

XÂY DỰNG HTTP TRÊN CÁC FRAMEWORK

ASP Core MVC

Giảng viên: *Mai Xuân Hùng*
Mail: *hungmx@uit.edu.vn*

Nội dung



ASP.NET Core



ASP.NET Core MVC

Giới thiệu ASP.NET Core

- ❖ Là framework đa nền tảng (cross-platform) chạy trên nền tảng .NET Core
- ❖ Là thư viện chuẩn để xây dựng ứng dụng web.
- ❖ Phiên bản đầu tiên 06/2016.
- ❖ Được thiết kế để đáp ứng các yêu cầu:
 - Phát triển ứng dụng trên đa nền tảng.
 - Nâng cao hiệu suất
 - Kiến trúc dựa trên các module.
 - Là mã nguồn mở.
 - Phù hợp với xu hướng hiện đại của ứng dụng web
- ❖ Hoạt động được trên Windows, macOS và Linux.

Giới thiệu ASP.NET Core

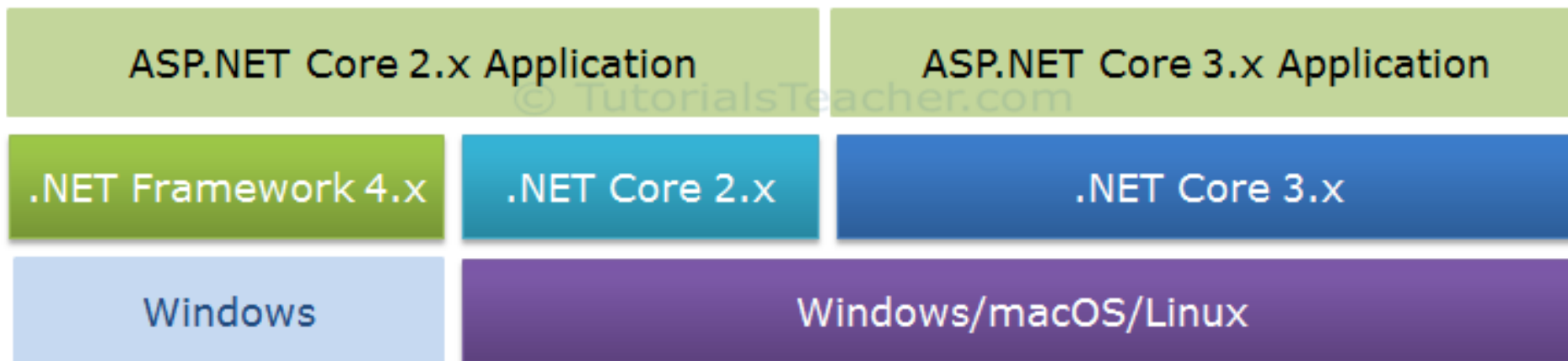
- ❖ Có thể triển khai ứng dụng viết trên ASP.NET Core như một web server độc lập hoặc kết hợp với các web server khác: IIS, Apache, Nginx.
- ❖ Có thể chạy trên được .NET Framework:
 - Hệ thống API của ASP.NET Core sử dụng các API cơ bản của .NET. Trong đó .NET Core và .NET Framework có chung hệ thống API cơ bản.
 - Cùng chung runtime thực thi cho ứng dụng, đều ở dạng mã trung gian IL (Intermediate Language).
 - ASP.NET Core phiên bản 2.0 tới 2.2 chạy trên được .NET Framework 4.6.1 hoặc cao hơn và trên .NET Core 2.0 hoặc cao hơn.
 - ASP.NET Core 3.0 chỉ chạy trên .NET Core 3.0

Giới thiệu ASP.NET Core

- ❖ Không còn dựa trên System.Web.dll, mà dựa trên các gói, các module (Nuget packages).
- ❖ Các class/thư viện của ASP.NET Core sử dụng chung cho các mô hình phát triển ứng dụng web: MVC, Web API, Razor Pages.
- ❖ Tích hợp liền mạch với các thư viện và framework phía client: Angular, React, Bootstrap...
- ❖ Các ứng dụng của ASP.NET Core:
 - Web HTML theo mô hình MVC hoặc Razor Pages.
 - REST API cho ứng dụng đơn trang (Single Page Application – SPA).
 - Dịch vụ gọi hàm từ xa: Remote Procedure Call – RPC.

- ❖ Các đặc điểm nổi bật của ASP.NET Core:
 - Dùng để xây dựng giao diện web (Web UI) và các API Web.
 - Tích hợp framework ở phía client.
 - Hệ thống cấu hình sẵn có trên đám mây.
 - Hiệu suất cao.
 - Có khả năng chạy trên nhiều host như IIS, Nginx, Apache, Docker hoặc self-host
 - Side-by-side App với .NET Core cho phép ứng dụng chạy đồng thời trên nhiều phiên bản.
 - Chạy trên đa nền tảng: Windows, MacOS, Linux.
 - Mã nguồn mở và có cộng đồng phát triển lớn.
 - Được cung cấp dưới dạng các gói NuGet. Khi xây dựng ứng dụng, chỉ cần cài đặt các gói cần thiết

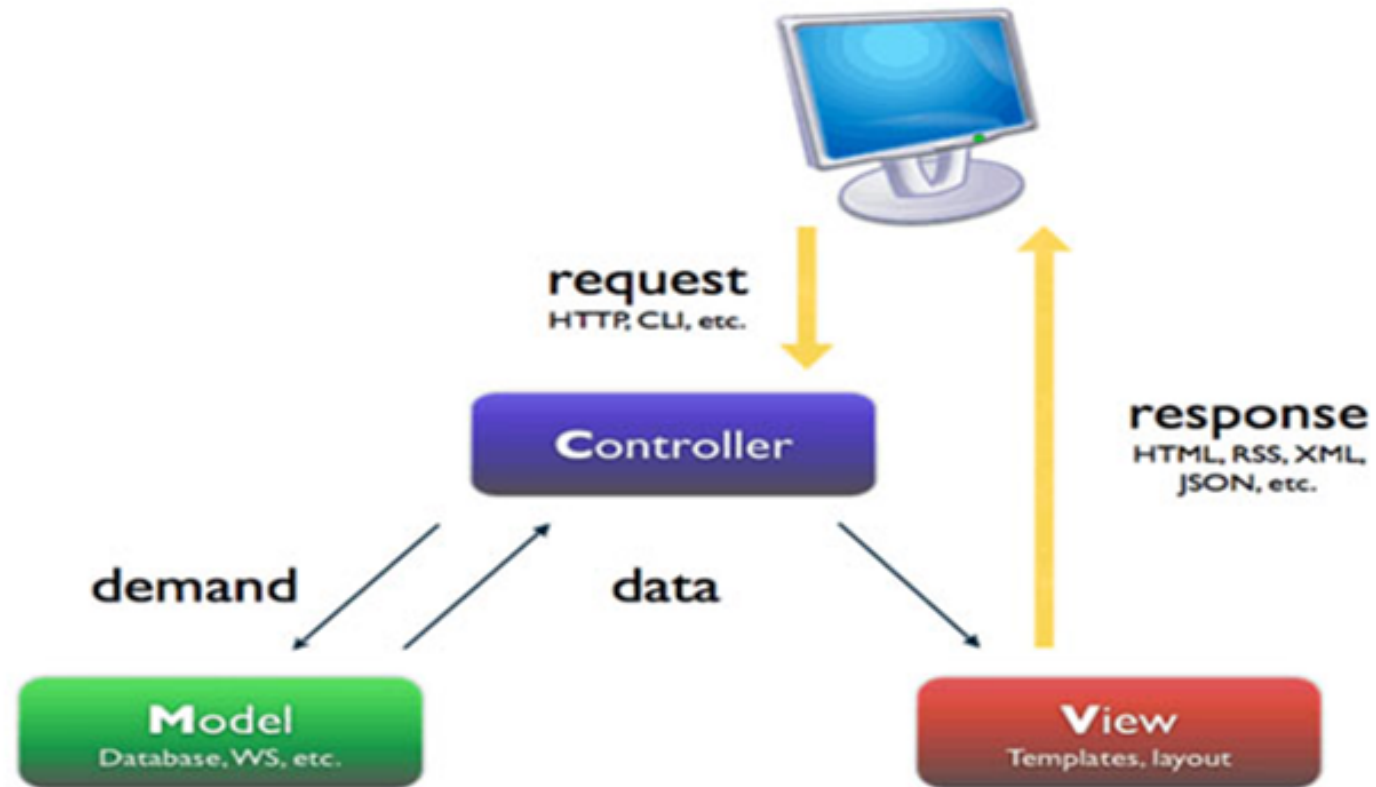
Giới thiệu ASP.NET Core



ASP.NET Core MVC

- ❖ Là framework cho việc xây dựng ứng dụng web.
- ❖ Sử dụng mô hình MVC để thiết kế và xây dựng web.
- ❖ MVC (Model – View - Controller) là mô hình tổ chức chương trình thành 3 thành phần khác nhau Model, View và Controller. Mỗi thành phần có một nhiệm vụ riêng biệt và độc lập với các thành phần khác.
 - **Model:** là thành phần chứa phương thức xử lý, truy xuất dữ liệu, hàm xử lý, chứa các đối tượng mô tả dữ liệu như các class và lưu trữ dữ liệu xuống hệ quản trị CSDL.
 - **View:** là thành phần nhận và hiển thị thông tin, tương tác với người dùng thông qua giao diện và gửi yêu cầu đến controller và nhận lại phản hồi từ controller.
 - **Controller:** là thành phần xử lý và điều hướng các hành động từ client; là thành phần kết nối giữa View và Model

