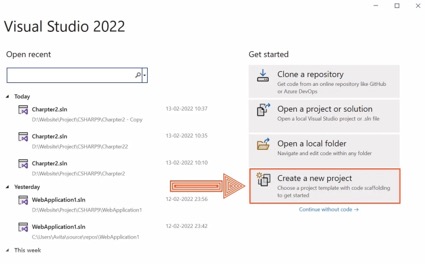
ỨNG DỤNG ASP.NET CORE MVC

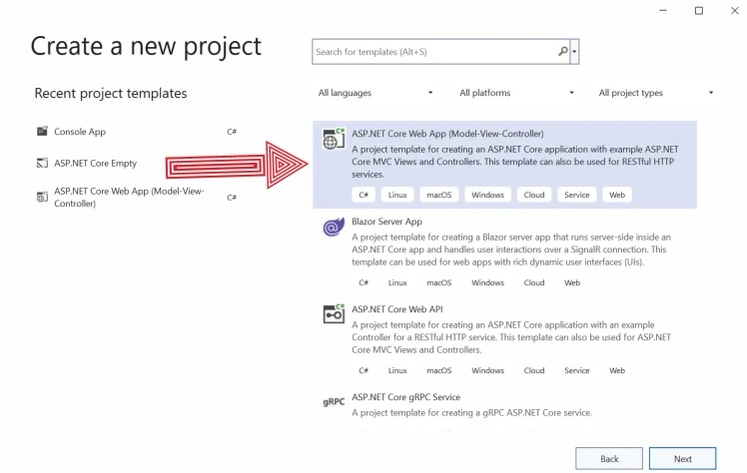
Trước khi bắt đầu học ASP.NET Core, nên có kiến thức cơ bản về HTML, CSS, Java Script và C#. HTML và CSS được sử dụng trong Views tạo thành thành phần giao diện người dùng của ASP.NET Core MVC. C# là ngôn ngữ lập trình để tạo logic như giao tiếp với cơ sở dữ liệu, tìm kiếm giá trị trong dữ liệu,…

## Tạo ứng dụng ASP.NET Core 6.0 trong Visual Studio 2022

Mở Visual Studio 2022, sau đó chọn Create a new project trong Get started.

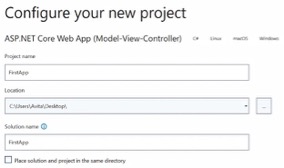


Thao tác này sẽ mở cửa sổ Create a new project. Bây giờ ở đây, chọn mẫu có tên ASP.NET Core Web App (Model-View-Controller), sau đó nhấp vào nút Next.

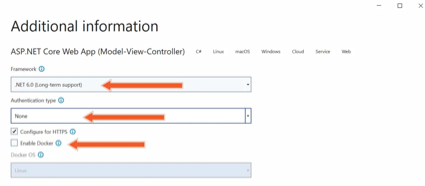


## Cấu hình dự án mới trong Visual Studio 2022

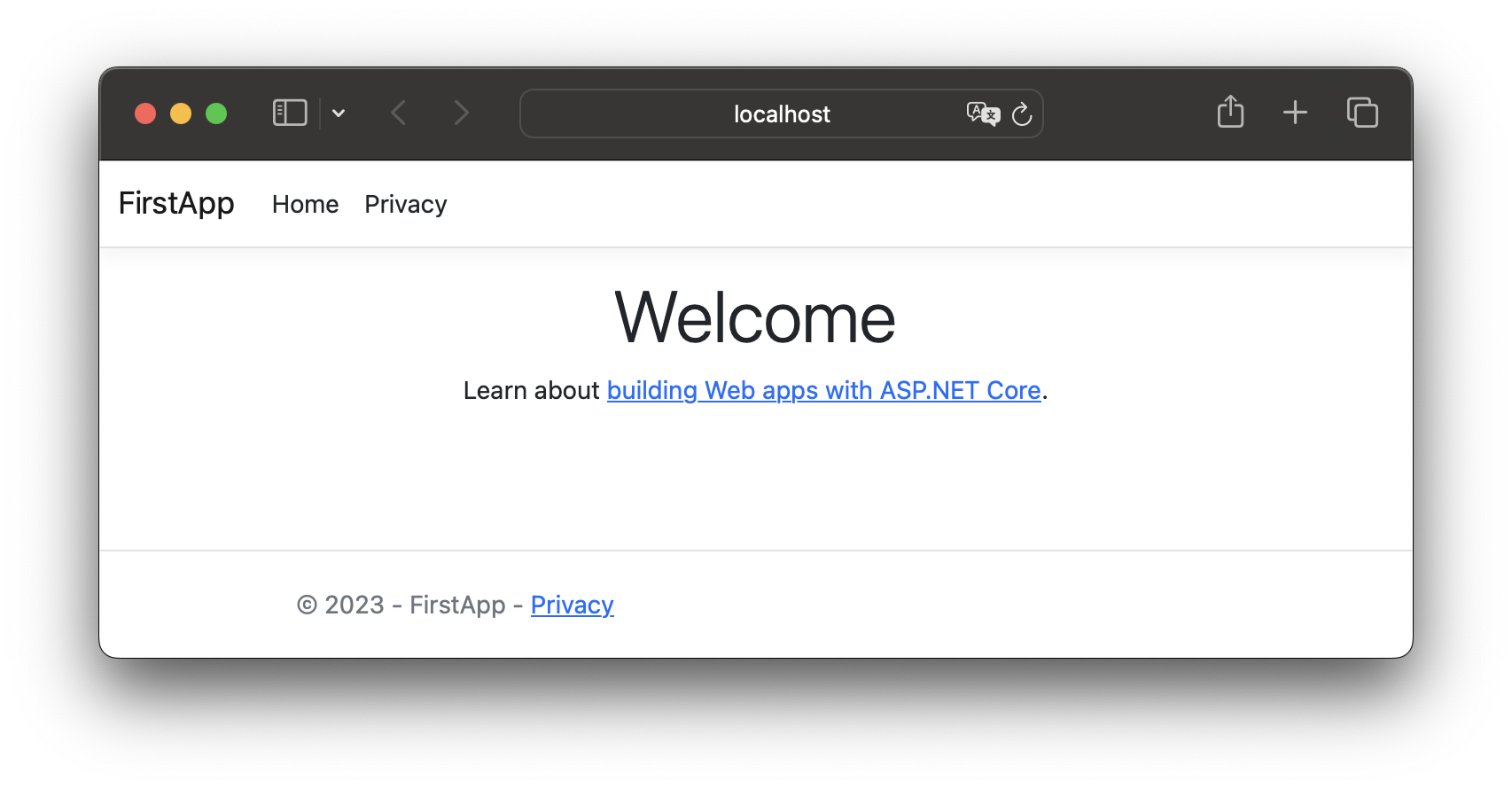
Tiếp theo, sẽ định cấu hình dự án mới. Vì vậy, hãy thêm tên của dự án là FirstApp và chọn vị trí trên ổ đĩa nơi muốn tạo ứng dụng này.



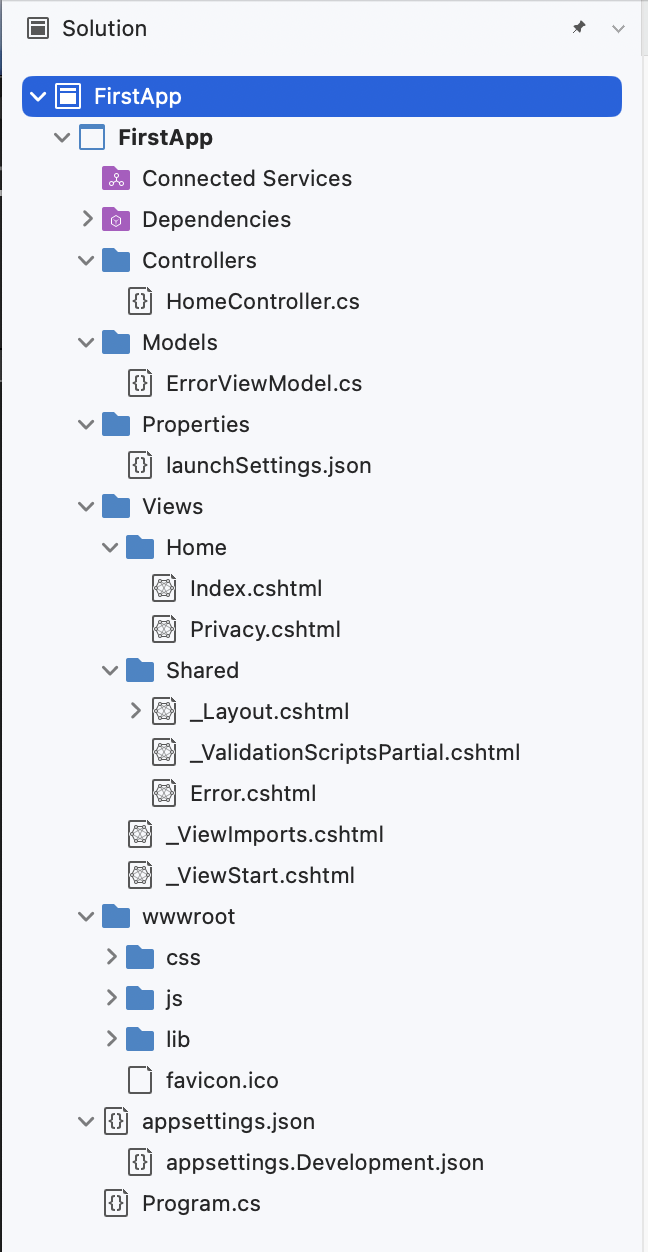
Click vào nút Next để tiếp tục. Sẽ đến trang Additional Information nơi phải chọn framework và authentication (xác thực).



* Đảm bảo rằng .NET 6.0 framework được chọn và xác thực được đặt None. Tùy chọn Docker cũng không được đánh dấu.
* Cuối cùng nhấn vào nút Create để tạo ứng dụng.
* Ứng dụng sẽ được tạo sau vài giây và sẵn sàng chạy. Chọn Start Debugging menu Debug (nếu nó nhắc bật gỡ lỗi, chỉ cần nhấp vào nút OK). Cũng có thể bắt đầu gỡ lỗi bằng cách sử dụng phím tắt là phím F5.
* Visual Studio sẽ xây dựng ứng dụng và mở nó trên trình duyệt mặc định. Nó sẽ trông giống như được hiển thị trên hình ảnh bên dưới.



