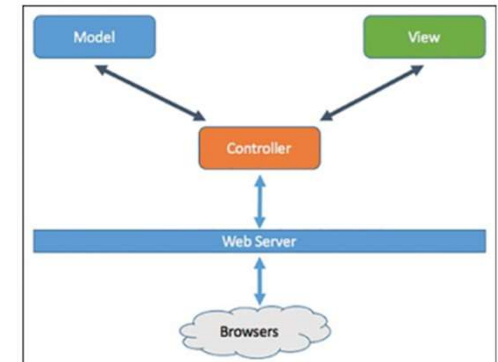


Controller & Action

Controller

Controller

Controller là trong mô hình MVC, đóng vai trò là đường dẫn giữa mô hình dữ liệu (model) và chế độ xem (view). Controller xác định các hành động logic nghiệp vụ xử lý dữ liệu trên model và cung cấp dữ liệu cho view để hiển thị cho người dùng.



Controllers là các lớp C# có phương thức public (được gọi là action hoặc phương thức action) có nhiệm vụ xử lý yêu cầu HTTP và phản hồi lại cho client.

Controller có 3 trách nhiệm chính:

- ❖ Nhận request từ user
- ❖ Xây dựng model : Controller action method thực thi logic của ứng dụng và xây dựng nên model.
- ❖ Gửi trả response : trả về kết quả trong HTML, File, JSON, XML hoặc bất cứ định dạng nào về cho user.

Controller

Controller là thành phần đầu tiên nhận request từ người dùng. Khi người dùng truy cập URL qua trình duyệt, ASP.NET Core routing sẽ map request đó vào controller cụ thể.

Ví dụ: Request URL như sau: `http://localhost/Customer/List`

Trường hợp này, Controller có tên là **CustomerController** được gọi. Sau đó nó sẽ gọi đến Action method tên **List** rồi tạo ra response trả về cho user.

Lớp Controller trong MVC được kế thừa từ **Microsoft.AspNetCore.Mvc**

using Microsoft.AspNetCore.Mvc;

Controller


Đối với một ứng dụng MVC truyền thống action trả lại HTML cho trình duyệt, các phương thức hành động thường sẽ trả về một `ViewResult` mà `MvcMiddleware` sẽ sử dụng để tạo phản hồi HTML hoặc `RedirectResult`, cho biết người dùng sẽ được chuyển hướng đến một trang khác trong ứng dụng.

Controller

Trong lớp Controller, một phương thức hành động (action) thường trả về một đối tượng kiểu **interface IActionResult** như **ContentResult** hoặc **ViewResult**

```
using Microsoft.AspNetCore.Mvc;
namespace CoreWebApplication.Controllers
{
    0 references
    public class HomeController : Controller
    {
        0 references
        public IActionResult Index()
        {
            //return View();
            return Content("Home controller, Index action");
        }

        0 references
        public IActionResult About()
        {
            return Content("Home controller, About action");
        }
    }
}
```

← → ↻  <https://localhost:44393/Home/Index>

Home controller, Index action

← → ↻  <https://localhost:44393/Home/About>

Home controller, About action

```

public class HomeController : Controller
{
    0 references
    public IActionResult Index()
    {
        return View();
    }

    0 references
    public string Hello()
    {
        return "Chào bạn đến với Lập trình ASP.NET Core MVC tại Trường iSpace";
    }

    0 references
    public IActionResult About()
    {
        return Content("Home controller, About action");
    }

    0 references
    public IActionResult Profile()
    {
        ViewData["Message"] = "Your profile page.";
        return View();
    }

    0 references
    public IActionResult Contact()
    {
        ViewData["Message"] = "Your contact page.";
        return View();
    }
}

```

HomeController có 5 action:

1. Index()
2. Hello()
3. About()
4. Profile()
5. Contact()

Được request qua các URL:

- <http://localhost:44393>
- <http://localhost:44393/Home>
- <http://localhost:44393/Home/Index>
- <http://localhost:44393/Home/Hello>
- <http://localhost:44393/Home/About>
- <http://localhost:44393/Home/Profile>
- <http://localhost:44393/Home/Contact>

Controller - Demo

Tạo ứng dụng ASP.NET Core Web App (Model-View-Controller)

```
namespace SampleWebCore.Controllers
{
    3 references
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        0 references
        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        0 references
        public IActionResult Index()
        {
            return View();
        }

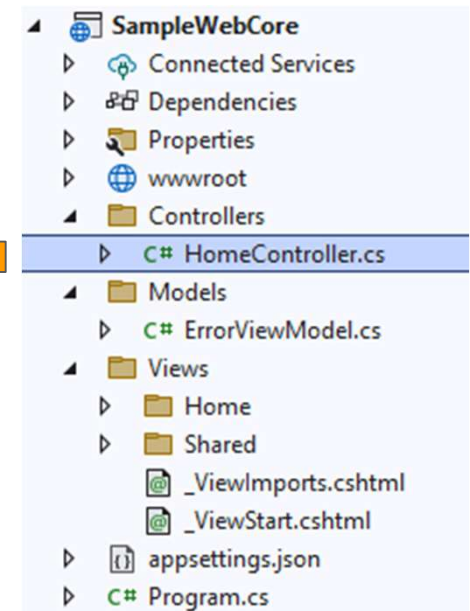
        0 references
        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        0 references
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}
```

Home Controller dẫn xuất từ Controller class

Khởi tạo dịch vụ ILogger được đưa vào thông qua Dependency Injection

Có 3 action method: Index, Privacy & Error.



Sử dụng ViewData

Mở file view Index.cshtml của Index action trong folder View ➤ Home

```
@{  
    ViewData["Title"] = "Home Page";  
}  
  
<div class="text-center">  
    <h1 class="display-4">Welcome</h1>  
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">  
        building Web apps with ASP.NET Core</a>.</p>  
</div>
```

ViewData là đối tượng của ViewDataDictionary dùng truyền dữ liệu từ Controller đến View và từ View đến LayoutView

File View có 2 loại code:

1. Biểu thức Razor đặt bên trong ký hiệu "@", viết với code C#.
2. HTML dùng cho thiết kế giao diện.

Có thể dùng viewdata để truyền nhiều kiểu dữ liệu như strings, int, float hoặc các kiểu object

Sử dụng ViewData

File layout view _Layout.cshtml (trong folder Views ► Shared)

```
_Layout.cshtml
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6      <title>@ViewData["Title"] SampleWebCore</title>
7      <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
8      <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
9      <link rel="stylesheet" href="~/SampleWebCore.styles.css" asp-append-version="true" />
10 </head>
```

