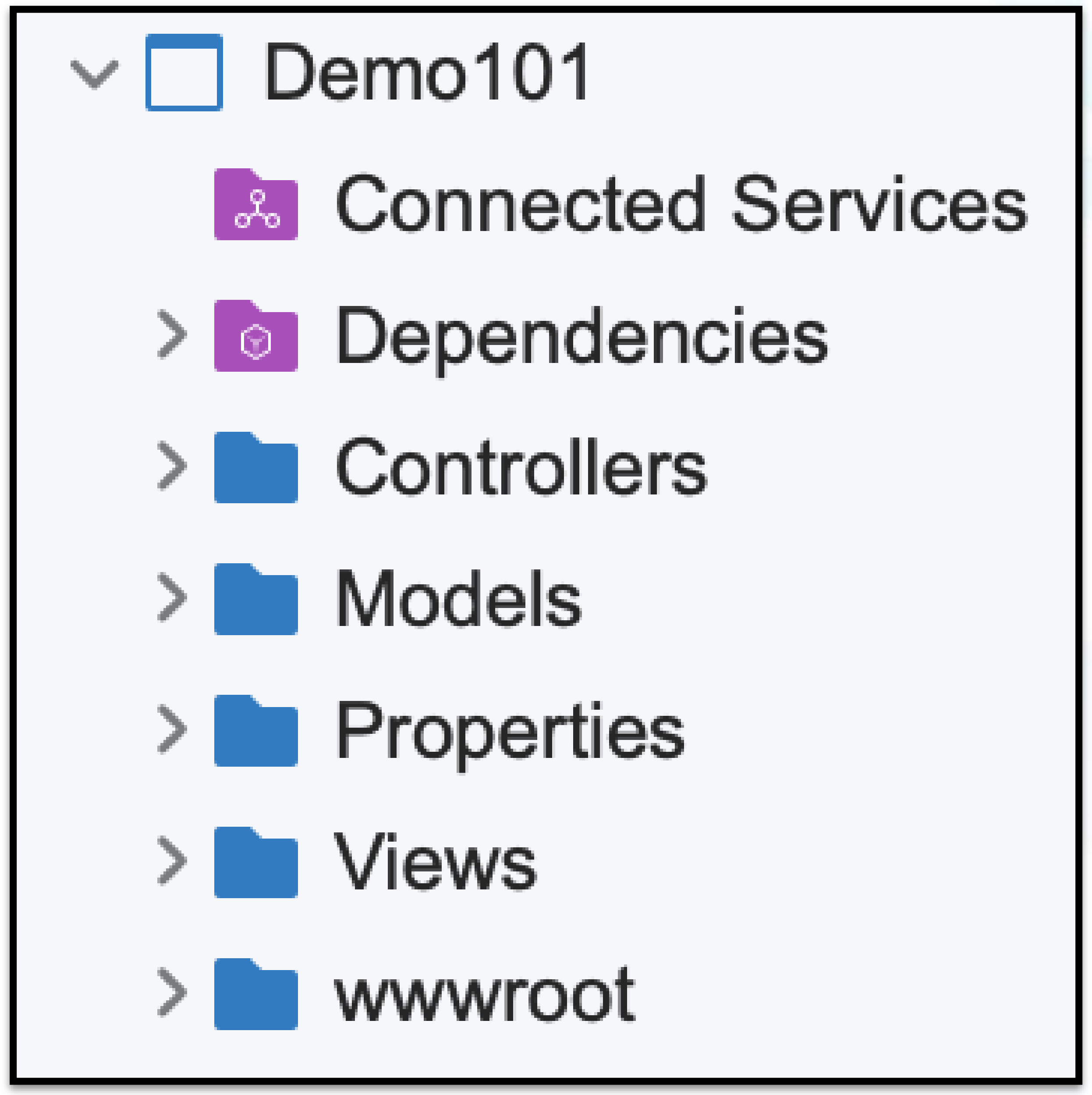
Razor

Tan Cher Wah (cherwah@nus.edu.sg)

CONVENTION IN ASP.NET

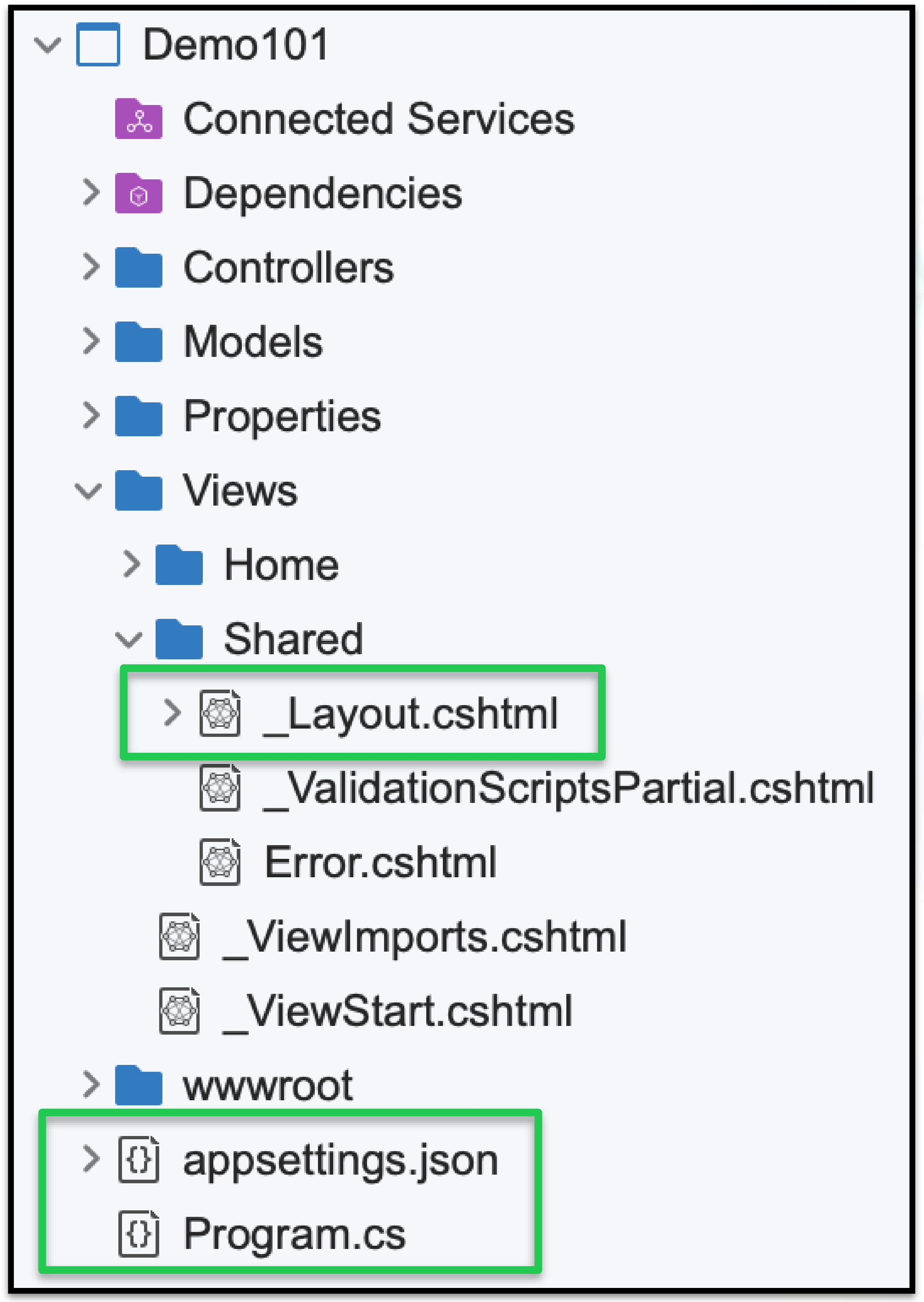
MVC Project Structure - Important Folders

- **/Controllers**
  - All controllers (.cs)
  - Contains business logic
- **/Views**
  - All views (.cshtml)
  - Contains user interfaces
- **/Models**
  - C# objects (.cs)
  - Contains data (Entity Framework)
- **/wwwroot (used by client only)**
  - JavaScript (.js)
  - StyleSheets (.css)
- **/<user-defined-folder>**
  - E.g. /Middleware folder to store all your Middleware classes



MVC Project Structure - Important Files

- **Program.cs**
  - Map routes to Controllers and Action Methods
  - Adding custom Middlewares to Middleware Pipeline
  - Adding Dependencies for Dependency Injection
- **appsettings.json**
  - Application-configuration such as database connection string
- **/Views/Shared/_Layout.cshtml**
  - Template used by every View
  - Where custom JavaScripts and Stylesheets can be added



Convention over Configuration

- ASP.NET adopts a Convention over Configuration philosophy - runtime assumes a particular **naming convention** to look for required components during execution

- It uses convention when it
  - Looks for the Controller to **route** a web request to based on a URL pattern (i.e. /<Controller>/<Action-Method>)
  - Looks for the correct **View** to use for an Action Method (i.e. /Views/<Controller>/<Action-Method>.cshtml)