Ở đây đã chọn Template Model-View-Controller (MVC) để các thư mục và file MVC cần thiết được tạo tự động bởi Visual Studio. Có thể mở Solution Explorer từ menu View 🡪 Solution Explorer, nó sẽ được thêm vào góc bên phải của Visual Studio và sẽ hiển thị tất cả các file/thư mục này trong ứng dụng mới tạo. Nếu đã chọn tùy chọn khác, đó là tùy chọn mẫu Empty, thì phải tạo từng file và thư mục MVC này từ file menu.

´´

## File và thư mục MVC

Bao gồm các thành phần sau :

* wwwroot – chứa các file tĩnh như images, scripts, external frameworks và các thư viện như Bootstrap, jQuery.
* Controllers – Thư mục này chứa các file controller.
* Models – Thư mục này chứa các tập tin model.
* Views – Thư mục này chứa các file view.
* appsettings.json – file này chứa cài đặt cấu hình của ứng dụng. Có thể sử dụng nó để lưu chuỗi kết nối cơ sở dữ liệu, giá trị biến ứng dụng và các thông tin khác.
* program.cs – Đây là điểm bắt đầu của ứng dụng khi chạy ứng dụng. Tại đây, tạo máy chủ ứng dụng (host), chọn máy chủ web, thêm dịch vụ, (authorization) ủy quyền và xác thực (authentications).
* startup.cs – Lưu ý – DOT NET 6.0 không có startup.cs. Nếu chỉ đang sử dụng các phiên bản .NET cũ hơn thì cần có file này trong ứng dụng. Startup.cs được gọi từ file program.cs. Tại đây, thêm các dịch vụ và định cấu hình đường dẫn HTTP. Cũng viết các Routes URL trong file này.

Trên đây chỉ là một mô tả ngắn gọn về các file và thư mục MVC này. Để hiểu đầy đủ về các file này khi tiếp tục với các hướng dẫn riêng lẻ của chúng sau.

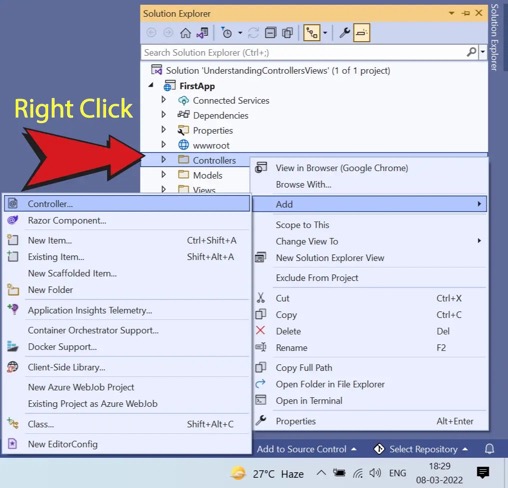
Có rất nhiều cấu hình trong ASP.NET Core đã trình bày trong hướng dẫn có tên Configuring Application in ASP.NET Core.

## Thêm Controller

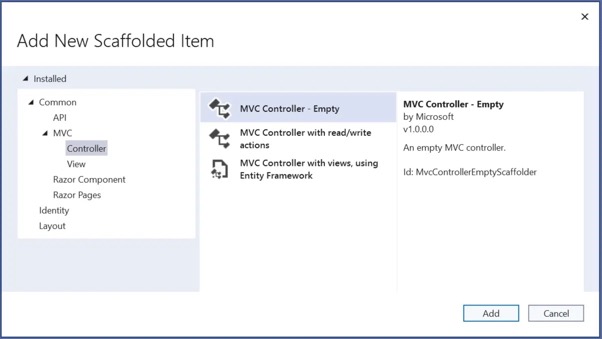
ASP.NET Core Controller là lớp C#, VB hoặc F#, có nhiệm vụ xử lý tất cả các yêu cầu HTTP gửi đến. Controller có thể có một hoặc nhiều phương thức hành động (action methods). Mỗi phương thức hành động có thể được gọi từ web bởi một số URL.

Ví dụ: Nếu controller có tên Home có một phương thức hành động có tên là Calculate thì phương thức hành động này có thể được gọi bằng cách mở URL - Home/Calculate trên trình duyệt.

Theo mặc định, Visual Studio có controller Home. Bây giờ sẽ thêm một controller mới, hãy nhấp chuột phải vào thư mục Controllers trên Solutions Explorer. Sau đó chọn Add 🡪 Controller trên menu.



Sẽ xuất hiện một hộp thoại, tại đây hãy chọn tùy chọn có nội dung - MVC Controller – Empty và nhấp vào nút Add.



Controller mới được thêm vào Application và VS sẽ mở nó ra để chỉnh sửa. Theo mặc định, nó sẽ có mã sau đây được cung cấp bên dưới:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

namespace FirstApp.Controllers

{

public class FirstController : Controller

{

// GET: /<controller>/

public IActionResult Index()

{

return View();

}

}

}

DOT NET 6 sử dụng phiên bản C#10. Trong C# 10, không cần phải viết Namespace Parenthesis. Nó chỉ giả định rằng - bất cứ thứ gì bên trong không gian tên, được xác định trong file, thực sự nằm trong không gian tên. Điều này làm giảm "noise" bổ sung trong code và làm cho C# dễ đọc hơn.

Viết mã controllers mà không sử dụng dấu ngoặc đơn không gian tên (namespace parenthesis). Xem dòng mã được đánh dấu dưới đây.

using Microsoft.AspNetCore.Mvc;

namespace FirstApp.Controllers;

public class FirstController : Controller

{

public IActionResult Index()

{

return View();

}

}

### Action method trả về String

Controller đã tạo trước đó có một phương thức hành động có tên là Index. Thay đổi phương thức này để nó trả về một chuỗi.

Controllers và Action methods là những phần cơ bản của ASP.NET Core. Xem qua Action methods in ASP.NET Core để hiểu cách thức hoạt động của từng phương thức và cách có thể khai thác hết tiềm năng của chúng.

Thay đổi phương thức hành động này thành:

using Microsoft.AspNetCore.Mvc;

namespace FirstApp.Controllers;

public class FirstController : Controller

{

public string Index() {

**return "Xin chao!";**

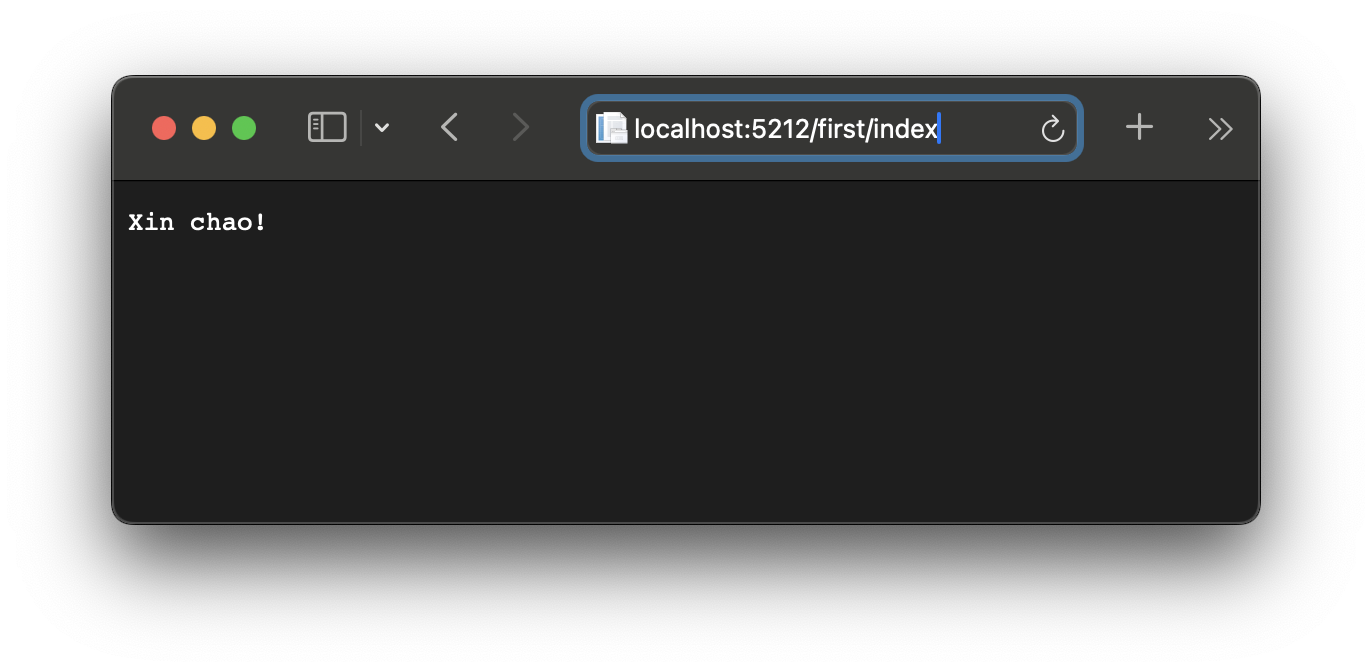
}

}

Bây giờ, phương thức hành động trả về một chuỗi – "Hello World".

Chạy ứng dụng bằng cách nhấp vào Debug 🡪 Start Debugging hoặc nhấn phím F5. Visual Studio sẽ mở trình duyệt hiển thị Home.

Chạy ứng dụngvà yêu cầu http://localhost:[portnumber]/First/Index. Sẽ thấy – “Xin chao!” trên trình duyệt như hình bên dưới.