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">
        building Web apps with ASP.NET Core</a>.</p>
</div>
```

Index.cshtml của Index action
trong folder View ► Home

Sử dụng ViewData

Controller

```
public IActionResult DetailAddress()
{
    ViewData["Name"] = "Hanna";
    ViewData["Address"] = new Address()
    {
        HouseNo = "455 Lý Thường Kiệt",
        City = "Hà Nội",
    };

    return View();
}
```

public class Address

Address.cs

```
{
    1 reference
    public string? HouseNo { get; set; }
    1 reference
    public string? City { get; set; }
}
```

View file

```
@{
    // casting address to Address.cs class object
    var address = ViewData["Address"] as Address;
}

<p>Hello: @ViewData["Name"]</p>
<p>With Address: @address.HouseNo, @address.City</p>
```

Truyền dữ liệu từ Views đến Controllers

Có 3 cách truyền dữ liệu từ View đến Controllers:

1. FormData objects
2. Query strings
3. Model binding

1 . Controllers Request – Request.Form

Thuộc tính `Request` của lớp `ControllerBase` trả về kiểu đối tượng `HttpRequest`

Đối tượng `HttpRequest` chứa thuộc tính `Form` trả về dictionary của đối tượng `FormData`. Có thể dùng thuộc tính `Form` để đọc dữ liệu truyền về từ View

```

1  @{
2      ViewData["Title"] = "Home Page";
3  }
4
5  <form asp-action="ReceivedDataByRequest">
6      <div class="form-group">
7          <label>Name:</label>
8          <input class="form-control" name="name" />
9      </div>
10     <div class="form-group">
11         <label>Sex:</label>
12         <select class="form-control" name="sex">
13             <option value="M">Male</option>
14             <option value="F">Female</option>
15         </select>
16     </div>
17     <div class="m-1">
18         <button class="btn btn-primary" type="submit">Submit</button>
19     </div>
20 </form>

```

tag helper

Index

localhost:44339

Name:

Sex:

Male

Submit

- Model là **biến mặc định** trong Razor View
- Cú pháp khai báo trong view

@model <kiểu dữ liệu>

=> Dùng **@Model** để truy cập dữ liệu

```
public IActionResult SendString()
{
    string message = "Truyền chuỗi trong ASP.NET Core!";
    return View("SendString", message); // Truyền chuỗi sang View
}
```



```
@model string

<h2>Truyền chuỗi</h2>
<p>@Model</p>
```

```
public IActionResult ShowNumber()
{
    int number = 42;
    return View(number);
}
```



```
@model int <!-- Khai báo kiểu dữ liệu là int -->

<h2>Giá trị số:</h2>
<p>Số được truyền từ Controller: @Model</p>
<p>Số gấp đôi: @(Model * 2)</p>
```

```
public IActionResult StudentInfo()
{
    var student = new Student { Id = 1, Name = "Phạm Bửu Tài" };
    return View(student); // Truyền đối tượng sang View
}
```



```
@model Student

<h2>Thông tin sinh viên:</h2>
<p>Mã số: @Model.Id</p>
<p>Họ tên: @Model.Name</p>
```



```

5      public class HomeController : Controller
6      {
7          public IActionResult Index()
8          {
9              return View();
10         }
11
12         public IActionResult ReceivedDataByRequest()
13         {
14             string name = Request.Form["name"];
15             string sex = Request.Form["sex"];
16             return View("ReceivedDataByRequest", $"{name} sex is {sex}");
17         }
18     }
19 }

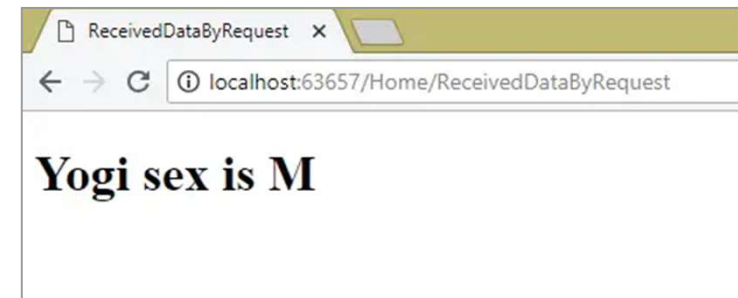
```

Khai báo View

```

@model string
<h1>@Model</h1>

```



2. Nhận dữ liệu từ Parameters của Action

```
13 <form asp-action="ReceivedDataByParameter">
14   <div class="form-group">
15     <label>Name:</label>
16     <input class="form-control" name="name" />
17   </div>
18   <div class="form-group">
19     <label>Sex:</label>
20     <select class="form-control" name="sex">
21       <option value="M">Male</option>
22       <option value="F">Female</option>
23     </select>
24   </div>
25   <div class="m-1">
26     <button class="btn btn-primary" type="submit">Submit</button>
27   </div>
28 </form>
```

Lưu ý: Tên tham số phải giống tên của control

0 references

```
public IActionResult ReceivedDataByParameter(string name, string sex)
{
    return View("ReceivedDataByParameter", $"{name} sex is {sex}");
}
```

2. Nhận dữ liệu từ Parameters của Action

Khai báo View

```
@model string  
<h1>@Model</h1>
```



3. Truyền dữ liệu từ Query String

```
<a  
href="/Home/ReceivedDataByParameter?  
name=sparrow&sex=m" class="link-  
primary">Primary link  
</a>
```

4. Truyền dữ liệu từ View đến Controller với Model Binding

Trong Model Binding, cần phải tạo 1 lớp Model trong folder Models, sau đó ánh xạ các thuộc tính của lớp với form.

Để controller nhận được giá trị, phải thêm tham số (parameter) thuộc kiểu model vào phương thức action

Ví dụ: Trong folder Models, tạo file class [Person.cs](#)