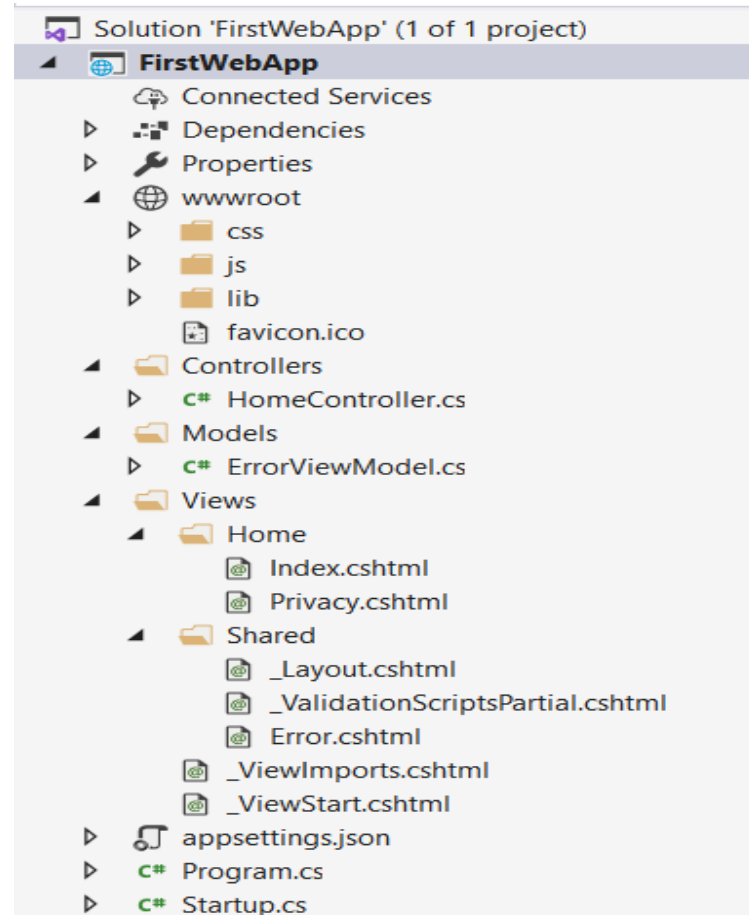
ASP.NET Core MVC



- ❖ Cung cấp các tính năng dựa trên mô hình xây dựng website động:
 - *Routing*: xác định URL và điều hướng thông tin tương ứng với request do người dùng đưa vào. Cấu hình Routing được lưu trữ trong `RoutTable`.
 - *Model binding*: dùng để chuyển đổi dữ liệu từ yêu cầu của client (form data, route data, query string, parameters, HTTP headers) thành đối tượng để controller xử lý. Tự ánh xạ dữ liệu từ HTTP Request tới tham số của method action.
 - *Model validation*: ràng buộc dữ liệu cho các thuộc tính trong model, thuộc tính sẽ được kiểm tra ở client trước khi được gửi về server.
 - *Dependency injection*: controller có thể gửi yêu cầu đến các service thông qua hàm khởi tạo của chúng.

- *Filters*: cho phép kích hoạt trước hoặc sau các action của controller. VD: Phân quyền
- *Areas*: cho phép chia các controller, view và model thành từng nhóm riêng biệt nhau. VD: nhóm các controller cho việc xử lý quản lý Administrator.
- *Web APIs*: hỗ trợ cho xây dựng các API.
- *Testability*: hỗ trợ việc kiểm tra ứng dụng được xây bằng framework sử dụng interfaces và dependency injection dễ dàng.
- *Razor view engine*: được dùng để render code theo ngôn ngữ Razor để tạo View
- *Tag Helpers*: cho phép code ở server side tham gia vào việc tạo và hiển thị các phần tử HTML.

❖ Cấu trúc thư mục project MVC trong Visual Studio



ASP.NET Core MVC

- ❖ **Connected Services**: danh sách các dịch vụ online mà project sử dụng
- ❖ **Dependencies**: danh sách gói thư viện NuGet được cài đặt và sử dụng trong project.
- ❖ **Properties**: cấu hình của project, nằm trong file json launchSetting.json, dùng để chạy ứng dụng từ VS hoặc .NET Core CLI.
- ❖ **appsettings.json**: thông tin cấu hình hoạt động của ứng dụng: connection string, biến môi trường, tham số dòng lệnh...
- ❖ **Program.cs**: file chứa class Program, là file cấu hình nền tảng (infrastructure) của ứng dụng, chứa entry point của ứng dụng và các cấu hình trong file hầu như không thay đổi trong mỗi project.

ASP.NET Core MVC

- ❖ **wwwroot**: chứa các file css, javascript, các thư viện, hình ảnh. Các file phải nằm trong thư mục này thì mới hoạt động. Gồm các thư mục:
 - css: chứa các file css.
 - js: chứa các file javascript
 - lib: các thư viện cho javascript như jQuery bootstrap
 - favicon.ico: file biểu tượng của site
- ❖ **Startup.cs**: chứa class với các phương thức cấu hình cho hoạt động của ứng dụng, vd: sử dụng middleware nào, thứ tự các middleware trong pipeline, các cấu hình sử dụng service, Dependency Injection, Logging.

ASP.NET Core MVC

- ❖ **Controllers**: chứa các file **.cs** của controller; tương ứng với mỗi controller sẽ có một thư mục view đi kèm.
- ❖ **Views**: chứa các file **.cshtml** của view; ứng với mỗi controller sẽ là một folder trong view. Các phương thức có `return View()` trong controller sẽ tương ứng với một file trong view.
 - Thư mục **Shared** là nơi chứa các thành phần giao diện chung cho trang web: `_Layout.cshtml`, `_ValidationScriptsPartial.cshtml` và `Error.cshtml`
- ❖ **Models**: chứa các file **.cs** của model; là các lớp thể hiện dữ liệu của ứng dụng.

ASP.NET Core MVC



❖ File Controller

```
HomeController.cs*  FirstWebApp
FirstWebApp.Controllers.HomeController
1  using ...
9
10 namespace FirstWebApp.Controllers
11 {
12     3 references
13     public class HomeController : Controller
14     {
15         private readonly ILogger<HomeController> _logger;
16
17         0 references
18         public HomeController(ILogger<HomeController> logger)
19         {
20             _logger = logger;
21         }
22
23         0 references
24         public IActionResult Index()
25         {
26             return View();
27         }
28
29         0 references
30         public IActionResult Privacy()
31         {
32             return View();
33         }
34
35         [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
36         0 references
37         public IActionResult Error()
38         {
39             return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
40         }
41     }
42 }
```


❖ File View

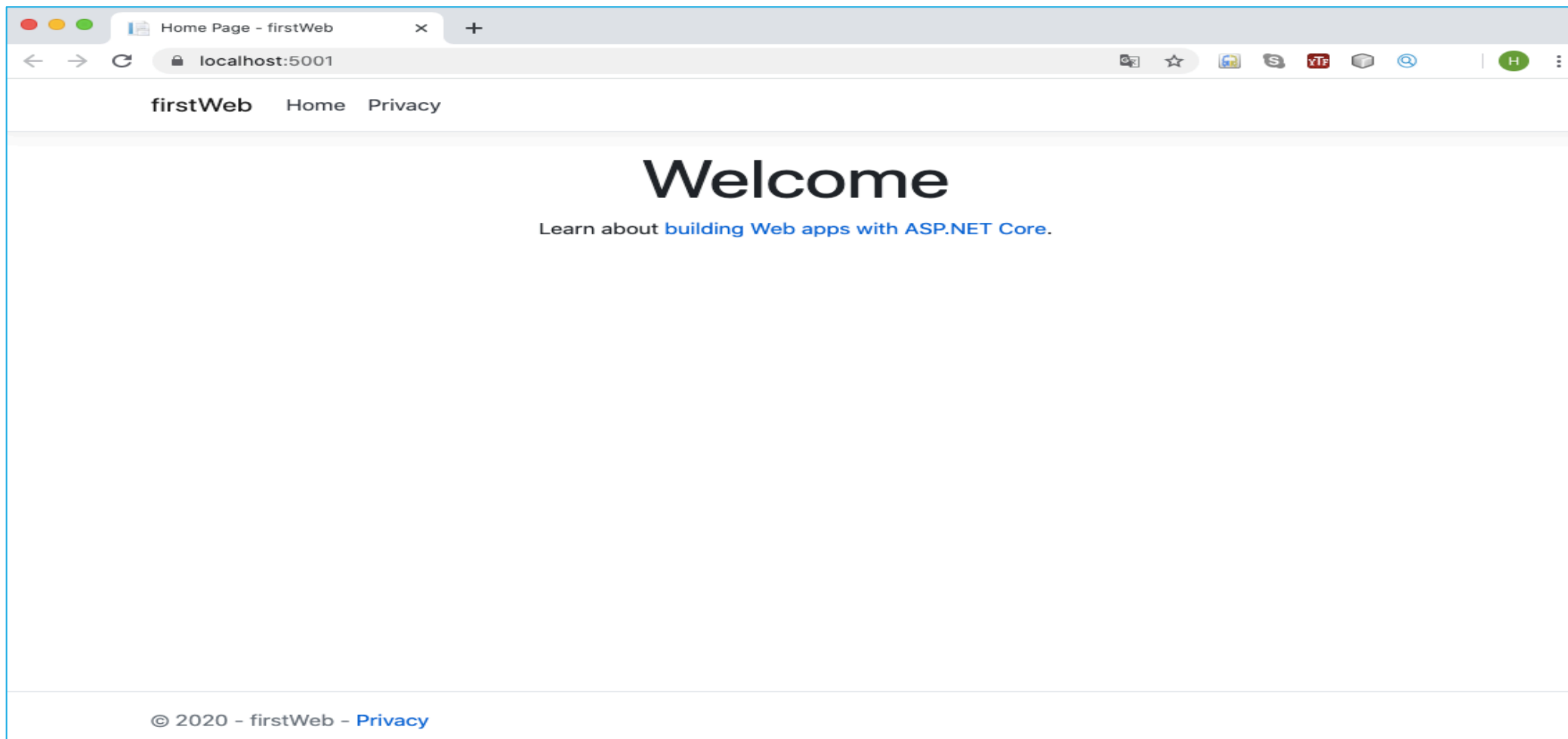
```
Index.cshtml*  ➦ ✕
1      @{
2          ViewData["Title"] = "Home Page";
3      }
4
5      <div class="text-center">
6          <h1 class="display-4">Welcome</h1>
7          <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">
8              building Web apps with ASP.NET Core</a>.</p>
9      </div>
```

```
Privacy.cshtml  ➦ ✕
1      @{
2          ViewData["Title"] = "Privacy Policy";
3      }
4      <h1>@ViewData["Title"]</h1>
5
6      <p>Use this page to detail your site's privacy policy.</p>
```

