



Chương 5: ASP.NET Core MVC (tt)

Giảng viên: Tạ Việt Phương
E-mail: phuongtv@uit.edu.vn

Nội dung



ASP.NET Core MVC - Controller



ASP.NET Core MVC - View

Nội dung



ASP.NET Core MVC - Controller

Controller

- 📖 **Controller:** xử lý mọi yêu cầu từ View đến thông qua URL.
- 📖 Thường đặt trong thư mục Controllers trong project.
- 📖 Controller là class thỏa các điều kiện sau:
 - Có thể được khởi tạo, có hàm khởi tạo public, không static và abstract.
 - Có tên gọi kết thúc bằng **Controller**. VD: **HomeController**
 - Là lớp dẫn xuất của lớp **Controller** hoặc **ControllerBase**.
 - Nếu là lớp dẫn xuất của lớp **Controller** thì tên gọi không cần kết thúc bằng từ **Controller**.
- 📖 Chứa các phương thức public được gọi là phương thức action.
- 📖 Action xử lý các yêu cầu từ trình duyệt, lấy dữ liệu từ Model trả về cho View.

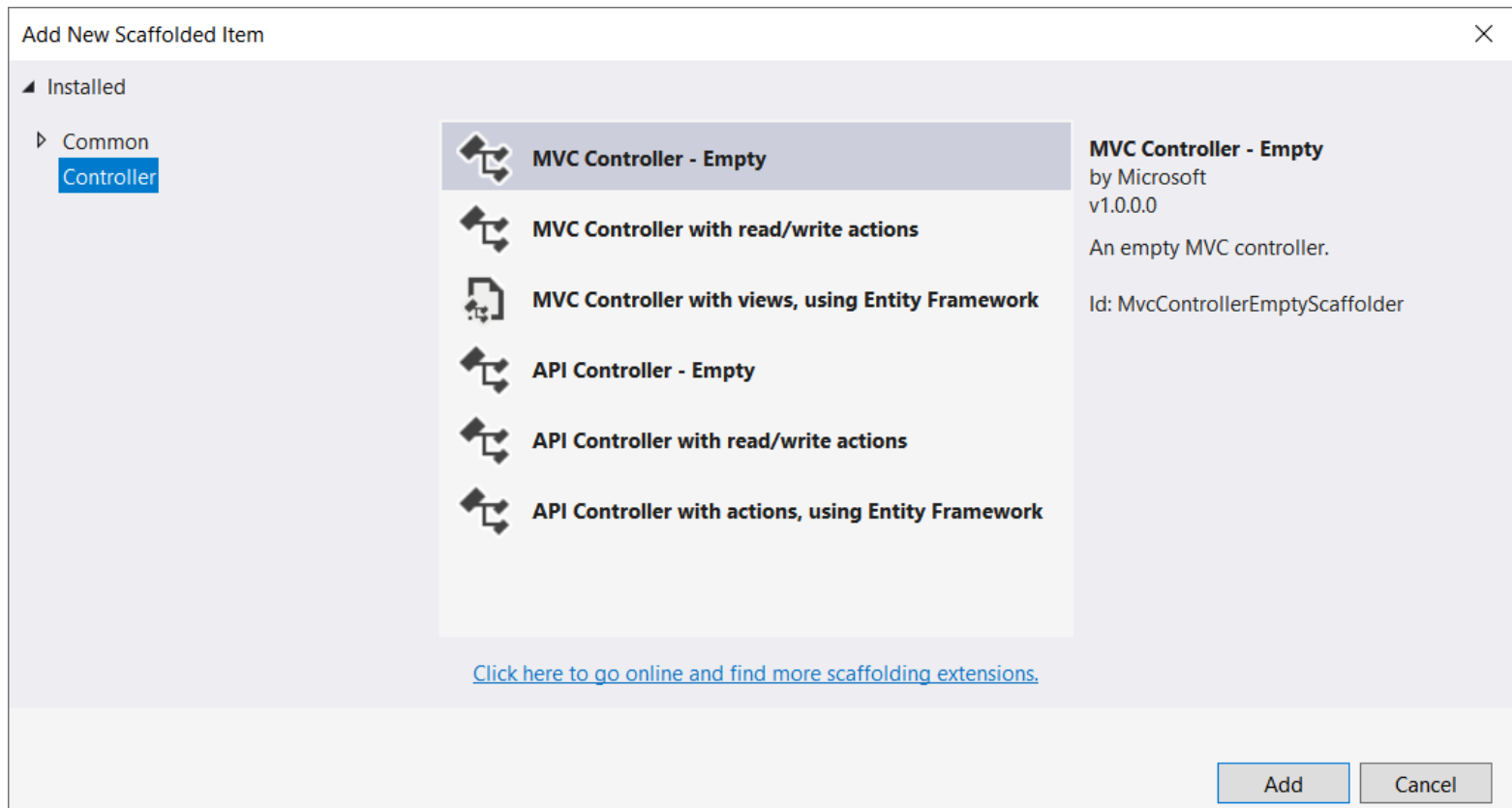
Controller

Trách nhiệm của Controller:

- Nhận và xử lý request từ người dùng thông qua URL.
- Gửi request đến các service tương ứng.
- Dựng model: Controller Action method thực thi logic của ứng dụng và xây dựng nên model.
- Gửi model đến view theo dạng HTML, File, JSON, XML hoặc bất cứ định dạng nào để tạo giao diện.
- Gửi các lỗi validation hoặc lỗi thực thi về view (nếu có)
- Controller không truy cập đến tầng data.
- Tương tác cả model và view, điều khiển luồng dữ liệu đi vào Model và cập nhật View khi có thay đổi dữ liệu
- Giữ View và Model tách biệt nhau.

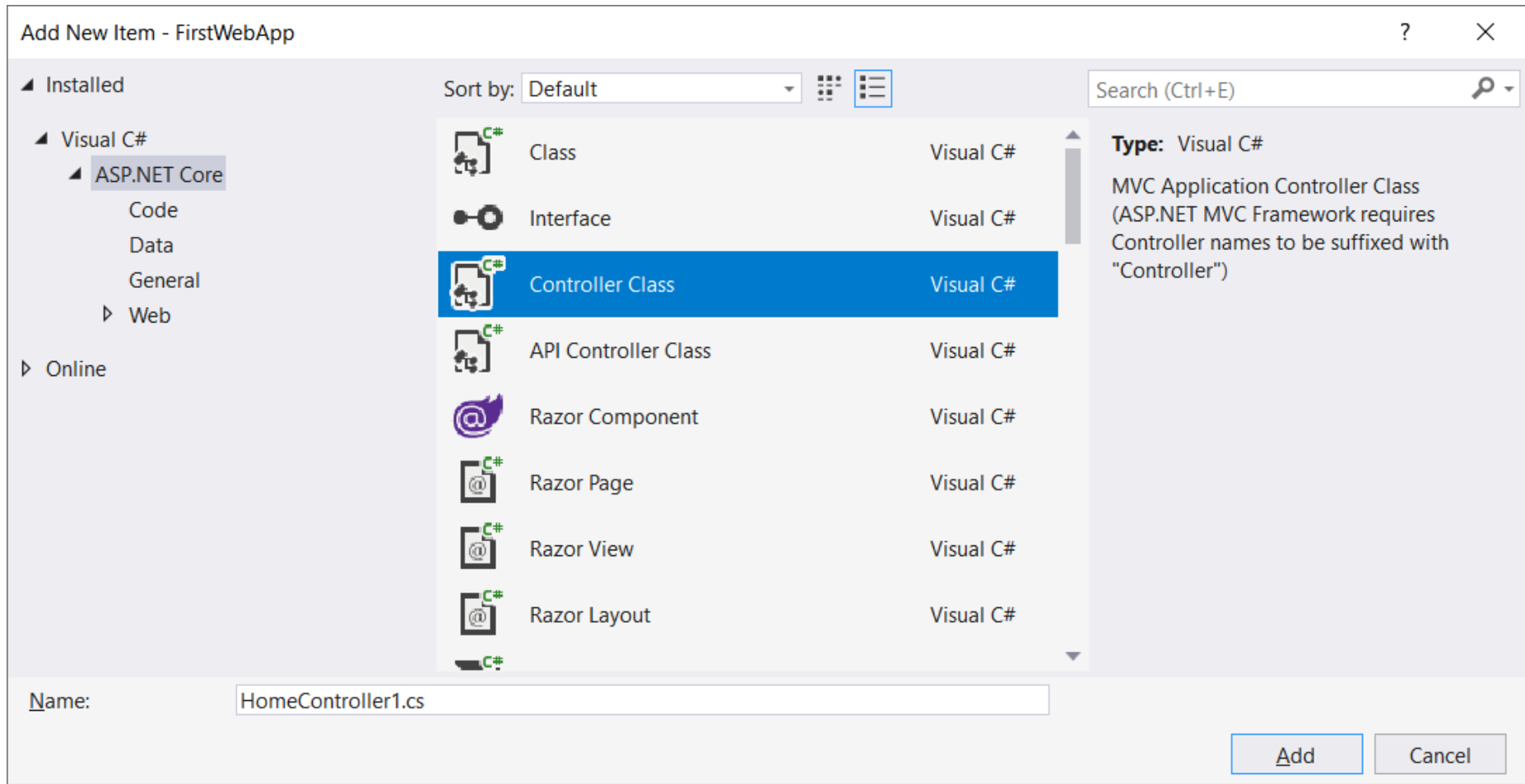
Thêm Controller

- 📖 Nhấp chuột phải tại thư mục Controllers > Add > Controller hoặc New Item...
- Tại cửa sổ Add Scaffold chọn MVC Controller – Empty (nếu chọn Controller)



Thêm Controller

- Với Controllers > Add > New Item... chọn Controller Class



Thêm Controller

- Thêm đoạn code sau vào Controller mới tạo

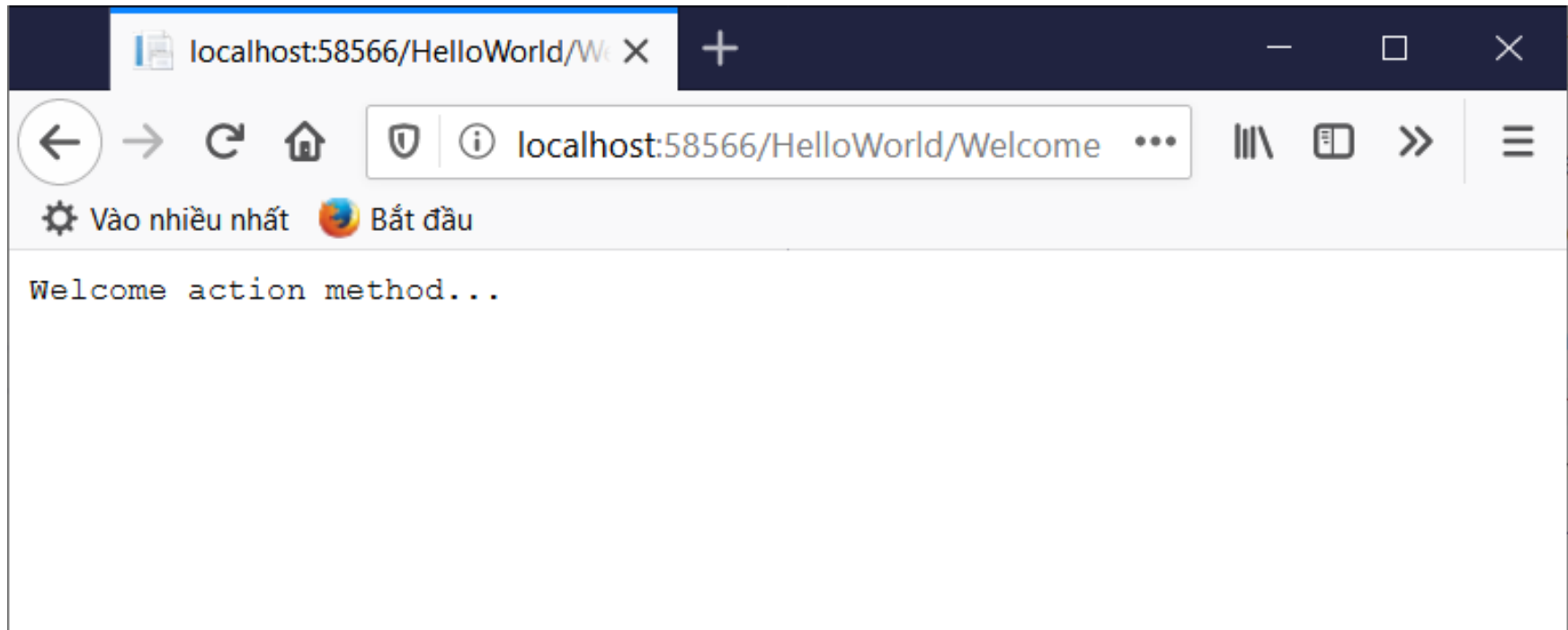
```
public class HelloWorldController : Controller
{
    // GET: HelloWorld/
    0 references
    public string Index()
    {
        return "Hello world default action";
    }

    // GET: HelloWorld/Welcome
    0 references
    public string Welcome()
    {
        return "Welcome action method...";
    }
}
```


