# MVC

Christen Zarif Foad

# OutLine

- State Management Technology in mvc
- Pass Data To View
  - ViewData
  - ViewBag
  - TempData
  - Session
  - Cookie
  - Model
- ViewModel

# State Management

- State refers to the current states(value) of the properties ,variables ,and other data maintained by an app for single user
- HTTP is a stateless protocol.
- Once the server serves any request from the user, it cleans up all the resources used to serve that request ,**and the state is lost**.
- These resources include the objects created during that request, the memory allocated during that request, etc
- State management is the process by which ASP.NET let the developers maintain state and page information over multiple request for the same or different pages.

# Strong type view

- Pass data  using model object
- Model:

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

- Controller

```
// GET: Movies/Random
public ActionResult Random()
{
    var movie = new Movie() { Name = "Shrek!" };

    return View(movie);
}
```

- View

```
@model Vidly.Models.Movie
@{
    ViewBag.Title = "Random";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>@Model.Name</h2>
```

# @model directive

- The @model directive provides a cleaner and more concise way to reference strongly-typed models from view files
- Razor will derive the view from the RazorPage<TModel> base class.

# Passing Data to Views

- To create a dynamic web site, we'll instead want to pass information from our controller actions to our view templates

- There are three ways to pass/store data between the **controllers and views and from action to action**
  - **ViewData**
  - **ViewBag**
  - **TempData**

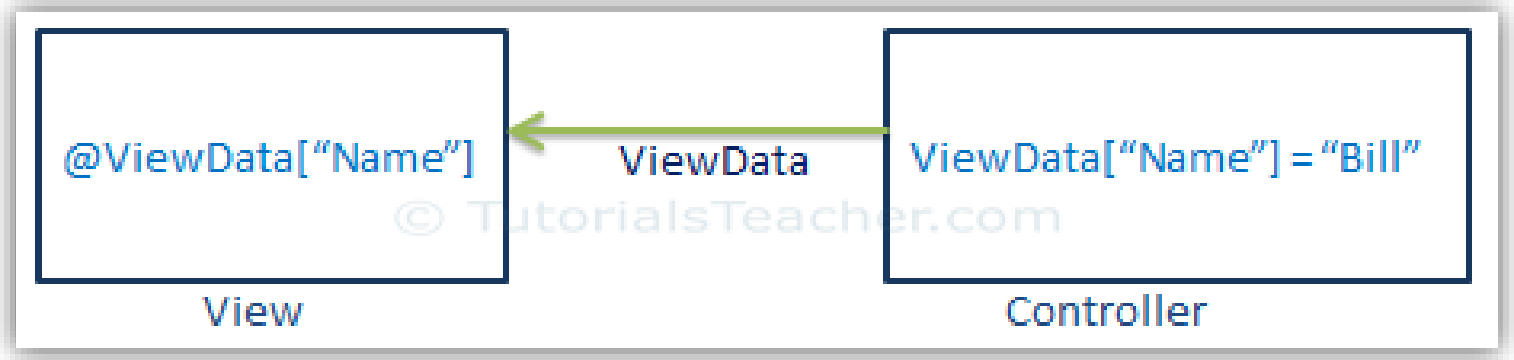
# ViewData / ViewBag

- There may be some scenario where you want to send a small amount of temporary data to the view.

- ViewBag - viewData can be useful when you want to transfer temporary data (which is not included in model) from the controller to the view.

# ViewData

- ViewData is used to pass data from controller to view
  - It is derived from ViewDataDictionary class
  - It is available for the **current** request only
  - Requires typecasting for complex data type and checks for null values to avoid error
  - If redirection occurs, then its value becomes null