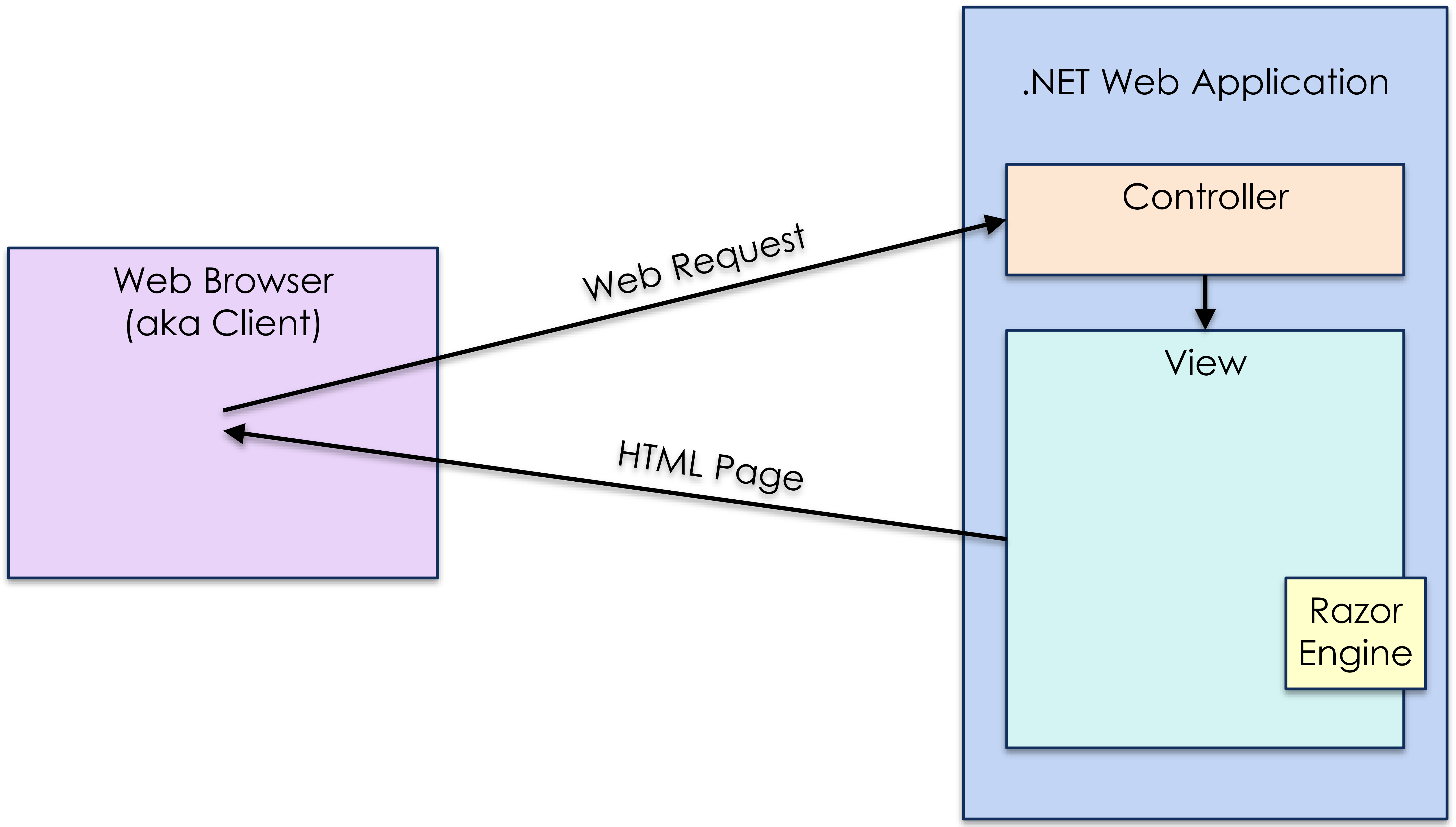
RAZOR

Razor

- Razor allows us to create web pages **programatically**
- Razor enables a **mix** of HTML tags and C# code in a **View**
- A **View** has a **.cshtml** file extension, and Razor code in it executes to produce a HTML page
- That **HTML page** is then sent over to the client (e.g. a web browser)

Relation of Razor and View

- A typical flow between a Web Browser and our .NET application
- Razor is used, as a scripting language, within Views to generate HTML pages



Razor

- Razor uses a subset of C# keywords

- Razor **keywords**
  - do while, while
  - for, foreach
  - if, else, else if
  - switch, case, default
  - try, catch, finally
  - class (reserved but unused)

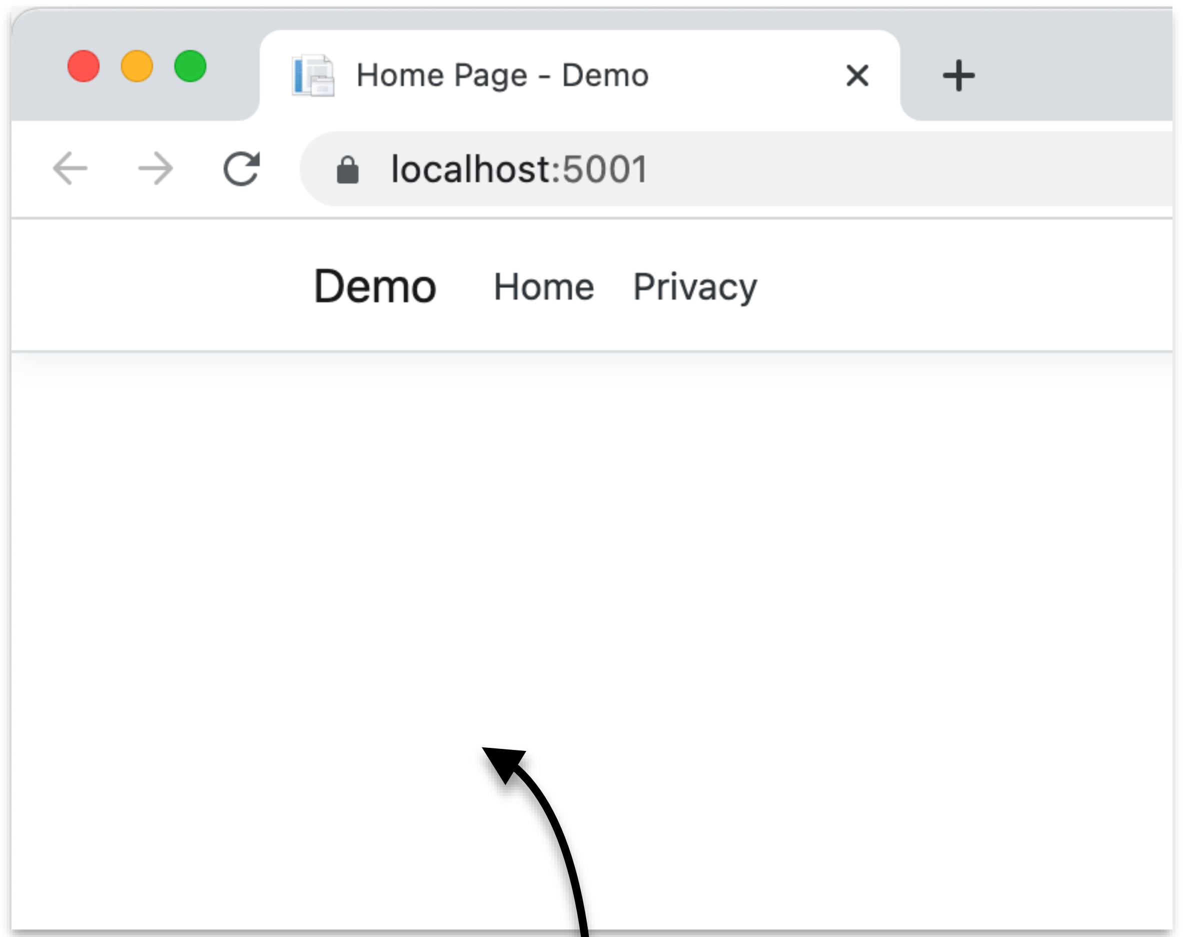
- All code blocks must be enclosed within a open and close braces – "{ … }" (such as *for* and *if*))

Razor

- In Razor, by default, anything within @{ } is code
- Notice that the Razor code in our View is not visible to the client

Razor knows everything here is **code**

```
@{
    int a = 1;
    int b = 2;

    int sum = a + b;
}
```