Portnumber ở đây là 5212 là cổng ứng dụng, trong trường hợp trên máy khác thì nó sẽ khác. Vì vậy, hãy đảm bảo rằng giữ cổng ứng dụng trong url của trình duyệt. Portnumber được cấu hình trong file launchSettings.json

{

"iisSettings": {

"windowsAuthentication": false,

"anonymousAuthentication": true,

"iisExpress": {

"applicationUrl": "http://localhost:62953",

"sslPort": 0

}

},

"profiles": {

"http": {

"commandName": "Project",

"launchBrowser": true,

"applicationUrl": **"http://localhost:5212",**

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

},

"dotnetRunMessages": true

},

"IIS Express": {

"commandName": "IISExpress",

"launchBrowser": true,

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

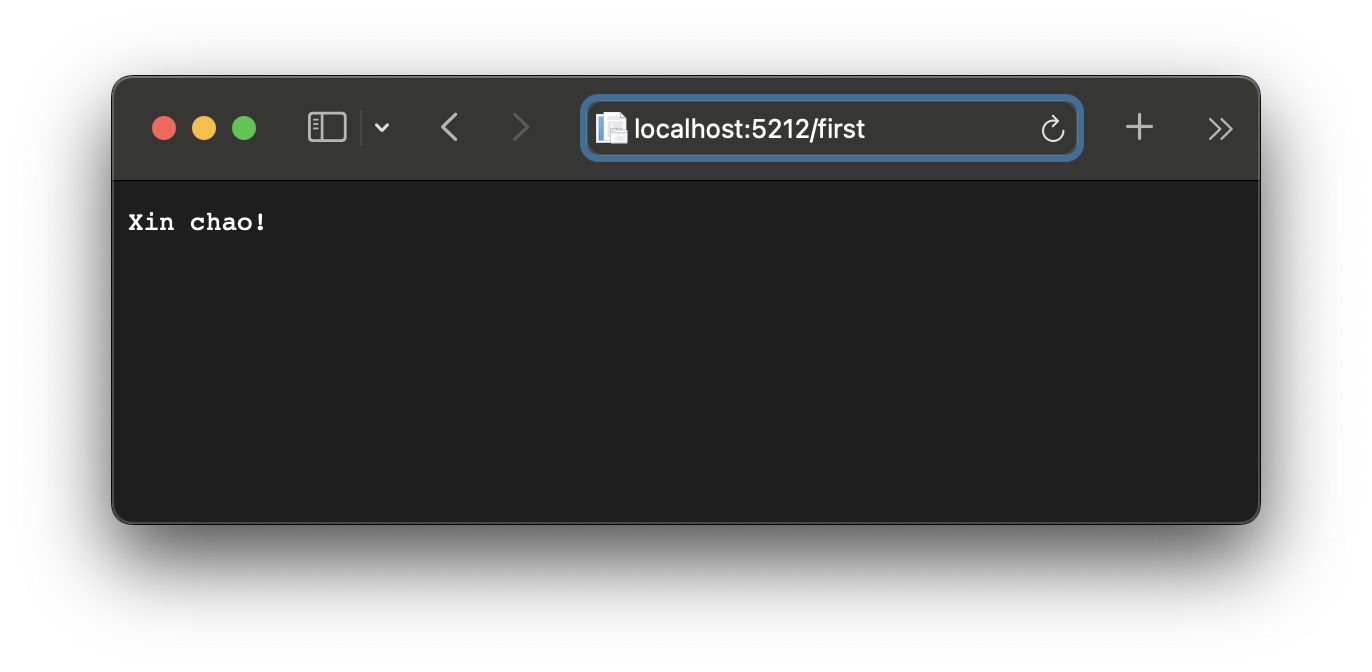
}

}

}

}

Phương thức hành động Index là phương thức hành động mặc định được đặt trên Program.cs (hoặc Startup.cs nếu đang sử dụng Dotnet 5 trở về trước), vì vậy không cần phải chỉ định rõ phương thức này trên URL của trình duyệt. Hành động trên có thể được gọi đơn giản bởi <http://localhost:[portnumber]/First>.



Phương thức hành động này trả về một chuỗi do đó không cần tạo View. Nói chung, hầu như các phương thức hành động trả về View. Vì vậy, đối với những hành động này, kiểu trả về phải được chỉ định là IActionResult

### Action method trả về view

Thêm một phương thức hành động mới vào controller và đặt tên là Hello. Lưu ý ở đây sẽ có kiểu trả về là IActionResult vì nó đang trả về một View.

using Microsoft.AspNetCore.Mvc;

namespace FirstApp.Controllers;

public class FirstController : Controller

{

public string Index()

{

return "Xin chao!";

}

public IActionResult Hello()

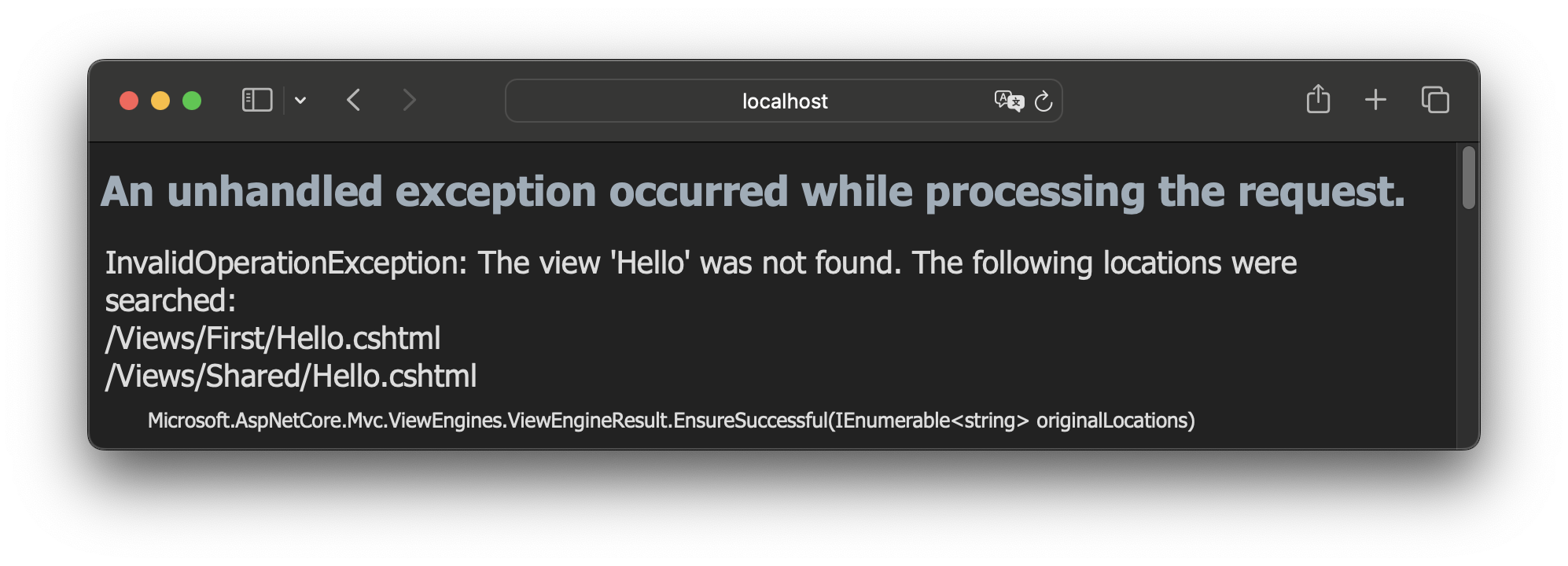
{

return View();

}

}

Chạy ứng dụng rồi truy cập URL của phương thức hành động – <http://localhost:[portnumber]/First/Hello>. Sẽ thấy rằng ứng dụng đang cố tìm view, như thể hiện trong thông báo lỗi hiển thị bên dưới.



**The error states:**

InvalidOperationException: The view 'Hello' was not found. The following locations were searched:

/Views/First/Hello.cshtml

/Views/Shared/Hello.cshtml

Đây là một thông báo giải thích lỗi xảy ra do không có file View trong ứng dụng.

ASP.NET Core MVC Views được lưu trữ trong thư mục Views và được sắp xếp thành các thư mục con. Các view được liên kết với controller có tên là 'FirstController.cs' được lưu trữ trong thư mục Views/First. Views, không dành riêng cho một controller, được lưu trữ bên trong thư mục có tên là Views/Shared. Vì vậy, phải loại bỏ lỗi này bằng cách tạo View Hello trong thư mục Views/First hoặc Views/Shared.

Nhưng có một nhược điểm, nếu đặt view Hello bên trong thư mục Views/Shared và một Controller khác, giả sử AnotherController.cs cũng có một phương thức hành động có cùng tên – Hello và hành động này không có view cụ thể (tức là bên trong thư mục Views/Another). Vì vậy, bất cứ khi nào hành động Hello của bất kỳ controller nào (First hoặc Another) được gọi thì Views/Shared/Hello sẽ được gọi bởi ASP.NET Core runtime engine.

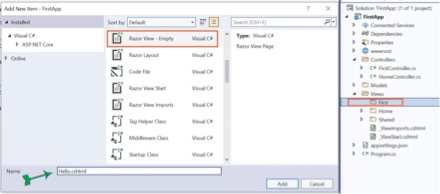
Tóm lại, cả hai URL bên dưới sẽ gọi View – Views/Shared/Hello, miễn là không có views cụ thể cho các hành động:

* /First/Hello
* /Another/Hello

MVC framework tìm kiếm View của một phương thức hành động, đầu tiên trong thư mục cụ thể của nó, nếu nó không tìm thấy view, thì nó sẽ tìm kiếm nó trong thư mục ‘Shared’. Views có phần mở rộng file .cshtml.

Bây giờ hãy tạo một view cụ thể cho controller ‘Hello’. Bắt đầu bằng cách nhấp chuột phải vào thư mục Views và chọn Add 🡪 New Folder. Đặt tên cho thư mục mới này First.

Tiếp theo, nhấp chuột phải vào thư mục ‘First’ này và chọn Add 🡪 New Item. Sau đó, trên hộp thoại hiện ra, chọn Razor View – Empty, và nhấn nút Add.