# ViewBag

- ViewBag is also used to pass data from the controller to the respective view

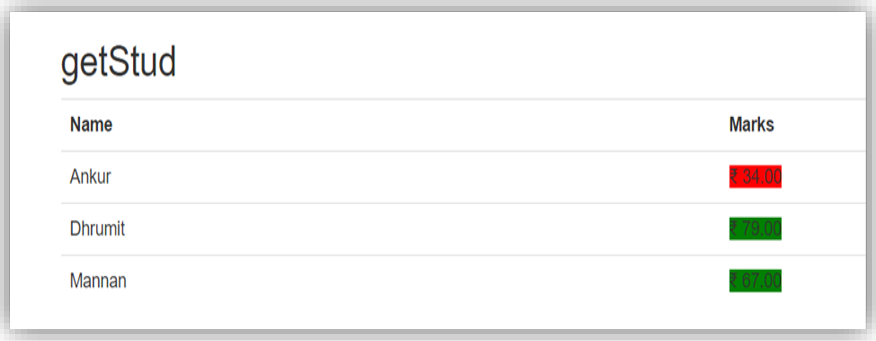

- ViewBag is a dynamic property
  - that takes advantage of the new dynamic features in C# 4.0
  - property of ControllerBase class which is the base class of all the controllers.It is also available for the current request only
- If redirection occurs, then its value becomes null
- Doesn't require typecasting for complex data type
- ViewBag is actually a wrapper around ViewData

- ViewData and ViewBag both use the same dictionary internally. So you cannot have ViewData Key matches with the property name of ViewBag

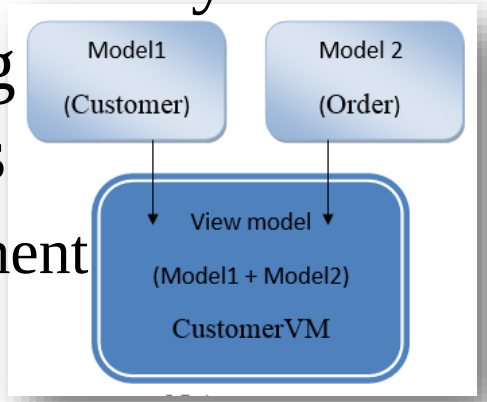
# ViewModel

# ViewModel

- ViewModels help you organize and manage data in MVC applications when you need to work with **more complex** data than the other objects **allow**.
  - **Ex**:Student Color Degree




- ViewModels area generally a more flexible way to **access multiple data sources** than using models with ViewBag/ViewData objects
  - Ex: Employee Data With List of all department



# Why Use a ViewModel

- Explicit declaration of data for view
- Easier for multiple teams and new developers
- Combining multiple models into a single view
- Formatting data for presentation

# ViewData & ViewModel

- The ViewData dictionary approach has the benefit of being fairly fast and easy to implement. Some developers don't like using string-based dictionaries, though, since typos can lead to errors that will not be caught at compile-time. The un-typed ViewData dictionary also requires using the "as" operator or casting when using a strongly-typed language like C# in a view template
- ViewData & ViewBag  it’s harming the Mvc-Pattern

# ViewModel

```
public class Movie
{
    public int ID { get; set; }
    public string Title { get; set; }
    public DateTime ReleaseDate { get; set; }
    public string Genre { get; set; }
    public decimal Price { get; set; }
    public string Rating { get; set; }
}
```

```
public class MovieViewModel
{
    public List<string> ListOfGenres { get; set; }
    public List<string> ListOfRatings { get; set; }
    public List<string> Theatre { get; set; }
    public string Time { get; set; }

    public int ID { get; set; }

    [Required(ErrorMessage = "Title is required")]
    public string Title { get; set; }

    [Required(ErrorMessage = "Date is required")]
    [DisplayFormat(DataFormatString = "{0:d}")]
    public DateTime ReleaseDate { get; set; }

    [Required(ErrorMessage = "Genre must be specified")]
    public string Genre { get; set; }

    [Required(ErrorMessage = "Price Required")]
    [Range(1, 100, ErrorMessage = "Price must be between $1 and $100")]
    [DisplayFormat(DataFormatString = "{0:c}")]
    public decimal Price { get; set; }

    [StringLength(5)]
    public string Rating { get; set; }

} MovieViewModel
```