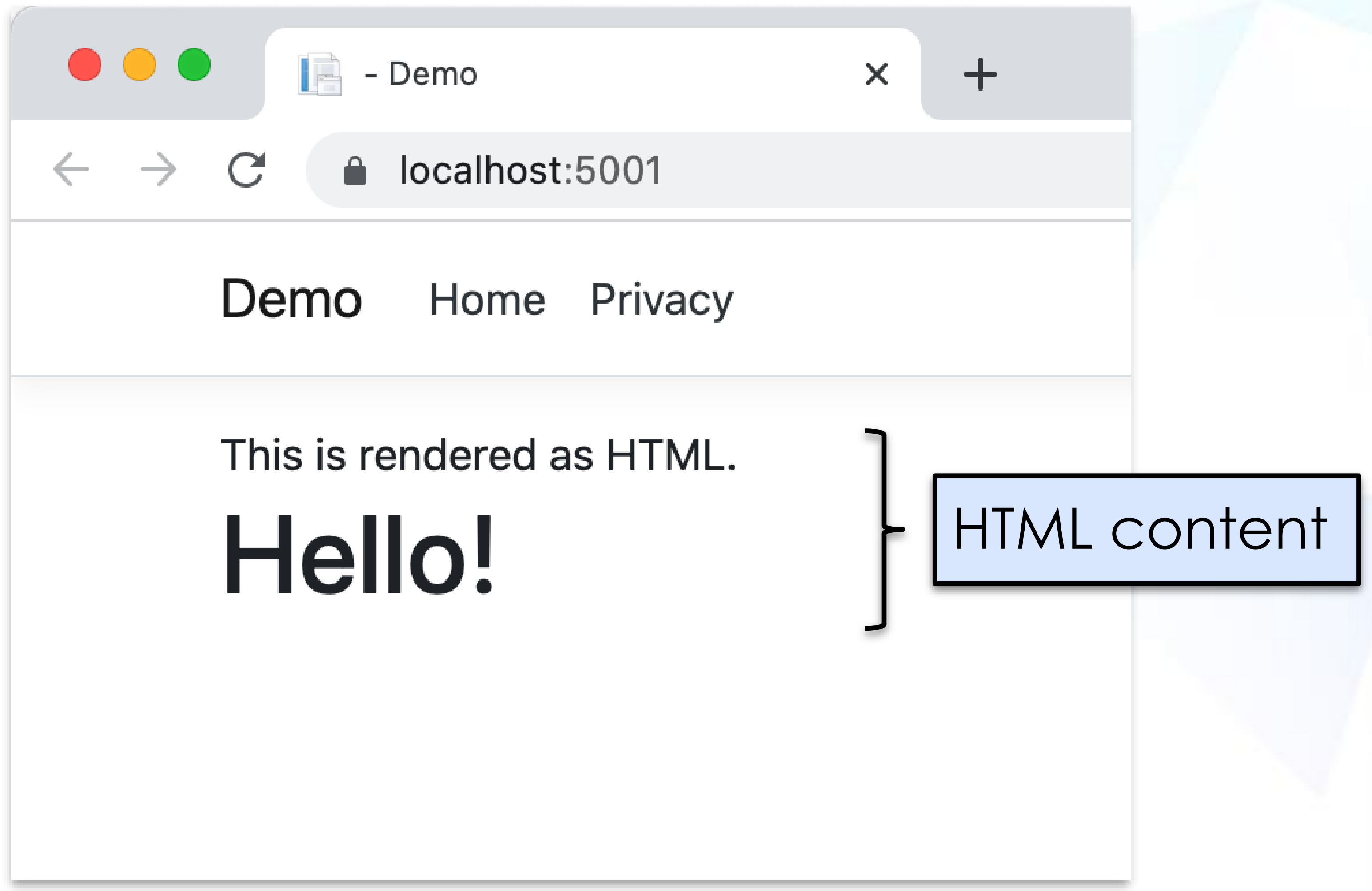
Nothing is displayed to the user because we only did computation

Razor

- In Razor, by default, anything outside a @{…} block is HTML

```
@{
    int a = 1;
    int b = 2;

    int sum = a + b;
}

This is rendered as HTML.

<h1>Hello!</h1>
```

Razor sees these as HTML text


- Demo
localhost:5001
Demo Home Privacy

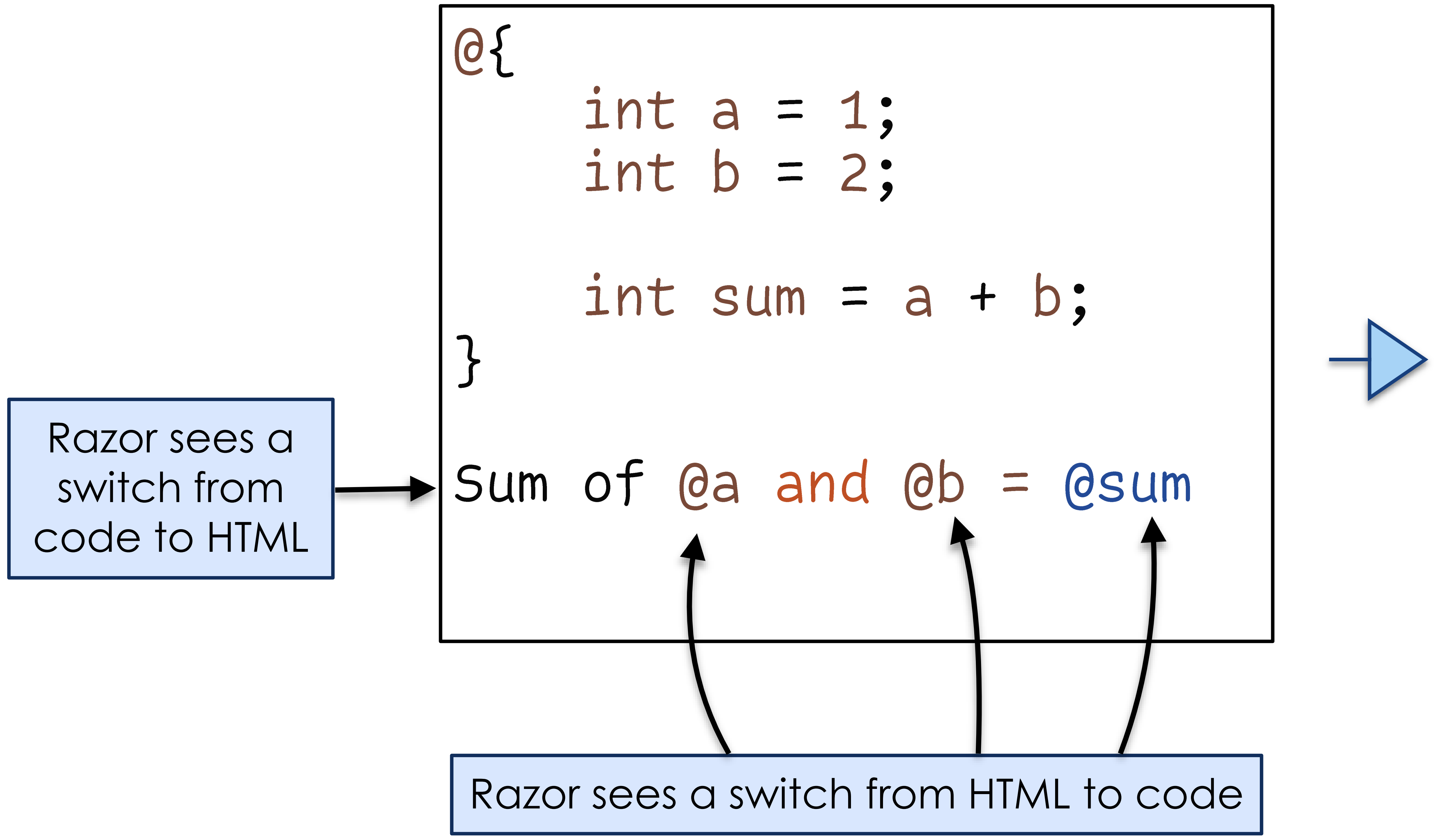
This is rendered as HTML.

Hello!

HTML content

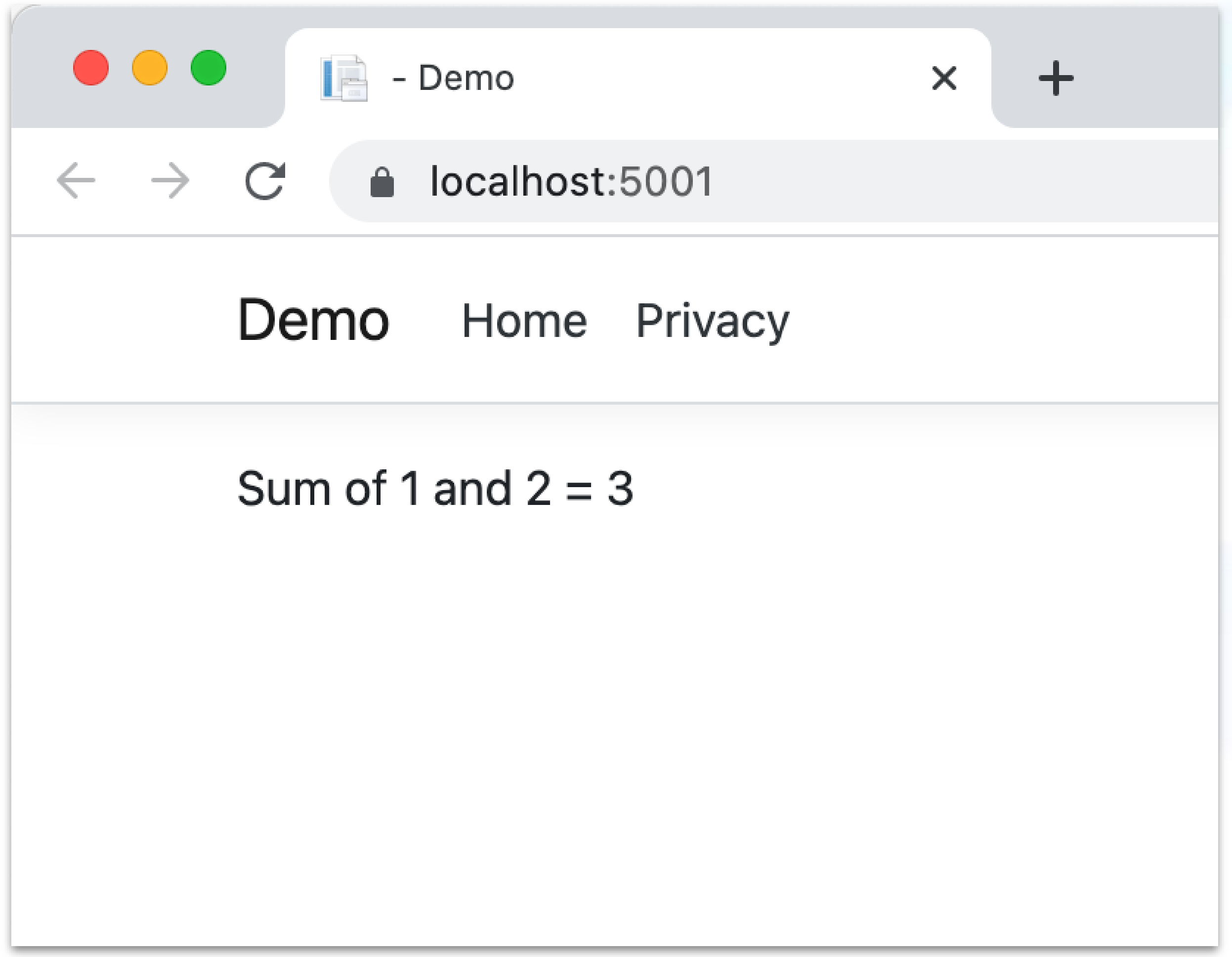
Razor

- Prefix our variables with the @ symbol to make their values visible to the user



```
@{
    int a = 1;
    int b = 2;

    int sum = a + b;
}

Sum of @a and @b = @sum
```