Thêm Controller

- 📖 Tất cả các phương thức public trong controller đều có thể được gọi như là một trang web.
- 📖 Đường dẫn URL để gọi một controller gồm tên lớp controller và tên phương thức cần gọi:

<http://localhost:58566/HelloWorld/Welcome>

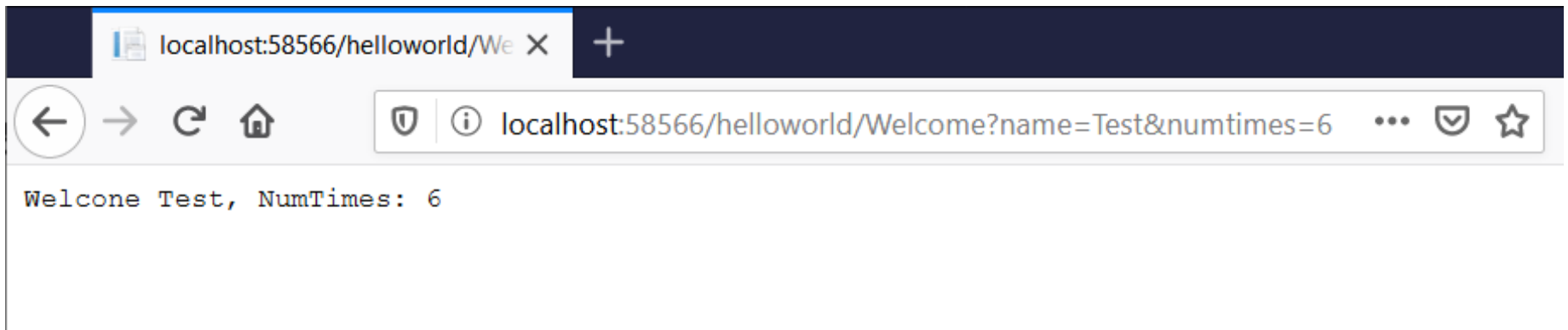


Thêm Controller

📖 Controller với tham số truyền vào:

```
public string Welcome(string name, int numTimes = 1)
{
    return HtmlEncoder.Default.Encode($"Welcone {name}, NumTimes: {numTimes}");
}
```

<http://localhost:58566/helloworld/Welcome?name=Test&numtimes=6>

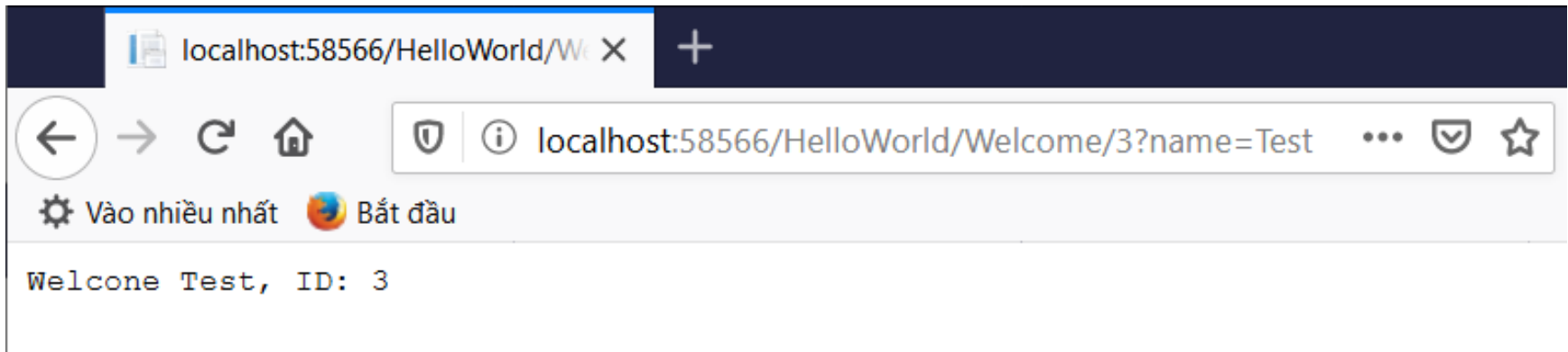


Thêm Controller

📖 Controller với tham số truyền vào có tham số ID:

```
public string Welcome(string name, int ID = 1)
{
    return HtmlEncoder.Default.Encode($"Welcone {name}, ID: {ID}");
}
```

<http://localhost:58566/HelloWorld/Welcome/3?name=Test>



Action method

- 📖 Là phương thức xử lý truy vấn trong lớp controller.
- 📖 Là phương thức public trong controller.
- 📖 Trong ASP.NET Core MVC, mỗi URL tương ứng với một phương thức.
- 📖 Khi truy vấn đến server, Mvc Middleware căn cứ vào URL để gọi phương thức cần thực thi.
- 📖 VD: url: `/helloworld/welcome` sẽ gọi action method tên `Welcome()` trong `HelloWorldController`.
- 📖 Mỗi action chịu trách nhiệm:
 - Xác nhận tính chính xác của truy vấn.
 - Thực thi các lệnh tương ứng với truy vấn.
 - Lựa chọn loại phản hồi phù hợp (bắt buộc): là các dạng như chuỗi, dạng object thuộc kiểu `ActionResult` hoặc `IActionResult`.

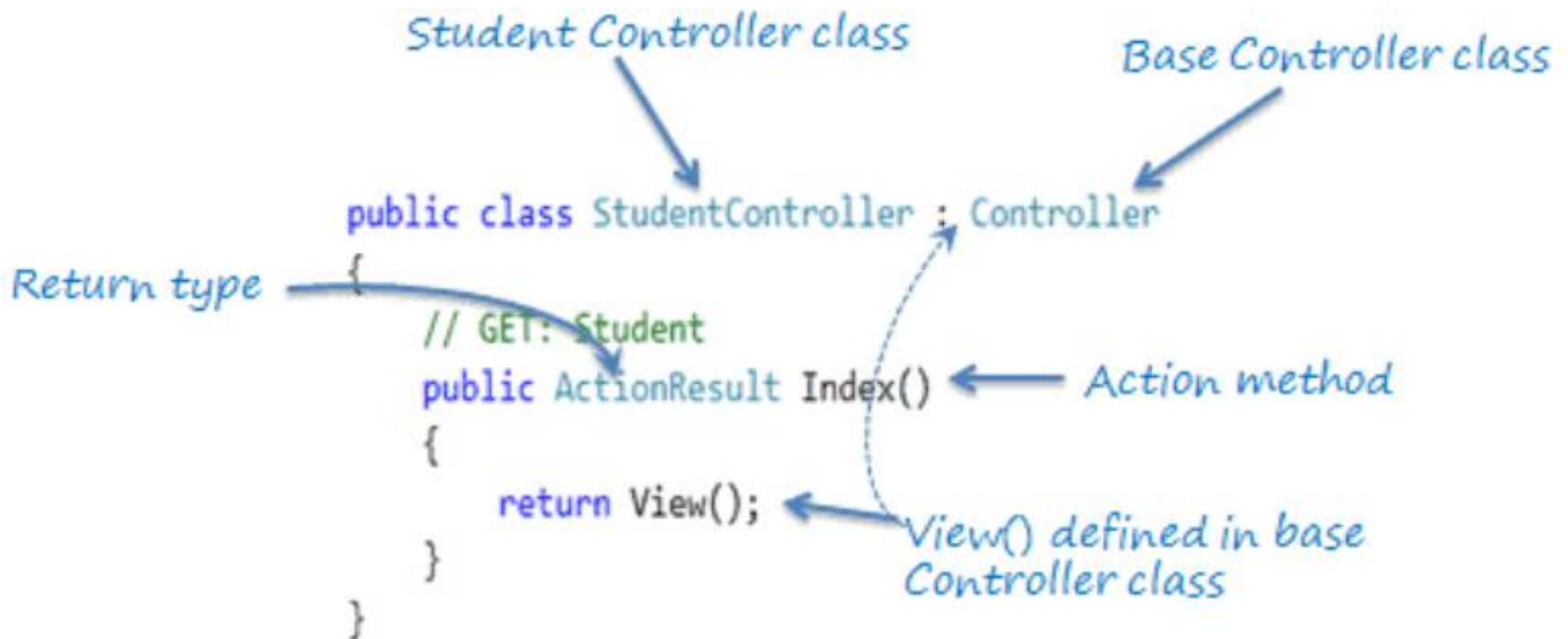
Action method

📖 Một số lưu ý với action method:

- Phải là một public method
- Không là static hoặc extension method
- Các phương thức constructor, getter, setter không được sử dụng.
- Các phương thức được kế thừa không được sử dụng như là action method
- Tham số không được chứa từ khóa **ref** hoặc **out**.
- Không thể nạp chồng (overloaded) action method.
- Không trực tiếp sinh ra view; chỉ chuẩn bị dữ liệu và lựa chọn view phù hợp.
- Không nên thực hiện các loại tính toán nghiệp vụ trong action method.

Action method

- 📖 Ví dụ: phương thức Index là một phương thức public và nó trả về ActionResult bằng phương thức View (). Phương thức View () được định nghĩa trong lớp cơ sở Controller, trả về ActionResult.



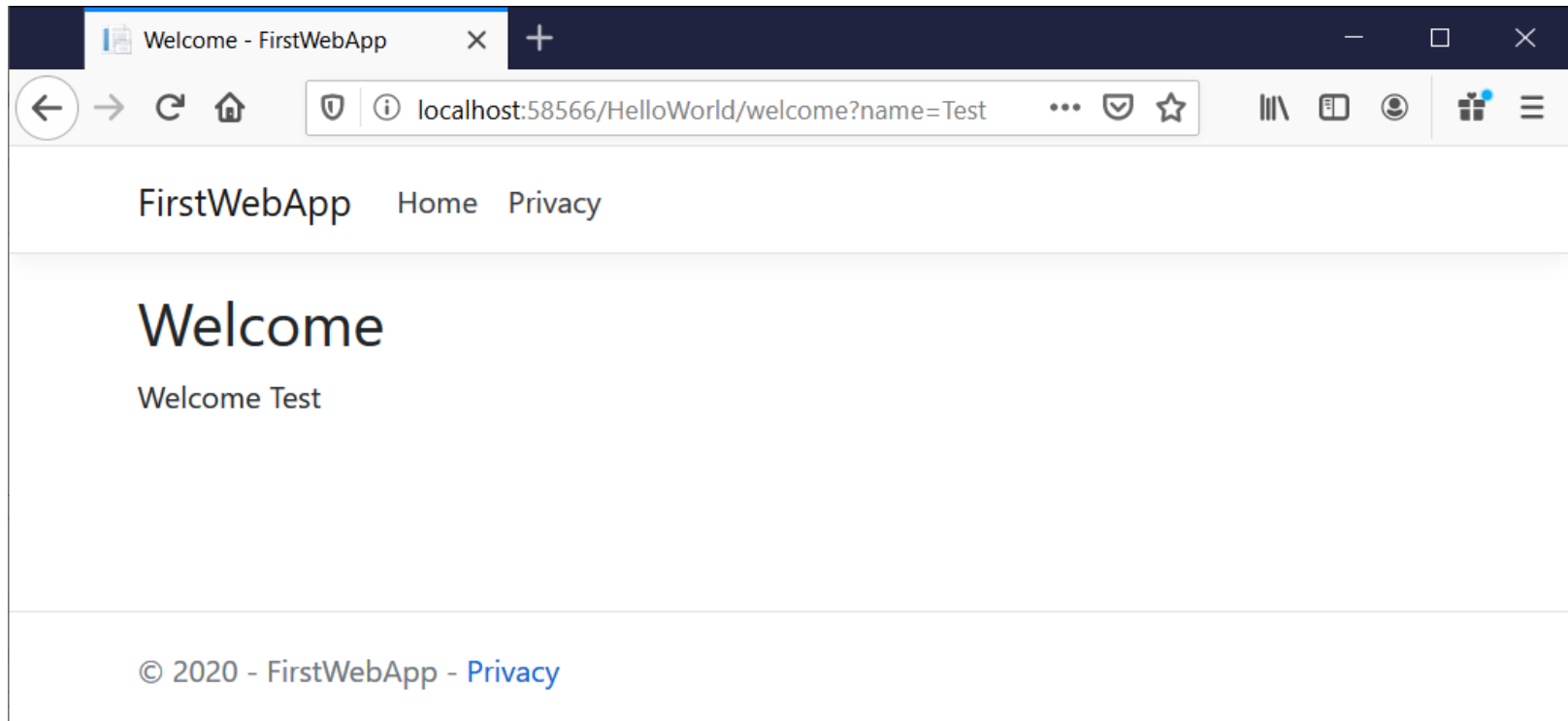
Action method

- 📖 Kiểu trả về là một trang HTML, JSON, XML hay một file.
- 📖 Tổng quan action method trả về một class abstract `ActionResult` hoặc `ActionResult`.
- 📖 Các kiểu trả về của action result thuộc `ActionResult` (`ActionResult`):
 - `ViewResult`: hiển thị kiểu HTML
 - `EmptyResult`: hiển thị không kết quả.
 - `RedirectResult`: chuyển hướng đến một URL mới.
 - `JsonResult`: trả về kiểu JSON object, thường dùng cho AJAX.
 - `JavaScriptResult`: trả về đoạn code javascript.
 - `ContentResult`: trả về văn bản.
 - `FileContentResult`: trả về file để tải theo dạng nhị phân
 - `FilePathResult`: trả về file để tải theo đường dẫn
 - `FileStreamResult`: trả về file để tải dạng stream