ASP.NET Core MVC



❖ Kết quả:



Cấu hình file Startup.cs

- ❖ MVC thực thi controller (và các phương thức của chúng) dựa vào đường dẫn URL.
- ❖ Đường dẫn URL xác định phương thức thực thi trong controller với dạng:
 - localhost:{Port}/[Controller]/[ActionName]/[Parameters]
- ❖ Cấu hình đường dẫn controller trong phương thức Configure của file **Startup.cs**

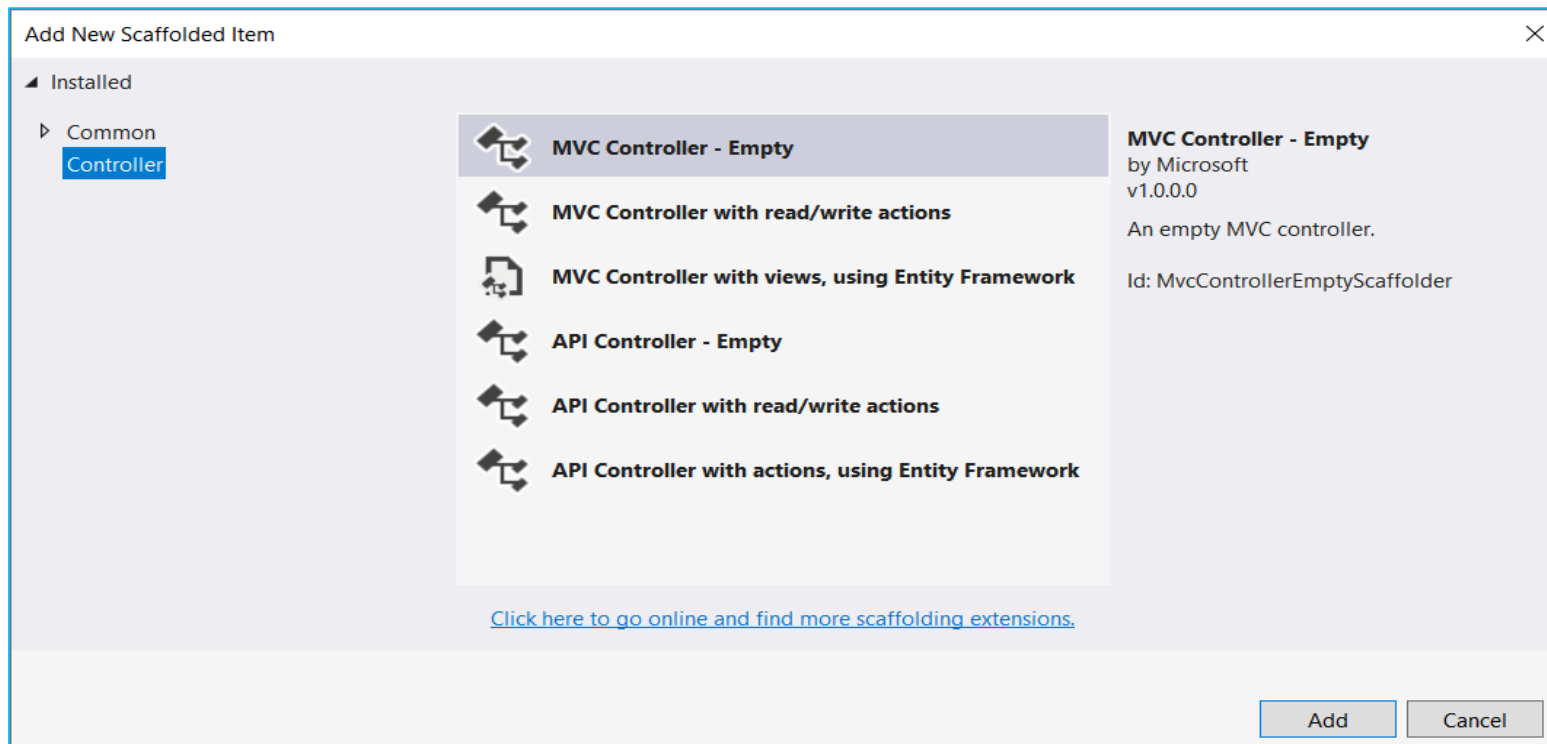
```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

- Mặc định là controller **Home** và phương thức **Index**. Tham số **id?** là tham số không bắt buộc phải xác định rõ. _____

Thêm controller



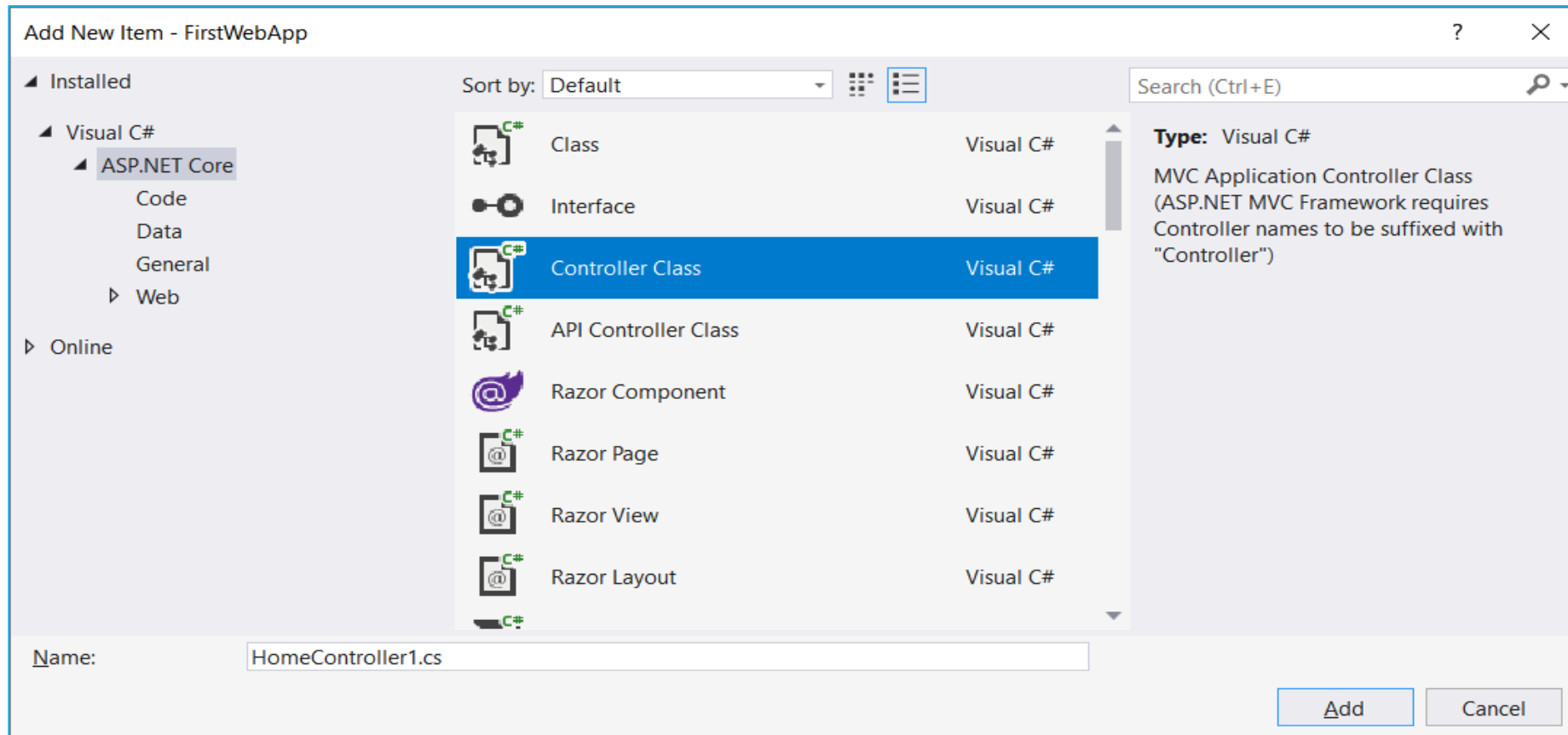
- ❖ Nhấp chuột phải tại thư mục Controllers > Add > Controller hoặc New Item...
 - Tại cửa sổ Add Scaffold chọn MVC Controller – Empty



Thêm controller



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



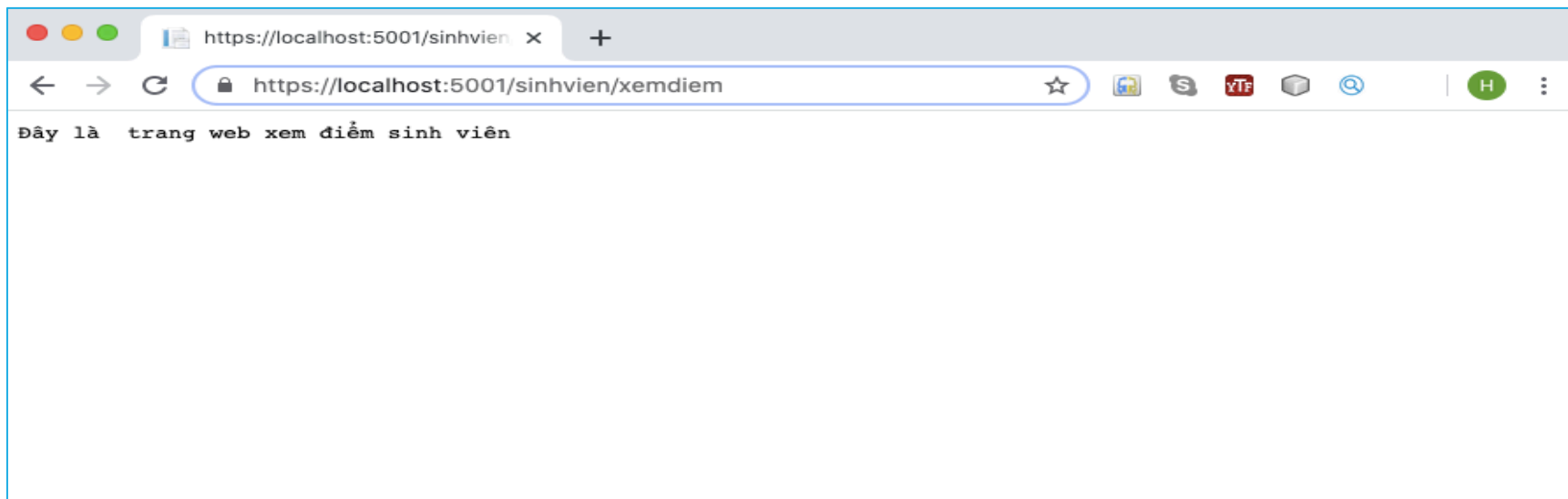
Thêm controller

- ❖ Thêm đoạn **code** sau vào **Controller** mới tạo