Razor sees a switch from code to HTML
Razor sees a switch from HTML to code



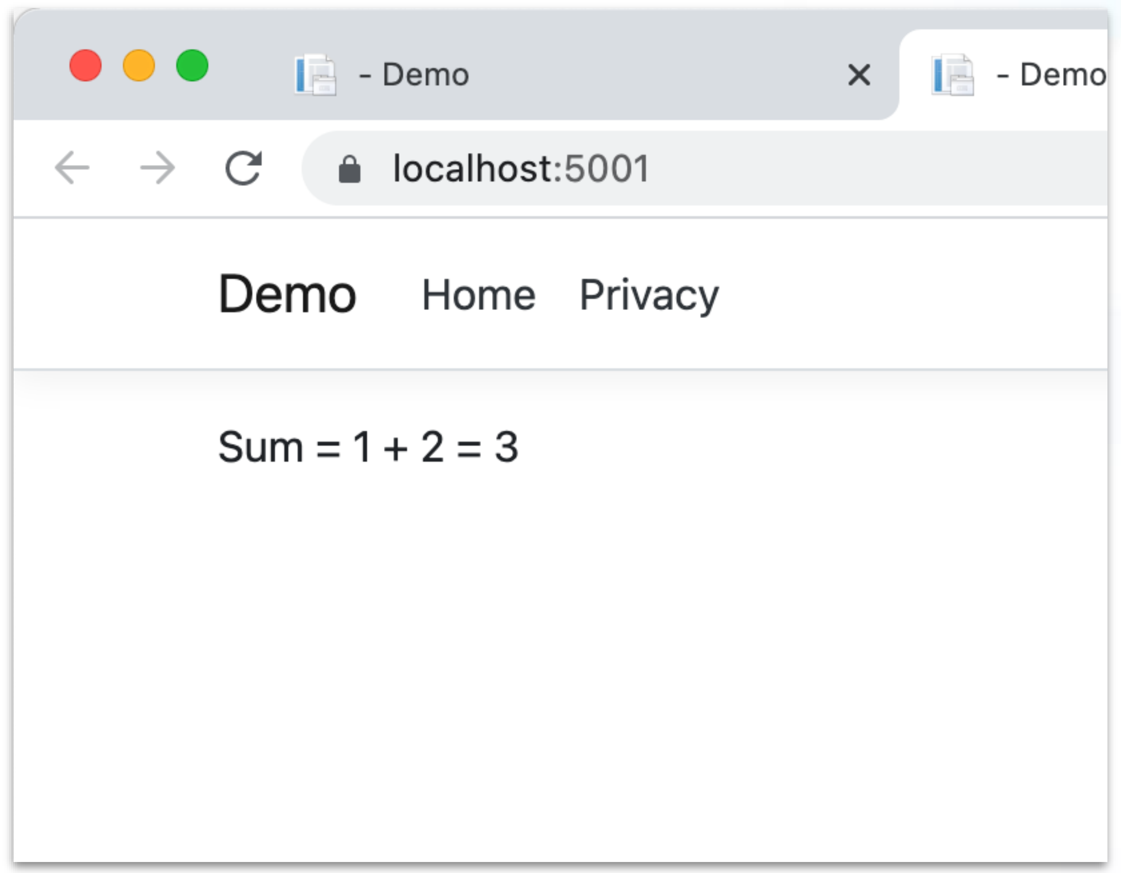
```
Sum of 1 and 2 = 3
```

Razor

- Razor interprets anything after @: as HTML - this behavior persists till the end of that line
- Razor allows the syntax @(…) where … is replaced with an expression (e.g. a + b, a++, a = 1)

```
@{
    int a = 1;
    int b = 2;

    @:Sum = @a + @b = @(a + b)
}
```

An **expression** is something that can be **evaluated** to determine its **value**


- Demo
- Demo
localhost:5001
Demo Home Privacy
Sum = 1 + 2 = 3

Data Types in Razor

- int, long (whole numbers)
- float (numbers with decimal points)
- decimal (higher precision than ‘float’)
- bool (true/false)
- string (values enclosed in “”)
- var (type inference) - adopts type of values first assigned

```
@{
    var data = 1;
    data = 2;

    data = "hello"; // error!!
}
```

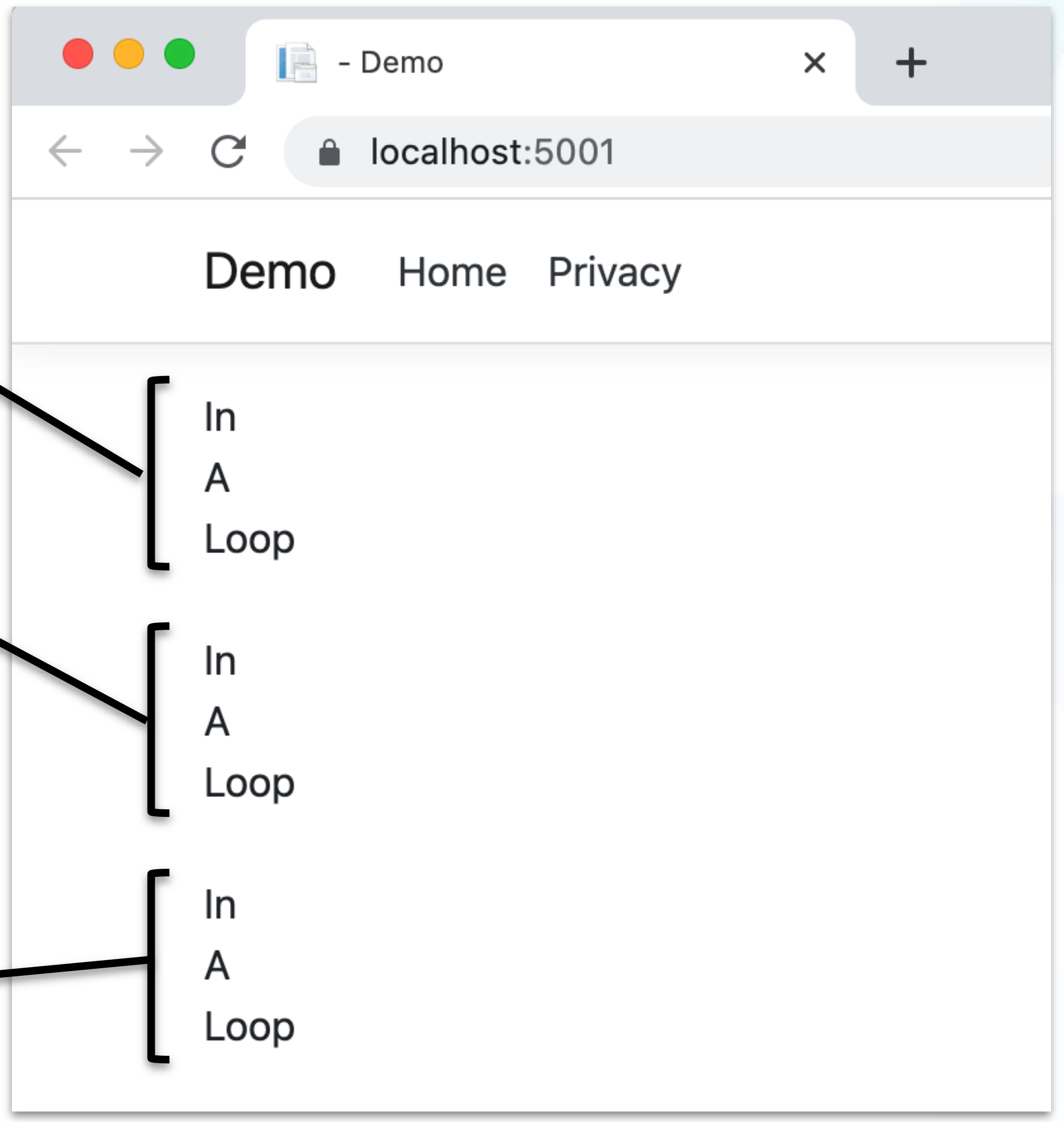
```
@{
    var data = "hello";
    data = "world";

    data = 10; // error!!
}
```

Only accept values of the **data type** that it was **first assigned**

Loop

- Using *foreach*, *for* and *while* to loop through our data

```
@{
    string[] items = { "In", "A", "Loop" };

    foreach (string item in items) {
        @:@item <br />
    }

    <br />

    for (int i = 0; i < items.Length; i++) {
        @:@items[i] <br />
    }

    <br />

    int j = 0;
    while (j < items.Length) {
        @:@items[j] <br />
        j++;
    }
}
```