Action method

📖 Ví dụ:

```
public IActionResult Welcome(string name)
{
    ViewData.Model = name;
    var view = new ViewResult();
    view.ViewData = ViewData;
    return view;
}
```



Action method

- 📖 Để đơn giản việc trả về của action, lớp `Controller` xây dựng nhiều hàm hỗ trợ.
- 📖 Các hàm được đặt tên theo class mà nó hỗ trợ bỏ đi từ **Result**.
- 📖 VD: `View()` hỗ trợ trả kết quả của `ViewResult`.

```
public IActionResult Welcome(string name, int numTimes = 1)
{
    string s = name;
    ViewData["Name"] = name;
    ViewData["NumTimes"] = numTimes;
    return View();
}
```

Action method

Các phương thức hỗ trợ của lớp Controller

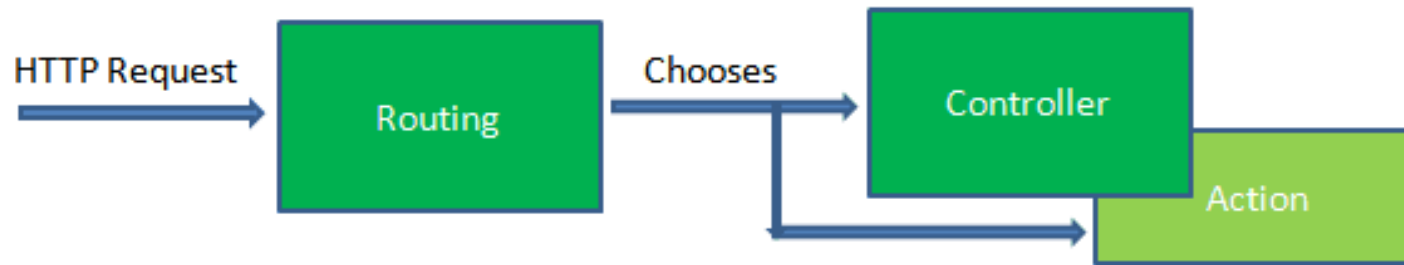
Result	Mô tả	Hàm hỗ trợ
ViewResult	Hiển thị kiểu HTML	View()
PartialViewResult	Hiển thị bên view không layout	PartialView()
RedirectResult	Chuyển hướng đến URL mới	Redirect()
RedirectToRouteResult	Chuyển sang một action hoặc một action của controller khác	RedirectToRoute() RedirectToAction()
JsonResult	Trả về dữ liệu kiểu JSON	Json()
JavaScriptResult	Trả về đoạn code javascript	JavaScript()
ContentResult	Trả về nội dung là chuỗi/văn bản	Content()
FileContentResult FilePathResult FileStreamResult	Trả về nội dung một file	File()

Routing

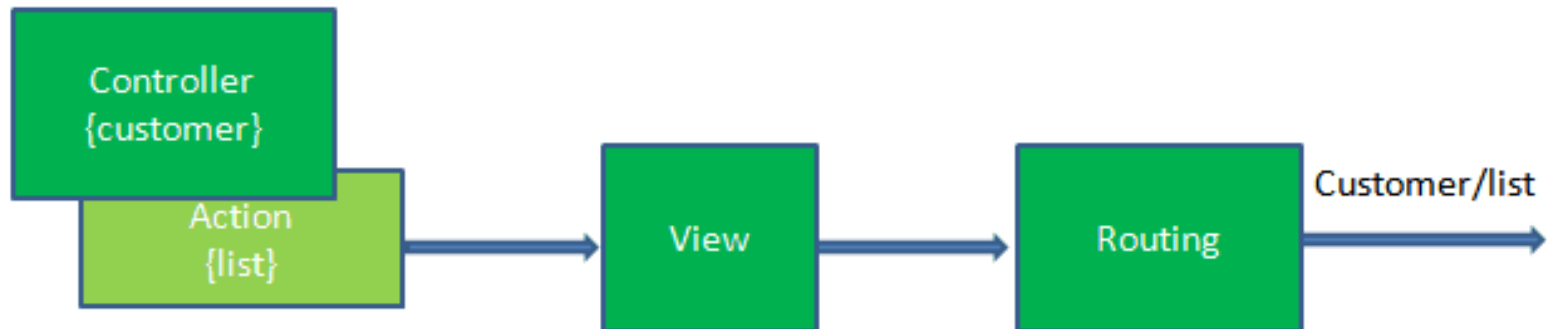
- 📖 Là thành phần quan trọng trong MVC, quyết định request sẽ được controller nào xử lý.
- 📖 Xem xét URL request gửi đến và chỉ định action method.
- 📖 Như vậy, routing có 2 chức năng chính:
 - Map request đến vào action method của Controller
 - Tạo ra URL tương ứng với kiểu trả về của action method
- 📖 Có hai loại routing trong ASP.NET Core MVC
 - *Conventional routing*: là dạng mặc định trong ASP.NET Core MVC sử dụng khi tạo project mới. Được cấu hình trong file **Startup.cs**
 - *Attribute routing*: được định nghĩa riêng cho từng action. Được sử dụng bằng cách đặt attribute [Route] trước mỗi action.

Routing

Routing in ASP.NET MVC Core



Mapping the Incoming Requests



Constructing the URLs

Conventional Routing

- 📖 Là dạng mặc định khi tạo một project template trong ASP.NET Core MVC.
- 📖 Định nghĩa routing chung cho controller và action (theo mẫu, template, quy ước) toàn project.
- 📖 Đơn giản, tiện lợi, dễ theo dõi và quản lý.
- 📖 Được cài đặt ở phương thức `Configure()` trong class `Startup`.
- 📖 Mỗi quy ước được định nghĩa là tham số cho phương thức `MapControllerRoute()`, chứa tên (name) và URL pattern.
- 📖 Có thể định nghĩa nhiều template cho nhiều routing.
- 📖 Mặc định ASP.NET Core MVC định nghĩa một template tên “default”

Conventional Routing

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())...
    else...
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

Tên route

URL Pattern

Conventional Routing

URL Pattern:

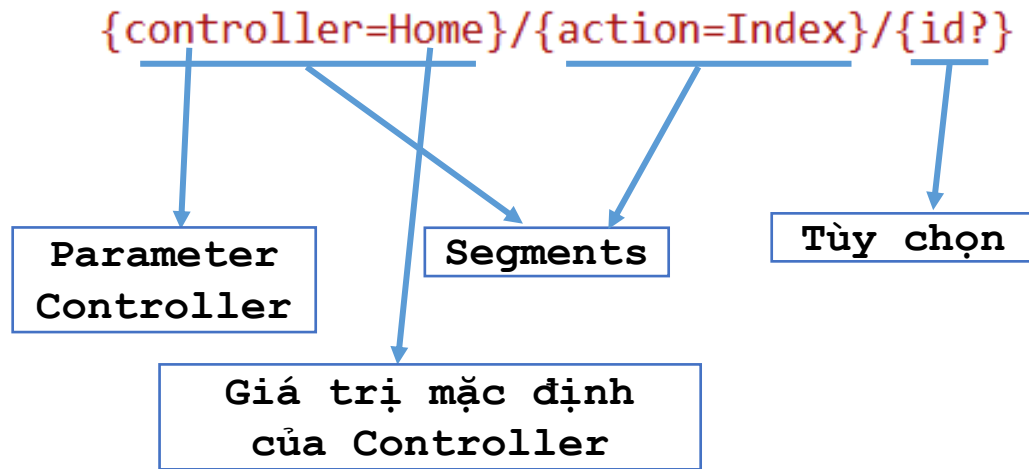
- Mỗi route phải chứa một URL Pattern và được so sánh với URL request.
- Gồm nhiều phần (segment), được phân tách bằng dấu gạch chéo, segment có thể là hằng số hoặc Route Parameter.

Segment là hằng số yêu cầu trùng tên với action, và không phân biệt hoa thường. VD: `hello`, `welcome`

Route Parameter:

- Được thể hiện trong cặp `{}` (`{controller}`, `{action}`).
- Có thể gán giá trị mặc định cho từng parameter (`{controller = Home}`).
- Có 2 loại parameter:
 - ASP.NET Core MVC định nghĩa và bắt buộc: `{controller}`, `{action}`.
 - Loại do người dùng định nghĩa: `{ten}`, `{tuoi}`.

Conventional Routing



- 📖 <http://localhost:58566>
- 📖 <http://localhost:58566/home>
- 📖 <http://localhost:58566/helloworld>
- 📖 <http://localhost:58566/helloworld/welcome>
- 📖 <http://localhost:58566/helloworld/welcome/6>

Conventional Routing

📖 VD: {admin}/{controller=Home}/{action=Index}

URL	Mô tả	Nhận diện
/	Lỗi. Không có mặc định cho admin	
/home	Có, “home” sẽ là admin	admin=home controller=Home action=Index
/welcome	Có, “welcome” sẽ là admin	admin=welcome controller=Home action=Index
/home/index	Không. admin=home Controller=index Không có IndexController	
/admin/home	Có	admin=admin controller=Home action=Index

Conventional Routing

📖 VD: `admin/{controller=Home}/{action=Index}`

URL	Mô tả	Nhận diện
/	Lỗi vì bắt buộc có thành phần đầu tiên trên URL	
/home	Lỗi vì thành phần đầu tiên phải là admin	
/welcome	Lỗi vì thành phần đầu tiên phải là admin	
/admin	Có. Controller=home	controller=Home action=Index
/admin/home	Có	controller=Home action=Index

Conventional Routing

📖 `{action}/{controller}`

📖 `"welcome/{name}-{address}", new {controller = "Helloworld", action = "welcome"})`

URL: `http://localhost:58566/welcome/test-uit`

📖 Lưu ý khi dùng Conventional Routing:

- Bắt buộc phải có chỉ định `{controller}` và `{action}` khi thêm routing.
- Nếu parameter không nằm trong URL pattern thì phải tạo ra một kiểu nặc danh để chỉ rõ giá trị của *controller* và *action*.
- Mỗi parameter trong URL pattern đều là bắt buộc.
- Để parameter không là bắt buộc thêm ký tự `?` sau tên parameter

Attribute Routing

- 📖 Sử dụng các attribute để định nghĩa trực tiếp các router trong action method.
- 📖 Được cài đặt trực tiếp trên action của controller.
- 📖 Có 3 tham số: URL pattern, tên và thứ tự.

📖 VD:

```
// GET: HelloWorld/  
[Route("Hola")]  
0 references  
public IActionResult Index()  
{  
    ...  
    return View();  
}
```

URL: <http://localhost:58566/hola>

- 📖 URL pattern không cần đúng với tên controller và action method
- 📖 Nếu sử dụng cả attribute và conventional thì attribute routing sẽ được ưu tiên trước.

Attribute Routing

- 📖 Sử dụng nhiều route cho một action method

```
[Route("")]
[Route("Hola")]
[Route("Hola/Index")]
0 references
public IActionResult Index()
{
    return View();
}
```

- 📖 Sử dụng token dạng biến thay thế [controller], [action]
- 📖 Các token đảm bảo tên Controller và tên Action ở đó sẽ được đồng bộ với tên của class controller tự động.

```
[Route("[controller]/[action]")]
0 references
public IActionResult Index()
{
    return View();
}
```

Attribute Routing

- 📖 Gắn tham số vào action method trong cặp {}

```
[Route("[controller]/[action]/{name?}")]
0 references
public IActionResult Welcome(string name)
{
    string s = name;
    ViewData["Name"] = name;
    return View();
}
```

- 📖 Các URL pattern được xác định trên controller đều được gắn vào URL Pattern của action method

```
[Route("Hola")]
0 references
public class HelloWorldController : Controller
{
    // GET: HelloWorld/
    [Route("Index")]
    0 references
    public IActionResult Index()
    {
        return View();
    }
}
```

- 📖 Pattern bắt đầu bằng “/” thì được xem như là đường dẫn tuyệt đối và không kết hợp được với URL pattern controller:
[Route("/Index")]

Action Selector

- 📖 Là một attribute áp dụng cho controller action.
- 📖 Giúp Routing engine chọn đúng action method để xử lý request.
- 📖 Gồm 3 kiểu:
 - Action Name
 - Non Action
 - Action Verb

Action Selector

Action Name:

- Định nghĩa tên một action.
- Route engine sử dụng tên này thay cho tên của action method.
- Là dạng tên alias cho tên của action method
- Khi đã dùng action name thì không thể gọi action method bằng tên phương thức được nữa.
- Có thể dùng route attribute để thay đổi action name

```
[ActionName("Them")]
```

0 references

```
public string Insert()  
{  
    return "Thêm thành công";  
}
```

<http://localhost:58566/helloworld/them>

```
[Route("hola/them")]
```

0 references

```
public string Insert()  
{  
    return "Thêm thành công";  
}
```

<http://localhost:58566/hola/them>

Action Selector

Non Action:

- Các public method trong controller mặc định được xem là action method.
- Đặt thuộc tính NonAction để chỉ định đây không phải là một action method

```
[NonAction]
0 references
public string Insert()
{
    return "Thêm thành công";
}
```

Action Selector

Action Verb:

- Được sử dụng khi muốn điều khiển action method dựa trên HTTP Request method (Http Attributes)
- Một số HTTP Verb: GET, POST, PUT, DELETE, HEAD, OPTION, PATCH.
- Khi URL request sử dụng một verb cụ thể, route engine sẽ tìm và gọi controller action có attribute tương ứng.
- HTTP Attribute cho phép định nghĩa nạp chồng phương thức action với HTTP verb khác nhau.

```
[HttpGet]
0 references
public string Insert()
{
    ...
    return "Thêm thành công";
}
```

```
[HttpPost]
0 references
public IActionResult Insert(string id)
{
    ...
    return View();
}
```

Action Selector

- Có thể dùng Http Action verb để thay thế cho Route Attribute

```
[HttpGet("")]
[HttpGet("HelloWorld")]
[HttpGet("HelloWorld/Insert")]
0 references
public string Insert()
{
    return "Thêm thành công";
}
```

Action Selector

Các loại HTTP Attribute:

○ HttpGet:

- Giới hạn action method chỉ được nhận các url request sử dụng Get.
- Các tham số trên URL tự động được thêm vào như các tham số phương thức.

```
[HttpGet("{id}")]
0 references
public string Insert(string id)
{
    return "Thêm " + id;
}
```

○ HttpPost: giới hạn action method chỉ được nhận các url request sử dụng Post.

```
[HttpPost]
0 references
public IActionResult Insert(string id)
{
    return View();
}
```

Action Selector

- HttpDelete: chỉ chấp nhận HTTP Request sử dụng Delete verb, cho phép xóa tài nguyên.
- HttpPut: chỉ chấp nhận HTTP Request sử dụng Put verb, cho phép cập nhật hoặc tạo mới tài nguyên.
- AcceptVerbs attribute cho phép sử dụng nhiều action verb trên action method:
 - `[AcceptVerbs(HttpVerbs.Post | HttpVerbs.Get)]`
 - `public ActionResult GetAndPostAction() {`
 - `return RedirectToAction("Index"); }`

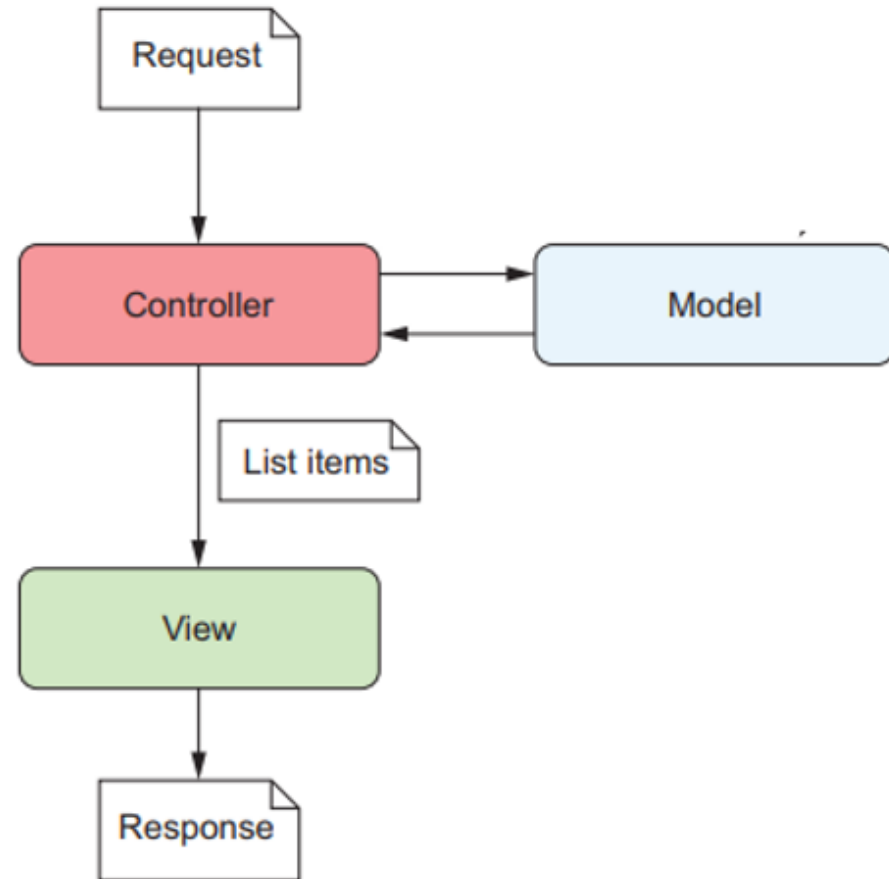
Nội dung



ASP.NET Core MVC - View

View

- Trong mô tả chung của mẫu kiến trúc MVC, view là thành phần chịu trách nhiệm sinh ra giao diện cho ứng dụng. Đây là khâu cuối cùng trong chuỗi xử lý yêu cầu của người dùng trước khi trả lại kết quả.
- Tuy nhiên, mỗi framework khi thực thi MVC lại diễn đạt view và mối quan hệ giữa nó với các thành phần khác theo cách riêng



View

- 📖 **View:** là thành phần chịu trách nhiệm tạo ra giao diện hiển thị cho người dùng.
- 📖 Controller sau khi xử lý sẽ trả về view model cho view hiển thị.
- 📖 Dữ liệu (object, danh sách object..) do action method của controller cung cấp.
- 📖 Thường đặt trong thư mục Views trong project.
- 📖 Ứng với mỗi Controller class sẽ là một thư mục con trong thư mục Views
- 📖 Mỗi action method của Controller class sẽ có một view để hiển thị.
- 📖 Có thể hiển thị nhiều loại định dạng HTML, JSON, XML...

View

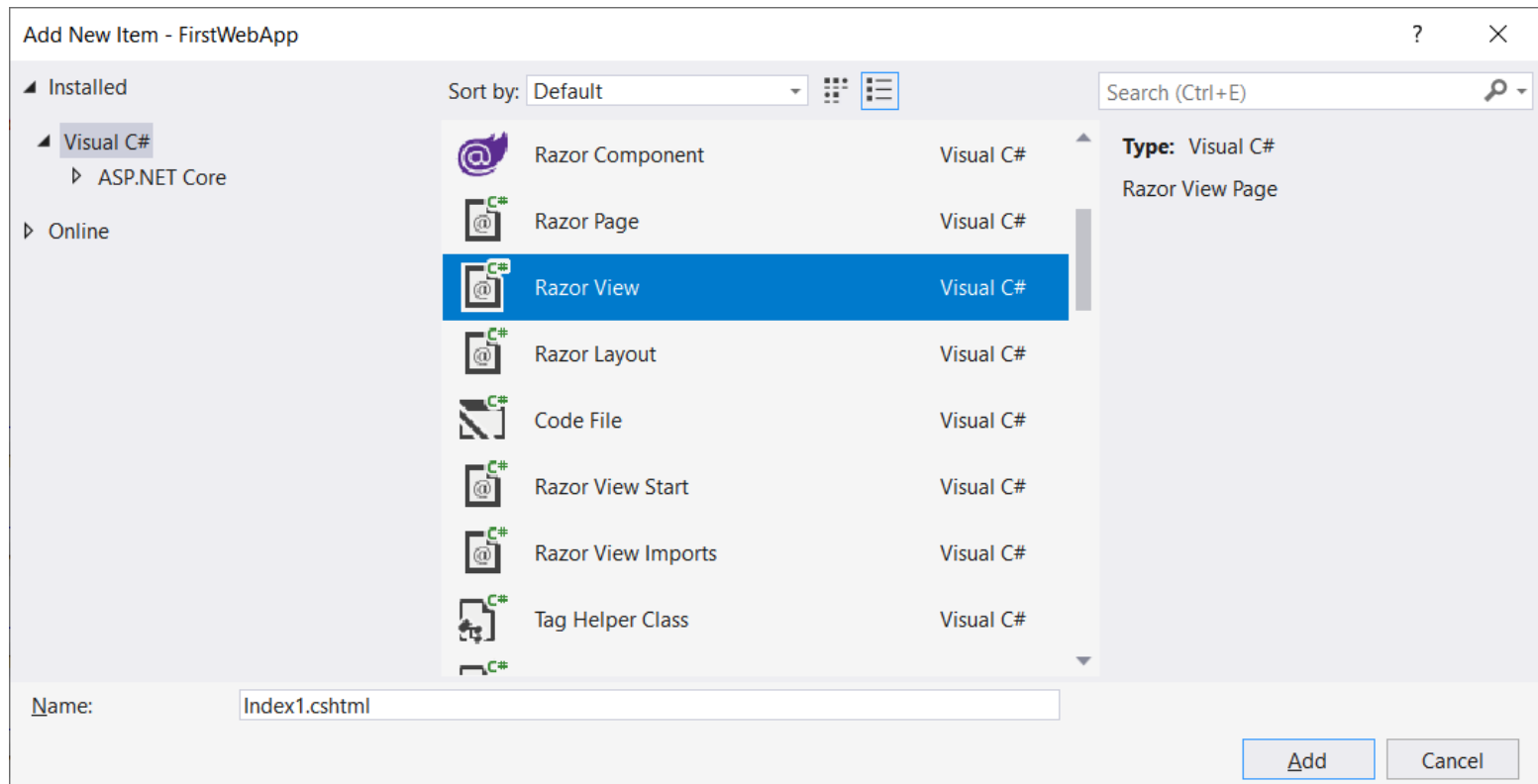


Trách nhiệm của View:

- Render ra giao diện và hiển thị model.
- Cung cấp khuôn mẫu (thông tin tĩnh, html) kết hợp với dữ liệu model từ controller để tạo ra giao diện.
- Sử dụng cú pháp **Razor** (kết hợp giữa HTML và C#) để tạo giao diện.
- Razor View có định dạng file **.cshtml**.

Thêm View

- ❏ Nhấp chuột phải tại thư mục Views > Add > New folder: đặt tên thư mục theo tên controller.
- ❏ Có thể tạo Razor View hoặc MVC View.
 - Nhấp chuột phải tại thư mục vừa tạo Add > New Item > Razor View.



Thêm View

Hoặc

- Nhấp chuột phải tại thư mục vừa tạo Add > View, để tạo View MVC với các template

Add MVC View

View name: View1

Template: Empty (without model)

Model class: Create
Delete
Details
Edit
Empty (without model)
List

Options:

☐ Create as a partial view

☒ Reference script files

☐ Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

Add Cancel

Thêm View

- 📖 Tên View: theo quy tắc cùng tên với Action Method trong Controller.
- 📖 View MVC với các template để tạo các view thêm, xóa, sửa... trên các model view được cung cấp:
 - **Create:** tạo HTML Form để tạo một model. Trên form có label, input cho mỗi thuộc tính model.
 - **Delete:** tạo HTML Form cho việc xóa model. Trên form có label và giá trị hiện tại cho mỗi thuộc tính model.
 - **Details:** Tạo ra view hiển thị giá trị của model.
 - **Edit:** tạo HTML Form cho việc chỉnh sửa model.
 - **Empty** (without model): tạo một view trống.
 - **List:** tạo HTML table để hiển thị danh sách model, mỗi cột là một thuộc tính của model. Cần trả về kiểu `IEnumerable<Model>`. View cũng chứa danh sách các thao tác thêm, xóa, sửa.

Thêm View

- 📖 Một số tùy chọn khác trong tạo template view:
 - **Model class:** tạo view template theo model class.
 - **Create a Partial View:** tùy chọn, dùng để tạo một phần view (không phải view hoàn chỉnh) vì không có thẻ <html> hoặc không có thẻ <head> và Layout sẽ bị bỏ đi.
 - **Thư viện Scripts:** tùy chọn, dùng để thêm các thư viện jquery phục vụ cho triển khai client-side validation.
 - **Use a layout page:** tùy chọn, cho phép chọn layout page cho View.