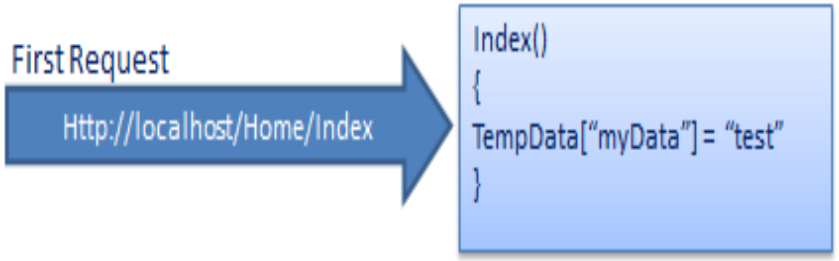
# TempData

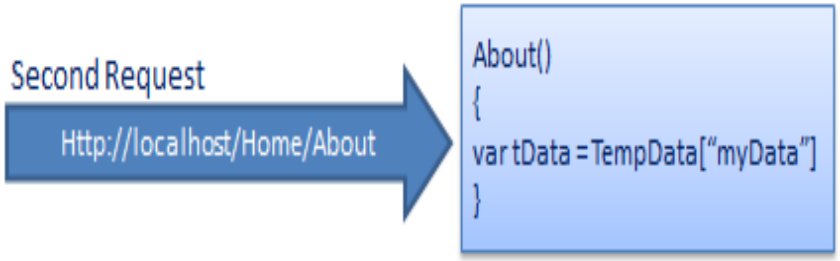
- Derived from TempDataDictionary class
- Used to pass data from the current request to the next request
- TempData dictionary is used to share data between controller actions.
- It helps to maintain the data when we move from one controller to another controller or from one action to another action
- **The value of TempData persists until it is read or until the current user's session times out.**
- By default, the TempData saves its content to the session state.
- It requires typecasting for complex data type and checks for null values to avoid error.
- Generally, it is used to store only one time messages like the **error messages** and **validation messages**
- Always do null check and do necessary action if TempData is null

```
public class HomeController : Controller
{
    // GET: Student
    public HomeController()
    {

    }
    public ActionResult Index()
    {
        TempData["name"] = "Test data";
        TempData["age"] = 30;

        return View();
    }

    public ActionResult About()
    {
        string userName;
        int userAge;

        if(TempData.ContainsKey("name"))
            userName = TempData["name"].ToString();

        if(TempData.ContainsKey("age"))
            userAge = int.Parse(TempData["age"].ToString());

        // do something with userName or userAge here

        return View();
    }
}
```

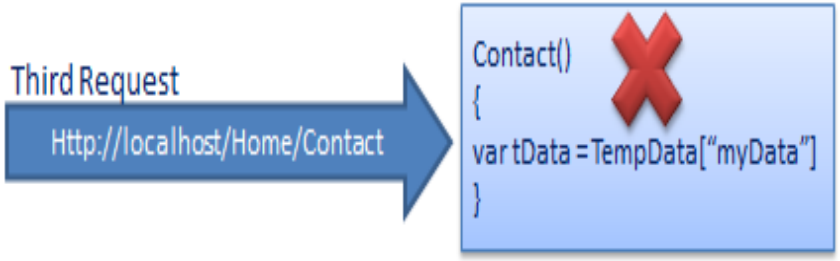
**we have converted values into the appropriate type**