```
public class Person
{
    public string name { get; set; }
    public string sex { get; set; }
}
```

4. Truyền dữ liệu từ View đến Controller với Model Binding

Tạo action `ReceivedDataByModelBinding` trong Home Controller

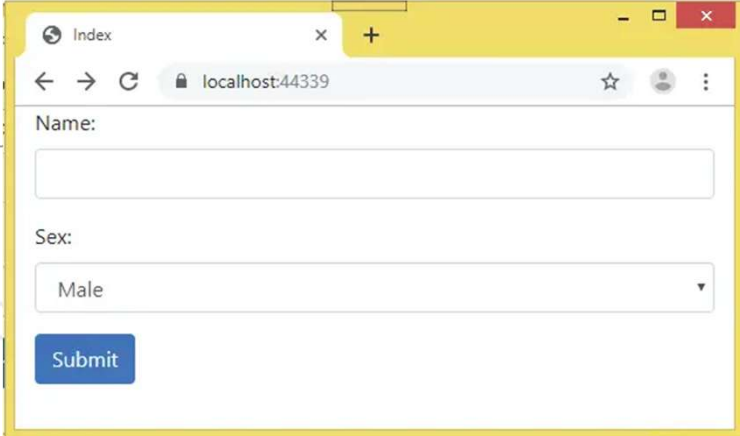
```
using Microsoft.AspNetCore.Mvc;
using SampleWebCore.Models;

namespace SampleWebCore.Controllers
{
    public class HomeController : Controller
    {
        //...

        public IActionResult ReceivedDataByModelBinding(Person person)
        {
            return View("ReceivedDataByModelBinding", person);
        }
    }
}
```

Tạo view Index.cshtml cho Index action của controller Home

```
12 @model Person
13 @{
14     ViewData["Title"] = "Home Page";
15 }
16
17 <form asp-action="ReceivedDataByModelBinding">
18     <div class="form-group">
19         <label>Name:</label>
20         <input class="form-control" asp-for="name" />
21     </div>
22     <div class="form-group">
23         <label>Sex:</label>
24         <select class="form-control" asp-for="sex">
25             <option value="M">Male</option>
26             <option value="F">Female</option>
27         </select>
28     </div>
29     <div class="m-1">
30         <button class="btn btn-primary" type="submit">Submit</button>
31     </div>
32 </form>
```



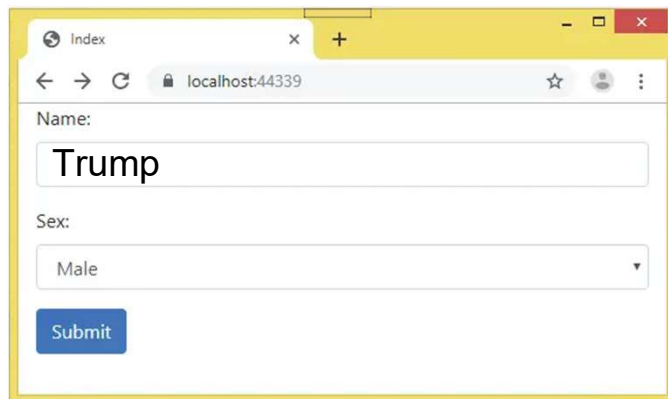
Thẻ `asp-for="class_property"` là tag helper dùng để liên kết (bind) các control với thuộc tính tương ứng của model class.

4. Truyền dữ liệu từ View đến Controller với Model Binding

Tạo view có tên `ReceivedDataByModelBinding` trong folder `Views ► Home`

```
@model Person
```

```
<h1>@Model.name sex is @Model.sex</h1>
```



Index

localhost:44339

Name:

Trump

Sex:

Male

Submit



Truyền dữ liệu từ Controller sang View

Để truyền dữ liệu từ Controller sang View chúng ta nên tạo ViewModel với các thuộc tính cần thiết

ViewModel được gán từ Controller sang View bằng ViewBag (sử dụng thuộc tính động) hoặc ViewData.

Ví dụ: Tạo model có tên **Customer.cs**

```
namespace SampleWebCore.Models
{
    1 reference
    public class Customer
    {
        1 reference
        public int CustomerID { get; set; }
        1 reference
        public string Name { get; set; }
        1 reference
        public string Address { get; set; }

        0 references
        public Customer()
        {
            CustomerID = 1;
            Name = "iSpace School";
            Address = "";
        }
    }
}
```

Sử dụng ViewBag để truyền ViewModel về View

Khai báo trong action Index

```
public IActionResult Index()
{
    ViewBag.Customer = new Customer();
    return View();
}
```

Khai báo View, tham chiếu đến đối tượng customer sử dụng **ViewBag.Customer**. Vì sử dụng ViewBag nên không cần phải ép kiểu sang kiểu tương ứng.

```
@{
    Customer customer = ViewBag.Customer;

    <p>Id :@customer.CustomerID </p>
    <p>Name :@customer.Name </p>
    <p>Address :@customer.Address </p>
}
```

Sử dụng thuộc tính Model để trả về ViewModel

ViewData trả về một **ViewDataDictionary** trong đó có một thuộc tính đặc biệt là **Model**, chúng ta có thể gán Customer ViewModel về view sử dụng thuộc tính **Model**:

```
public IActionResult DetailCustomer()
{
    ViewData.Model = new Customer();
    return View();
}
```

Truy cập Model trong View bằng cách sử dụng **ViewData.Model** hoặc **Model** (trả về ViewData.Model):

```
@{
    Customer customer = Model;
    //Hoặc khai báo
    //Customer customer =ViewData.Model;

    <p>Id :@customer.CustomerID </p>
    <p>Name :@customer.Name </p>
    <p>Address :@customer.Address </p>
}
```

Strongly Typed View

View mà gắn vào một kiểu cụ thể của ViewModel thay vì một thuộc tính động gọi là strongly typed view. Trong ví dụ trên, chúng ta đang gắn Customer ViewModel vào View sử dụng **ViewBag.Customer** hoặc **ViewData.Model**. Trình biên dịch không biết gì về kiểu của model. Trong strongly typed view, chúng ta để cho View biết được kiểu của ViewModel được gắn cho nó.

Khai báo @model

Strongly typed view được tạo sử dụng khai báo **@model**. Khai báo **@model** được đặt trên đầu của file view chỉ ra kiểu của ViewModel được gắn.

@model Customer

Sau đó tham chiếu trực tiếp đến model trong View:

```
@{  
    <p>Id :@Model.CustomerID </p>  
    <p>Name :@Model.Name </p>  
    <p>Address :@Model.Address </p>  
}
```

Model và model

- ❖ Khai báo **model** là khai báo được dùng để **khai báo kiểu của ViewModel**.
- ❖ **Model** là một biến sử dụng để truy cập vào **ViewModel**. Kiểu của **Model** được khai báo tùy thuộc vào khai báo **@model** ở trên.
- ❖ Thực tế, tất cả dữ liệu được gán vào View qua ViewBag. Khi dùng cách khai báo **model**, Razor engine tạo ra một **thuộc tính tên Model**. Model sau đó trả về kiểu được khai báo.

```
@model Customer
```

Sẽ được hiểu là
Customer Model;

Khuyến nghị cách gán ViewModel sang View

Để gán dữ liệu từ ViewModel sang View nên sử dụng phương thức View. Phương thức View nhận model như một tham số và tự động gán vào ViewData.Model.