Thêm View

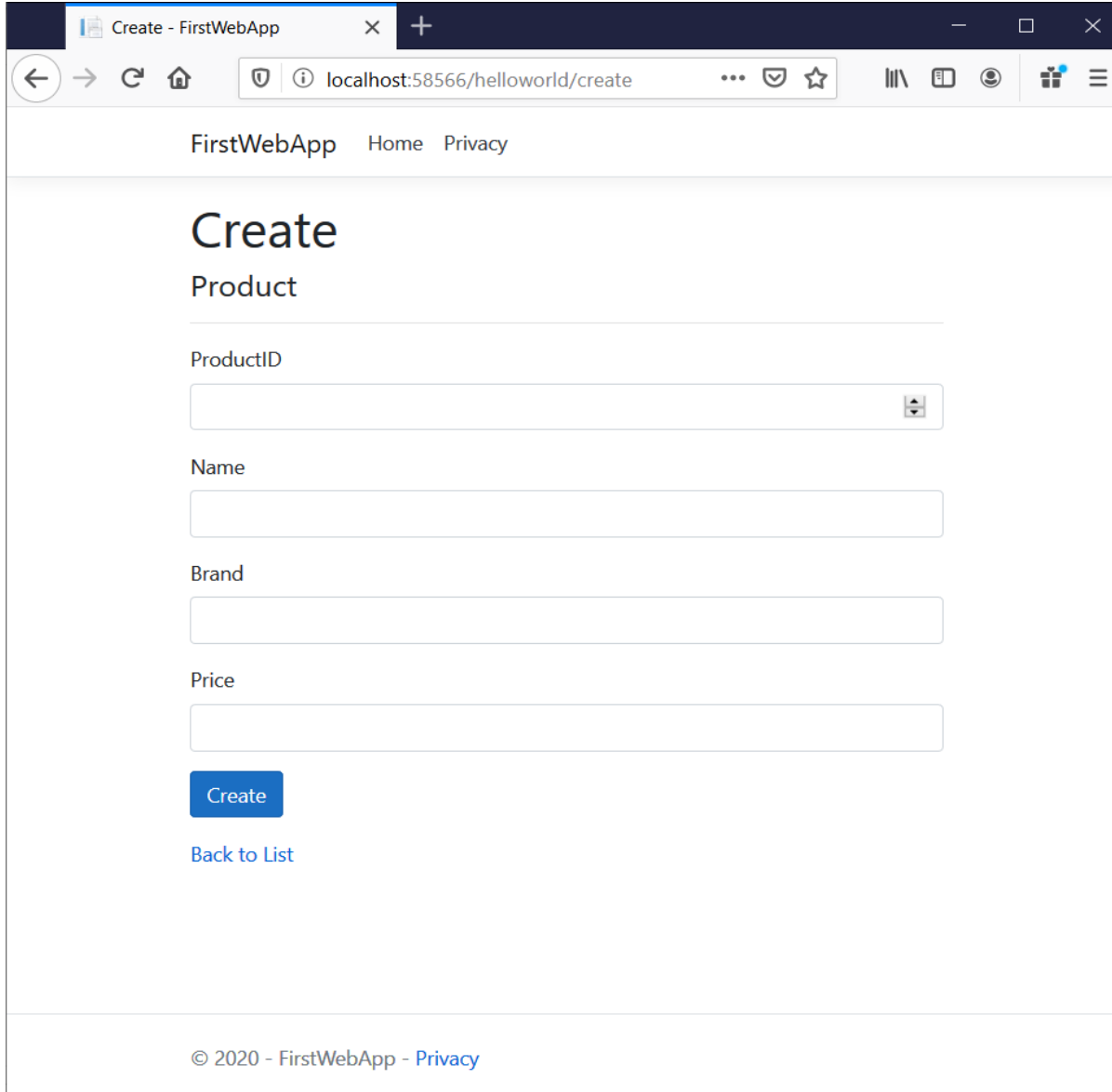
📖 Ví dụ tạo template view create:

```
Create.cshtml  + ×
1  @model FirstWebApp.Controllers.Product
2  @{
3      ViewData["Title"] = "Create";
4  }
5  <h1>Create</h1>
6  <h4>Product</h4>
7  <hr />
8  <div class="row">
9      <div class="col-md-4">
10         <form asp-action="Create">
11             <div asp-validation-summary="ModelOnly" class="text-danger"></div>
12             <div class="form-group">
13                 <label asp-for="ProductID" class="control-label"></label>
14                 <input asp-for="ProductID" class="form-control" />
15                 <span asp-validation-for="ProductID" class="text-danger"></span>
16             </div>
17             <div class="form-group">
18                 <label asp-for="Name" class="control-label"></label>
19                 <input asp-for="Name" class="form-control" />
20                 <span asp-validation-for="Name" class="text-danger"></span>
21             </div>
22             <div class="form-group">
23                 <label asp-for="Brand" class="control-label"></label>
24                 <input asp-for="Brand" class="form-control" />
25                 <span asp-validation-for="Brand" class="text-danger"></span>
26             </div>
27             <div class="form-group">
28                 <label asp-for="Price" class="control-label"></label>
29                 <input asp-for="Price" class="form-control" />
30                 <span asp-validation-for="Price" class="text-danger"></span>
31             </div>
32             <div class="form-group">
33                 <input type="submit" value="Create" class="btn btn-primary" />
34             </div>
35         </form>
36     </div>
37 </div>
38 <div>
39     <a asp-action="Index">Back to List</a>
40 </div>
```

100 % No issues found

Thêm View

 Ví dụ tạo template view create:



The screenshot shows a web browser window with the title 'Create - FirstWebApp'. The address bar displays 'localhost:58566/helloworld/create'. The page has a navigation bar with 'FirstWebApp', 'Home', and 'Privacy'. The main content area is titled 'Create' and 'Product'. It contains four input fields: 'ProductID' (with a dropdown arrow), 'Name', 'Brand', and 'Price'. Below the fields is a blue 'Create' button and a blue link 'Back to List'. The footer shows '© 2020 - FirstWebApp - Privacy'.

Create - FirstWebApp

localhost:58566/helloworld/create

FirstWebApp Home Privacy

Create

Product

ProductID

Name

Brand

Price

Create

[Back to List](#)

© 2020 - FirstWebApp - Privacy

Thêm View

 Ví dụ tạo template view delete:

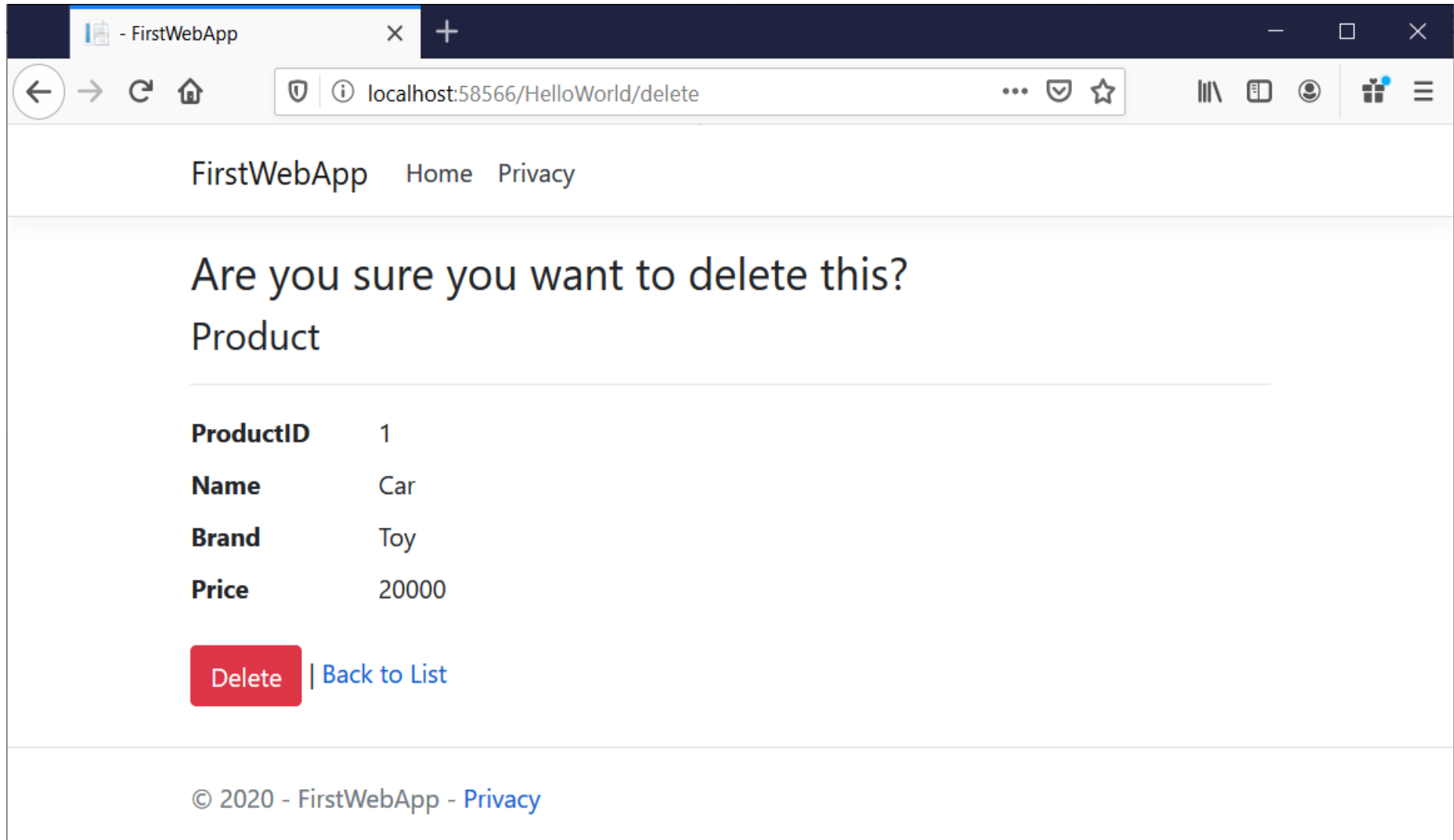
//Action Delete trong controller

```
0 references
public IActionResult Delete()
{
    var product = new Product { ProductID = 1, Name = "Car", Brand = "Toy", Price = 20000 };
    return View(product);
}
```

```
@model FirstWebApp.Controllers.Product
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Delete</title>
</head>
<body>
    <h3>Are you sure you want to delete this?</h3>
    <div>
        <h4>Product</h4>
        <hr />
        <dl class="row">
            <dt class="col-sm-2">@Html.DisplayNameFor(model => model.ProductID)</dt>
            <dd class="col-sm-10">@Html.DisplayFor(model => model.ProductID)</dd>
            <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.Brand)</dt>
            <dd class="col-sm-10">@Html.DisplayFor(model => model.Brand)</dd>
            <dt class="col-sm-2">@Html.DisplayNameFor(model => model.Price)</dt>
            <dd class="col-sm-10">@Html.DisplayFor(model => model.Price)</dd>
        </dl>
        <form asp-action="Delete">
            <input type="submit" value="Delete" class="btn btn-danger" /> |
            <a asp-action="Index">Back to List</a>
        </form>
    </div>
</body>
</html>
```


Thêm View

 Ví dụ tạo template view delete:



Thêm View

 Ví dụ tạo template view detail:

```
public IActionResult Details()
{
    var product = new Product { ProductID = 1, Name = "Car", Brand = "Toy", Price = 20000 };
    return View(product);
}

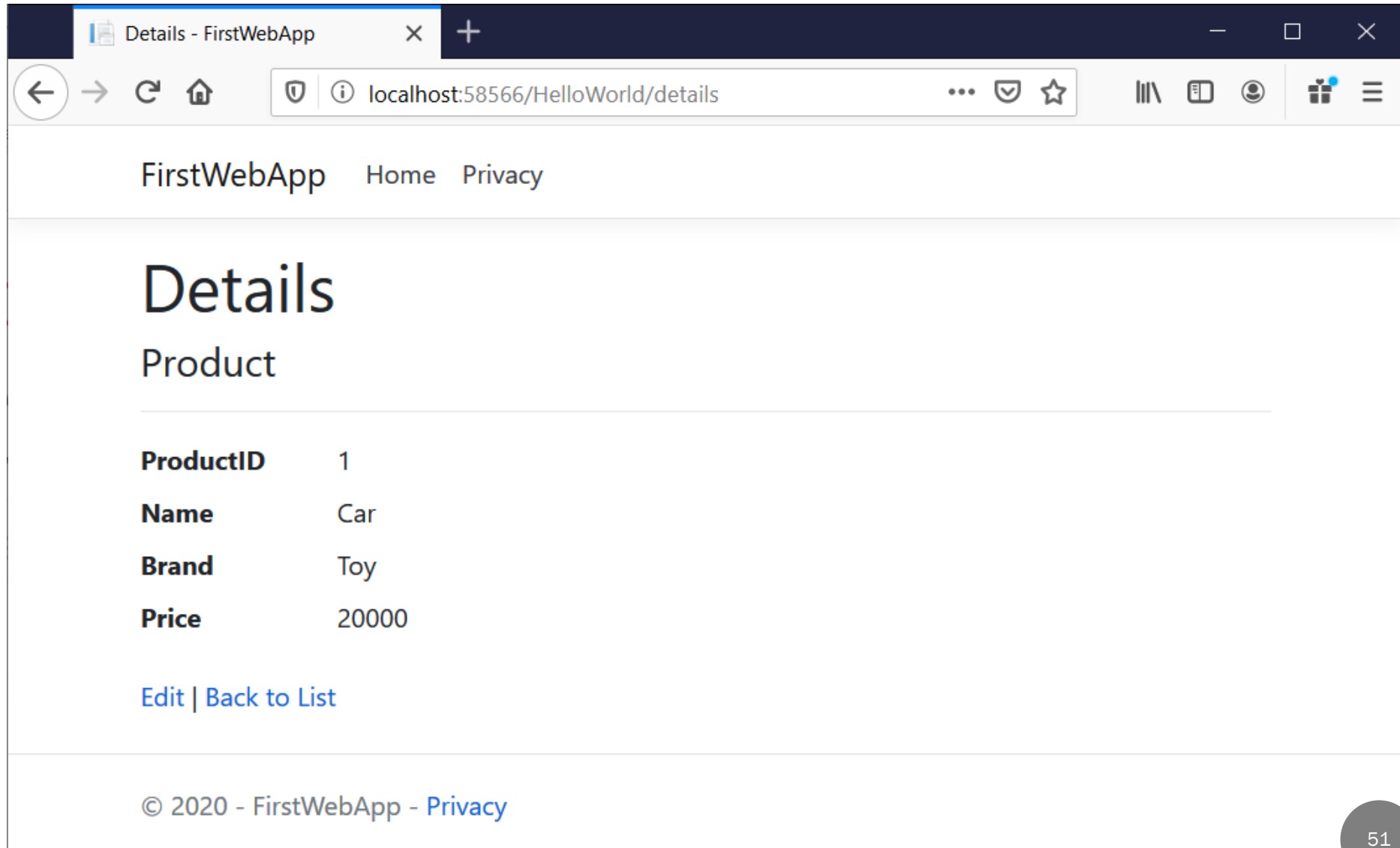
@model FirstWebApp.Controllers.Product

@{
    ViewData["Title"] = "Details";
}

<h1>Details</h1>
<div>
    <h4>Product</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">@Html.DisplayNameFor(model => model.ProductID)</dt>
        <dd class="col-sm-10">@Html.DisplayFor(model => model.ProductID)</dd>
        <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.Brand)</dt>
        <dd class="col-sm-10">@Html.DisplayFor(model => model.Brand)</dd>
        <dt class="col-sm-2">@Html.DisplayNameFor(model => model.Price)</dt>
        <dd class="col-sm-10">@Html.DisplayFor(model => model.Price)</dd>
    </dl>
</div>
<div>
    @Html.ActionLink("Edit", "Edit", new { /* id = Model.PrimaryKey */ }) |
    <a asp-action="Index">Back to List</a>
</div>
```