```
public class SinhVienController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
    public string XemDiem()
    {
        return "Đây là trang web xem điểm sinh viên";
    }
}
```

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:
- ❖ <https://localhost:5001/sinhvien/xemdiem>

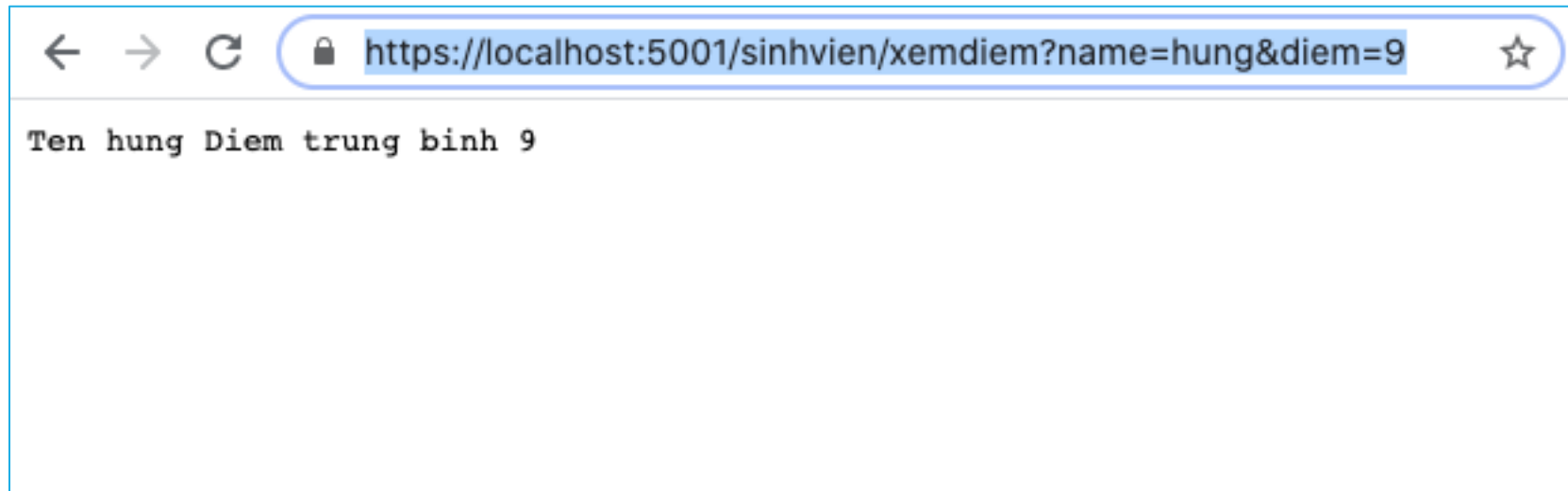


Thêm controller

❖ Controller với tham số truyền vào

```
public string XemDiem(string name, double diem=9)
{
    return HtmlEncoder.Default.Encode($"Ten {name} Diem trung binh {diem}");
}
```

<https://localhost:5001/sinhvien/xemdiem?name=hung&diem=9>



Thêm controller

❖ **Controller** với tham số truyền vào có tham số **ID**:

```
public string XemDiem(string name, double diem=9, int ID=1)
{
    return HtmlEncoder.Default.Encode($"ID {ID} Ten {name} Diem trung binh {diem}");
}
```

<https://localhost:5001/sinhvien/xemdiem/3?&name=hung&diem=9>



Thêm view

- ❖ Tạo **template view** sử dụng **Razor View**
- ❖ **Razor View Engine** là ngôn ngữ ngắn gọn, rõ ràng và hữu ích cho việc tạo ra giao diện của ASP.NET Core MVC.
- ❖ **Razor View template** có định dạng file **.cshtml**, cung cấp cách tạo ra **HTML** bằng **C#**
- ❖ Để gọi **view** tạo bằng **Razor**, phương thức trong controller (action method) phải gọi phương thức **View()** và trả về kiểu **ActionResult**.
- ❖ Nhấp chuột phải tại thư mục **Views** > Add > **New folder**: đặt **tên** thư mục theo tên **controller**
- ❖ Nhấp chuột phải tại **thư mục** vừa tạo Add > New Item > **Razor View**
- ❖ Trong **controller** có method trả về **ActionResult** thì phải có **View** đi kèm có tên **trùng** với **method** này

Thêm view

- ❖ Thêm **View** có tên **index** vào thư mục **SinhVien** và bổ sung code như hình

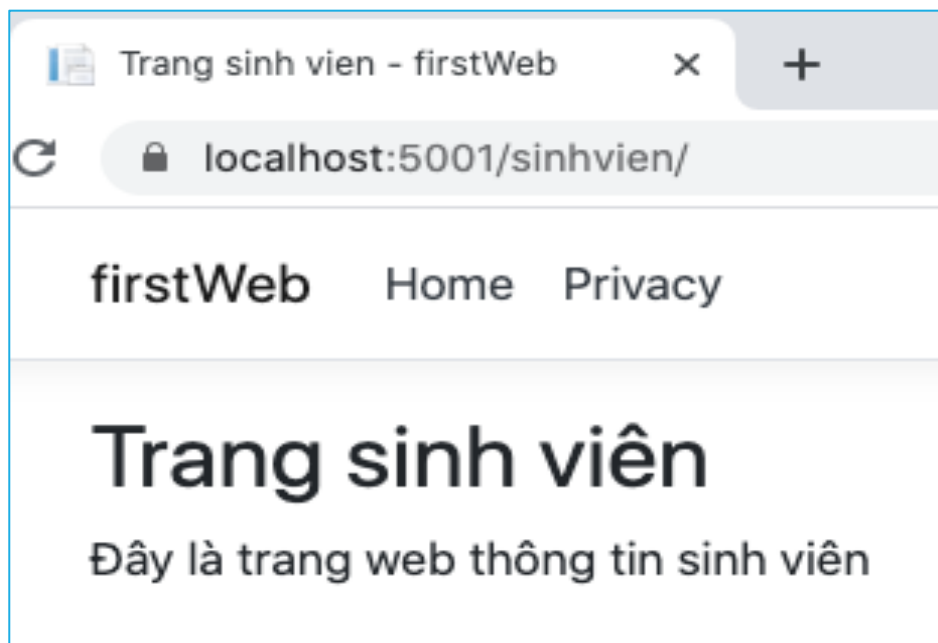
```
@{  
    ViewData["Title"] = "Trang sinh vien";  
}  
<h2> Trang sinh viên</h2>  
<p>Đây là trang web thông tin sinh viên</p>
```

- ❖ Chỉ sửa phương thức Index trong **controller**

```
public IActionResult Index()  
{  
    return View();  
}
```

Thêm view

❖ Kết quả:



Truyền dữ liệu từ controller sang View

- ❖ Lưu trữ và truyền dữ liệu từ **controller** tới **view** bằng **ViewData**, **ViewBag**
- ❖ **ViewData**, **ViewBag** là một kiểu đối tượng động (không cần định nghĩa trước kiểu dữ liệu) và là kiểu **dictionary** (lưu trữ theo dạng cặp key – value).
- ❖ Được **Razor Pages** tạo sẵn và có thể truy cập từ bất kỳ đâu trong ứng dụng Razor.
- ❖ Khi gọi phương thức **View()** trong **Controller** action **ViewData** sẽ tự động gán vào **view**.
- ❖ Lưu trữ theo kiểu string thì lấy và dùng trực tiếp ở view

```
ViewData["Name"] = "Test";
```

 - Truy xuất ở View: `@ViewData["Message"]`_____

Truyền dữ liệu từ controller sang View – ViewData

- ❖ Chỉnh sửa phương thức **XemDiem** trong **controller**