# TempData




**TempData.Keep();**
**// to keep all the values of TempData in a third request.**




**because TempData will be cleared out after second request.**

# TempData (Con.)

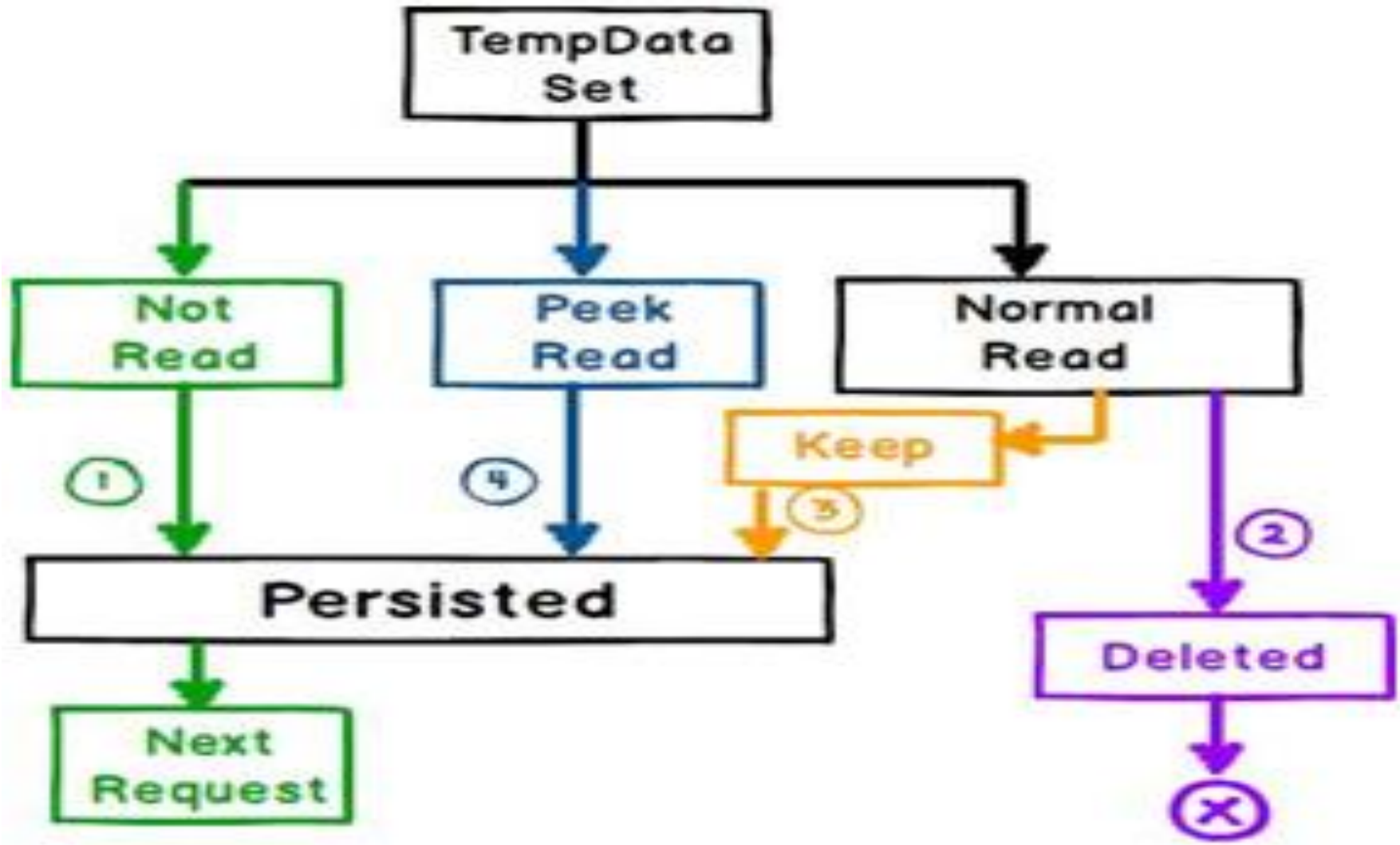
- If you want to keep TempData values even after reading them call Keep(). Then those values will keep for next request also.

```
TempData.Keep("CreditCardInfo");
```