Thêm View

 Ví dụ tạo template view details:



The screenshot shows a web browser window with the title 'Details - FirstWebApp'. The address bar displays 'localhost:58566/HelloWorld/details'. The page content includes a navigation bar with 'FirstWebApp', 'Home', and 'Privacy'. The main heading is 'Details', followed by the subheading 'Product'. Below this is a table with product details. At the bottom of the table area are links for 'Edit' and 'Back to List'. The footer contains the copyright notice '© 2020 - FirstWebApp - Privacy'.

ProductID	1
Name	Car
Brand	Toy
Price	20000

[Edit](#) | [Back to List](#)

© 2020 - FirstWebApp - [Privacy](#)

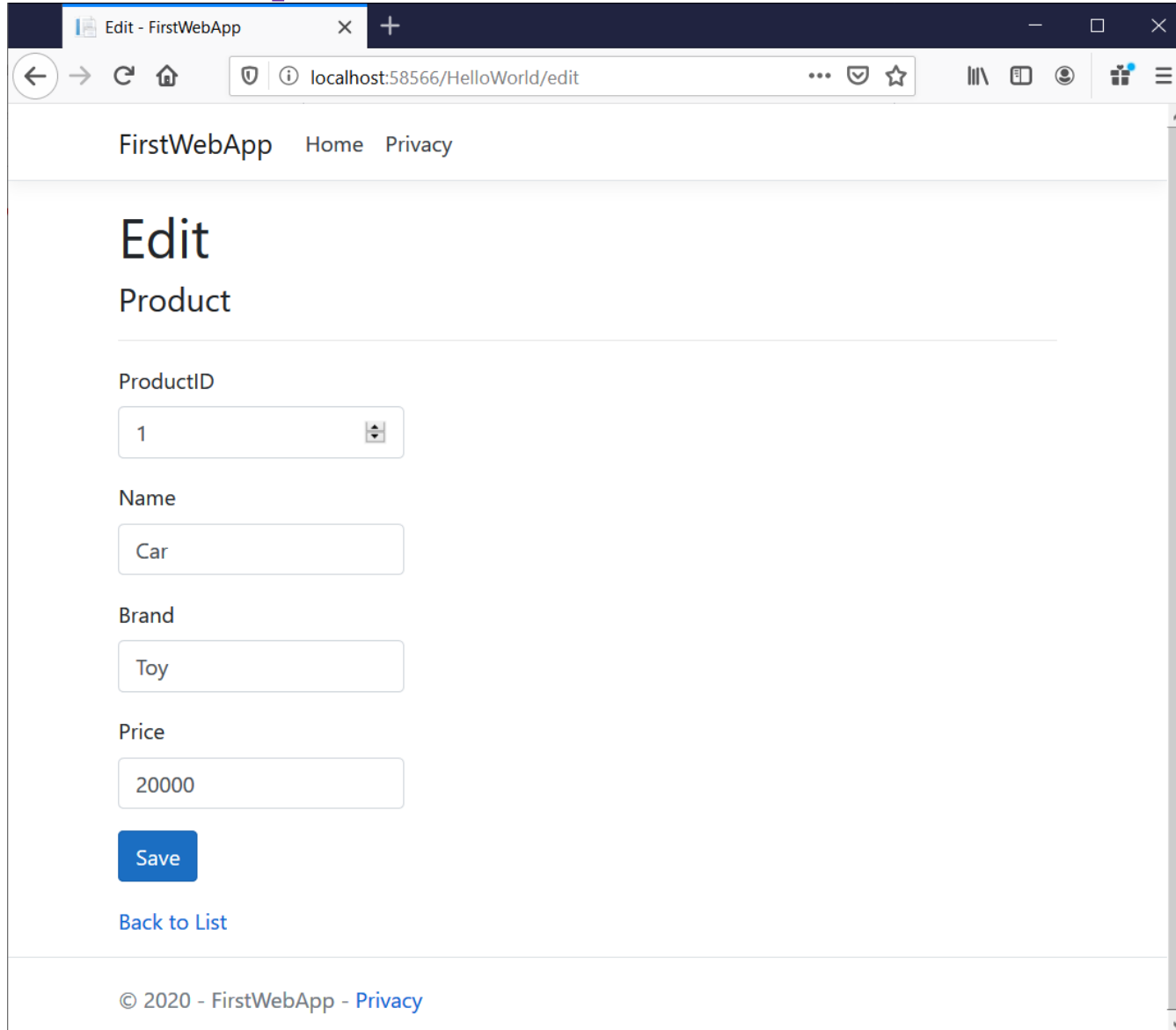
Thêm View

 Ví dụ tạo template view edit:

```
@model FirstWebApp.Controllers.Product
@{
    ViewData["Title"] = "Edit";
}
<h1>Edit</h1>
<h4>Product</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="ProductID" class="control-label"></label>
                <input asp-for="ProductID" class="form-control" />
                <span asp-validation-for="ProductID" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Brand" class="control-label"></label>
                <input asp-for="Brand" class="form-control" />
                <span asp-validation-for="Brand" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Price" class="control-label"></label>
                <input asp-for="Price" class="form-control" />
                <span asp-validation-for="Price" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>
<div>
    <a asp-action="Index">Back to List</a>
</div>
```

Thêm View

 Ví dụ tạo template view edit:



The screenshot shows a web browser window with the title 'Edit - FirstWebApp'. The address bar displays 'localhost:58566/HelloWorld/edit'. The page has a navigation bar with 'FirstWebApp', 'Home', and 'Privacy'. The main content area is titled 'Edit Product' and contains a form with the following fields:

- ProductID**: A dropdown menu with the value '1' selected.
- Name**: A text input field containing 'Car'.
- Brand**: A text input field containing 'Toy'.
- Price**: A text input field containing '20000'.

Below the form is a blue 'Save' button and a blue link labeled 'Back to List'. The footer of the page reads '© 2020 - FirstWebApp - Privacy'.

Thêm View

 Ví dụ tạo template view list:

```
public IActionResult List()
{
    List<Product> product = new List<Product>{
        new Product { ProductID = 1, Name = "Car", Brand = "Toy", Price = 20000 },
        new Product { ProductID = 2, Name = "Car", Brand = "Hon", Price = 10000 }
    };
    return View(product);
}
```

Thêm View

Ví dụ tạo template view list:

```
@model IEnumerable<FirstWebApp.Controllers.Product>
@{
    ViewData["Title"] = "List";
}
<h1>List</h1>
<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>@Html.DisplayNameFor(model => model.ProductID)</th>
            <th>@Html.DisplayNameFor(model => model.Name)</th>
            <th>@Html.DisplayNameFor(model => model.Brand)</th>
            <th>@Html.DisplayNameFor(model => model.Price)</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model) {
            <tr>
                <td>@Html.DisplayFor(modelItem => item.ProductID)</td>
                <td>@Html.DisplayFor(modelItem => item.Name)</td>
                <td>@Html.DisplayFor(modelItem => item.Brand)</td>
                <td>@Html.DisplayFor(modelItem => item.Price)</td>
                <td>
                    @Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) |
                    @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
                    @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
                </td>
            </tr>
        }
    </tbody>
</table>
```

Thêm View

📖 Ví dụ tạo template view list:

FirstWebApp Home Privacy

List

[Create New](#)

ProductID	Name	Brand	Price	
1	Car	Toy	20000	Edit Details Delete
2	Car	Hon	10000	Edit Details Delete

© 2020 - FirstWebApp - [Privacy](#)

ViewData

- 📖 Là đối tượng cho phép View lấy dữ liệu từ Controller.
- 📖 ViewData là thuộc tính của Controller base class. Trả về kiểu `ViewDataDictionary`.
 - Lưu trữ dữ liệu theo **key-value**.
 - Key không phân biệt hoa thường.
 - Không giới hạn số lượng key-value trong ViewData.
- 📖 Khi gọi phương thức `View()` trong action method, ViewData tự động được gán vào.
- 📖 Trong View, sử dụng key để truy xuất dữ liệu.
- 📖 Dữ liệu chỉ tồn tại trong request, khi view được tạo xong thì dữ liệu cũng bị hủy.
- 📖 ViewData có thể được dùng ở Partial View hoặc Layout

ViewData

- VD: tạo ViewData ở action method

```
public IActionResult Welcome(string name)
{
    ViewData["Name"] = name;
    return View();
}
```

- Gọi ở View:

```
<h2>Welcome</h2>
<p>Welcome @ViewData["Name"]</p>
```

- Có thể lưu trữ bất cứ kiểu dữ liệu trong ViewData: kiểu số nguyên, kiểu chuỗi, kiểu logic hay kiểu đối tượng.
- Kiểu chuỗi dữ liệu được dùng trực tiếp, không cần chuyển kiểu
- Khi dùng kiểu đối tượng, cần phải chuyển kiểu giá trị từ ViewData sang kiểu đối tượng tương ứng.

ViewData

- VD: dùng ViewData để gán đối tượng từ Controller sang View

```
public IActionResult Welcome()  
{  
    ViewData["Person"] = new Person()  
    {  
        ID = 1234,  
        Name = "ABC",  
        Address = "VN"  
    };  
  
    return View();  
}
```

- Gọi ở View:

```
<h2>Welcome</h2>  
@{ var person = ViewData["Person"] as FirstWebApp.Controllers.Person; }  
<p>Welcome @person.Name</p>  
<p>Address: @person.Address</p>
```

ViewBag

- 📖 Là đối tượng cho phép View lấy dữ liệu từ Controller.
- 📖 ViewBag: sử dụng kiểu động (dynamic), kiểu trả về **ViewDataDictionary**
- 📖 Cung cấp thuộc tính động và là lớp bao của ViewData
- 📖 Không cần chuyển kiểu dữ liệu.

📖 VD:

```
public IActionResult Welcome()
{
    ViewBag.Person = new Person()
    {
        ID = 1234,
        Name = "ABC",
        Address = "VN"
    };
    return View();
}
```

📖 Gọi ở View

```
@{
    Person person = ViewBag.Person;
}
<p>Welcome @person.Name</p>
<p>Address: @person.Address</p>
```

ViewData và ViewBag

- 📖 Luôn sử dụng **ViewDataDictionary** để lưu trữ dữ liệu.
- 📖 Luôn được dùng truyền dữ liệu từ Controller sang View
- 📖 Được xử lý lúc runtime, không kiểm tra ở lúc biên dịch.
- 📖 Khác nhau:
 - ViewData sử dụng cú pháp Dictionary để truy cập giá trị, ViewBag sử dụng cú pháp truy cập thuộc tính của đối tượng.
 - ViewData dẫn xuất từ **ViewDataDictionary** nên có các thuộc tính: ContainsKey, Add, Remove và Clear.
 - ViewBag nhận từ DynamicViewData, cho phép tạo động các thuộc tính bằng cách dùng dấu chấm: **ViewBag.Key=<giá trị>** và không cần chuyển kiểu khi sử dụng ở View.
 - ViewData cho phép Key có khoảng trắng: “Ho Ten”.
 - Cần chuyển kiểu xác định khi ViewData sử dụng kiểu dữ liệu object; và cần kiểm tra giá trị null.
 - Kiểm tra giá trị null trong ViewBag: `@ViewBag.person?.Name`

Model

- 📖 Là tập hợp chứa dữ liệu của ứng dụng, có thể chứa thêm xử lý logic.
- 📖 Có 3 loại: Domain Model, View Model và Edit Model
- 📖 Domain Model:
 - Thể hiện một đối tượng trong CSDL.
 - Mỗi đối tượng tương ứng với một bảng trong CSDL.
 - Có thể gọi là entity model hoặc data model.
- 📖 View Model:
 - Tham chiếu đến các đối tượng chứa dữ liệu.
 - Dùng cho việc hiển thị dữ liệu ở View.
 - Được định nghĩa dựa trên cách thức hiển thị dữ liệu ở view.
 - Có thể liên quan đến logic hiển thị.
- 📖 Edit Model (Input Model):
 - Là dạng để thay đổi hoặc thêm mới dữ liệu

View Model

- 📖 Được dùng để truyền dữ liệu từ Controller sang View.
- 📖 Tham chiếu đến các đối tượng chứa dữ liệu để hiển thị ở View.
- 📖 Độc lập với Domain Model cho nên có thể lấy dữ liệu từ nhiều Domain Model.
- 📖 View Model được gán bằng ViewBag hoặc ViewData.
- 📖 Hoặc sử dụng thuộc tính Model của ViewData.
- 📖 Sử dụng thuộc tính Model để tạo ra strongly typed view.