```
public IActionResult XemDiem(string name, double diem = 9)
{
    ViewData["name"] = name;
    ViewData["diem"] = diem;
    return View();
}
```

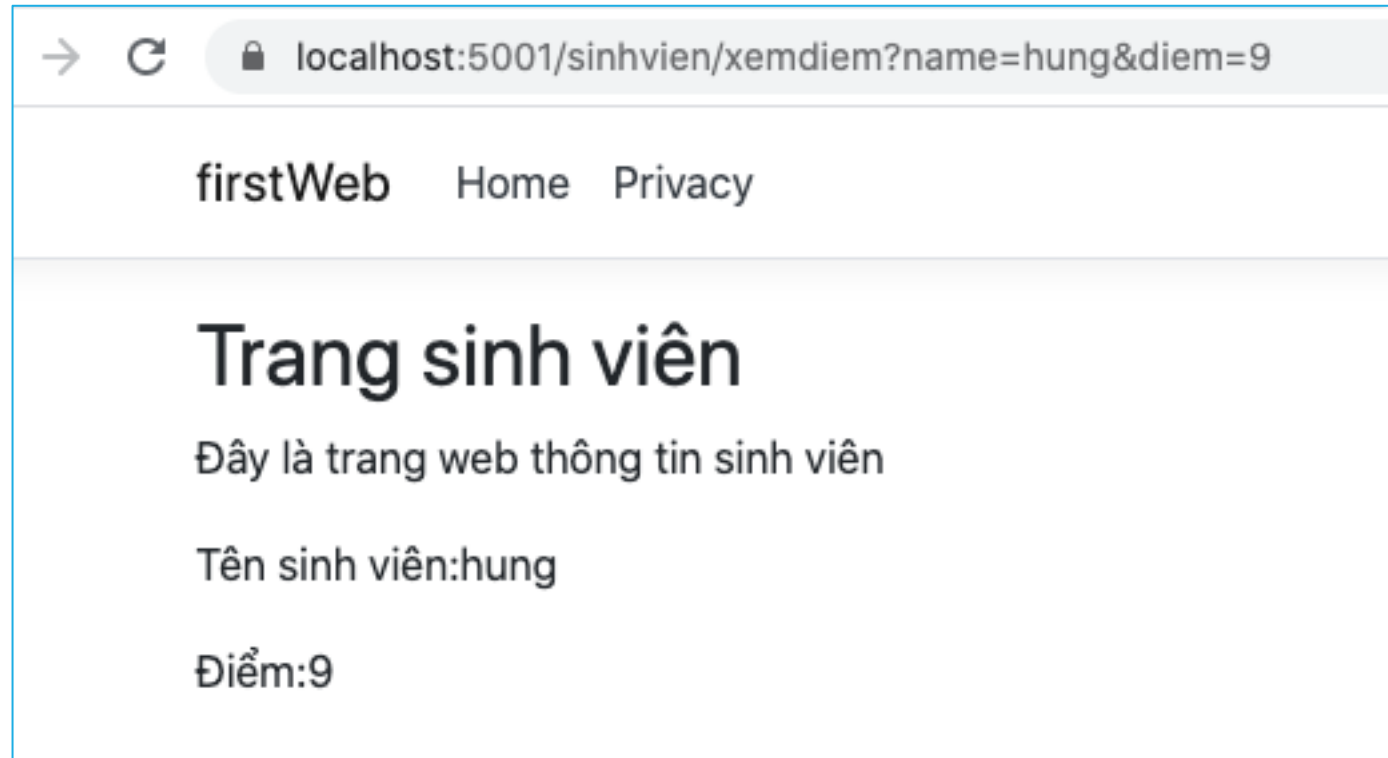
- ❖ Thêm View **XemDiem.cshtml** trong thư mục **View/SinhVien**

```
@{
    ViewData["Title"] = "Trang sinh vien";
}
<h2> Trang sinh viên</h2>
<p>Đây là trang web thông tin sinh viên</p>
<p>Tên sinh viên:@ViewData["name"]</p>
<p>Điểm:@ViewData["diem"]</p>
```

Truyền dữ liệu từ controller sang View – ViewData



❖ Kết quả:



Truyền dữ liệu từ controller sang View – ViewData

- ❖ Lưu trữ theo kiểu object thì phải ép kiểu sang kiểu dữ liệu xác định

```
ViewData["Product"] = new Product()
{
    ProductID = 1,
    Name = "ABC",
    Brand = "XYZ",
    Price = 1000
};
```

- Truy xuất ở View phải chuyển sang kiểu Product.

```
@{ var product = ViewData["Product"] as Product; }
@product.ProductID<br>
@product.Name<br>
@product.Brand<br>
@product.Price<br>
```


Truyền dữ liệu từ controller sang View – ViewData



❖ Method Index trong controller Khoa

```
public IActionResult Index()
{
    ViewData["Khoa"] = new Khoa("MK11", "Mạng máy tính");
    return View();
}
```

❖ View có tên file là Index.cshtml

```
@{
    Khoa khoa = ViewData["Khoa"] as Khoa;
    <p> Mã khoa: @khoa.MaKhoa</p>
    <p> Tên khoa: @khoa.TenKhoa</p>
}
```

Truyền dữ liệu từ controller sang View – ViewData



- ❖ Method Index trong controller Khoa, dùng thông qua `Model`

```
public IActionResult Index()
{
    ViewData.Model = new Khoa("MK11","Mạng máy tính");
    return View();
}
```

- ❖ View có tên file là `Index.cshtml`

```
@{
    Khoa khoa = Model;
    // Hoặc Khoa khoa = ViewData.Model
    <p> Mã khoa: @khoa.MaKhoa</p>
    <p> Tên khoa: @khoa.TenKhoa</p>
}
```

Truyền dữ liệu từ controller sang View – ViewData



- ❖ Method Index trong controller Khoa, dùng thông qua **Model** cách **Strongly Typed**

```
public IActionResult Index()
{
    ViewData.Model = new Khoa("MK11", "Mạng máy tính");
    return View();
}
```

- ❖ View có tên file là **Index.cshtml**

```
@model Khoa
@{
    <p> Mã khoa: @Model.MaKhoa</p>
    <p> Tên khoa: @Model.TenKhoa</p>
}
```

Truyền dữ liệu từ controller sang View – ViewBag



- ❖ Method Index trong controller Khoa, dùng thông qua **Model** cách **Strongly Typed**

```
public IActionResult Index()
{
    ViewBag.Khoa = new Khoa("MK10", "Hệ thống Thông tin");
    return View();
}
```

- ❖ View có tên file là Index.cshtml

```
@{
    Khoa kh = ViewBag.Khoa;
    <p>Mã khoa: @kh.MaKhoa</p>
    <p>Tên khoa: @kh.TenKhoa</p>
}
```

Razor Page: Layout

- ❖ Trang web thường chia làm năm phần: **header**, footer, **navigation menu**, **sidebar** và **content**. Trừ **content**, các phần còn lại thông thường được thống nhất chung trong nhiều trang được gọi là layout
- ❖ **Razor page** cung cấp cơ chế **Layout** để thực hiện việc này.
- ❖ Trong **ASP.NET Core MVC** là file **_Layout.cshtml** trong thư mục **View** -> **Shared**.
- ❖ Phương thức **@RenderBody()** là phương thức để xuất nội dung của bất kỳ **page** nào sử dụng **layout**.
- ❖ Sử dụng:

```
@{  
    Layout = "_Layout";  
}
```

Razor Page: Layout

```
_Layout.cshtml  X
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - FirstWebApp</title>
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
  <header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
      <div class="container">
        <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">FirstWebApp</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
          aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
          <ul class="navbar-nav flex-grow-1">
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
  </header>
  <div class="container">
    <main role="main" class="pb-3">
      @RenderBody()
    </main>
  </div>

  <footer class="border-top footer text-muted">
    <div class="container">
      &copy; 2020 - FirstWebApp - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </div>
  </footer>
  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
  <script src="~/js/site.js" asp-append-version="true"></script>
  @RenderSection("Scripts", required: false)
</body>
</html>
```

Razor Page: ViewStart

- ❖ Sử dụng **Layout** thì đầu mỗi trang đều phải khai báo **layout** cần sử dụng.
- ❖ Sử dụng **ViewStart** để đặt **layout** mặc định cho nhiều trang
- ❖ Trong **ASP.NET Core MVC** là file **_ViewStart.cshtml** trong thư mục **View**
- ❖ **_ViewStart.cshtml** có tác dụng trong thư mục chứa nó và các **thư mục con**. Nếu thư mục con cũng có file **_ViewStart** thì các file trong **thư mục con** sẽ chịu tác động của file **_ViewStart** này

```
_ViewStart.cshtml  ➦ X
1      @{
2          Layout = "_Layout";
3      }
```

Razor View Engine

- ❖ **View Engine**: kết hợp ngôn ngữ lập trình với ngôn ngữ HTML để xuất ra mã HTML:
 - Hoạt động như một chương trình dịch (compiler hoặc interpreter).
 - Khi **Controller** gọi **Action Method** và kiểu trả về là một **Action Result**; cụ thể là **ViewResult**.
 - **ViewResult** được cung cấp bởi **View Engine** sẽ sinh ra **code HTML**.
- ❖ **Razor View Engine**: là View Engine mặc định của ASP.NET Core; lấy mã Razor trong file view chuyển thành HTML response.

Razor

- ❖ **Razor** là loại ngôn ngữ/cú pháp đánh dấu (markup/language syntax) sử dụng **Razor View Engine** của ASP.NET Core
- ❖ cú pháp **Razor** kết hợp **C#** và **HTML** với nhiệm vụ xác định (đánh dấu) **code C#** và **code HTML** trong file để **view engine** xử lý.
- ❖ Tạo ra các trang web động bằng HTML.
- ❖ **Gần** tương tự với ngôn ngữ **ASP** hoặc **PHP**.
- ❖ Là ngôn ngữ cho các **framework** trong **ASP.NET Core**: Razor Pages, MVC và Blazor.
- ❖ **File Razor** có đuôi **.cshtml**.
- ❖ Khi duyệt file **.cshtml**, mã **C#** sẽ được thực thi để tạo dữ liệu **HTML** và gửi về client.

Cú pháp Razor

- ❖ Trong file .cshtml (razor file/page) tồn tại song song hai ngôn ngữ C# và HTML. Vì vậy, cú pháp Razor có nhiệm vụ xác định cụ thể code C# và HTML để View Engine dịch và xử lý.
- ❖ Razor sử dụng ký tự @ để chuyển cách viết từ HTML sang C#. Có 2 cách khai báo:

- *Razor expression*: bắt đầu bằng @ và theo sau là code C#, chèn chung với code HTML

```
<h4>@DateTime.Now.Year</h4>
```

- *Khối lệnh Razor*: bắt đầu bằng @ và nằm trong cặp dấu {}; có thể được dùng để thao tác Model, khai báo biến, đặt thuộc tính của View, không nên sử dụng cho việc xử lý logic.

