ViewBag, TempData, Cookies, QueryStrings

- Server-Side

  Where data is stored→ On the server's memory

  Examples → Session, Application

🔸 CLIENT-SIDE STATE MANAGEMENT

Data is stored on the user's side (browser or query string). Let's go one-by-one:

✅ 1. ViewBag

📍 Purpose: Used to pass data from Controller to View only during one request.

📌 Type: Dynamic (dynamic keyword)

📌 Use case: Simple values like messages or flags.

🧠 Example:

csharp

Copy

Edit

```
// Controller
public ActionResult EmployeeDetails() {
    Employee emp = new Employee() { Empid = 101, Ename = "Alex" };
    ViewBag.Message = "Employee Details";
    ViewBag.Emp = emp;
    return View();
}
```

html

Copy

Edit

```
<!-- EmployeeDetails.cshtml -->
<h3>@ViewBag.Message</h3>
@{
```

```
    var emp = ViewBag.Emp;
}
<table>
    <tr><td>ID:</td><td>@emp.Empid</td></tr>
    <tr><td>Name:</td><td>@emp.Ename</td></tr>
</table>
```

📝 Notes:

Only for same request

No casting required

Can store simple or complex objects

✅ 2. TempData

📍 Purpose: Used to pass data between two different requests (e.g., after a redirect).

📌 Type: Dictionary (TempDataDictionary)

📌 Backed by: Session (internally)

📌 One-time use – gets cleared after it's read

🧠 Example:

csharp

Copy

Edit

```
// Controller
public ActionResult TestTempData() {
```

```
    Employee emp = new Employee() { Empid = 101, Ename = "Alex" };
    TempData["Emp"] = emp;
    return RedirectToAction("About");
}

public ActionResult About() {
    var emp = (Employee)TempData["Emp"];
    TempData.Keep(); // Keeps it for next request too
    return View(emp);
}
```

html

Copy

Edit

```
<!-- About.cshtml -->
<p>
    Employee ID: @Model.Empid <br />
    Name: @Model.Ename
</p>
```

📝 Notes:

Used when doing RedirectToAction()

TempData.Keep() preserves it

TempData.Peek() reads it without removing

✅ 3. Cookies

📍 Purpose: Store small pieces of data (like session ID, user ID) on client's browser

📌 Lifetime: Can be temporary (until browser closes) or persistent (set expiry)

📌 Stored as: Plain text

🧠 Example:

csharp

Copy

Edit

```
public ActionResult Cookietest() {
    HttpCookie cookie = new HttpCookie("TestCookie", "This is test cookie");
    Response.Cookies.Add(cookie);

    if (Request.Cookies["TestCookie"] != null) {
        string msg = Request.Cookies["TestCookie"].Value;
        ViewBag.CookieMessage = msg;
    }
    return View();
}
```

html

Copy

Edit

```
<!-- Cookietest.cshtml -->
<h3>Cookie: @ViewBag.CookieMessage</h3>
```

📝 Notes:

Accessible in JavaScript

Small size limit (4 KB)

Not secure – use HTTPS and HttpOnly for protection

✅ 4. Query Strings

📍 Purpose: Pass data in the URL using key-value pairs

📌 Syntax: ?key=value

🧠 Example:

csharp

Copy

Edit

```
// HomeController
public ActionResult QueryStringTest() {
    return RedirectToAction("Index", "Employee", new { Empid = 1, Ename = "Gary" });
}

// EmployeeController
public ActionResult Index() {
    Employee emp = new Employee() {
        Empid = int.Parse(Request.QueryString["Empid"]),
        Ename = Request.QueryString["Ename"]
    };
    return View(emp);
```

}

html

Copy

Edit

<!-- Index.cshtml -->

ID: @Model.Empid

Name: @Model.Ename

📝 Notes:

Visible in URL (not secure)

Good for small and non-sensitive data

Max length limit of URL (~2048 characters)

🔸 SERVER-SIDE STATE MANAGEMENT

Data is stored on the server, more secure and private.