Đặt tên cho view là Hello và cuối cùng nhấp vào nút Add để tạo view. VS sẽ tạo nó và mở nó để chỉnh sửa. Thêm đoạn mã sau vào file View mới được tạo này:

@{

ViewData["Title"] = "Hello";

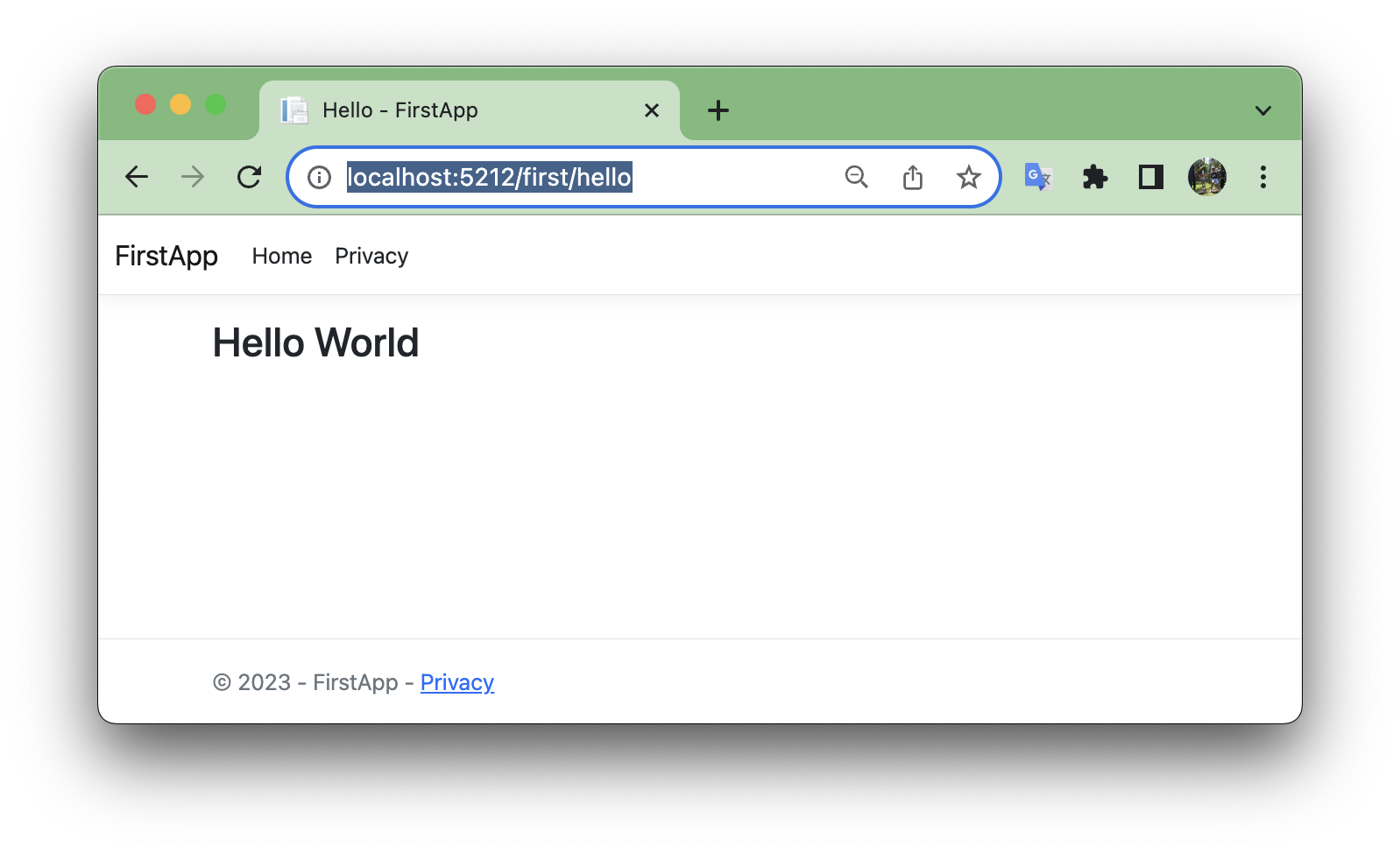
}

<h2>Hello World</h2>

Trong file tên có 2 thao tác :

* Đã thêm biến ViewData “Title” với giá trị chuỗi “Hello”. Điều này sẽ được đặt làm tiêu đề trang.
* Thẻ H2 với Hello World sẽ được hiển thị trên trang.

Chạy ứng dụng và truy cập URL – <http://localhost:[portnumber]/First/Hello>. Sẽ thấy view hiển thị trên trình duyệt như hình bên dưới:



Lưu ý 2 điều, tiêu đề trang trên trình duyệt được đặt là “Hello” và trang hiển thị “Hello World” trong thẻ h2. Vậy là tạo thành công chế độ xem đầu tiên và gọi nó từ trình duyệt.

## Thêm Dynamic Data vào View

Truyền dữ liệu từ Controllers sang View được thực hiện theo một số cách. Ở đây, sẽ sử dụng ViewBag, một đối tượng động để gán bất kỳ loại giá trị nào.

Hãy thay đổi phương thức hành động Hello để bao gồm mã 'ViewBag' gán một biến cho chuỗi "Hello World". Đặt tên biến là Message (nó có thể là bất kỳ tên nào cho biến).

public IActionResult Hello()

{

ViewBag.Message = "Hello World";

return View();

}

Hành động lưu trữ chuỗi trên biến 'ViewBag' được gọi là Message. Bây giờ hiển thị chuỗi này trên View. Vì vậy, hãy thay đổi code Hello.cshtml thành:

@{

ViewData["Title"] = "Hello";

}

<h2>@ViewBag.Message</h2>

Thay đổi nằm trong thẻ <h2> hiện chứa @ViewBag.Message. Nó chỉ hiển thị giá trị của Message của ViewBag được truyền từ Controller.

Chạy lại ứng dụng và v truy cập URL – /First/Hello trên trình duyệt. Sẽ nhận được kết quả tương tự như trước nhưng ở đây đã làm cho nó động.

Có thể gán bất kỳ giá trị nào cho biến ViewBag như chuỗi, int, đối tượng lớp, xml, json, v.v. Biến ViewBag sẽ tự động được .NET Runtime hủy sau khi View được hiển thị trên trình duyệt.

## Thêm Model và chuyển dữ liệu của nó sang View

Thêm một Model vào ứng dụng, thêm Model này từ Controller rồi chuyển nó sang View, nơi nó sẽ được hiển thị trên trình duyệt. Lưu ý rằng Models không là gì ngoài các lớp C#.

Thêm một lớp đơn giản vào thư mục Models. Nhấp chuột phải vào thư mục Models và chọn Add 🡪 Class. Đặt tên lớp là 'Person.cs'. Thêm các thuộc tính dưới đây vào nó:

using System;

namespace FirstApp.Models;

public class Person

{

public string Name { get; set; } = String.Empty;

public int Age { get; set; }

public string Location { get; set; } = String.Empty;

}

Nó là một lớp đơn giản chứa 3 thuộc tính – Name, Age và Location. Bây giờ sẽ điền giá trị vào 3 thuộc tính này trong Controller.

Chuyển đến First Controller và thêm một phương thức hành động mới với tên Info. Mã của nó như sau:

using FirstApp.Models;

using Microsoft.AspNetCore.Mvc;

namespace FirstApp.Controllers;

public class FirstController : Controller

{

// Index Action

// Hello Action

public IActionResult Info()

{

Person person = new Person();

person.Name = "ND Duy";

person.Age = 45;

person.Location = "Vietnam";

return View(person);

}

}

Trong Info Action Method, đã tạo một đối tượng mới cho Model class ‘Person’. Sau đó, gán thuộc tính Name là 'ND Duy', thuộc tính Age là 45 và thuộc tính Location là Vietnam cho đối tượng này. Cuối cùng, đối tượng ‘person’ được trả lại cho View (từ code View(person)).

Để sử dụng lớp Person trong controller First, phải nhập namespace Models trên Controller ở trên cùng - using FirstApp.Models

Bây giờ Model được thêm vào Controller và phải hiển thị dữ liệu của nó trên View. Vì vậy, hãy tạo file view Info bên trong Views 🡪 First.

View Info sẽ có mã sau:

@model Person

@{

ViewData["Title"] = "Info";

}

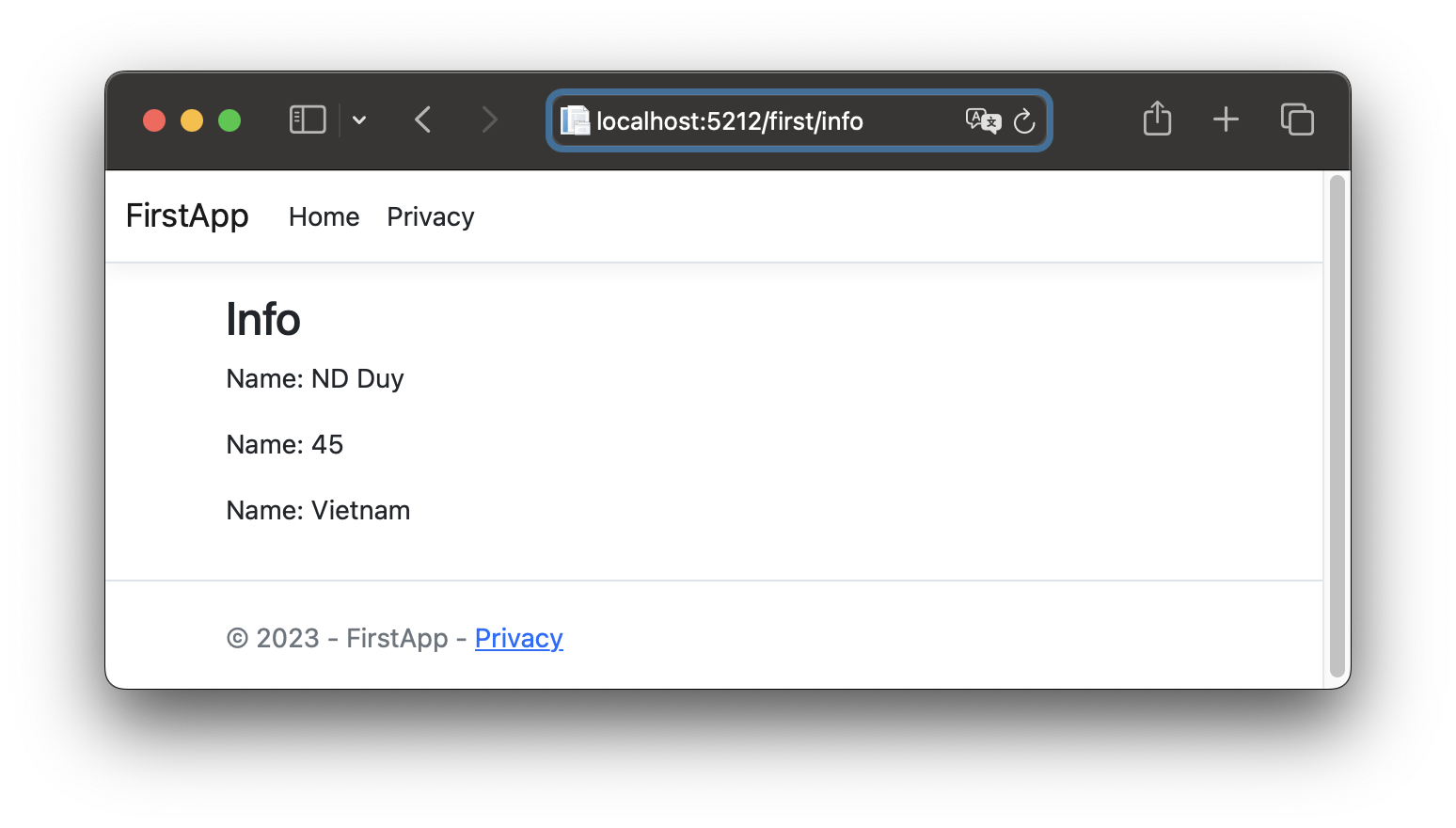
<h2>Info</h2>

<p>Name: @Model.Name</p>

<p>Name: @Model.Age</p>

<p>Name: @Model.Location</p>

Chạy ứng dụng và điều hướng đến /First/Info trên trình duyệt. Dữ liệu Model sẽ được hiển thị như hình bên dưới.