```
@{
```

```
    var message = "Welcome";
```

```
    var weekDay = DateTime.Now.DayOfWeek;
```

```
}
```

Một số lưu ý của Razor

- ❖ Ngôn ngữ mặc định trong file `cshtml` là `HTML`.
- ❖ Không có đánh dấu đặc biệt thì `Razor View Engine` sẽ xem đây hoàn toàn là `HTML` và gửi khối `HTML` về `client`
- ❖ Ký tự `@` để Razor xác định đây là `code C#` và sẽ chuyển sang chế độ dịch `C#`
- ❖ Kết quả biên dịch cuối cùng của `file cshtml` là mã `HTML`
- ❖ Viết 2 lần ký tự `@` (`@@` - `@` escape) để biểu diễn cho ký tự `@` trong Razor.
- ❖ Razor tự phân biệt được `@` trong `email`
- ❖ Cách ghi chú thích: vùng code `C#` chú thích kiểu `C#`, vùng code `HTML` chú thích kiểu `HTML`.
- ❖ Ký tự `@` đi cùng một số từ đặc biệt: `@page`, `@model`, `@using` (gọi là directive) được `Razor` sử dụng riêng

Biểu thức Razor

- ❖ Biểu thức **Razor** (Razor expression): bắt đầu bằng @ và theo sau là biểu thức của **C#**, chèn chung với code **HTML**
- ❖ Được dùng để chèn giá trị (tính bằng C#) vào vị trí tương ứng với HTML.
- ❖ VD: `<h4>@message</h4>`
- ❖ Biểu thức C# (expression) là những lệnh có trả về kết quả: phương thức, kết quả thực thi các phép toán, giá trị của biến; phương thức không trả về kết quả (câu lệnh – statement) không được dùng làm biểu thức Razor.
- ❖ Có 2 dạng biểu thức Razor:
 - Biểu thức ẩn (Implicit expression)
 - Biểu thức rõ (Explicit expression)

Biểu thức Razor

❖ Biểu thức ẩn (Implicit expression):

- Không cho phép chứa dấu cách (khoảng trống) hoặc ký tự đặc biệt (nhầm lẫn với code HTML xung quanh). VD: không thể chứa kiểu generics.
- Không thể là các phép toán thông thường (+, -, *, /)
- Là dạng đơn giản hóa, tiện lợi khi tích hợp biểu thức C# với HTML
- VD:

```
<p>Hello @name</p>
```

```
<p>@DateTime.Now.DayOfWeek</p>
```

```
<p>Name : @cust.name</p>
```

```
<p>Address : @cust.address</p>
```

Biểu thức Razor

❖ Biểu thức rõ (Explicit expression):

- Đặt trong cặp dấu ()
- Là dạng đầy đủ khi tích hợp biểu thức C# và HTML
- VD:

`<p>ISBN: @(No)</p>`

`<p>1 + 1: @(1 + 1)</p>`

`<p>Diện tích = @(Math.PI * r * r)</p>`

Khối lệnh Razor

❖ Khối lệnh (khối code – Razor code block): là những câu lệnh C# nằm trong cặp dấu {} và bắt đầu bằng ký tự @.

```
@{  
    var message = "Welcome";  
    int Cong(int a, int b)  
    {  
        return a + b;  
    }  
}
```

❖ Đặc điểm:

- Được dùng để khai báo, khởi tạo biến và khai báo hàm cục bộ.
- Các biến và hàm tạo trong khối lệnh có thể được dùng làm biểu thức Razor hoặc trong khối lệnh khác trong cùng trang.
- Không giới hạn khối lệnh và vị trí viết trên một trang.
- Hàm có thể được gọi trong khối lệnh khác trước khi được khai báo.
- Biến phải được khai báo trước khi sử dụng.
- Chú thích khối lệnh dùng cặp: @*.....*@

Khối lệnh Razor

- ❖ Trong khối lệnh **Razor** có thể có **code HTML**. Razor sẽ tự động chuyển đổi từ **C#** sang **HTML**

```
@{  
    void XuấtTen(string name)  
    {  
        <div>@name </div>  
    }  
}  
@{  
    XuấtTen("Mai Hung");  
}
```


Khối lệnh Razor

- ❖ Chuyển đổi chủ động: trong trường hợp không muốn sử dụng thẻ HTML trong khối lệnh Razor.
 - Do không sử dụng code HTML trong hàm, nên Razor sẽ không xuất ra HTML được.
 - Dùng “@:” để xuất giá trị trên một dòng.

```
void WelcomeMessage(string name)
{
    @:Hello @name.
    @:Welcome to Razor
}
```

Khối lệnh Razor

- Dùng thẻ `<text></text>` để bao quanh khối giá trị cần xuất trên nhiều dòng.

```
void WelcomeMessage(string name)
{
    <text>Hello @name.
    Welcome to Razor</text>
}
```

Cấu trúc điều khiển trong Razor

- ❖ Là những đoạn khối lệnh dùng để điều chỉnh việc phát sinh mã HTML như phân nhánh, lặp....
- ❖ Trong khối lệnh (C# là ngôn ngữ mặc định) thì viết câu lệnh theo cấu trúc điều khiển C# bình thường.
- ❖ Ngoài khối lệnh (HTML là ngôn ngữ mặc định) thì sử dụng kiểu viết markup của Razor.
- ❖ Các markup của Razor:
 - Cấu trúc điều kiện: @if - else if – else, @switch
 - Cấu trúc lặp: @for, @foreach, @while và @do...while
 - Cấu trúc @using
 - Cấu trúc @try – catch - finally

Cấu trúc điều kiện



❖ Cấu trúc điều kiện: **@if - else if – else:**

- Cấu trúc @if - else if – else là một khối lệnh.
- Chỉ cần viết @ ở trước if.
- Điều kiện logic viết theo ngôn ngữ C#

```
@{  
    int b = 2;  
    int c = 1;  
}  
@if (b == 0)  
{  
    if (c == 0)  
    {  
        <div>Phuong trinh vô số nghiệm</div>  
    }  
    else  
    {  
        <div>Phuong trinh vô nghiệm</div>  
    }  
}  
else  
{  
    double x = -(double)c / b;  
    <div>Phuong trinh có nghiệm @x</div>  
}
```

Cấu trúc điều kiện

❖ Cấu trúc điều kiện: @switch

- Đặt ký tự @ vào trước cấu trúc switch của C#
- Mỗi case trong switch là một khối lệnh, có thể kết hợp C# và HTML

❖ VD:

```
@{  
    int thang = 5;  
    int nam = 2020;  
}  
@switch (thang)  
{  
    case 1:case 3:case 5:case 7:case 8:case 10:  
    case 12:<div>Tháng có 31 ngày</div>  
        break;  
    case 2:
```

```
if (nam % 4 == 0)  
{  
    <div>Tháng có 29 ngày</div>  
}  
else{  
    <div>Tháng có 28 ngày</div>  
}  
break;  
case 4:case 6:case 9:case 11:  
    <div>Tháng có 30 ngày</div>break;  
default:  
    <div>Tháng sai</div>break;  
}
```

❖ Cấu trúc lặp: **@for, @foreach**

- Đặt ký tự @ vào trước cấu trúc for hoặc foreach của C#
- Bên trong for là khối lệnh, có thể kết hợp C# và HTML

❖ VD:

```
@for (int i = 0; i < 5; i++)  
{  
    <span> @i </span>  
}
```

```
@{  
    var custList = new []{  
        "ABC", "XYZ"  
    };  
}  
  
@foreach (var cust in custList)  
{  
    <h4>cust</h4>  
}
```

Cấu trúc lặp

❖ Cấu trúc lặp: @while, @do...while

- Đặt ký tự @ vào trước cấu trúc while hoặc do...while của C#
- Bên trong for là khối lệnh, có thể kết hợp C# và HTML

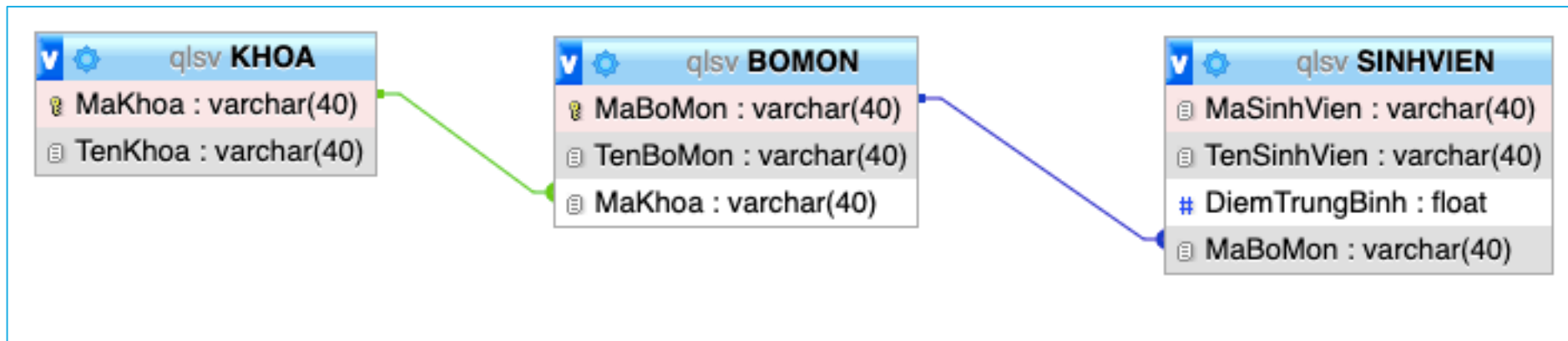
❖ VD:

```
@{ var i = 0; }  
@while (i < 5){  
    <p>@i</p>  
    i++;  
}
```

```
@{ var i = 0; }  
@do  
{  
    var cust = custList[i++];  
    <h4>@cust</h4>  
}  
while (i < custList.Length);
```

Xây dựng HTML Form

- ❖ Xét CSDL quản lý sinh viên của khoa có sơ đồ CSDL như sau:



- ❖ Ta tiến hành xây dựng Form thêm khoa có giao diện như sau:

Thêm khoa

Mã khoa

Tên khoa

Xây dựng HTML Form



Bước 1: Tạo mới project

Bước 2: Thêm `services.AddMvc()` trong phương thức `ConfigureServices` ở file `startup.cs`

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddMvc();
}
```

Xây dựng HTML Form



Bước 1: Tạo mới project

Bước 2: Thêm `services.AddMvc()` trong phương thức `ConfigureServices` ở file `startup.cs`