```
public IActionResult Detail()
{
    Customer customer = new Customer();
    return View(customer);
}
```

Action

Action Method

- ❖ Là các phương thức public được truy xuất trực tiếp qua controller
- ❖ Mỗi URL tương ứng với một phương thức
- ❖ Khi một truy vấn đến server, Mvc Middleware căn cứ vào URL để xác định phương thức cần thực thi
- ❖ Phương thức public trong controller có thể được gọi bởi bất cứ ai biết được URL của nó, cần cẩn trọng khi đặt phương thức public
- ❖ Action không trực tiếp sinh ra view, action chỉ lựa chọn loại view phù hợp và chuẩn bị dữ liệu cho view

Ví dụ:

<http://localhost:67876/Product/GetAll> sẽ gọi action method tên **GetAll** trong **ProductController**

Action Method

Lưu ý khi tạo một Action method:

- **Action method** phải là một **phương thức public**
- Action method không thể là static method hoặc một extension method.
- Constructor, getter, setter không được sử dụng.
- Các phương thức được kế thừa không được sử dụng như là một action method.
- Action method không được chứa từ khóa **ref** hoặc **out** trên tham số.
- Action method không được chứa thuộc tính [NonAction]
- Action method không thể được nạp chồng (overloaded)

Action Method - Tham số cho action

Một action có thể có hoặc không có tham số, tham số cho các phương thức action có thể là loại đơn giản (string, int) hoặc có thể là loại phức tạp (model, object, form)

```
public ActionResult Detail(int id, string Name)
{
    Category cate = new Category
    {
        CategoryID = id,
        CategoryName = Name,
    };
    return View(cate);
}
```

❖ Theo định tuyến (routing) 3 liên kết bên dưới có tác dụng như nhau, đều gọi đến action Detail() và truyền 2 tham số Id, Name cho action.

- </Category/Detail?Id=2&Name=Mango>>Liệt kê sản phẩm
- </Category/Detail/2?Name=Mango>>Liệt kê sản phẩm
- </Category/Detail/2/Mango>>Liệt kê sản phẩm

Action Method - Tham số cho action

Ví dụ, gọi action và truyền tham số với các trường form

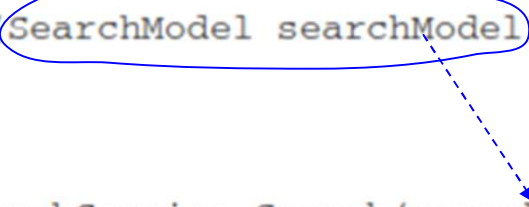
```
<form asp-action="Create" asp-controller="Category">
  <div asp-validation-summary="ModelOnly" class="text-danger"></div>
  <div class="form-group">
    <label class="control-label">Tên loại</label>
    <input asp-for="CategoryName" class="form-control" />
    <span asp-validation-for="CategoryName" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label class="control-label">Mô tả</label>
    <input asp-for="CategoryDescription" class="form-control" />
    <span asp-validation-for="CategoryDescription" class="text-danger"></span>
  </div>
  <div class="form-group">
    <input type="submit" value="Tạo mới" class="btn btn-default" />
  </div>
</form>
```

Khi nhấn vào nút “Tạo mới” thì yêu cầu được gửi đến action **Create()** của controller **Category**, đồng thời chuyển các tham số **CategoryName**, **CategoryDescription** cho action này xử lý

Action Method - Tham số cho action

Tham số là kiểu Model

```
public IActionResult Search(SearchModel searchModel)
{
    if (ModelState.IsValid)
    {
        var viewModel = _searchService.Search(searchModel);
        return View(viewModel);
    }
    return Redirect("/")
}
```

A blue oval highlights the parameter `SearchModel searchModel` in the method signature. A dashed blue arrow points from this parameter to the `searchModel` argument in the `_searchService.Search(searchModel)` call within the method body.

IActionResult và ActionResult

IActionResult là một interface nó định nghĩa một khuôn mẫu cho toàn bộ các Action Result của một action method.

ActionResult là một lớp cơ sở trừu tượng cài đặt cho **IActionResult**.

Các Action result như ViewResult, PartialViewResult hay JsonResult...đều kế thừa lớp ActionResult.

Kết quả trả về của action là ActionResult, một kiểu dữ liệu chung chung từ kiểu đơn giản string, int, ... cho đến kiểu phức tạp như JSON, html, file (dùng để download).

Sử dụng ActionResult

ASP.NET Core có nhiều loại IActionResult

1. **ViewResult** — kết quả là một HTML Response
2. **RedirectResult**—Chuyển đến địa chỉ URL chỉ định.
3. **RedirectToRouteResult**—Action result này chuyển khách hàng đến một route cụ thể. Nó nhận tên route, giá trị của route và chuyển chúng ta đến vị trí mà route cung cấp:
4. **FileResult**—Trả về phản hồi ở dạng file
5. **ContentResult**—Trả về dạng chuỗi văn bản, không bao gồm layout, phù hợp cho test và làm việc với ajax
6. **StatusCodeResult**—gửi kết quả và chỉ ra một HTTP Status code
7. **NotFoundResult**—trả về lỗi HTTP 404 cho client.

....

1. ViewResult

Phương thức View tạo một đối tượng ViewResult bằng cách xác định một model sẽ được chuyển đến View. Có 4 phương thức View

- ❖ `View()`
- ❖ `View("tên-tập-tin-view")`
- ❖ `View(model)`
- ❖ `View("tên-tập-tin-view", model)`

1. ViewResult

Phương thức View() tìm kiếm View trong thư mục Views để tìm file **.cshtml** giống với tên của Action và chuyển nó cho Razor View Engine. Bạn có thể gán cho nó model dữ liệu. View sẽ trả về một ViewResult và kết quả là một HTML Response.