Lưu ý rằng khi xác định model type trên view, sử dụng chữ 'm' nhỏ viết thường như @model Person. Khi phải hiển thị giá trị thuộc tính Model properties thì hãy sử dụng chữ 'M' viết hoa như @Model.Name.

Ứng dụng CRUD trong ASP.NET Core MVC

Xây dựng một ví dụ ASP.NET Core CRUD. Để thực hiện, tạo một form Nhân viên (Employee) có thể điền thông tin nhân viên và gửi (submit). Khi gửi, thông tin nhân viên sẽ được lưu trữ trong một kho lưu trữ (respository). Có các thao tác Create, Read, Update & Delete.

Form Nhân viên (Employee) có 5 trường:

* Tên (Name): tên nhân viên.
* Tuổi (Age): tuổi nhân viên.
* Lương (Salary): lương của nhân viên.
* Bộ phận (Department): bộ phận làm việc của nhân viên
* Giới tính (Sex): giới tính nhân viên.

## Hoạt động CRUD trong ASP.NET Core là gì?

CRUD là từ viết tắt của Create, Read, Update, và Delete. Các hoạt động CRUD trong ASP.NET Core về cơ bản là thực hiện tất cả các hoạt động này trên một kho lưu trữ như cơ sở dữ liệu SQL Server. Ở đây, chúng ta sẽ thực hiện các Thao tác CRUD trên một kho lưu trữ (repository) là một lớp static C# tĩnh.

## Làm cách nào để thực hiện các thao tác CRUD trong ASP.NET Core?

Các bước để thực hiện Thao tác CRUD trong ASP.NET Core như sau:

1. Tạo kho lưu trữ (repository) giống như cơ sở dữ liệu.
2. Tạo ứng dụng ASP.NET Core.
3. Kết nối ứng dụng với kho lưu trữ.
4. Tạo tất cả các hoạt động CRUD trong ứng dụng.
5. Thực hiện Kiểm thử.
6. Chạy ứng dụng trong IIS hoặc Kestrel.

## Tạo ứng dụng trong ASP.NET Core

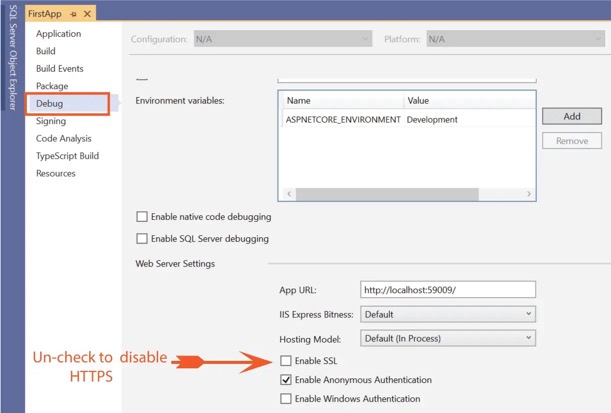
Mở Visual Studio để tạo Ứng dụng web ASP.NET Core mới và chọn mẫu Model-View-Controller như hình bên dưới. Tùy chọn Cấu hình cho HTTPS (Configure for HTTPS) được chọn theo mặc định. ASP.NET Core sẽ tạo Chứng chỉ SSL, chứng chỉ này sẽ hoạt động trên trình duyệt để phát triển dự án.

**ASP.NET Core Vô hiệu hóa HTTPS**

Khi tạo ứng dụng ASP.NET Core, sẽ thấy tùy chọn được chọn có nội dung Configure for HTTPS. Nếu muốn tắt https thì chỉ cần “bỏ chọn” tùy chọn này.

Cũng có thể tắt HTTPS trên ứng dụng ASP.NET Core từ thuộc tính dự án:

1. Click phải vào tên dự án trong “Solution Exporer” và chọn Properties.
2. Nhấp vào tab Debug.
3. Trong Web Server Settings, bỏ chọn tùy chọn Enable SSL.
4. Lưu bằng cách giữ CTRL+S.



## Tạo kho lưu trữ (Repository)

Phải lưu trữ dữ liệu Nhân viên (Employee ) ở đâu đó trong ứng dụng. Để làm điều này, để đơn giản, tạo một static class thay vì cơ sở dữ liệu. Dữ liệu trong một static class vẫn tồn tại miễn là ứng dụng đang chạy, vì vậy ứng dụng sẽ hoạt động chính xác giống như một ứng dụng kết nối cơ sở dữ liệu. Điều này cũng có nghĩa là sẽ tạo các hoạt động CRUD mà không cần sử dụng Entity Framwork Core.

Đối với ứng dụng hướng cơ sở dữ liệu, phải sử dụng Entity Framework Core để thực hiện các thao tác CRUD.

Tiến hành tạo một lớp mới bên trong thư mục Models và đặt tên là Repository.cs.

using System.Collections.Generic;

namespace FirstApp.Models

{

public static class Repository

{

private static List<Employee> allEmpoyees = new List<Employee>();

public static IEnumerable<Employee> AllEmpoyees

{

get { return allEmpoyees; }

}

public static void Create(Employee employee)

{

allEmpoyees.Add(employee);

}

}

}

Trong static class này có một private field gọi là allEmployees thuộc kiểu List. Trường này sẽ chứa tất cả nhân viên được gửi từ form.

Thuộc tính public được gọi là AllEmployees có kiểu IEnumerable sẽ trả về biến 'allEmployees'.

Ngoài ra còn có chức năng Create sẽ chỉ cần thêm nhân viên, được truyền vào tham số, vào danh sách 'allEmployees'.

## Tạo Employee Model

Tạo một lớp bên trong thư mục Models và đặt tên là Employee.cs. Thêm mã dưới đây vào lớp này:

using System;

namespace FirstApp.Models

{

public class Employee

{

public string Name { get; set; }

public int Age { get; set; }

public decimal Salary { get; set; }

public string Department { get; set; }

public Char Sex { get; set; }

}

}

Lưu ý: Lấy thuộc tính Char cho Sex với 'M' cho nam hoặc 'F' cho nữ.

## Tạo Employee Form View

Bây giờ tạo form, tạo Create.cshtml bên trong thư mục Views 🡪 Employee.

Thêm model loại Employee vào trên cùng – @model Employee.

Tạo một người dùng có thể điền và gửi thông tin chi tiết về nhân viên (employee details). Code Create View như bên dưới:

@model Employee

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<form method="post">

<p>

<label asp-for="Name">Your Name:</label>

<input asp-for="Name" />

</p>

<p>

<label asp-for="Age">Your Age:</label>

<input asp-for="Age" />

</p>

<p>

<label asp-for="Salary">Your Salary:</label>

<input asp-for="Salary" />

</p>

<p>

<label asp-for="Department">Your Department:</label>

<select asp-for="Department">

<option value="Development">Development</option>

<option value="HR">HR</option>

<option value="Research">Research</option>

</select>

</p>

<p>

<label asp-for="Sex">Your Sex:</label>

<input type="radio" asp-for="Sex" value="M" />Male

<input type="radio" asp-for="Sex" value="F" />Female

</p>

<p><button type="submit">Submit</button></p>

</form>

Trong code của form, thấy các thuộc tính asp-for được áp dụng trên các thẻ HTML. asp-for là một thuộc tính của trình trợ giúp thẻ (tag helper) liên kết các thẻ HTML (labels, inputs, select,…) với các thuộc tính của mô hình.

Cần import tag helpers trong \_ViewImports.cshtml bằng cách thêm dòng mã – @addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers.

Thuộc tính asp-for trên phần tử label đặt giá trị của nó cho thuộc tính. Thuộc tính asp-for trên các phần tử input, select đặt các phần tử 'id' và 'name' của nó.

Vì vậy, đối với thẻ asp-for được áp dụng trên phần Name của form (được cung cấp bên dưới):

<label asp-for="Name">Your Name:</label>

<input asp-for="Name" />

HTML được hình thành trên trình duyệt sẽ là (được đưa ra bên dưới):

<p>

<label for="Name">Your Name:</label>

<input type="text" id="Name" name="Name" value="">

</p>

Theo cách tương tự, tag helper asp-for được áp dụng cho thẻ select .

<select asp-for="Department">

<option value="Development">Development</option>

<option value="HR">HR</option>

<option value="Research">Research</option>

</select>

Để tạo các nút Radio, tạo 2 đầu vào loại radio và cũng liên kết cả hai đầu vào với thuộc tính Sex bằng cách sử dụng tag helpers asp-for. Cũng có thể lưu ý một cái có giá trị 'M' (dành cho nam) và 'F' khác (dành cho nữ).

<input type="radio" asp-for="Sex" value="M" />Male

<input type="radio" asp-for="Sex" value="F" />Female

Ngoài ra còn có một nút loại submit ngay trước thẻ kết thúc form. Khi nhấp vào nút này, form được gửi.

## Tạo Employee Controller

Tạo một file ASP.NET Core Controller mới bên trong thư mục Controllers và đặt tên là 'EmployeeController'. Trong Controller, bạn Tạo action method cho View.

Ở phần trên đã tạo một View tên là Create.cshtml. Bây giờ trong Controller, phải tạo không chỉ một mà là hai phiên bản action methods cho View này:

1. Phiên bản đầu tiên của phương thức hành động sẽ thuộc loại HTTP GET và sẽ được gọi khi trình duyệt đưa ra yêu cầu bình thường. Phương action method này sẽ được gọi khi truy cập liên kết – /Employee/Create.
2. Phiên bản thứ hai của action method sẽ thuộc loại HTTP POST và sẽ được gọi khi biểu mẫu được gửi. Phiên bản này của action method sẽ chịu trách nhiệm nhận dữ liệu đã gửi.