Thêm Model

- 📖 Chọn chuột phải ở thư mục Models -> Add -> Class; đặt tên class theo tên model.

```
public class Product
{
    1 reference
    public int ProductID { get; set; }
    1 reference
    public string Name { get; set; }
    1 reference
    public string Brand { get; set; }
    1 reference
    public float Price { get; set; }
    0 references
    public Product()
    {
        ProductID = 1;
        Name = "Car";
        Brand = "Toy";
        Price = 20000;
    }
}
```


Thuộc tính Model

📖 Sử dụng thuộc tính Model để trả về View Model.

○ Tại Controller:

```
public IActionResult Welcome()  
{  
    ViewData.Model = new Product();  
    return View();  
}
```

```
public IActionResult Welcome()  
{  
    Product product = new Product();  
    return View(product);  
}
```

○ Tại View:

```
<h2>Welcome</h2>
```

```
@{
```

```
    Product product = ViewData.Model;
```

```
    //Hoặc
```

```
    //Product product = Model;
```

```
}
```

```
<p>Welcome @product.Name</p>
```

```
<p>From brand: @product.Brand</p>
```

○ Sử dụng thuộc tính Model cũng có hạn chế như ViewBag. Trình biên dịch không biết kiểu của Model.

Thuộc tính Model

- View được gắn vào một kiểu cụ thể của ViewModel thay vì thuộc tính động gọi là *strongly typed view*.
- Sử dụng khai báo **@model** để tạo strongly typed view
- @model được khai báo trên đầu file View để chỉ ra kiểu của ViewModel được gán: `@model Product`.
- Tham chiếu trực tiếp đến model trong View

```
@model Product
<h2>Welcome</h2>
<p>Welcome @Model.Name</p>
<p>From brand: @Model.Brand</p>
```

- Với strongly typed view, kiểu của Model sẽ được kiểm tra lúc biên dịch.
- Chỉ có một thuộc tính Model nên mỗi View chỉ có một ViewModel

View Model

- 📖 View Model độc lập với Domain Model cho nên có thể lấy dữ liệu từ nhiều Domain Model.
- 📖 Lưu ý khi dùng View Model:
 - Tách biệt View Model và Domain Model, không lấy Domain Model thay cho View Model.
 - Mỗi View có một View Model riêng.
 - Tạo *strongly type view*, để View biết được kiểu của View Model.
 - Sử dụng Data Annotation để khai báo thuộc tính cho View Model, giúp tận dụng cơ chế client validation.
 - Cần hiển thị dữ liệu nào thì mới để trong View Model.
 - Có thể chứa xử lý logic đặc thù trong View Model.

Nội dung



Tag Helper



Model Binding



Model Validation

Nội dung



Tag Helper

Tag Helper

- 📖 Là loại thẻ đặc biệt được Razor engine xử lý để sinh ra code HTML/CSS/Javascript tương ứng.
- 📖 Các dạng code gần với code của HTML.
- 📖 Được sử dụng để thay thế cách viết code của HTML
- 📖 Khi thực thi sẽ sinh ra code HTML hoàn chỉnh.
- 📖 VD: thẻ input của Form trong Tag Helper.

```
<input asp-for="Name" class="form-control" />
```

Khi thực thi sẽ tạo ra code HTML

```
<input class="form-control" type="text" id="Name" name="Name" value="" />
```

- 📖 Hỗ trợ tạo view HTML đơn giản, có thể dựa trên dữ liệu từ Model đã gắn vào View.

Sử dụng Tag Helper

📖 Thuộc thư viện `Microsoft.AspNetCore.Mvc.TagHelpers`

📖 Sử dụng `@addTagHelper` để khai báo trong View

`@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers`

📖 Sử dụng “*” để chỉ ra tất cả các TagHelper trong thư viện được sử dụng.

📖 Hoặc chỉ định rõ TagHelper muốn sử dụng.

`@addTagHelper Microsoft.AspNetCore.Mvc.TagHelpers.FormTagHelper, Microsoft.AspNetCore.Mvc.TagHelpers`

📖 Để sử dụng TagHelper trên toàn bộ View trong project thì khai báo TagHelper ở file **_ViewImports.cshtml**.

Sử dụng Tag Helper

- Để bỏ không dùng Tag Helper sử dụng `@removeTagHelper`.

```
@removeTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

- Sử dụng “!” trước mỗi phần tử HTML để vô hiệu hóa tag helper cho phần tử đó

```
<!label asp-for="Brand" class="control-label"></!label>.
```

- Sử dụng `@tagHelperPrefix` để chỉ định tag nào có tiền tố theo định nghĩa sẽ thực hiện tag helper

```
@tagHelperPrefix ab:
```

```
<label asp-for="ProductID" class="control-label"></label>
```

```
<ab:label asp-for="Name" class="control-label"></ab:label>
```


Sử dụng Tag Helper

VD: Tạo ra form Edit dựa trên View Model Product.

```
<div class="row">
  <div class="col-md-4">
    <form asp-action="Edit">
      <div asp-validation-summary="ModelOnly" class="text-danger"></div>
      <div class="form-group">
        <label asp-for="ProductID" class="control-label"></label>
        <input asp-for="ProductID" class="form-control" />
        <span asp-validation-for="ProductID" class="text-danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="Name" class="control-label"></label>
        <input asp-for="Name" class="form-control" />
        <span asp-validation-for="Name" class="text-danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="Brand" class="control-label"></label>
        <input asp-for="Brand" class="form-control" />
        <span asp-validation-for="Brand" class="text-danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="Price" class="control-label"></label>
        <input asp-for="Price" class="form-control" />
        <span asp-validation-for="Price" class="text-danger"></span>
      </div>
      <div class="form-group">
        <input type="submit" value="Save" class="btn btn-primary" />
      </div>
    </form>
  </div>
</div>
```

Sử dụng Tag Helper

```
<form asp-action="Edit">
```



```
<form action="/HelloWorld/Edit" method="post">
```

```
<input name="__RequestVerificationToken" type="hidden"  
value="CfDJ8FPk8RWCU0lFmbzr8JG6_sVa46kSwp47kZ3E7FzniXtkhcjM2H1_JaLIjemBk-26BYLd5ABIx-  
TSTUGfpZZrq36rxq22ipwm4PIoR0Ch4TZrHxUunGE_zFdKilRKD7w_RKB3zR4jYuRRNfvh8NbD5LQ" /></form>
```

```
<label asp-for="ProductID" class="control-label"></label>
```



```
<label class="control-label" for="ProductID">ProductID</label>
```

```
<input asp-for="ProductID" class="form-control" />
```



```
<input class="form-control" type="number" data-val="true" data-val-required="The ProductID field is  
required." id="ProductID" name="ProductID" value="1" />
```

Sử dụng Tag Helper

```
<form action="/HelloWorld/Edit" method="post">

    <div class="form-group">
        <label class="control-label" for="ProductID">ProductID</label>
        <input class="form-control" type="number" data-val="true" data-val-required="The ProductID field is required."
id="ProductID" name="ProductID" value="1" />
        <span class="text-danger field-validation-valid" data-valmsg-for="ProductID" data-valmsg-replace="true">
</span>
    </div>
    <div class="form-group">
        <label class="control-label" for="Name">Name Car</label>
        <input class="form-control" type="text" data-val="true" data-val-required="The Name Car field is required."
id="Name" name="Name" value="Car" />
        <span class="text-danger field-validation-valid" data-valmsg-for="Name" data-valmsg-replace="true"></span>
    </div>
    <div class="form-group">
        <label class="control-label" for="Brand">Brand</label>
        <input class="form-control" type="text" data-val="true" data-val-required="Nhập" id="Brand" name="Brand"
value="Toy" />
        <span class="text-danger field-validation-valid" data-valmsg-for="Brand" data-valmsg-replace="true"></span>
    </div>
    <div class="form-group">
        <label class="control-label" for="Price">Price</label>
        <input class="form-control" type="text" data-val="true" data-val-number="The field Price must be a number."
data-val-required="The Price field is required." id="Price" name="Price" value="20000.00" />
        <span class="text-danger field-validation-valid" data-valmsg-for="Price" data-valmsg-replace="true"></span>
    </div>
    <div class="form-group">
        <input type="submit" value="Save" class="btn btn-primary" />
    </div>
    <input name="__RequestVerificationToken" type="hidden"
value="CfDJ8FPk8RWCU0lFmbzr8JG6_sWiWm4MAG0qjUCKmX5uIVdldOMKXgo6kXw3KvvyoHVT-
z2AcMRRWBMh1eiID6ptPF5oiUytyv9z4ZwfPVL6ezoQeTUN1jUWfLTyJ28tZsTAEjqmarxCS4gD2vOj-fbUmQXo" /></form>
```

Ưu điểm Tag Helper

- 📖 Hỗ trợ tạo view HTML đơn giản.
- 📖 Dễ sử dụng, viết code như viết HTML.
- 📖 Dễ dàng thêm CSS hoặc thuộc tính HTML vào Tag Helper.
- 📖 Được hỗ trợ bởi IntelliSense của VS.
- 📖 Rút gọn cách viết HTML.
- 📖 Dễ dàng tạo ra tag helper mới phù hợp với nhu cầu sử dụng.

Tag Helper

Tag Helper	Thẻ HTML	Thuộc tính
Form Tag Helper	<form>	asp-action, asp-all-route-data, asp-area, asp-controller, asp-fragment, asp-page, asp-page-handler, asp-route, asp-route-{value}
Anchor Tag Helper	<a>	asp-action, asp-all-route-data, asp-area, asp-controller, asp-fragment, asp-host, asp-page, asp-page-handler, asp-protocol, asp-route, asp-route-*
Label Tag Helper	<label>	asp-for
Input Tag Helper	<input>	asp-for
Image Tag Helper		asp-append-version
Link Tag Helper	<link>	href, asp-fallback-href, asp-fallback-test-class, asp-fallback-test-property, asp-fallback-test-value
Select Tag Helper	<select>	asp-for, asp-items
Option Tag Helper	<option>	asp-for, asp-items
Textarea Tag Helper	<textarea>	asp-for

Tag Helper

Tag Helper	Thẻ HTML	Thuộc tính
Validation Message Tag Helper		asp-validation-for
Validation Summary Tag Helper	<div>	asp-validation-summary
Script Tag Helper	<script>	asp-fallback-src, asp-fallback-test
Partial Tag Helper	<partial>	name, for, model, view-data
Cache Tag Helper	<cache>	enabled, expires-after, expires-on, expires-sliding, priority, vary-by-header, vary-by-queue, vary-by-route, vary-by-cookie, vary-by-user, vary-by
Environment Tag Helper	<environm ent>	names, include, exclude

Tham khảo: <https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/built-in/?view=aspnetcore-3.1>

Input Tag Helper

- 📖 Tạo ra thẻ input HTML.
- 📖 Tạo ra các thuộc tính trong thẻ input: type, name, id... dựa trên thuộc tính model (kiểu dữ liệu, data annotation) của View Model.
- 📖 Hỗ trợ validate dữ liệu ở client.
- 📖 Định dạng dữ liệu nhập vào.
- 📖 Sử dụng trên thẻ input HTML, thuộc tính **asp-for**

Input Tag Helper

- 📖 Thuộc tính: **asp-for**: dựa vào thuộc tính tương ứng trong View Model và data annotation để tạo ra thuộc tính cho thẻ input.
- 📖 Tạo ra phần tử HTML dựa trên các thông số:
 - Kiểu trong HTML.
 - Data annotation của thuộc tính model
 - Kiểu dữ liệu của thuộc tính.
- 📖 VD:
 - Model có thuộc tính: `public string Brand { get; set; }`
 - Thẻ input ở View: `<input asp-for="Brand" class="form-control" />`
 - Thẻ HTML khi thực thi:
`<input class="form-control" type="text" id="Brand" name="Brand" value="" />`

Input Tag Helper

📖 Kiểu trong HTML: chỉ định thuộc tính của thẻ HTML

📖 Data annotation của thuộc tính model

```
[EmailAddress]                                <input asp-for="Email" class="form-control" />
0 references
public string Email { get; set; }
```

```
<input class="form-control" type="email" data-val="true" data-val-email="The Email
field is not a valid e-mail address." id="Email" name="Email" value="" />
```

○ Các data annotation của các input type

Data annotation	Input type
[EmailAddress]	type="email"
[Url]	type="url"
[HiddenInput]	type="hidden"
[Phone]	type="tel"
[DataType(DataType.Password)]	type="password"
[DataType(DataType.Date)]	type="date"
[DataType(DataType.Time)]	type="time"

Input Tag Helper

- 📖 Kiểu dữ liệu của thuộc tính: nếu không có data annotation thì Input Tag Helper sẽ dựa vào kiểu dữ liệu của thuộc tính model.