```
HomeController.cs
SampleWebCore

0 references
16 public IActionResult Index()
17 {
18     ViewBag.EmployeeID = "1";
19     ViewBag.FirstName = "Paul";
20     ViewBag.LastName = "Andrew";
21     return View();
22 }
```

```
Index.cshtml
11 <div>
12     <dl class="row">
13         <dt class="col-sm-2">
14             @Html.Label("EmployeeID")
15         </dt>
16         <dd class="col-sm-10">
17             @ViewBag.EmployeeID
18         </dd>
19         <dt class="col-sm-2">
20             @Html.Label("FirstName")
21         </dt>
22         <dd class="col-sm-10">
23             @ViewBag.FirstName
24         </dd>
25         <dt class="col-sm-2">
26             @Html.Label("LastName")
27         </dt>
28         <dd class="col-sm-10">
29             @ViewBag.LastName
30         </dd>
31     </dl>
32 </div>
```

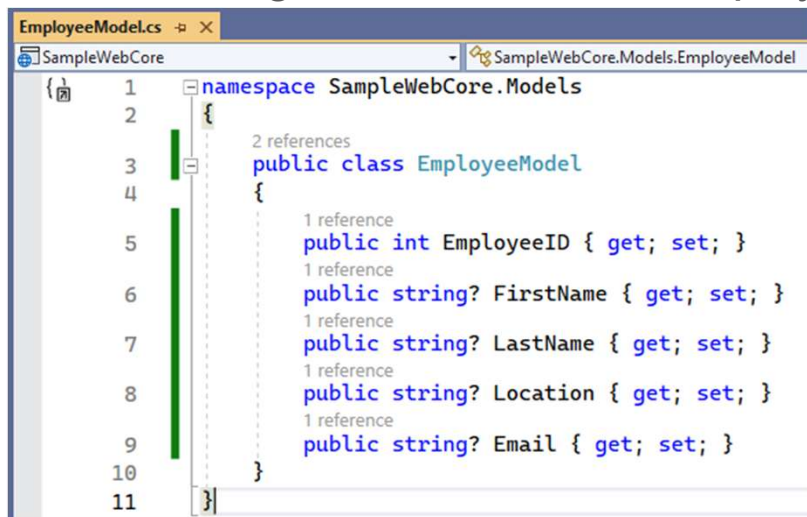
Kết quả thực thi

EmployeeID	1
FirstName	Paul
LastName	Andrew

ViewResult with Model

Có thể tạo các model kiểu mạnh (strongly type) hoặc view model và liên kết với view. Model có thể là loại đơn giản hoặc phức tạp.

Ví dụ: Định nghĩa model có tên EmployeeModel.cs



```
1 namespace SampleWebCore.Models
2 {
3     2 references
4     public class EmployeeModel
5     {
6         1 reference
7         public int EmployeeID { get; set; }
8         1 reference
9         public string? FirstName { get; set; }
10        1 reference
11        public string? LastName { get; set; }
12        1 reference
13        public string? Location { get; set; }
14        1 reference
15        public string? Email { get; set; }
16    }
17 }
```

ViewResult với Model

Khai báo sử dụng Model trong Index action của HomeController



The screenshot shows the Visual Studio IDE with the file `HomeController.cs` open. The breadcrumb navigation at the top indicates the file path is `SampleWebCore > SampleWebCore.Controllers.HomeController`. The code editor displays the `Index` action method, which is a public `IActionResult` method. It creates a new `EmployeeModel` object and initializes its properties: `EmployeeID` (1), `FirstName` ("Paul"), `LastName` ("Andrew"), `Location` ("Ấn Độ"), and `Email` ("paul@gmail.com"). The method then returns the view using `View(employee)`. The code is as follows:

```
15  
16 0 references  
16 public IActionResult Index()  
17 {  
18     EmployeeModel employee = new EmployeeModel()  
19     {  
20         EmployeeID = 1,  
21         FirstName = "Paul",  
22         LastName = "Andrew",  
23         Location = "Ấn Độ",  
24         Email = "paul@gmail.com"  
25     };  
26     return View(employee);  
27 }
```

```

11  @{
12      ViewData["Title"] = "Index";
13  }
14
15  @model SampleWebCore.Models.EmployeeModel
16
17  <div>
18      <dl class="row">
19          <dt class="col-sm-2">
20              @Html.DisplayNameFor(model => model.EmployeeID)
21          </dt>
22          <dd class="col-sm-10">
23              @Html.DisplayFor(model => model.EmployeeID)
24          </dd>
25          <dt class="col-sm-2">
26              @Html.DisplayNameFor(model => model.FirstName)
27          </dt>
28          <dd class="col-sm-10">
29              @Html.DisplayFor(model => model.FirstName)
30          </dd>
31          <dt class="col-sm-2">
32              @Html.DisplayNameFor(model => model.Email)
33          </dt>
34          <dd class="col-sm-10">
35              @Html.DisplayFor(model => model.Email)
36          </dd>
37          <dt class="col-sm-2">
38              @Html.DisplayNameFor(model => model.Location)
39          </dt>
40          <dd class="col-sm-10">
41              @Html.DisplayFor(model => model.Location)
42          </dd>
43      </dl>
44  </div>

```

Kết quả thực thi

https://localhost:7173/Home/Index

Core Home Privacy

EmployeeID	1
FirstName	Paul
Email	paul@gmail.com
Location	Ấn Độ

Gọi file View từ thư mục khác

Cần phải chỉ rõ đường dẫn đến file View muốn kết xuất dữ liệu

```
public IActionResult Index()
{
    EmployeeModel employee = new EmployeeModel()
    {
        EmployeeID = 1,
        FirstName = "Paul",
        LastName = "Andrew",
        Location = "Ấn Độ",
        Email = "paul@gmail.com"
    };
    return View("~/Areas/Employee/Views/Employee/Index.cshtml", employee);
}
```

2. PartialView Result

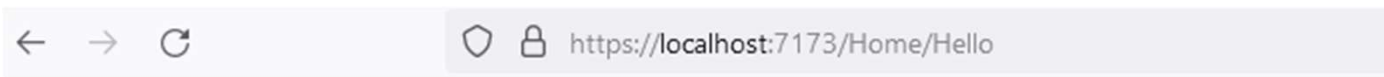
PartialView Result sử dụng model để tạo ra một phần của View. Chúng ta sử dụng `ViewResult` để tạo ra một view hoàn chỉnh còn *PartialView chỉ trả về một phần của View*. Kiểu trả về này hữu ích khi muốn cập nhật một phần của View thông qua AJAX.

```
public IActionResult Index()
{
    EmployeeModel employee = new EmployeeModel()
    {
        EmployeeID = 1,
        FirstName = "Paul",
        LastName = "Andrew",
        Location = "Ấn Độ",
        Email = "paul@gmail.com"
    };
    return PartialView(employee);
}
```

3. ContentResult

ContentResult ghi một nội dung cụ thể trực tiếp vào response như một chuỗi định dạng văn bản thuần.