Tên của cả hai phiên bản của action method sẽ giống nhau. Tuy nhiên, phiên bản POST sẽ mang thuộc tính [HttpPost].

Vì vậy, thêm 2 phiên bản của action method Create này trên Controller, sau đó sẽ giống như hình bên dưới:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using FirstApp.Models;

using Microsoft.AspNetCore.Mvc;

namespace FirstApp.Controllers

{

public class EmployeeController : Controller

{

public IActionResult Index()

{

return View();

}

// HTTP GET VERSION

public IActionResult Create()

{

return View();

}

// HTTP POST VERSION

[HttpPost]

public IActionResult Create(Employee employee)

{

Repository.Create(employee);

return View("Thanks", employee);

}

}

}

Phiên bản HTTP GET của phương thức hành động 'Create' chỉ trả về View. Nó không yêu cầu bất kỳ thuộc tính nào như phiên bản POST.

HTTP POST của phương thức hành động ‘Create’ chứa thuộc tính [HttpPost]. Nó cũng có một tham số kiểu ‘Employee’. Khi nút của form được nhấp vào, tham số này sẽ tự động nhận giá trị của tất cả các thẻ HTML do người dùng điền vào. Điều này được thực hiện tự động bởi tính năng ASP.NET Core Model Binding.

Tiếp theo, dữ liệu Employee (được submit từ form) được thêm vào kho lưu trữ (repository) và người dùng được đưa trở lại View Thanks cùng với dữ liệu nhân viên đã gửi.

Tham số đầu tiên được truyền trên View – View(“Thanks”, employee) chỉ định tên View, ở đây là 'Thanks.cshtml'. Trong khi tham số thứ 2 chỉ định dữ liệu Model được chuyển đến View, ở đây là dữ liệu nhân viên. Điều này có nghĩa là sau khi biểu mẫu được gửi, 'Thanks.cshtml' sẽ được gọi để hiển thị.

## Tạo view Thanks.cshtml

Thêm View Thanks bên trong thư mục View 🡪 Employee và thêm code sau vào:

@model Employee

@{

ViewData["Title"] = "Thanks";

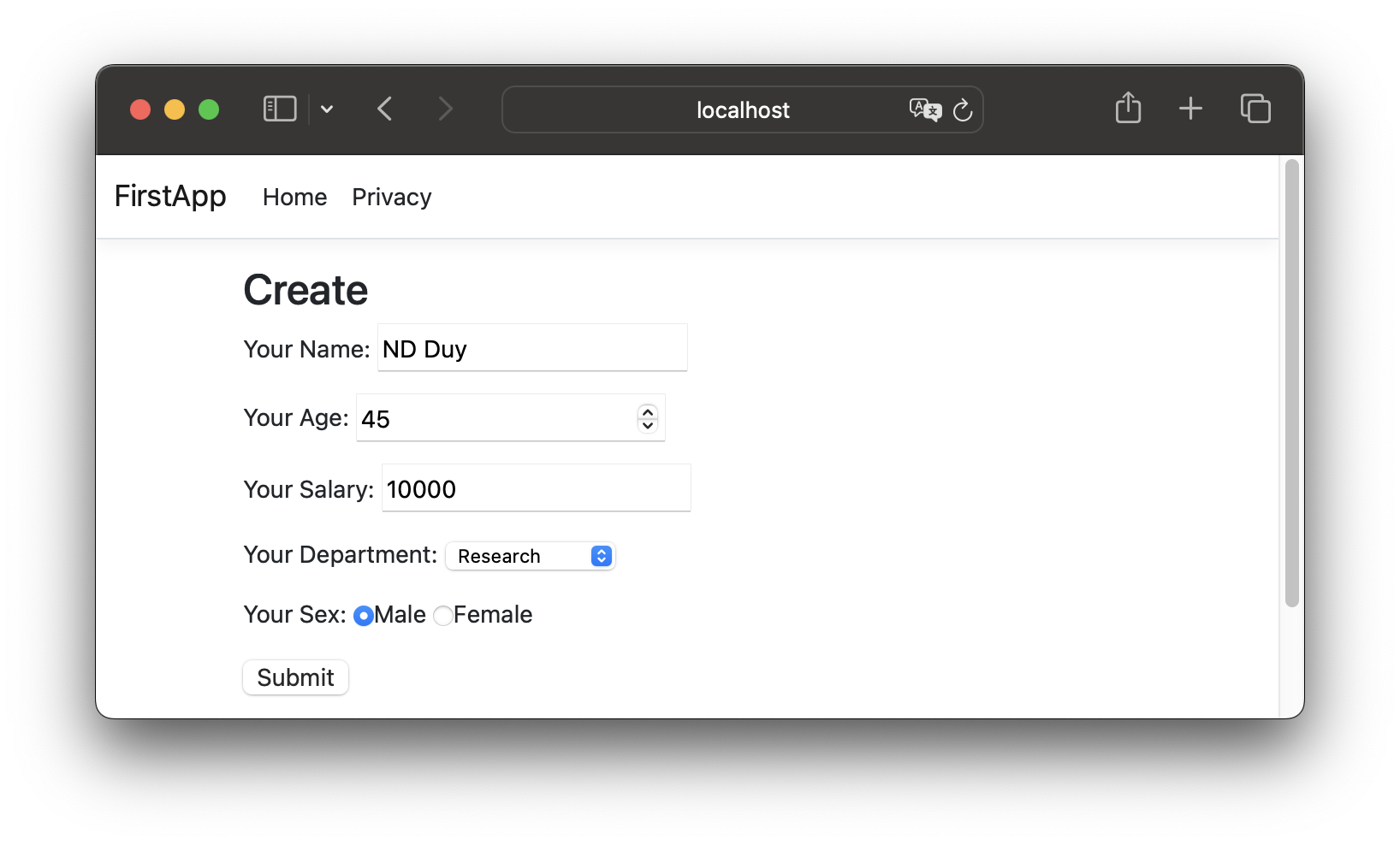
}

<h2>Thanks, @Model.Name</h2>

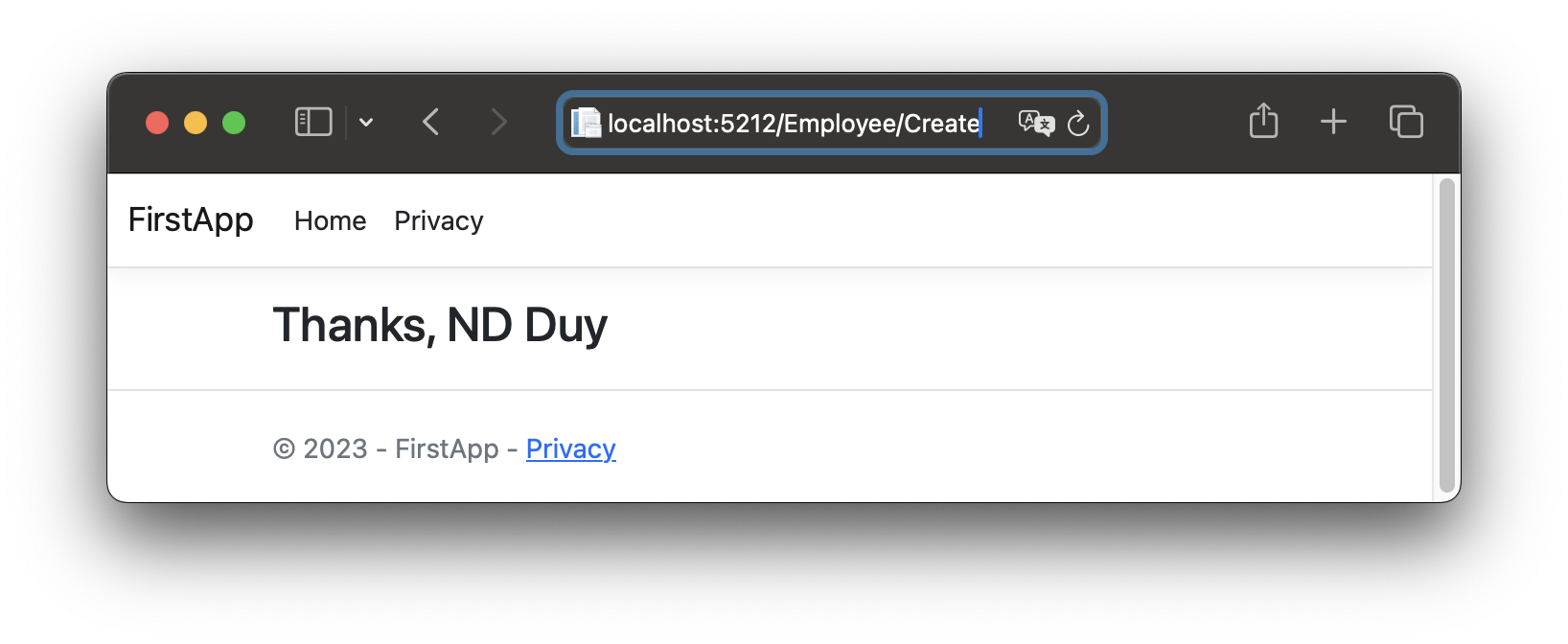
Thêm @model Employee ở trên cùng vì nó sẽ nhận dữ liệu của lớp Employee.

Bên trong thẻ <h2>, một thông báo Thánk sẽ được hiển thị với Tên nhân viên – @Model.Name.

Giờ, hãy chạy ứng dụng và điều hướng đến /Employee/Create trên trình duyệt. Sẽ thấy biểu mẫu (form), điền vào và nhấp vào nút gửi (Submit):



Khi gửi, sẽ thấy thông báo cảm ơn với tên nhân viên.



## Hiển thị tất cả Employees

Đến đây tính năng Create Employee đã hoàn tất. Giờ sẽ tạo tính năng Read để có thể đọc thông tin chi tiết của tất cả nhân viên.