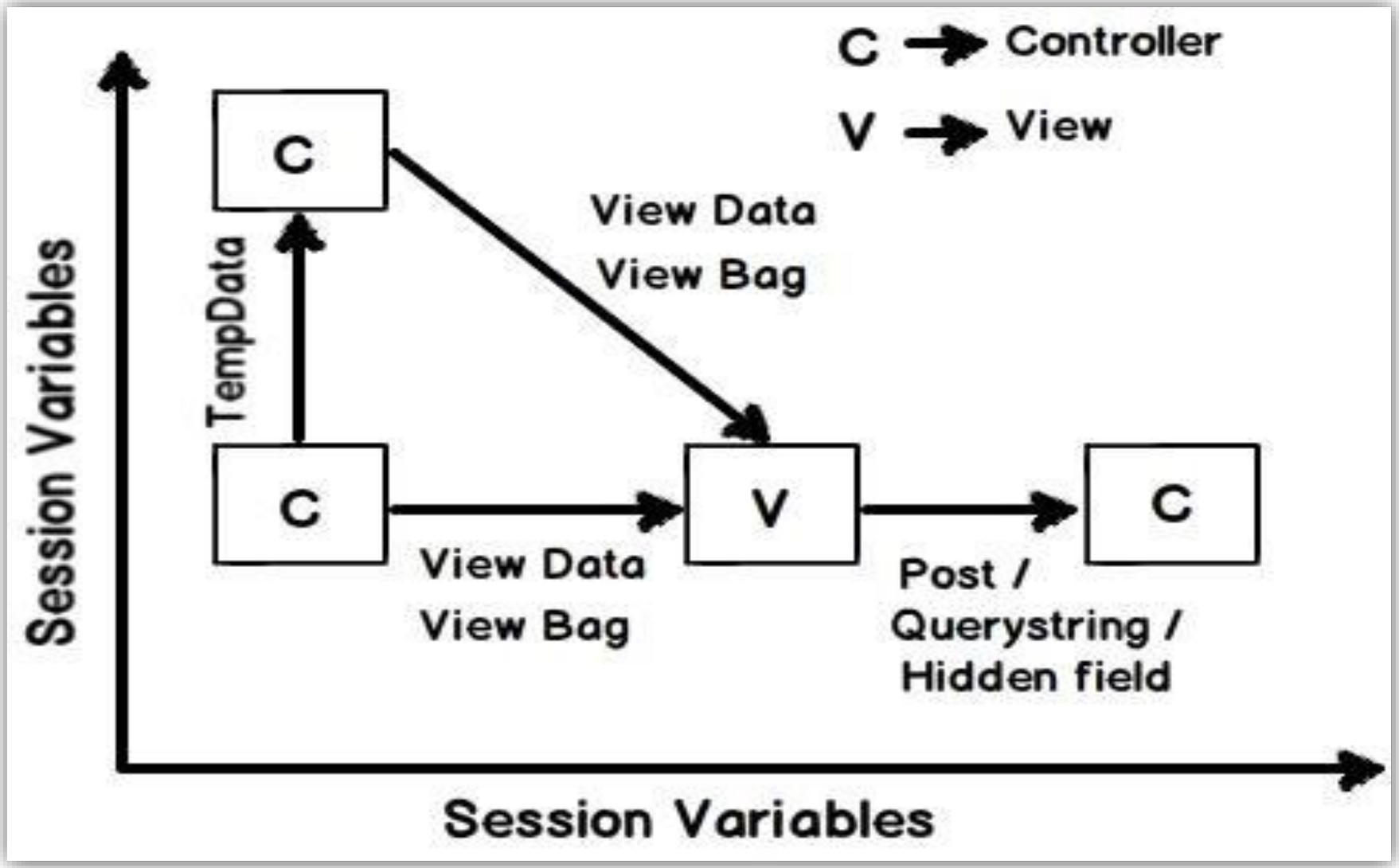
- If you want to remove TempData values then call Remove(). Then those values will remove from current and next request.

```
TempData.Remove("CreditCardInfo");
```