```
public ContentResult Index()  
{  
    string content = "<div><b><i>Hello from iSpace team," +  
        "this is example of content action method.</ b ></ i ></ div > ";  
  
    return Content(content, "text/html");  
}
```



Hello from iSpace team,this is example of content action method.

Câu 1: Tạo một ứng dụng ASP.NET Core MVC đơn giản mà khi người dùng truy cập, nó sẽ hiển thị "Hello World" trên trình duyệt.

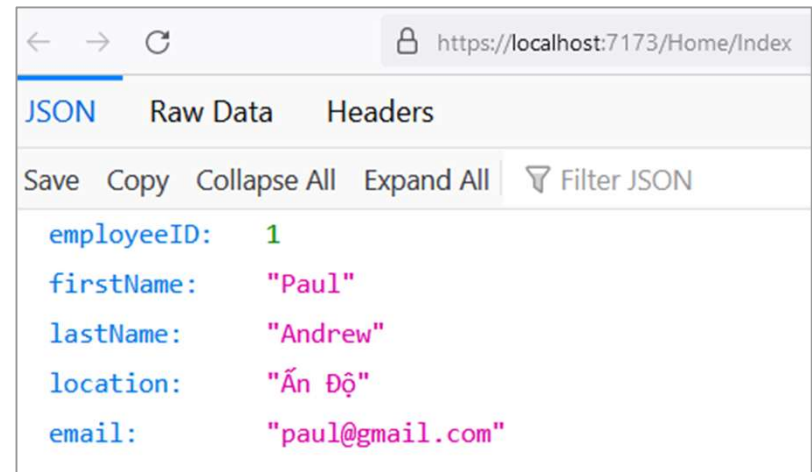
- Tạo một Controller mới có tên là ``HelloWorldController``.
- Trong Controller đó, thêm một Action ``Index`` trả về ``ViewResult`` và hiển thị một View đơn giản với nội dung "Hello World".
- Tạo một View mới có tên là ``Index.cshtml`` trong thư mục ``Views/HelloWorld`` và viết mã HTML để hiển thị "Hello World".

Câu 2: Xây dựng một ứng dụng quản lý sách đơn giản với các chức năng thêm, xem danh sách.

- Tạo một Model ``Book`` với ít nhất hai thuộc tính: ``Title`` và ``Author``.
- Tạo một Controller ``BooksController`` với hai Actions: ``Index`` để hiển thị danh sách sách, và ``Create`` để thêm sách mới.
- Tạo các View tương ứng cho Actions: ``Index.cshtml`` để hiển thị danh sách và ``Create.cshtml`` với một form để nhập thông tin sách mới.

3. ContentResult - JsonResult

```
public IActionResult Index()
{
    EmployeeModel employee = new EmployeeModel()
    {
        EmployeeID = 1,
        FirstName = "Paul",
        LastName = "Andrew",
        Location = "Ấn Độ",
        Email = "paul@gmail.com"
    };
    return Json(employee);
}
```



The screenshot shows a web browser window with the address bar displaying `https://localhost:7173/Home/Index`. The browser's developer tools are open, showing the JSON response of the `Index` action. The JSON is displayed in a table format with the following data:

JSON	
employeeID:	1
firstName:	"Paul"
lastName:	"Andrew"
location:	"Ấn Độ"
email:	"paul@gmail.com"

3. ContentResult - EmptyResult

EmptyResult giống như tên của nó không chứa cái gì cả. Sử dụng nó khi bạn muốn thực thi một số logic trong controller nhưng không muốn trả về gì cho client.

```
public EmptyResult EmptyData()
{
    //code to execute some logic

    return new EmptyResult();
}
```

4. Redirect Results

```
return Redirect("/Product/Index");
```

```
return RedirectPermanent("/Product/Index");
```

```
return RedirectPermanentPreserveMethod("/Product/Index");
```

```
return RedirectPreserveMethod("/Product/Index");
```

5. File Results

```
public FileResult FileResultTest()  
{  
    return File("~/downloads/pdf-sample.pdf", "application/pdf");  
}
```

6. Trả về lỗi và HTTP Code

Loại Action result này được dùng trong Web API Controller. Kết quả sẽ được gửi về kèm HTTP Status Code. Một trong số chúng thì có thể gửi một đối tượng vào response.

```
public StatusCodeResult StatusCodeResultTest()  
{  
    return StatusCode(200);  
}
```

Status Code Result có nhiều loại phương thức trả về: `StatusCodeResult`, `ObjectResult`, `OkResult`, `OkObjectResult`, `CreatedResult`, `CreatedAtActionResult`, `CreatedAtRouteResult`, `BadRequestResult`, `BadRequestObjectResult`, `NotFoundResult`, `NotFoundObjectResult`, `UnsupportedMediaTypeResult`, `NoContentResult`

Action Selectors & Action Verbs

Action selector

Action selector là một thuộc tính có thể được áp dụng cho controller action. Các thuộc tính này giúp Routing Engine chọn đúng action method để xử lý request.

ASP.NET Core bao gồm 3 kiểu Action Selector:

1. Action Name
2. Non Action
3. Action Verbs

Action Name

Thuộc tính **ActionName** định nghĩa tên của một action. Routing engine sẽ sử dụng tên này thay vì tên phương thức để khớp với action name trong routing. Dùng attribute này khi muốn đặt alias cho tên phương thức:

```
[ActionName("tên-alias")]  
public string ActionMethod()  
{  
    //code .....  
}
```

```
[ActionName("Modify")]  
0 references  
public string Edit()  
{  
    return "Edit Method có alias là Modify";  
}
```

❖ Ngoài ra, có thể sử dụng route attribute để thay đổi Action Name

```
[Route("Controller/Alias")]  
public string ActionMethod()  
{  
    //code .....  
}
```

```
[Route("Home/Modify")]  
0 references  
public string Edit()  
{  
    return "Edit Method with Modify alias";  
}
```

Non Action

Khai báo thuộc tính **NonAction** cho Routing Engine biết đó là một phương thức đặc biệt không phải là một Action method, sẽ không hiển thị dữ liệu khi cố gắng truy cập URL.