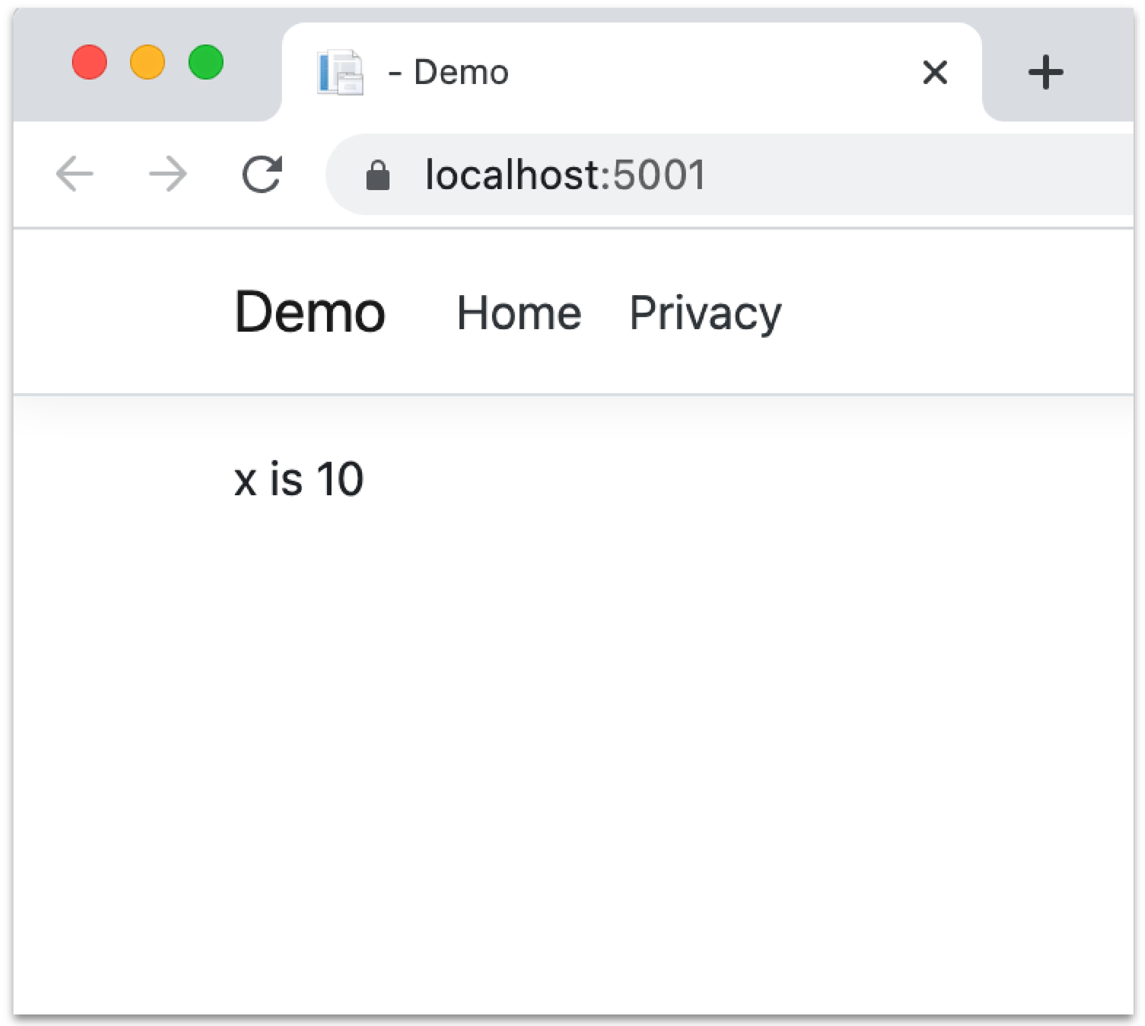


```
In
A
Loop

In
A
Loop

In
A
Loop
```

If-Else statements

- In Razor, a @if begins a if-else statement
- Its if-else logics follow the rules defined in the C# language

```
@{
    int x = 10;
}

@if (x == 1)
{
    @:x is 1
}
else if (x == 2)
{
    <b>x = 2</b>
}
else
{
    @:x is @x
}
```

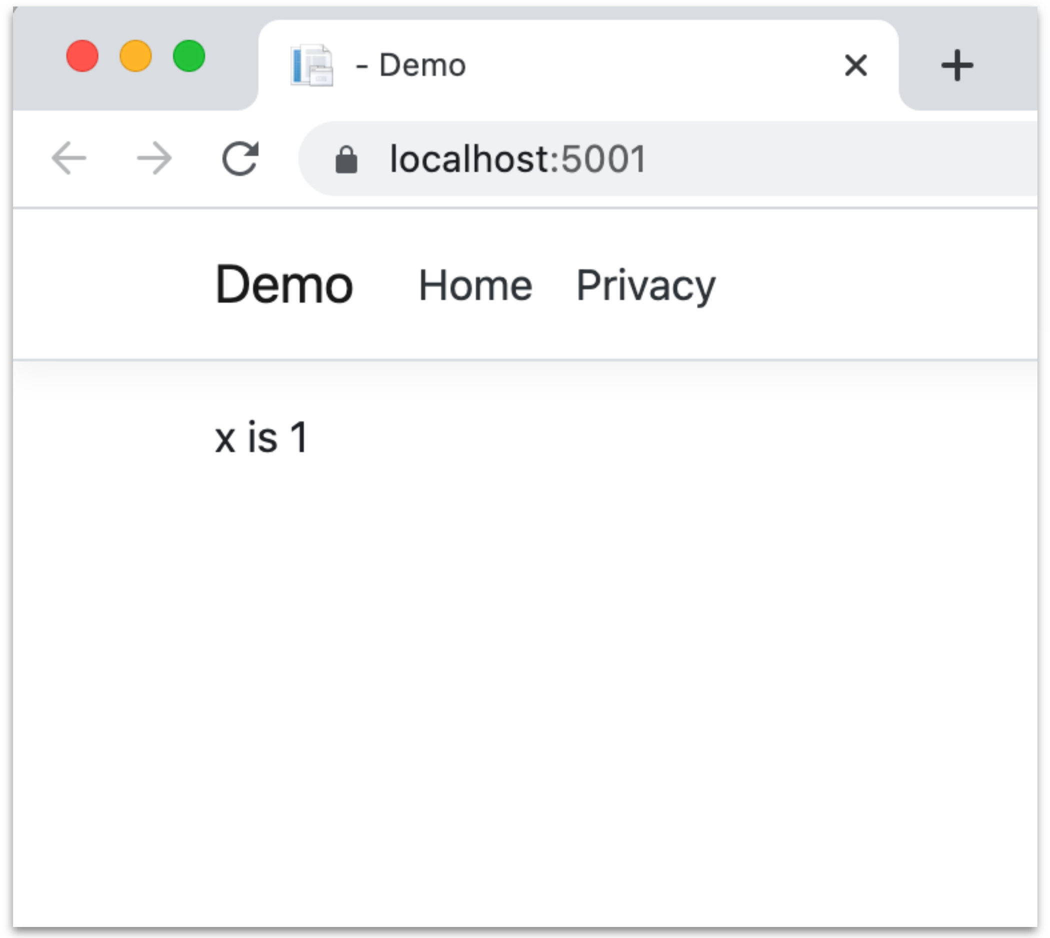


x is 10

Switch statements

- In Razor, a @switch begins a switch statement
- Its switch logics follow the rules defined in the C# language

```
@{
    int x = 1;
}

@switch(x)
{
    case 0:
        @:x is 0
        break;

    case 1: // fall through

    default:
        @:x is @x
        break;
}
```



x is 1

Razor Comments

- Razor uses the syntax @* ... *@ to denote comments
- Razor comments stay on the server (unlike HTML comments, which were sent over to clients)

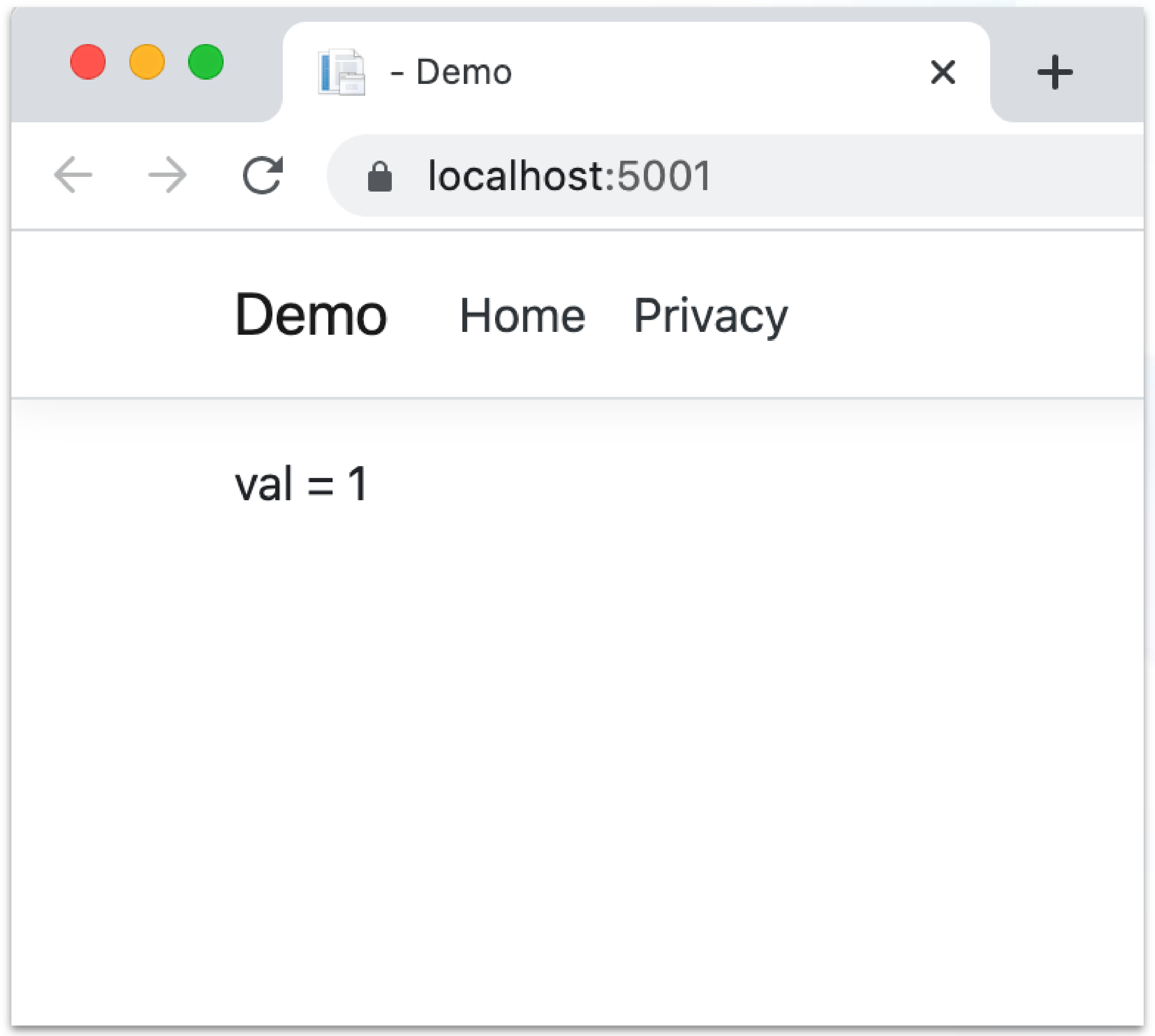
```
@{
    @* this is my comment *@
    var val = 1;
}
```