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddMvc();
}
```

Bước 3: Trong thư mục Models tạo 1 class có tên Khoa

```
public class Khoa
{
    private string maKhoa;
    private string tenKhoa;
    public string MaKhoa{
        get { return maKhoa; }
        set { maKhoa = value; }
    }
}
```

```
public string TenKhoa {
    get { return tenKhoa; }
    set { tenKhoa = value; }
}
```

Xây dựng HTML Form



Bước 4: Tạo Controller Khoa và thêm phương thức EnterKhoa

```
public IActionResult EnterKhoa() {  
    return View();  
}
```

Bước 5: Tạo thư mục Khoa trong thư mục Views và thêm một file có tên EnterKhoa.cshtml

```
@{  
    ViewData["Title"] = "Thêm khoa";  
    <h2>Thêm khoa</h2>  
    <form action="/Khoa/InsertKhoa" method="post">  
        <label for="maKhoa">Mã khoa</label> <input type="text" name="MaKhoa"/><br>  
        <label for="tenKhoa">Tên khoa</label> <input type="text" name="TenKhoa"/><br>  
        <input type="submit" name="submit" value="Thêm" />  
    </form>  
}
```

Xây dựng HTML Form



Bước 6: Lấy dữ liệu khi nhấn nút **Submit** trên Form, thuộc tính action trên form là `action="/Khoa/InsertKhoa"`. Tức khi ta nhấn nút Submit trên form thì phương thức `InsertKhoa` của controller `Khoa` được thi hành

```
public IActionResult InsertKhoa(Khoa kh)
{
    string message = "";
    if (ModelState.IsValid)
    {
        message = "Mã khoa:" + kh.MaKhoa + " Tên khoa" + kh.TenKhoa + " created successfully";
    }
    else
    {
        message = "Failed to create the khoa. Please try again";
    }
    return Content(message);
}
```

Tag Helper

- ❖ Tag Helpers là tính năng mới của ASP.NET Core, nó giúp chúng ta thêm code phía server vào HTML dễ dàng
- ❖ Tag Helper giúp chúng ta viết phần tử HTML trong Razor sử dụng cú pháp thân thiện với HTML. Nó nhìn như là HTML chuẩn vậy nhưng code được xử lý bởi Razor Engine trên server và nó tận dụng được các ưu điểm của việc xử lý phía server.
- ❖ Tag Helper sẽ giúp tạo ra view HTML đơn giản hơn dựa trên dữ liệu lấy từ Model gắn vào nó
- ❖ Ví dụ dùng tagHelper

```
<form asp-action="create" asp-controller="home">
```

- Tương đương với

```
<form action="/home/create" method="post">
```

Tag Helper

- ❖ Để sử dụng **Tag Helper** ta thêm **@addTagHelper ***, **Microsoft.AspNetCore.Mvc.TagHelpers** vào View muốn sử dụng tagHelper
- ❖ Thêm Tag Helper toàn bộ các View: Thêm @addTagHelper vào **_ViewImports.cshtml**
- ❖ Bỏ Tag Helper: @removeTagHelper "*, Microsoft.AspNetCore.Mvc.TagHelpers"
- ❖ **Form Tag Helper**

`<form asp-controller="Home" asp-action="Create">`

- ❖ **Label Tag Helper**

`<label asp-for="@Model.Name"></label>`

`<label asp-for="Name"></label>`

- **Tương đương với:** `<label for="Name">Name</label>`

Tag Helper

❖ Input Tag Helper :

- `<input asp-for="Name" />`
- Tương đương với: `<input type="text" id="Name" name="Name" value="" />`
- Thuộc tính type, id và name tự động lấy từ tên và kiểu dữ liệu của trường đó trong Model

❖ Ví dụ:

```
<form asp-controller="Home" asp-action="Create">
  <label asp-for="Name"></label>
  <input asp-for="Name" />
  <label asp-for="Rate"></label>
  <input asp-for="Rate" />
  <label asp-for="Rating"></label>
  <input asp-for="Rating" />
  <input type="submit" name="submit" />
</form>
```

Truyền dữ liệu bằng phương thức Session



❖ Cấu hình file “Startup.cs”

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDistributedMemoryCache();
    services.AddSession(options => {
        //thời gian tồn tại session
        options.IdleTimeout = TimeSpan.FromMinutes(1);
    });
    services.AddMvc();
}
```


Truyền dữ liệu bằng phương thức Session



❖ Thêm dòng `app.UseSession()` vào phương thức `Configure`

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();

    app.UseSession();

    ...
}
```

Truyền dữ liệu bằng phương thức Session



❖ Khai báo biến Session

```
string SessionName = "_Name";
```

```
string SessionAge = "_Age";
```

❖ Ghi dữ liệu lên biến Session

```
HttpContext.Session.SetString(SessionName, "Admin");
```

```
HttpContext.Session.SetInt32(SessionAge, 24);
```

❖ Lấy dữ liệu từ biến Session

```
HttpContext.Session.GetString(SessionName);
```

```
HttpContext.Session.GetInt32(SessionAge);
```

➤ *Project name : code/netcore/MySession*

Truyền dữ liệu bằng phương thức Cookies



- ❖ Tạo đối tượng CookieOptions `option = new CookieOptions()` để đặt thời gian tồn tại của Cookies
- ❖ Ghi dữ liệu vào Cookies: `Response.Cookies.Append(tên biến, value, option);`
- ❖ Lấy dữ liệu của biến lưu trên Cookies: `Request.Cookies["Tên biến cookies"];`
- ❖ Xóa 1 biến lưu trên Cookies: `Response.Cookies.Delete(tên biến cookies);`

```
public void Set(string key, string value, int? expireTime)
{
    CookieOptions option = new CookieOptions();
    if (expireTime.HasValue)
        option.Expires = DateTime.Now.AddMinutes(expireTime.Value);
    else
        option.Expires = DateTime.Now.AddMilliseconds(10);
    Response.Cookies.Append(key, value, option);
}
```

Project name : code/netcore/MyCookies

Kết nối cơ sở dữ liệu

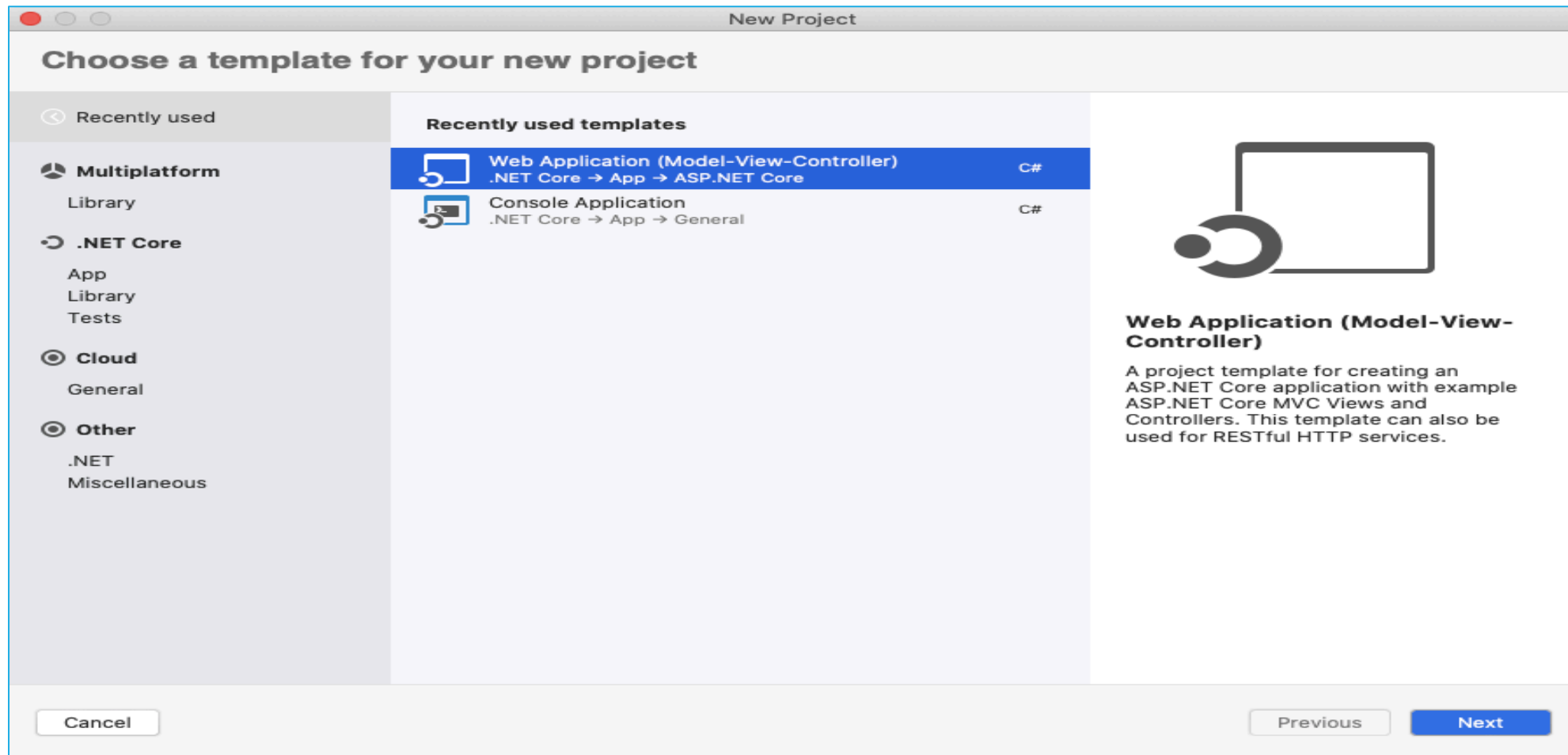


- ❖ **Bước 1:** Tạo project
- ❖ **Bước 2:** Dùng công cụ NuGet để add thư viện MySql.Data
- ❖ **Bước 3:** Xây dựng lớp Model tương ứng với quan hệ trong CSDL và lớp context cho CSDL
- ❖ **Bước 4:** Thiết lập các thông số kết nối trong file *appsettings.json*
- ❖ **Bước 5:** Thêm dịch vụ kết nối ở phương thức ConfigureServices trong file *Startup.cs*
- ❖ **Bước 6:** Lấy connection và mở kết nối
- ❖ **Bước 7:** Chuẩn bị câu lệnh truy vấn
- ❖ **Bước 8:** Tạo SqlCommand và thực thi câu lệnh truy vấn
- ❖ **Bước 9:** Xử lý kết quả trả về
- ❖ **Bước 10:** Đóng kết nối

Kết nối cơ sở dữ liệu – bước 1



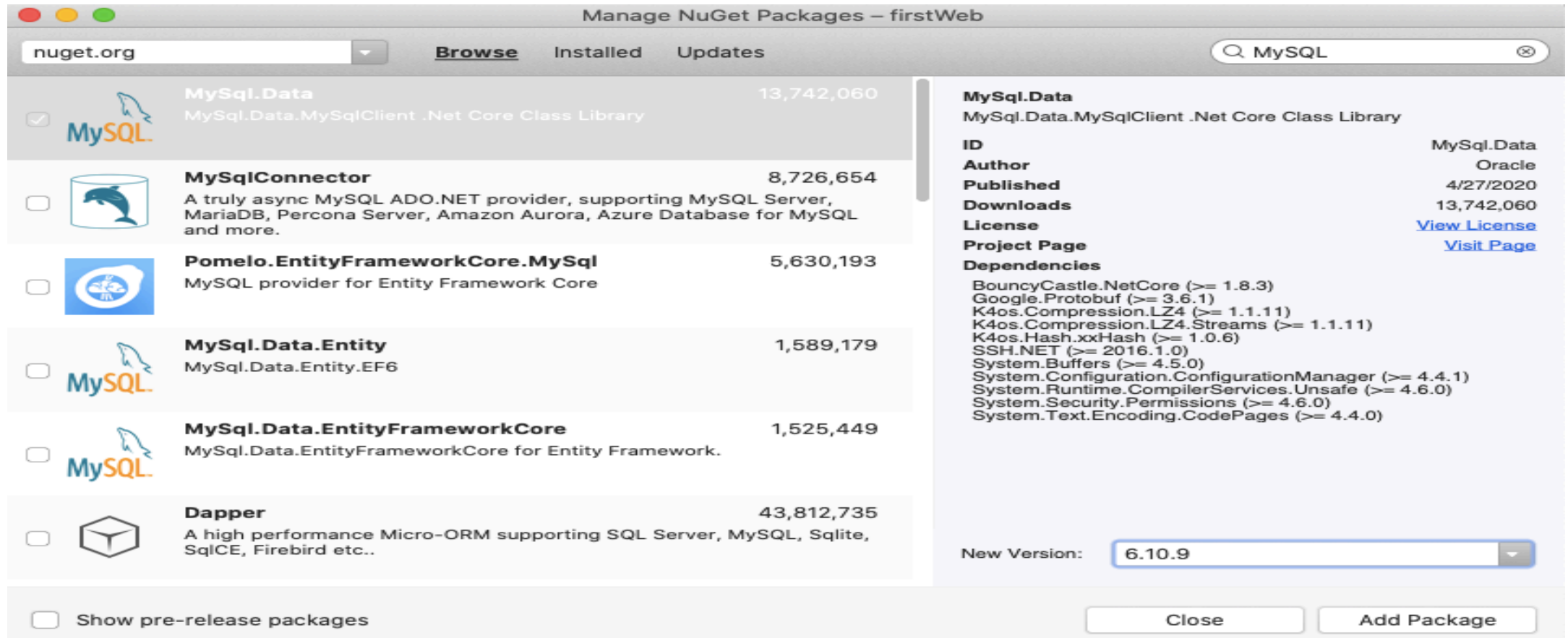
❖ Bước 1: Tạo project



Kết nối cơ sở dữ liệu – bước 2




❖ Bước 2: Dùng công cụ NuGet để add thư viện MySql.Data



Kết nối cơ sở dữ liệu – bước 3



- ❖ Xây dựng lớp Model tương ứng với quan hệ trong CSDL
- ❖ Ví dụ ta xây dựng lớp Model tương ứng với quan hệ Khoa trong CSDL quản lý sinh viên trong Khoa. Class “*Khoa*” lưu với tên file “*Khoa.cs*”

Name	Type	Collation
MaKhoa 	varchar(40)	utf8_unicode_ci
TenKhoa	varchar(40)	utf8_unicode_ci