✅ 5. Session

📍 Purpose: Store data for a single user during a session (browser time)

📌 Stored in: Server memory (or DB/in-process)

📌 Key-Value pairs

🧠 Example:

csharp

Copy

Edit

```
// Controller
public ActionResult Form1(Employee emp) {
    Session["empid"] = emp.Empid;
    return View("Index");
}
```

html

Copy

Edit

```
<!-- Index.cshtml -->
@if (Session["empid"] != null) {
    <label>Employee ID: @Session["empid"]</label>
}
```

📝 Notes:

Stored per user, destroyed after timeout (default 20 mins)

Good for user login, cart, preferences

Can use Session.Abandon() to clear

✅ 6. Application State

📍 Purpose: Share data between all users & all sessions on the server

📌 Stored in: Server memory

📌 Used for global counters, config values

🧠 Example:

csharp

Copy

Edit

Application["totalUsers"] = 100;

Access in other places:

csharp

Copy

Edit

int userCount = (int)Application["totalUsers"];

📝 Notes:

Data shared across entire app

Less commonly used now

Exists as Global.asax events like:

Application_Start

Application_End

🔹 Feature	🔸 ViewBag	🔸 TempData	🔸 Cookies	🔸 QueryString	🔸 Session	🔸 Application
Scope	Controller → View	Controller → Another Action	Multiple Visits	URL-based	Per User Session	All Users (Shared)
Lifetime	Current Request Only	One Redirect (or until read)	As per Expiry (set manually)	One Request (unless reused)	Until Timeout or Logout	Until Application Ends
Stored Where?	Server Memory	Server Memory (Session)	Client's Browser	URL Parameters	Server (Session Object)	Server (Application Object)
Security	Secure (Internal Only)	Secure	❌ Less Secure (Client-side)	❌ Insecure (Visible in URL)	✅ Secure	✅ Secure

What is ViewData in ASP.NET MVC?

ViewData is a dictionary object (key-value pair) that is used to pass data from the Controller to the View during the same HTTP request.

🔹 Key Characteristics:

Feature	Value
Type	ViewDataDictionary (not dynamic)
Lifetime	Only for the current request
Stored as	string key → object value
Casting Required	✅ Yes (type casting is needed)
Scope	Controller → View

🧠 Syntax:

csharp

```
// Controller
ViewData["Message"] = "Welcome to ViewData!";
```

html

```
<!-- View -->
<h2>@ViewData["Message"]</h2>
```

🔍 Detailed Example:

✅ 1. Controller Code:

csharp

```
public ActionResult EmployeeDetails()
{
    Employee emp = new Employee() { Empid = 101, Ename = "Alex" };
    ViewData["Message"] = "Employee Info via ViewData";
    ViewData["Employee"] = emp;
    return View();
}
```

✅ 2. View Code (EmployeeDetails.cshtml):

html

```
<h2>@ViewData["Message"]</h2>

@{
    var emp = (YourNamespace.Models.Employee)ViewData["Employee"];
}
```

```
<table>
    <tr><td>ID:</td><td>@emp.Empid</td></tr>
    <tr><td>Name:</td><td>@emp.Ename</td></tr>
</table>
```

✅ Note: You must cast the object to its original type.

🟡 Differences: ViewData vs ViewBag vs TempData

Feature	ViewData	ViewBag	TempData
Type	Dictionary	Dynamic (ExpandoObject)	Dictionary backed by Session
Lifetime	Same request only	Same request only	Across multiple requests
Cast Required	✅ Yes	❌ No	✅ Yes
Use Case	Pass data to view	Pass data to view	Pass data across redirects

💡 When to Use ViewData?

If you need to pass multiple values (especially in loops or logic-heavy views).

When you want to store data that needs explicit type safety and will be accessed with known keys.

If your project already follows a ViewData-based convention (e.g., older ASP.NET MVC apps).

⚠️ Caution:

If the key is wrong or not present, accessing ViewData["key"] will return null.

Always check for null before using it.

Type casting errors can cause runtime exceptions.

✅ Best Practice Tip:

For large or strongly-typed data (like an entire employee object or a form model), prefer using a ViewModel. Use ViewData/ViewBag only for small, temporary, or auxiliary data like messages, flags, UI control visibility, etc