```
[NonAction]
0 references
public string Edit()
{
    return "Edit Method using NonAction";
}
```


Action Verbs

- ❖ Action verbs selector được sử dụng khi muốn điều khiển action method dựa trên HTTP Request method. Điều này được đảm nhiệm sử dụng tập các attribute bởi MVC, ví dụ như `HttpGet` và `HttpPost`. Nó được gọi là `Http Attributes`.
- ❖ Một số HTTP Verbs có sẵn trong ứng dụng ASP.NET Core. Chúng là `GET`, `POST`, `PUT`, `DELETE`, `HEAD`, `OPTIONS`, `PATCH`. Mỗi verbs này kết hợp với HTTP Method Attributes được định nghĩa trong namespace `Microsoft.AspNetCore.Mvc.Routing`.
- ❖ Có thể áp dụng các attribute này cho Controller action method. Khi client gửi request sử dụng một verb cụ thể, routing engine sẽ tìm controller action với một attribute tương ứng và gọi nó.
- ❖ HTTP Attribute cho phép định nghĩa 2 phương thức với cùng tên nhưng khác kiểu response với HTTP Verb khác nhau.

Một phương thức Edit trả về một **Get** request và tạo ra Edit Form. Phương thức khác nhận **Post** request và cập nhật database.

[HttpGet]

0 references

```
public ActionResult Edit(string id)
{
    //Return the Edit Form
    return View();
}
```

[HttpPost]

0 references

```
public ActionResult Edit(Model model)
{
    //Update the database here
}
```

Gán routing value trong HTTP action verbs

Ngoài cách sử dụng thuộc tính routing để cấu hình route trong Route Attribute. Có thể sử dụng HTTP Action verb để gán routing

```
[HttpGet("")]
[HttpGet("Home")]
[HttpGet("Home/Index")]
0 references
public IActionResult Index()
{
    EmployeeModel employee = new EmployeeModel()
    {
        EmployeeID = 1,
        FirstName = "Paul",
        LastName = "Andrew",
        Location = "Ấn Độ",
        Email = "paul@gmail.com"
    };
    return Json(employee);
}
```

Sử dụng nhiều Action Verbs

AcceptVerbs attribute cho phép sử dụng nhiều action verb trên action method:

```
[AcceptVerbs("GET", "POST")]  
0 references  
public IActionResult Detail()  
{  
    Customer customer = new Customer();  
    return View(customer);  
}
```

Action Filter

Filter trong [ASP.NET](#) Core MVC cho phép thực thi code trước hoặc sau giai đoạn nhất định trong request processing pipeline.

Những filters được xây dựng sẵn trong [ASP.NET](#) Core MVC xử lý các tác vụ như:

- Authorization (ngăn chặn truy cập tới tài nguyên của một user chưa được xác thực).
- Đảm bảo rằng tất cả các requests sử dụng HTTPS.
- Response caching (Trả về response đã được cache trước đó).

Custom filter có thể được tạo ra để xử nhiều vấn đề cụ thể khác. Filter có thể tránh việc trùng lặp code trong nhiều actions. Ví dụ, filter cho việc xử lý lỗi (error handling exception) là một trong những trường hợp như thế. Thay vì việc phải handle trong từng action của controller, bạn có thể làm điều đó tại một nơi duy nhất là Filter.

Action Filter

Built-in Filter – [RequireHttps]

Thuộc tính [RequireHttps] là filter đã có sẵn, khi **áp dụng cho controller hoặc action**, chỉ cho phép request đến ở dạng HTTPS, ngược lại sẽ bị khóa

Minh họa:

Mở file launchSettings.json

```
"profiles": {  
  "http": {  
    "commandName": "Project",  
    "dotnetRunMessages": true,  
    "launchBrowser": true,  
    "applicationUrl": "http://localhost:5140",  
    "environmentVariables": {  
      "ASPNETCORE_ENVIRONMENT": "Development"  
    }  
  },  
  "https": {  
    "commandName": "Project",  
    "dotnetRunMessages": true,  
    "launchBrowser": true,  
    "applicationUrl": "https://localhost:7173;http://localhost:5140",  
    "environmentVariables": {  
      "ASPNETCORE_ENVIRONMENT": "Development"  
    }  
  }  
}
```

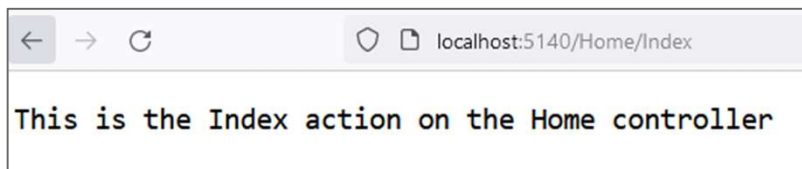
Mở file Program.cs **comment** dòng `app.UseHttpsRedirection()` để không dùng https

```
Program.cs* [X]
SampleWebCore
9  if (!app.Environment.IsDevelopment())
10 {
11     app.UseExceptionHandler("/Home/Error");
12     // The default HSTS value is 30 days. You may
13     app.UseHsts();
14 }
15
16 //app.UseHttpsRedirection();
17 app.UseStaticFiles();
18
19 app.UseRouting();
20
```

Tạo Index action đơn giản trong HomeController

```
public string Index()
{
    return "This is the Index action on the Home controller";
}
```

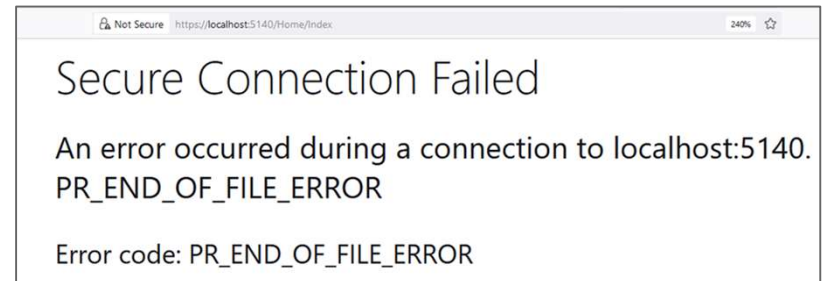
Chạy với non-https



Khai báo `[RequireHttps]` cho action Index

```
[RequireHttps]
0 references
public string Index()
{
    return "This is the Index action on the Home controller";
}
```

Chạy với https => Lỗi



Các loại Filter

Filter	Interfaces	Description
Authorization	<code>IAuthorizationFilter</code> , <code>IAsyncAuthorizationFilter</code>	Dùng cho xác thực và bảo mật
Action	<code>IActionFilter</code> , <code>IAsyncActionFilter</code>	Chạy trước và sau khi một action được gọi.
Result	<code>IResultFilter</code> , <code>IAsyncResultFilter</code>	Chạy ngay trước và sau khi action controller đã thực thi thành công
Exception	<code>IExceptionFilter</code> , <code>IAsyncExceptionFilter</code>	Dùng để xử lý các ngoại lệ

Action Filter

Action Filter là gì? Action Filters được thực thi trước và sau khi một phương thức action được thực thi. Chúng đứng thứ hai theo thứ tự thực hiện trong filter pipeline (luồng), tức là được thực thi sau Authorization Filters.

Action Filter được dẫn xuất từ interface `IActionFilter` hoặc `IAsyncActionFilter` không đồng bộ.

Định nghĩa interface **`IActionFilter`**