```
public class Khoa
{
    private string maKhoa;
    private string tenKhoa;
    public string MaKhoa{
        get { return maKhoa; }
        set { maKhoa = value; }
    }
    public string TenKhoa {
        get { return tenKhoa; }
        set { tenKhoa = value; }
    }
}
```

Kết nối cơ sở dữ liệu –bước 3



- ❖ Xây dựng lớp Context cho CSDL có tên *StoreContext*, và lưu trong file “*StoreContext.cs*”

```
public class StoreContext
{
    public string ConnectionString { get; set; } //biết thành viên

    public StoreContext(string connectionString) //phuong thuc khoi tao
    {
        this.ConnectionString = connectionString;
    }

    private MySqlConnection GetConnection() //lấy connection
    {
        return new MySqlConnection(ConnectionString);
    }
}
```


Kết nối cơ sở dữ liệu –bước 4



- ❖ Thiết lập các thông số kết nối trong file *appsettings.json*

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": "server=127.0.0.1;user id=root; password=; port=3306;
database=qlsv;"
  }
}
```

Kết nối cơ sở dữ liệu – bước 5



- ❖ Thêm dịch vụ kết nối ở phương thức `ConfigureServices` trong file `Startup.cs`

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddMvc();
    services.Add(new ServiceDescriptor(typeof(StoreContext),
        new StoreContext(Configuration.GetConnectionString("DefaultConnection"))));
}
```

Kết nối cơ sở dữ liệu – bước 6



❖ Lấy connection và mở kết nối

```
public List<Khoa> GetKhoas()
{
    List<Khoa> list = new List<Khoa>();
    using (MySqlConnection conn = GetConnection())
    {
        conn.Open();
    }
}
```

Kết nối cơ sở dữ liệu – bước 7

- ❖ Chuẩn bị câu lệnh truy vấn
- ❖ `str = "Insert into Khoa values(...)"`
- ❖ `str = "select * from KHOA"`
- ❖ `str = "delete from Khoa where..."`
- ❖ `str = "update Khoa set MaKhoa =@makhoa where ..."`

Kết nối cơ sở dữ liệu – bước 8



- ❖ Tạo SqlCommand và thực thi câu lệnh truy vấn
 - MySqlCommand `cmd` = new `MySqlCommand`(str, conn);
 - cmd.`ExecuteReader`(): Dùng cho câu lệnh SQL **Select**
 - cmd.`ExecuteNonQuery`(): Dùng cho câu lệnh SQL **Insert**
 - cmd.`ExecuteNonQuery`(): Dùng cho câu lệnh SQL **Update**
 - cmd.`ExecuteNonQuery`(): Dùng cho câu lệnh SQL **Delete**
 - cmd.`ExecuteScalar`(): thực thi các câu truy vấn có hàm tính toán (**Select** COUNT|**MIN**|**MAX**|**AVG**)

Kết nối cơ sở dữ liệu – bước 9 – select



❖ Xử lý kết quả trả về

```
List<Khoa> list = new List<Khoa>();
using (SqlConnection conn = GetConnection())
{
    conn.Open();
    string str = "select * from KHOA";
    MySqlCommand cmd = new MySqlCommand(str, conn);
    using (var reader = cmd.ExecuteReader())
    {
        while (reader.Read())
        {
            list.Add(new Khoa()
            {
                MaKhoa = reader["MaKhoa"].ToString(),
                TenKhoa = reader["TenKhoa"].ToString(),
            });
        }
    }
}
```

Kết nối cơ sở dữ liệu –bước 9 –Insert



❖ Xử lý kết quả trả về

```
public int InsertKhoa(Khoa kh)
{
    using (MySqlConnection conn = GetConnection())
    {
        conn.Open();
        var str = "insert into KHOA values(@makhoa, @tenkhoa)";
        MySqlCommand cmd = new MySqlCommand(str, conn);
        cmd.Parameters.AddWithValue("makhoa", kh.MaKhoa);
        cmd.Parameters.AddWithValue("tenkhoa", kh.TenKhoa);
        return (cmd.ExecuteNonQuery());
    }
}
```

Kết nối cơ sở dữ liệu –bước 9 –Update



❖ Xử lý kết quả trả về

```
public int UpdateKhoa(Khoa kh)
{
    using (MySqlConnection conn = GetConnection())
    {
        conn.Open();
        var str = "update KHOA set TenKhoa = @tenkhoa where MaKhoa=@makhoa";
        MySqlCommand cmd = new MySqlCommand(str, conn);
        cmd.Parameters.AddWithValue("tenkhoa", kh.TenKhoa);
        cmd.Parameters.AddWithValue("makhoa", kh.MaKhoa);
        return (cmd.ExecuteNonQuery());
    }
}
```


Kết nối cơ sở dữ liệu –bước 9 –Delete



❖ Xử lý kết quả trả về

```
public int XoaKhoa(string Id)
{
    using (MySqlConnection conn = GetConnection())
    {
        conn.Open();
        var str = "delete from KHOA where MaKhoa=@makhoa";
        MySqlCommand cmd = new MySqlCommand(str, conn);
        cmd.Parameters.AddWithValue("makhoa", Id);
        return (cmd.ExecuteNonQuery());
    }
}
```

Kết nối cơ sở dữ liệu –bước 10



❖ Đóng kết nối

```
conn.Open();
string str = "select * from KHOA";
MySqlCommand cmd = new MySqlCommand(str, conn);
using (var reader = cmd.ExecuteReader())
{
    while (reader.Read())
    {
        list.Add(new Khoa()
        {
            MaKhoa = reader["MaKhoa"].ToString(),
            TenKhoa = reader["TenKhoa"].ToString(),
        });
    }
    reader.Close();
}
conn.Close();
```

Kết nối cơ sở dữ liệu