Using .NET libraries in Razor

- Our Views can leverage on .NET libraries via the directive **@using**
- For example, the Diagnostics namespaces in Razor can be declared in our View to use Debug.WriteLine() for debugging

```
@using System.Diagnostics

@{
    var val = 1;

    @:val = @val

    Debug.WriteLine("Value of 'val' = " + val);
}
```

```
Output

Value of 'val' = 1
```

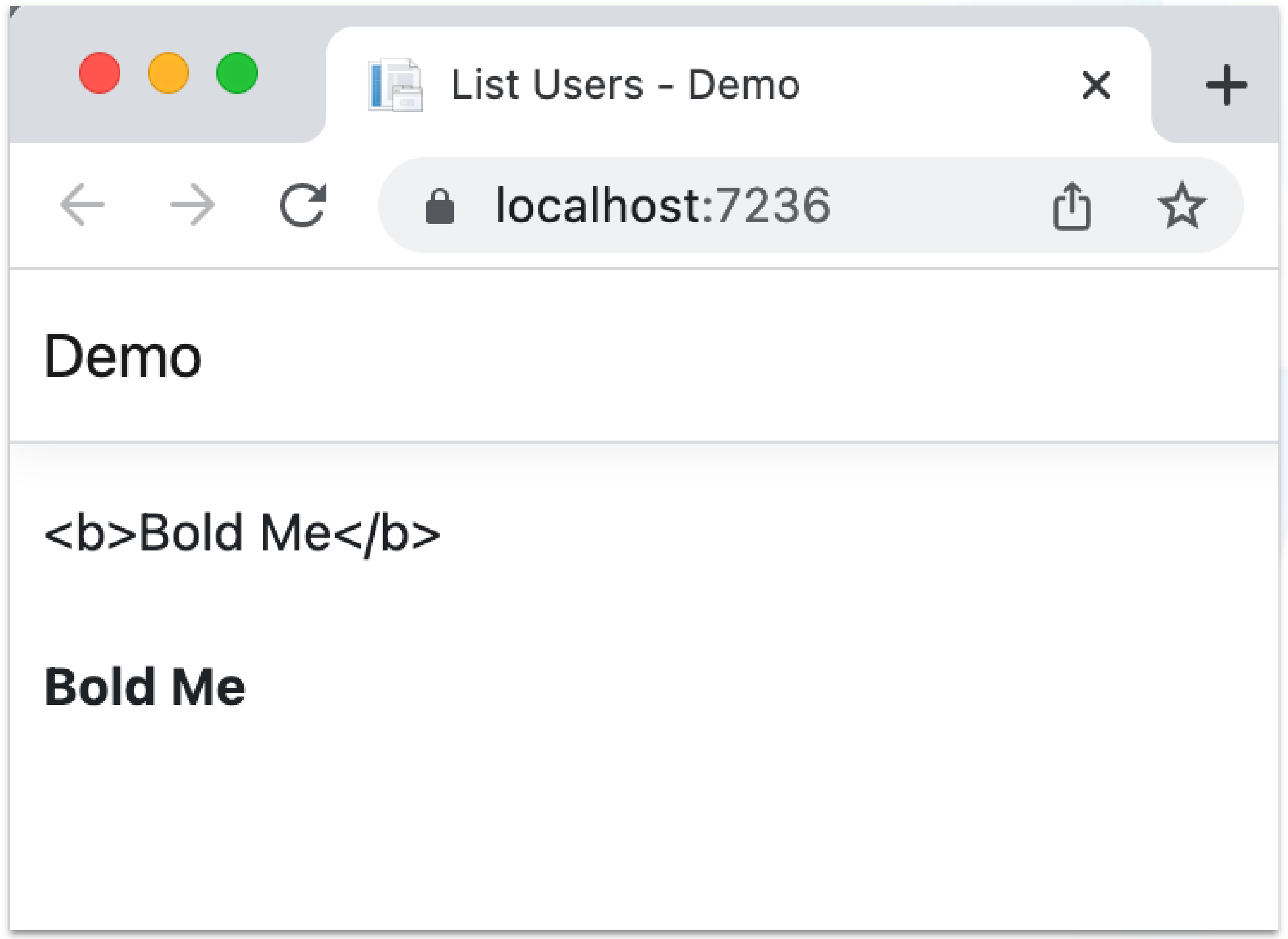


val = 1

Html.Raw

- Wraps HTML markups such that they are interpreted as HTML content (instead of plain text)

```
@{
    string s = "<b>Bold Me</b>";
    @s

    @Html.Raw("<b>Bold Me</b>")
}
```



List Users - Demo

localhost:7236

Demo

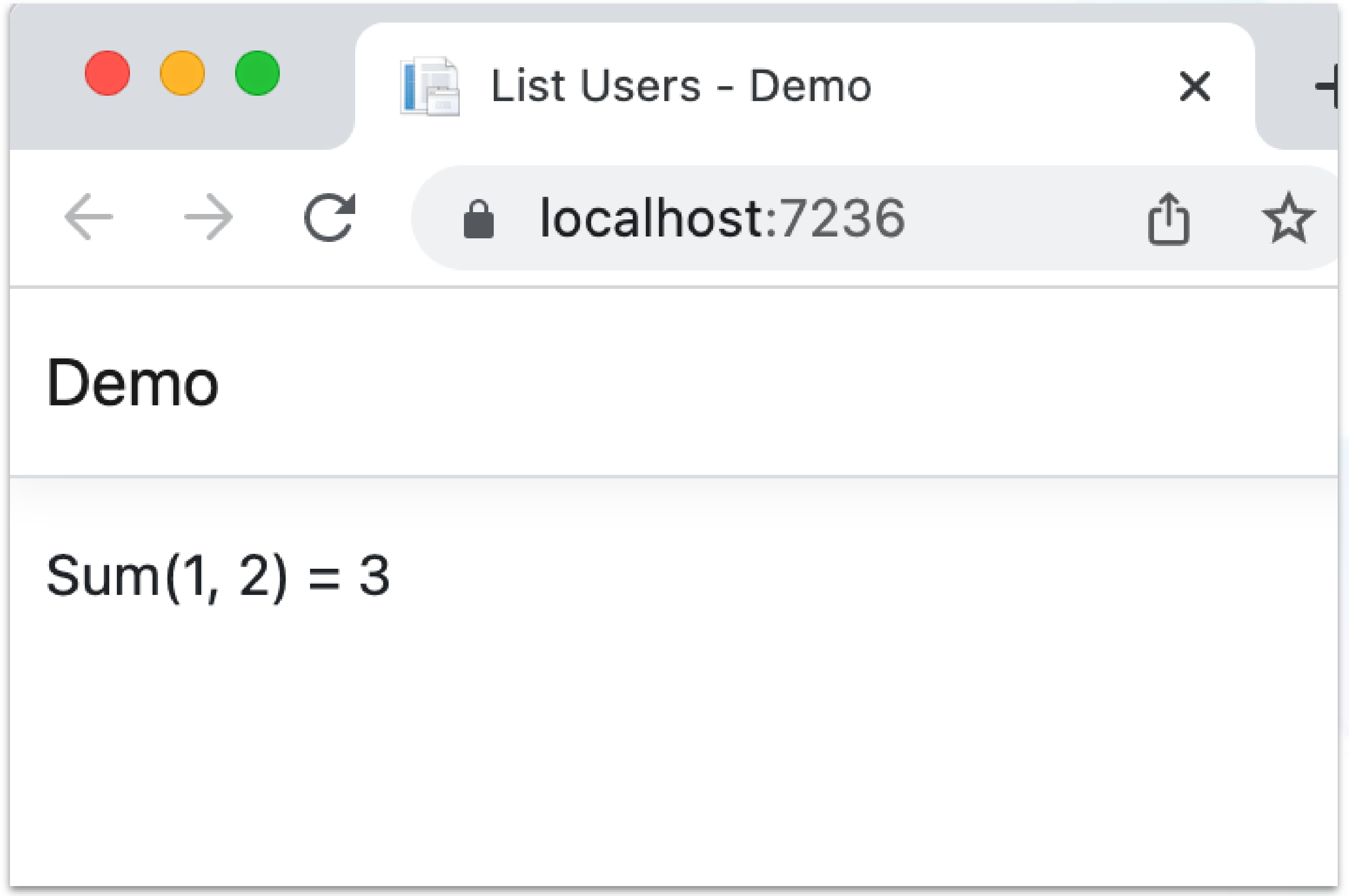
<b>Bold Me</b>

Bold Me

Functions

- In Razor, functions can be defined within @functions{...}

```
@{
    int a = 1;
    int b = 2;

    long sum = add(a, b);
    @:Sum(@a, @b) = @sum
}

@functions {
    public long add(int a, int b) {
        return a + b;
    }

    public int minus(int a, int b) {
        return a - b;
    }
}
```