# TempData LifeTime



# Session Variables

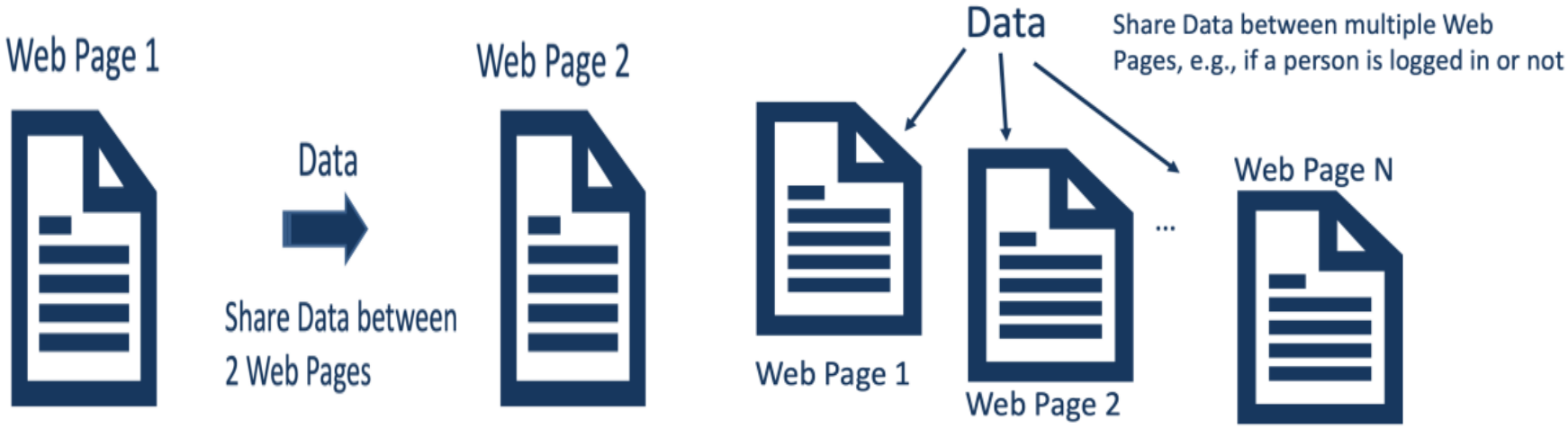


# Limitation

- Both ViewData & ViewBag does not provide compile time error checking
- Viewbag & viewdata not the best way to send data from controller to view
- So to pass data need to make Strong type view models because it provide  compiler time error checking

# Session Data

- HTTP is a stateless protocol. Without taking additional steps,
- HTTP requests are independent messages that do not retain user values or app state.
- We will see how we can use something called Session variables for sharing information between different web pages.



# State management

- The state can be stored using several approaches:
  - Cookies
  - Session State
  - TempData
  - Query String
  - Hidden fields • Etc

# SESSION STATE IN ASP.NET CORE

# Session State

- a mechanism that enables you to store and retrieve user specific values temporarily.
- The default time is 20 minutes (but can be configured if necessary).
- This information is Store in a global storage that is accessible from all pages in the Web application per User.

# Session State

- Session state is stored in the Session key/value dictionary.
- This information will be available to the current user only, i.e., current session only.
- Session management in ASP.NET Core is not enabled by default.

# How To Enable Session State

1. You need to install the **Microsoft.AspNetCore.Session NuGet** Package in order to use Session state.
2. You need to enable Session State in the Startup.cs file
   1. Call UseSession after UseRouting and before UseEndpoints
3. You need to include the Namespace Microsoft.AspNetCore.Http

# Session Middleware & services

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    services.AddSession();
    services.AddMemoryCache();
    ...
}
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ...
    app.UseSession();
    ...
}
```

# Write to Session Variables in C#

```
...
string name;
int age;

HttpContext.Session.SetString("Name", name);

HttpContext.Session.SetInt32("Age", age);
```

# Read from Session Variables in C#

- Read Session Data From Controller

```
string name;
int age;

name = HttpContext.Session.GetString("Name");

age = (Int32)HttpContext.Session.GetInt32("Age");
```

- Read Session Data From View

```
<h1>Index</h1>
@Context.Session.GetString("Key")
```

# How To Store Object in Session

```
// Save
var key = "my-key";
var str = JsonConvert.SerializeObject(obj);
context.Session.SetString(key, str);

// Retrieve
var str = context.Session.GetString(key);
var obj = JsonConvert.DeserializeObject<MyType>(str);
```

The extension methods on `ISession` are defined in the `Microsoft.AspNet(Core).Http` namespace.

1. Use **session state** to store user data that should **persist** across **multiple requests** in **a session**.
2. Use **cookies** to store user data that should **persist across multiple sessions.**