```
public int ProductID { get; set; }    <input asp-for="ProductID" class="form-control" />
```

```
    <input class="form-control" type="number" data-val="true" data-val-required="The  
ProductID field is required." id="ProductID" name="ProductID" value="" />
```

- Kiểu dữ liệu cho các input type

Kiểu dữ liệu	Input type
bool	type="checkbox"
string	type="text"
DateTime	type="datetime-local"
byte, int, long	type="number"
decimal, double, float	type="text"

Input Tag Helper

- 📖 Kiểu dữ liệu của thuộc tính: nếu không có data annotation thì Input Tag Helper sẽ dựa vào kiểu dữ liệu của thuộc tính model.

```
public int ProductID { get; set; }      <input asp-for="ProductID" class="form-control" />
```

```
      <input class="form-control" type="number" data-val="true" data-val-required="The  
ProductID field is required." id="ProductID" name="ProductID" value="" />
```

- Các data annotation của các input type

Kiểu dữ liệu	Input type
bool	type="checkbox"
string	type="text"
DateTime	type="datetime-local"
byte, int, long	type="number"
decimal, double, float	type="text"

Input Tag Helper

- ❏ Input Tag Helper cũng tạo ra các thuộc tính để kiểm tra dữ liệu nhập vào HTML.
- ❏ Các thuộc tính để kiểm tra dữ liệu trong thẻ HTML bắt đầu bằng **data-val-***.
- ❏ Các thông tin kiểm tra: giá trị lớn nhất, nhỏ nhất, có bắt buộc nhập, miền giá trị, biểu thức, quy tắc nhập hoặc thông báo lỗi.
- ❏ Dựa trên các thuộc tính data-val-* để kiểm tra dữ liệu ở client.
- ❏ VD: thuộc tính **[Required]**

[Required]

24 references

```
public string Name { get; set; }
```

```
<input class="form-control" type="text" data-val="true" data-val-required="The Name field is required." id="Name" name="Name" value="" />
```

Input Tag Helper

- Để thay đổi thông báo `data-val-required` sử dụng thuộc tính `ErrorMessage` của `Required`.

```
[Required(ErrorMessage = "Nhập Name")]
```

- Một số kiểu dữ liệu không cần khai báo `[Required]`, input tag helper cũng tự động chèn `data-val-required`: kiểu `DateTime`, các kiểu số.
- Sử dụng dạng **nullable** cho kiểu dữ liệu để không tự động chèn `required`.

```
public double? Price { get; set; }
```

0 references

```
public DateTime? Date { get; set; }
```

```
<input class="form-control" type="text" data-val="true" data-val-number="The field Price must be a number." id="Price" name="Price" value="" />
```

```
<input class="form-control" type="datetime-local" id="Date" name="Date" value="" />
```

- Thuộc tính `data-val-number` để kiểm tra các kiểu dữ liệu số.

Input Tag Helper

- 📖 Các data annotation để tạo thuộc tính kiểm tra dữ liệu nhập.

Data annotation	Mô tả
Required	Bắt buộc phải nhập giá trị
DisplayName	Hiển thị tên của field (ở Label)
MaxLength	Số ký tự tối đa có thể nhập
MinLength	Số ký tự ít nhất phải nhập
Range	Miền giá trị của dữ liệu có thể nhập
Compare	Chỉ ra field khác để so sánh giá trị có giống nhau.
RegularExpression	Giá trị nhập phải tuân theo regular expression
DataType	Kiểm tra dữ liệu nhập có đúng kiểu dữ liệu đã định nghĩa.
Remote	Gọi về server để kiểm tra dữ liệu. VD: kiểm tra username có tồn tại không
StringLength	Độ dài tối đa của chuỗi nhập

Nội dung



Model Binding

Model Binding

- 📖 Được dùng để truyền dữ liệu từ View lên Controller.
- 📖 Trích dữ liệu từ HTTP request chuyển thành object và đưa vào các tham số của action method.
- 📖 Cho phép chuyển dữ liệu từ nhiều nguồn:
 - HTML Form
 - Giá trị route
 - Query string
 - Body của request
 - Header của request
 - Services

Model Binding

- VD: Chuyển dữ liệu từ Form ở View lên Controller.
- Tạo form create ở View dựa trên ViewModel Product

```
<form asp-action="Create">
  <div asp-validation-summary="ModelOnly" class="text-danger"></div>
  <div class="form-group">
    <label asp-for="ProductID" class="control-label"></label>
    <input asp-for="ProductID" class="form-control" />
    <span asp-validation-for="ProductID" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="Name" class="control-label"></label>
    <input asp-for="Name" class="form-control" />
    <span asp-validation-for="Name" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="Brand" class="control-label"></label>
    <input asp-for="Brand" class="form-control" />
    <span asp-validation-for="Brand" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="Price" class="control-label"></label>
    <input asp-for="Price" class="form-control" />
    <span asp-validation-for="Price" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="Date" class="control-label"></label>
    <input asp-for="Date" class="form-control" />
    <span asp-validation-for="Date" class="text-danger"></span>
  </div>
  <div class="form-group">
    <input type="submit" value="Create" class="btn btn-primary" />
  </div>
</form>
```

Model Binding




Action method Create ở Controller

[HttpPost]

0 references

```
public IActionResult Create(Product product)
{
    string message = "";
    if (ModelState.IsValid) message = $"Product {product.Name} from brand {product.Brand} created successfully";
    else message = "Create product failed";

    return Content(message);
}
```

-  Khi nhấn “Submit” ở form, các giá trị trong form sẽ được tạo ứng với tham số của Action method.
-  Để Model binding chuyển đổi đúng dữ liệu:
 - Tên thuộc tính ở form phải giống với thuộc tính Model View.
 - Thuộc tính Model View phải là public set.
-  Sử dụng thuộc tính **ModelState.IsValid** để kiểm tra việc chuyển dữ liệu vào Model.

Nội dung



Model Validation

Model Validation

- 📖 Dùng để kiểm tra dữ liệu nhập vào theo yêu cầu đặt ra.
- 📖 Sử dụng các attribute để kiểm tra dữ liệu trong model.
- 📖 2 cơ chế validation:
 - Phía client (client-side validation): phản hồi lỗi nhanh, sửa chữa ngay.
 - Phía server (server-side validation): thực hiện kiểm lỗi mà client không làm được, hoặc kiểm tra lần nữa trước khi thao tác lên CSDL.
- 📖 Trên Model: dùng các Data annotation.
- 📖 Trên Controller: kiểm lỗi phía server. Dùng các phương thức, thuộc tính của ModelState: `ModelState.IsValid`, `ModelState.AddModelError()`...
- 📖 Trên View: sử dụng Validation Tag Helper.

Model

Kiểm tra lỗi trên Model:

- Sử dụng Data Annotation. Thuộc thư viện `System.ComponentModel.DataAnnotations`
- Thêm các thuộc tính vào Data Annotation.

```
[Required(AllowEmptyStrings =false, ErrorMessage ="Không được trống")]
```

21 references

```
public string Brand { get; set; }
```

```
[Range(18, 60, ErrorMessage ="Tuổi từ 18 đến 60!")]
```

0 references

```
public int Age { get; set; }
```

Model

○ Các thuộc tính Data Annotation.

Data annotation	Mô tả
Key	Xác định khóa chính của model
Required	Bắt buộc phải nhập giá trị
DisplayName	Hiển thị tên của field (ở Label)
MaxLength	Số ký tự tối đa có thể nhập
MinLength	Số ký tự ít nhất phải nhập
Range	Miền giá trị của dữ liệu có thể nhập
Compare	Chỉ ra field khác để so sánh giá trị có giống nhau.
RegularExpression	Giá trị nhập phải tuân theo regular expression
DataType	Kiểm tra dữ liệu nhập có đúng kiểu dữ liệu đã định nghĩa. Có thêm các thuộc tính: <code>DataType.Password</code> , <code>DataType.Date</code>
EmailAddress	Định dạng Email
CreditCard	Định dạng số thẻ tín dụng.
StringLength	Độ dài tối đa của chuỗi nhập

Tham khảo: <https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.dataannotations?view=netcore-3.1>

Controller

- 📖 Kiểm tra lỗi trên Controller: trong action method
 - Dùng `ModelState.IsValid` để kiểm tra lỗi.
 - Sử dụng các phương thức của `ModelState` để hiển thị thông báo lỗi.
 - Thuộc lớp `ModelStateDictionary`

Thuộc tính/Phương thức	Mô tả
Keys	Danh sách lỗi theo key
IsValid	Kiểm tra có lỗi.
Values	Giá trị lỗi.
AddModelError(string key, string errorMessage)	Thêm một lỗi với key vào danh sách lỗi Errors
Clear()	Xóa tất cả các key và value trong ModelStateDictionary
ClearValidationState(string key)	Xóa phần tử trong ModelStateDictionary theo key

Tham khảo: <https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.modelbinding.modelstatedictionary?view=aspnetcore-3.1>

Controller

- VD: Sử dụng phương thức `AddModelError` để xuất thông báo theo định nghĩa của người dùng

```
string message = "";  
if (ModelState.IsValid)  
{  
    message = $"Product {product.Name} from brand {product.Brand} created successfully";  
    ModelState.AddModelError("", message);  
}  
else  
{  
    message = "Create product failed";  
    ModelState.AddModelError("", message);  
}  
return View();
```


View

📖 Sử dụng Validation Tag Helper để thông báo lỗi trên client, gồm:

- Validation Message Tag Helper:

- Sử dụng thẻ `` của HTML và thuộc tính **asp-validation-for** của tag helper.

```
<span asp-validation-for="Brand" class="text-danger"></span>
```

- Đoạn code HTML sẽ được tạo ra với thẻ `` và class **field-validation-valid**.

```
<span class="text-danger field-validation-valid" data-valmsg-for="Brand" data-valmsg-replace="true"></span>
```

- Nội dung thông báo lỗi dựa vào Data annotation của model view và tag helper của thẻ input

```
<label class="control-label" for="Brand">Brand</label>
```

```
<input class="form-control" type="text" data-val="true" data-val-required="Brand must input" id="Brand" name="Brand" value="" />
```

```
<span class="text-danger field-validation-valid" data-valmsg-for="Brand" data-valmsg-replace="true"></span>
```

View

○ Validation Summary Tag Helper:

- Sử dụng thẻ <div> của HTML để hiển thị các thông tin lỗi của toàn bộ form ở một vị trí xác định.
- Một thông báo lỗi được hiển thị với một danh sách.
- Sử dụng thuộc tính **asp-validation-summary** của tag helper cho thẻ <div> với các thuộc tính **All**, **ModelOnly** hoặc **None**.
- **All**: cho phép hiển thị tất cả các thông báo lỗi (Của Model và data annotation).

```
<div asp-validation-summary="All"></div>
```

Khi không có lỗi, code HTML sẽ được tạo ra:

```
<div class="validation-summary-valid" data-valmsg-summary="true"><ul><li style="display:none"></li></ul></div>
```

Khi có lỗi, code HTML sẽ được tạo ra

```
<div class="validation-summary-errors" data-valmsg-summary="true"><ul><li>Brand must input</li></ul></div>
```

View

- **ModelOnly**: cho phép hiển thị thông báo lỗi ở mức Model. Sử dụng khi đang hiển thị lỗi ở thuộc tính dùng **asp-validation-for** của tag helper

```
<div asp-validation-summary="ModelOnly"></div>
```

Thêm báo lỗi trực tiếp ở Model

```
message = "Create product failed";  
ModelState.AddModelError("", message);
```


Khi có lỗi, code HTML sẽ được tạo ra

```
<div class="validation-summary-errors"><ul><li>Create product failed</li>  
</ul></div>
```

- **None**: Không hiển thị bất cứ lỗi nào.

Unobtrusive Client Validation

- 📖 Client-side validation gồm nhiều cách:
 - Viết code javascript
 - Sử dụng HTML5 validation
 - Sử dụng các thư viện JQuery validation của ASP.NET Core.
- 📖 Unobtrusive Client Validation: sử dụng các thư viện JQuery `jquery.validate.unobtrusive.js` và `jquery.validate.js`.
- 📖 Thêm hai thư viện vào từng View, hoặc file `_ValidationScriptsPartial.cshtml` để sử dụng chung cho cả project.
- 📖 Các input tag helper sẽ được tạo ra với các thuộc tính bắt đầu bằng `data-val` sẽ được các `unobtrusive` kiểm tra và trả về cho client nếu bị lỗi.



Q & A

Giảng viên: Tạ Việt Phương
E-mail: phuongtv@uit.edu.vn