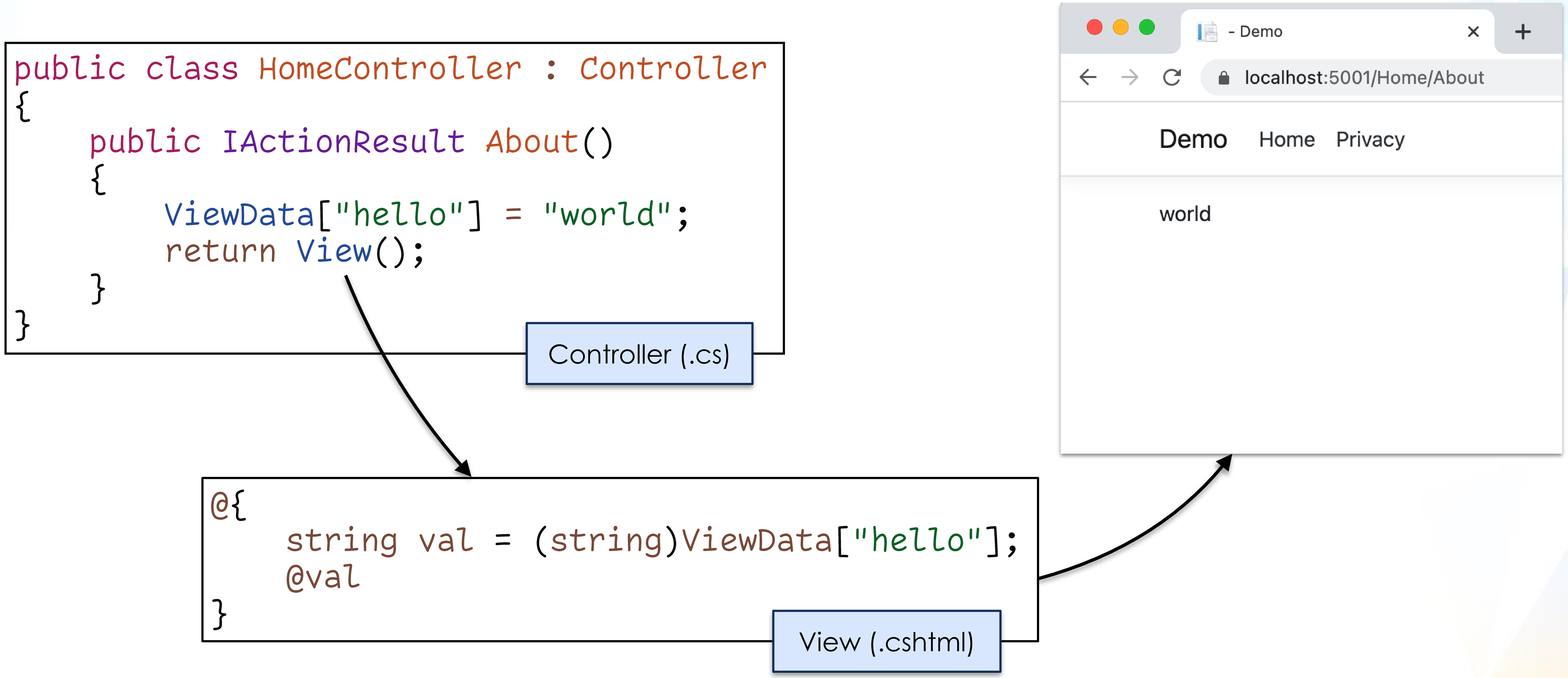


```
Sum(1, 2) = 3
```

PASSING DATA FROM CONTROLLER TO VIEW

Passing Data

- Use ViewData to pass data from a Controller to a View
- ViewData is basically a dictionary (key/value pairs)



```
public class HomeController : Controller
{
    public IActionResult About()
    {
        ViewData["hello"] = "world";
        return View();
    }
}
```

Controller (.cs)

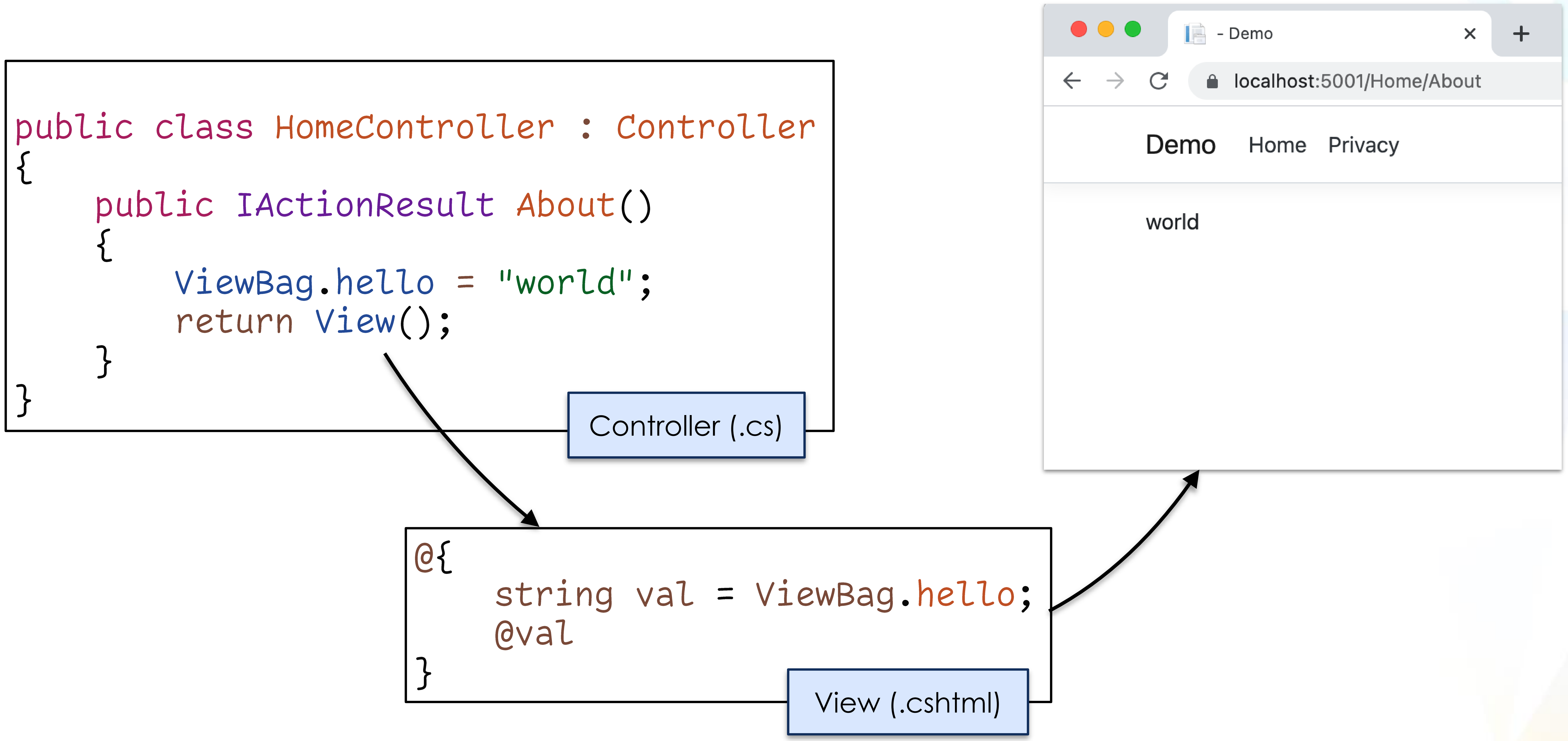
```
@{
    string val = (string)ViewData["hello"];
    @val
}
```

View (.cshtml)

world

Passing Data

- ViewBag is similar to ViewData but allows us to use the dot notation (e.g. ViewBag.data = 1)



```
public class HomeController : Controller
{
    public IActionResult About()
    {
        ViewBag.hello = "world";
        return View();
    }
}
```

Controller (.cs)