Sử dụng phương thức hành động Index sẽ trả lại Repository.AllEmpoyees cho View. Repository.AllEmpoyees chứa danh sách tất cả nhân viên. Code được đưa ra dưới đây:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using FirstApp.Models;

using Microsoft.AspNetCore.Mvc;

namespace FirstApp.Controllers

{

public class EmployeeController : Controller

{

public IActionResult Index()

{

return View(Repository.AllEmpoyees);

}

// Removed for clarity

}

}

Tiếp theo, tạo View Index bên trong thư mục Views 🡪 Employee và thêm code bên dưới vào đó:

@model IEnumerable<Employee>

@{

ViewData["Title"] = "Index";

}

<h2>Here is the list of Employees</h2>

<table style="width:50%">

<thead>

<tr>

<th>Name</th>

<th>Age</th>

<th>Salary</th>

<th>Department</th>

<th>Sex</th>

</tr>

</thead>

<tbody>

@foreach (Employee e in Model)

{

<tr>

<td>@e.Name</td>

<td>@e.Age</td>

<td>@e.Salary</td>

<td>@e.Department</td>

<td>@e.Sex</td>

</tr>

}

</tbody>

</table>

View có kiểu IEnumerable vì nó sẽ nhận được danh sách các nhân viên. Để hiển thị tất cả các Nhân viên này, sử dụng HTML Table, trong bảng, lặp lại tất cả các nhân viên bằng cách sử dụng vòng lặp foreach và hiển thị dữ liệu lần lượt bên trong phần tử tbody .

Để kiểm tra cách thức hoạt động của tính năng này, hãy tạo một vài nhân viên rồi truy cập URL /Employee trên trình duyệt. Sẽ thấy tất cả các nhân viên như hình bên dưới:

A screenshot of a computer

Description automatically generated

## Kết nối các views thông qua các liên kết

Bây giờ chúng ta hãy liên kết các View bằng thẻ neo (anchor tags).

Trên View Thanks.cshtml, hãy thêm dòng code:

<a asp-action="Index">View All Employees</a>

Đặt nó ngay trước thẻ h2.

Dòng này sẽ tạo thẻ neo (anchor tag) sẽ liên kết với View Index. asp-action là một tag helper yêu cầu neo liên kết với View Index của cùng Controller nơi hiện đang ở. Mã Thanks.cshtml sẽ trở thành:

@model Employee

@{

ViewData["Title"] = "Thanks";

}

<a asp-action="Index">View All Employees</a>

<h2>Thanks, @Model.Name</h2>

Theo cách tương tự, hãy thêm mã anchor tag vào View Index. Điều này sẽ liên kết nó với View Create.cshtml.

<a asp-action="Create">Create an Employees</a>

Mã Index.cshtml sẽ trở thành:

@model IEnumerable<Employee>

@{

ViewData["Title"] = "Index";

}

<a asp-action="Create">Create an Employees</a>

<h2>Here is the list of Employees</h2>

<table style="width:50%">

...

</table>

Cuối cùng liên kết View Create với View Index bằng cách thêm code sau:

<a asp-action="Index">View All Employees</a>

Mã Create.cshtml sẽ trở thành:

@model Employee

@{

ViewData["Title"] = "Create";

}

<a asp-action="Index">View All Employees</a>

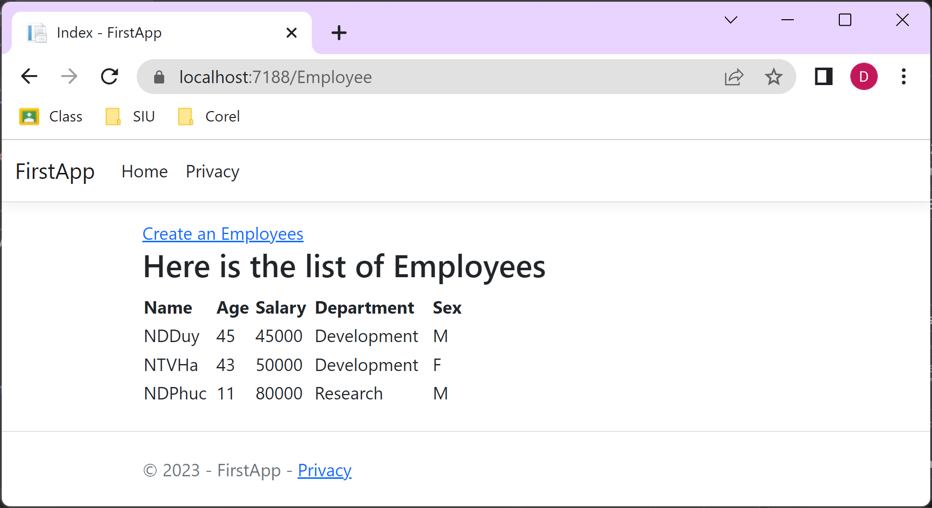
<h2>Create</h2>

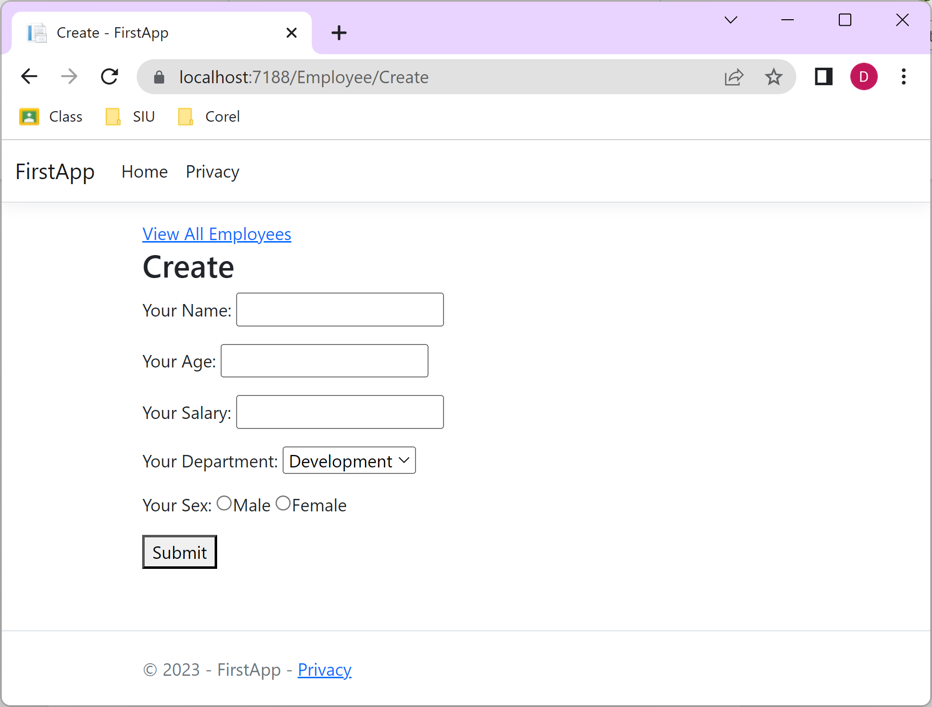
<form method="post">

...

</form>

Hình ảnh đã cho hiển thị các thẻ neo trên mỗi trong số 3 View:





## Cập nhật Employee

Toàn bộ khái niệm cập nhật là gửi tên nhân viên tới Update action method ở dạng chuỗi truy vấn. Ở đó sẽ lấy tất cả các chi tiết của nhân viên có tên như được chỉ định trong chuỗi truy vấn.

Sẽ cung cấp liên kết Cập nhật (Update link) bên cạnh mọi nhân viên được hiển thị trên bảng được cung cấp trên View Index. Vì vậy, hãy thêm một phần tử td mới bên trong khối foreach:

<td><a asp-action="Update" asp-route-empname="@e.Name">Update</a></td></span>

Bây giờ code View Index sẽ như sau:

@model IEnumerable<Employee>

@{

ViewData["Title"] = "Index";

}

<a asp-action="Create">Create an Employees</a>

<h2>Here is the list of Employees</h2>

<table style="width:50%">

<thead>

<tr>

<th>Name</th>

<th>Age</th>

<th>Salary</th>

<th>Department</th>

<th>Sex</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (Employee e in Model)

{

<tr>

<td>@e.Name</td>

<td>@e.Age</td>

<td>@e.Salary</td>

<td>@e.Department</td>

<td>@e.Sex</td>

<td><a asp-action="Update" asp-route-empname="@e.Name">Update</a></td>

</tr>

}

</tbody>

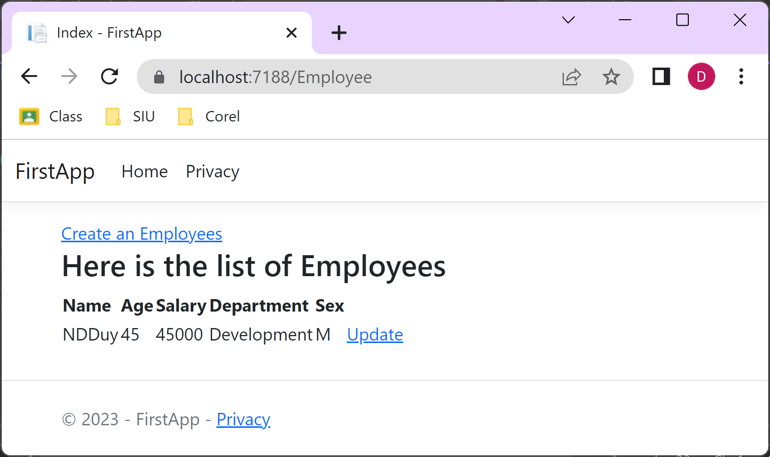
</table>

Asp-route-\* là một route tag helper sẽ tạo một chuỗi truy vấn theo tên được cung cấp tại '\*', ở đây nó sẽ là 'empname'. Xem đoạn code dưới đây:

<a asp-action="Update" asp-route-empname="@e.Name">Update</a>

Nó sẽ tạo một thẻ neo, với thuộc tính 'href' nhắm mục tiêu đến phương thức hành động ' Update' và có 'empname' làm chuỗi truy vấn. Tóm lại, 'href' của anchor sẽ là – /Update?empname=name-of-employee.

Chạy ứng dụng và chuyển đến View Index. Sẽ thấy các liên kết cập nhật dọc theo từng nhân viên như trong hình bên dưới:



Bây giờ, hãy chuyển đến Controller và thêm các phiên bản HTTP GET và HTTP POST của phương thức hành động Cập nhật (Update action method).