```
namespace Microsoft.AspNetCore.Mvc.Filters {  
    public interface IActionFilter : IFilterMetadata {  
        void OnActionExecuting(ActionExecutingContext context);  
        void OnActionExecuted(ActionExecutedContext context);  
    }  
}
```

Khi chúng ta áp dụng Action Filter trên phương thức Action, phương thức `OnActionExecuting` được gọi ngay trước khi phương thức action được gọi và phương thức `OnActionExecuted` được gọi ngay sau khi phương thức hành động kết thúc thực thi.

Phương thức `OnActionExecuting` có tham số thuộc loại `ActionExecutingContext`.

Các thuộc tính quan trọng của đối tượng `ActionExecutingContext` là:

Name	Description
Controller	Tên của controller có phương thức action sắp được gọi.
Result	Khi thuộc tính này được đặt một giá trị kiểu <code>IActionResult</code> thì dot Net sẽ kết xuất (hiển thị) <code>IActionResult</code> thay vì gọi phương thức action

Phương thức OnActionExecuted có tham số là loại lớp ActionExecutedContext.
Các thuộc tính quan trọng của lớp ActionExecutedContext là:

Name	Description
Controller	Tên của controller có phương thức action được gọi.
Exception	Chứa ngoại lệ xảy ra trong phương thức action
ExceptionHandled	Khi đặt thành true, các ngoại lệ sẽ không được lan truyền thêm
Result	Trả về IActionResult do phương thức hành động trả về và có thể thay đổi hoặc thay thế nó theo nhu cầu.

Ví dụ Custom Action Filter

Tạo một Action Filter để đo số mili giây mà một phương thức hành động cần để thực thi. Để làm điều này, chúng ta sẽ bắt đầu hẹn giờ trong phương thức `OnActionExecuting` và dừng trong phương thức `OnActionExecuting`.

Tạo 1 lớp `TimeElapsed.cs` trong folder `CustomFilters` và thêm code sau:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using System.Diagnostics;
```

```
namespace Filters.CustomFilters
```

```
{
```

```
    public class TimeElapsed : Attribute, IActionFilter
```

```
    {
```

```
        private Stopwatch timer;
```

```
        public void OnActionExecuting(ActionExecutingContext context)
```

```
        {
```

```
            timer = Stopwatch.StartNew();
```

```
        }
```

```
        public void OnActionExecuted(ActionExecutedContext context)
```

```
        {
```

```
            timer.Stop();
```

```
            string result = " Elapsed time:" + $"{timer.Elapsed.TotalMilliseconds} ms";
```

```
            IActionResult iActionResult = context.Result;
```

```
            ((ObjectResult) iActionResult).Value += result;
```

```
        }
```

```
    }
```

```
}
```

Tạo đối tượng của lớp Stopwatch để đo thời gian. Bắt đầu đo trong OnActionExecuting và stop trong OnActionExecuted.

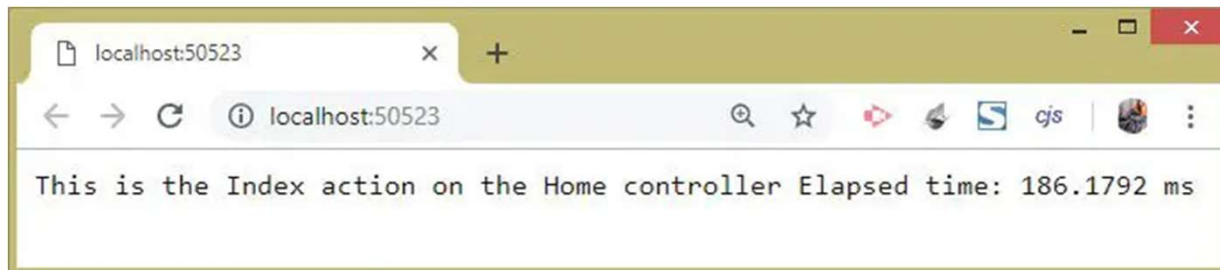
Ép kiểu

This is the Index action on the Home controller

Thuộc tính Result của đối tượng ActionExecutedContext

Khai báo thuộc tính `[TimeElapsed]` vào controller

```
public class HomeController : Controller
{
    [TimeElapsed]
    public string Index()
    {
        return "This is the Index action on the Home controller";
    }
}
```



Middleware

- ❖ **Middleware** là một khái niệm mới trong ASP.NET Core dùng để chỉ một module/class có khả năng xử lý truy vấn HTTP và trả lại kết quả ở dạng HTTP Response.
- ❖ Mỗi một tiến trình middleware thao tác với các request nhận được từ middleware trước đó và nó cũng có thể quyết định gọi middleware tiếp theo trong pipeline hay trả về response cho middleware ngay trước nhằm thực hiện hành vi dừng request pipeline.
- ❖ Cơ chế đăng ký và ghép nối middleware này giúp ứng dụng ASP.NET Core chỉ cần sử dụng những gì mình cần, không dùng những chức năng dư thừa, qua đó làm ứng dụng nhẹ và nhanh hơn.

- ❑ Có thể tưởng tượng quy trình xử lý với middleware giống như một phân xưởng có hai băng chuyền ngược chiều. Truy vấn HTTP tới từ trình duyệt sẽ chạy trên một băng chuyền (pipeline). Mỗi middleware là một công nhân đứng dọc băng chuyền đó và chỉ chịu trách nhiệm xử lý một phần xác định. Nếu hoàn tất quá trình xử lý, sản phẩm sẽ chuyển sang băng chuyền thứ hai trả ngược lại cho trình duyệt.

BÀI TẬP

- Cho model như sau

```
public class Customer
{
    public int CustomerID { get; set; }
    public string CustName { get; set; }
    public string Email { get; set; }
    public string Mobile { get; set; }
    public string Address { get; set; }
}
```

Hãy tạo Controller và các Action:

- 1) Tạo Constructor để khởi tạo dữ liệu ban đầu;
- 2) Index(): Hiển thị thông tin
- 3) Add(): Hiển thị form nhập liệu và gán dữ liệu cho các thuộc tính của model. Sau đó hiển thị lại thông tin.
- 4) Edit(): Hiển thị lại dữ liệu được chọn trên form và cập nhật lại thay đổi
- 5) Delete(): Xóa dữ liệu được chọn