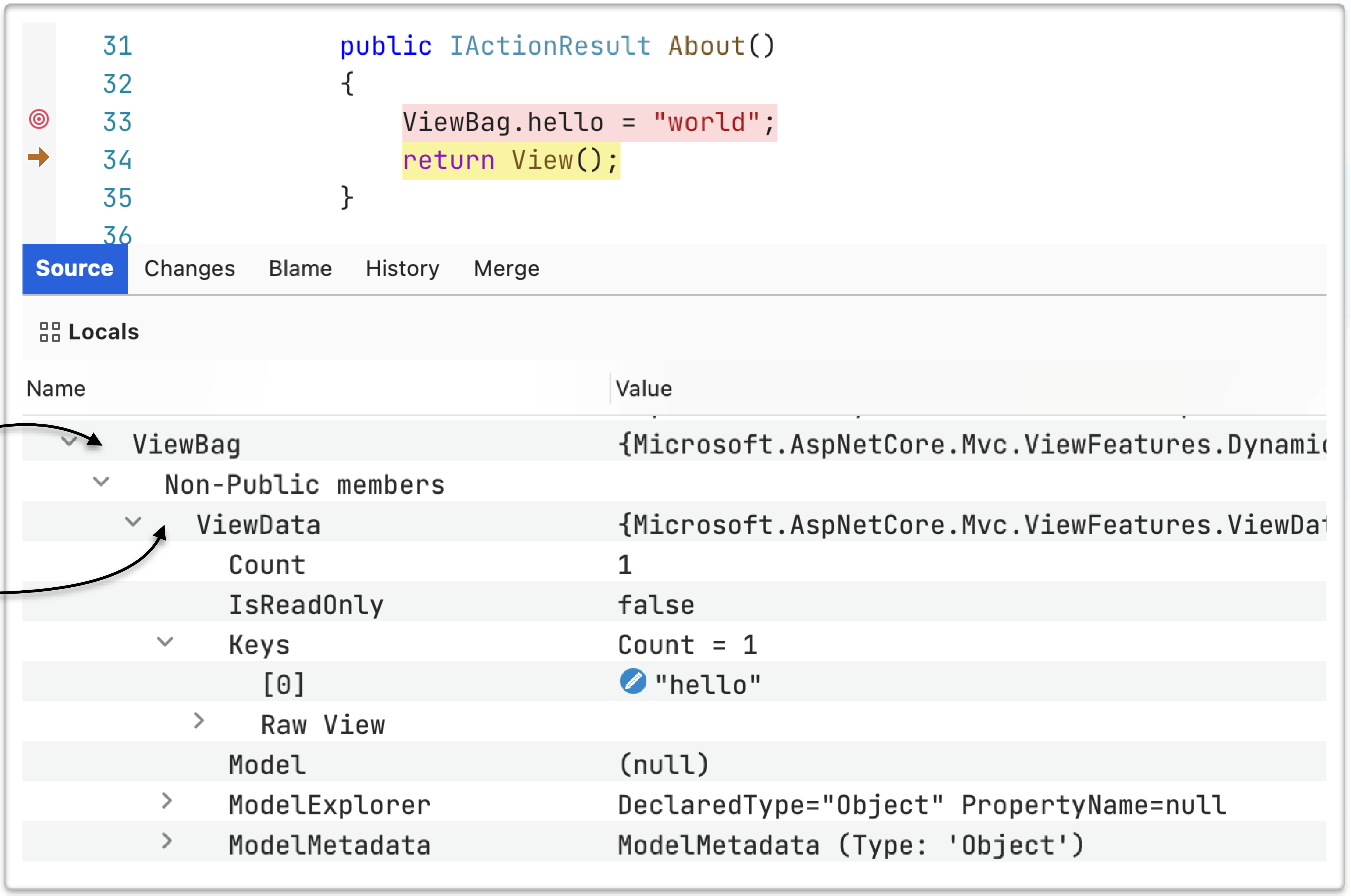
```
@{
    string val = ViewBag.hello;
    @val
}
```

View (.cshtml)

world

ViewData vs ViewBag

- Implementation-wise, ViewBag is a wrapper over ViewData



```
public IActionResult About()
{
    ViewBag.hello = "world";
    return View();
}
```

```
Name                              Value
ViewBag                           {Microsoft.AspNetCore.Mvc.ViewFeatures.Dynamic
  Non-Public members
    ViewData                      {Microsoft.AspNetCore.Mvc.ViewFeatures.ViewDat
      Count                       1
      IsReadOnly                  false
      Keys                        Count = 1
        [0]                       "hello"
        Raw View
      Model                       (null)
      ModelExplorer               DeclaredType="Object" PropertyName=null
      ModelMetadata               ModelMetadata (Type: 'Object')
```

ViewData wrapped within a ViewBag

RAZOR SCENARIO

Scenario: List Staff Info

- Consider the scenario where we were given a list of Person objects
- Each Person object contains the particulars of an actual person in our system
- We want to output each Person object as a row in a HTML table
- The top row of the HTML table should display the properties of the Person class (e.g. Name, Gender)
- Subsequent rows contain the actual content of Person objects
- Alternate rows, in the HTML table, should have different colors

Creating our Data

```
public class HomeController : Controller
{
    public IActionResult Index() {
        List<Person> persons = new List<Person>();

        persons.Add(new Person {
            Name = "Jerry",
            JobTitle = "Engineer",
            Gender = "M"
        });

        persons.Add(new Person {
            Name = "Hogan",
            JobTitle = "Data Scientist",
            Gender = "M"
        });

        ...

        ViewBag.persons = persons;
        return View();
    }
}
```

HomeController.cs

Our **Model**

```
public class Person
{
    public string? Name { get; set; }
    public string? JobTitle { get; set; }
    public string? Gender { get; set; }
}
```

Person.cs

Creating our CSS

- CSS stands for Cascading Style Sheet, and is used to describe the presentation of HTML documents, including colours, layout, fonts, and other design elements
- CSS allows the separation of presentation of a web page from its content, making it easier to manage and update the design of a website

```css
td, th {
    border: 1px solid #0060ff;
    text-align: center;
    padding: 8px;
}

.odd_row {
    background-color: #f5fcff;
}

.even_row {
    background-color: #fff;
}
```

wwwroot/css/site.css

Creating our View

- Write Razor code to construct our View

Views/Home/Index.cshtml

Unpack data received from controller

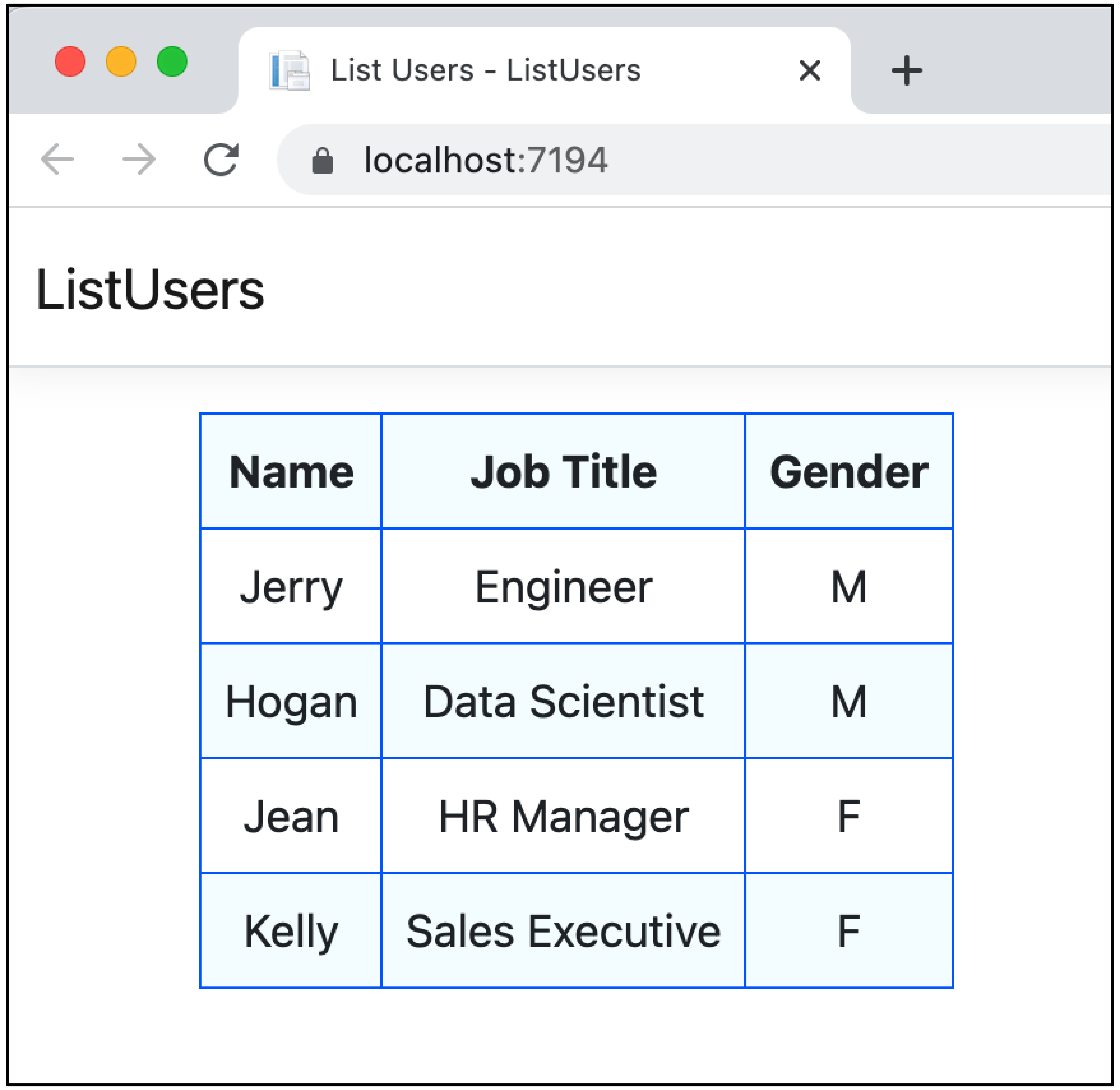
Creating table- headers

Creating table- rows

```
@{
    var persons = ViewBag.persons;
}

<table>
    <tr class="odd_row">
        <th>Name</th>
        <th>Job Title</th>
        <th>Gender</th>
    </tr>

    @for (int i = 0; i < persons.Count; i++)
    {
        string row_color = (i % 2 == 0) ? "class=even_row" : "class=odd_row";

        var person = persons[i];

        <tr @row_color>
            <td>@person.Name</td>
            <td>@person.JobTitle</td>
            <td>@person.Gender</td>
        </tr>
    }
</table>
```

The rows in light blue are encoded with the CSS class “odd_row”

Resultant HTML Page


List Users - ListUsers

localhost:7194

ListUsers

Name	Job Title	Gender
Jerry	Engineer	M
Hogan	Data Scientist	M
Jean	HR Manager	F
Kelly	Sales Executive	F

THE END