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using FirstApp.Models;

using Microsoft.AspNetCore.Mvc;

namespace FirstApp.Controllers

{

public class EmployeeController : Controller

{

// Removed for clarity

public IActionResult Update(string empname)

{

Employee employee = Repository.AllEmpoyees.Where(e => e.Name == empname).FirstOrDefault();

return View(employee);

}

[HttpPost]

public IActionResult Update(Employee employee, string empname)

{

Repository.AllEmpoyees.Where(e => e.Name == empname).FirstOrDefault().Age = employee.Age;

Repository.AllEmpoyees.Where(e => e.Name == empname).FirstOrDefault().Salary = employee.Salary;

Repository.AllEmpoyees.Where(e => e.Name == empname).FirstOrDefault().Department = employee.Department;

Repository.AllEmpoyees.Where(e => e.Name == empname).FirstOrDefault().Sex = employee.Sex;

Repository.AllEmpoyees.Where(e => e.Name == empname).FirstOrDefault().Name = employee.Name;

return RedirectToAction("Index");

}

}

}

Trong phiên bản HTTP GET của Update action method, chuỗi tham số empname sẽ nhận giá trị của biến chuỗi truy vấn 'empname' bằng model binding.

Sau đó, nhận được thông tin chi tiết về nhân viên có tên được gửi trong chuỗi truy vấn. Ở đây, sử dụng LINQ để tìm kiếm Repository. Các chi tiết về nhân viên được điền vào một loại tượng loại được gọi employee được trả về Update view.

Trong phiên bản HTTP POST của Update action method có 2 tham số – Employee employee, string empname. Tham số “employee” sẽ chứa dữ liệu Employee được gửi bởi form trong khi “empname” sẽ chứa tên của nhân viên trong chuỗi truy vấn.

Tiếp theo, tìm thông tin chi tiết về nhân viên có tên khớp với giá trị 'empname' và cập nhật các trường đó (Age, Salary, Department, Sex, Name) với các trường mới mà tham số employee nắm giữ. Cuối cùng, người dùng được chuyển hướng đến index action.

Tạo View update bên trong thư mục Views 🡪 Employee . Nội dung của View Update hoàn toàn giống với Create View ngoại trừ – ViewData[“Title”] = “Update” và <h2>Update</h2>.

Update View được đưa ra bên dưới:

@model Employee

@{

ViewData["Title"] = "Update";

}

<a asp-action="Index">View All Employees</a>

<h2>Update</h2>

<form method="post">

<p>

<label asp-for="Name">Your Name:</label>

<input asp-for="Name" />

</p>

<p>

<label asp-for="Age">Your Age:</label>

<input asp-for="Age" />

</p>

<p>

<label asp-for="Salary">Your Salary:</label>

<input asp-for="Salary" />

</p>

<p>

<label asp-for="Department">Your Department:</label>

<select asp-for="Department">

<option value="Development">Development</option>

<option value="HR">HR</option>

<option value="Research">Research</option>

</select>

</p>

<p>

<label asp-for="Sex">Your Sex:</label>

<input type="radio" asp-for="Sex" value="M" />Male

<input type="radio" asp-for="Sex" value="F" />Female

</p>

<p><button type="submit">Submit</button></p>

</form>

## Delete Employee

Để xóa một nhân viên khỏi kho lưu trữ (repository), trước tiên hãy tạo Delete static method trong file Repository.cs.

using System.Collections.Generic;

namespace FirstApp.Models

{

public static class Repository

{

// Removed for clarity

public static void Delete(Employee employee)

{

allEmpoyees.Remove(employee);

}

}

}

Tiếp theo, thêm HTTP POST action method theo tên Delete trong Controller.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using FirstApp.Models;

using Microsoft.AspNetCore.Mvc;

namespace FirstApp.Controllers

{

public class EmployeeController : Controller

{

// Removed for clarity

[HttpPost]

public IActionResult Delete(string empname)

{

Employee employee = Repository.AllEmpoyees.Where(e => e.Name == empname).FirstOrDefault();

Repository.Delete(employee);

return RedirectToAction("Index");

}

}

}

Delete action method là loại POST và có tên của nhân viên trong tham số của nó. Delete action điền thông tin chi tiết của nhân viên (có tên giống với tên mà nó nhận được trong tham số của nó) trong một đối tượng Employee có tên là “employee”, sau đó gọi phương thức Delete của repository để xóa nó khỏi đó. Cuối cùng, người dùng được chuyển hướng đến Index action method.

Để gọi phương Delete action method, hãy chuyển đến View Index và thêm phần tử td mới bên trong khối foreach. Bên trong td này tạo một phần tử form.

Code form là:

<form asp-route-empname="@e.Name" asp-action="Delete" method="post">

<button type="submit">Delete</button>

</form>

Form này có một nút mà khi được nhấp vào sẽ gọi phương thức hành động Delete. asp-action là một tag helper được sử dụng để đặt hành động của biểu mẫu, trong trường hợp này là Delete action method. asp-route-\* là một tag helper sẽ thêm tham số chuỗi truy vấn “empnane” trên URL. Biến “empname” này sẽ chứa tên của nhân viên.

Vì vậy, tham số của Delete action string empname sẽ liên kết với giá trị chuỗi truy vấn này do model binding.

Code của Index View đã trở thành:

@model IEnumerable<Employee>

@{

ViewData["Title"] = "Index";

}

<a asp-action="Create">Create an Employees</a>

<h2>Here is the list of Employees</h2>

<table style="width:50%">

<thead>

<tr>

<th>Name</th>

<th>Age</th>

<th>Salary</th>

<th>Department</th>

<th>Sex</th>

<th></th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (Employee e in Model)

{

<tr>

<td>@e.Name</td>

<td>@e.Age</td>

<td>@e.Salary</td>

<td>@e.Department</td>

<td>@e.Sex</td>

<td><a asp-action="Update" asp-route-empname="@e.Name">Update</a></td>

<td>

<form asp-route-empname="@e.Name" asp-action="Delete" method="post">

<button type="submit">Delete</button>

</form>

</td>

</tr>

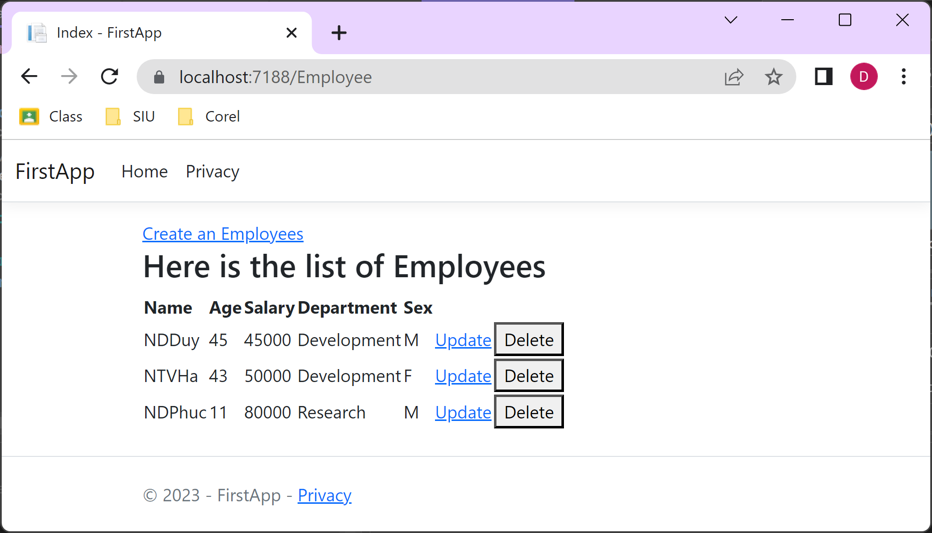
}

</tbody>

</table>

Có thể kiểm tra hoạt động của tính năng Delete bằng cách chạy ứng dụng của mình và nhấp vào nút Delete đối với bất kỳ Employee nào.

Hình ảnh bên dưới hiển thị các nút xóa được thêm vào từng nhân viên:



## Thêm Validations

Có thể nhận thấy rằng trên trang Create Employee , người dùng có thể chỉ cần gửi biểu mẫu mà không cần điền vào bất kỳ trường nào. Vì vậy, điều này sẽ làm phát sinh các mục xấu nên ngăn chặn trong ứng dụng của mình. Ở đây Validations sẽ có ích.

Trong ASP.NET Core Model Validation được thực hiện trong lớp Model. Điều này có nghĩa là validations được áp dụng ở một nơi nhưng có hiệu lực ở mọi nơi trong ứng dụng. Các quy tắc validations được áp dụng với các thuộc tính từ không gian tên System.ComponentModel.DataAnnotations.

Chuyển đến lớp Employee.cs trong thư mục Models và nhập không gian tên – System.ComponentModel.DataAnnotations vào lớp đó.

Thêm các thuộc tính như được hiển thị trên file Employee.cs đã sửa bên dưới:

using System;

using System.ComponentModel.DataAnnotations;

namespace FirstApp.Models

{

public class Employee

{

[Required(ErrorMessage = "Vui long nhap ten.")]

public string Name { get; set; }

[Range(18, 60, ErrorMessage = "Tuoi phai tu 18-60.")]

public int Age { get; set; }

[RegularExpression(@"\d+(\.\d{1,2})?", ErrorMessage = "Khong hop le, nhap dang # hoac #.##")]

public decimal Salary { get; set; }

public string Department { get; set; }

[RegularExpression(@"^[MF]+$", ErrorMessage = "Lua chon gioi tinh.")]

public Char Sex { get; set; }

}

}

ASP.NET Core MVC tự động phát hiện các thuộc tính và sử dụng chúng để xác thực dữ liệu trong quá trình model-binding.

Đã sử dụng 3 loại thuộc tính khác nhau ở đây:

1. Thuộc tính Required được sử dụng để xác định rằng trường mà nó được áp dụng không được chứa giá trị rỗng hoặc trống. Đã áp dụng chúng cho thuộc tính chuỗi Name để MVC buộc người dùng điền vào mục Name.
2. Thuộc tính Range xác định giá trị phạm vi số cho trường. Đã áp dụng nó trên trường Age để mục Age chỉ chấp nhận độ tuổi từ 18 đến 60.
3. Thuộc tính RegularExpression (Biểu thức chính quy) chỉ định rằng giá trị trường khớp với giá trị Biểu thức chính quy. Đã áp dụng trên trường Sẽ với biểu thức ^[MF]+$ để người dùng phải chọn nút radio. Các nút radio có giá trị 'M' & 'F', vì vậy nếu người dùng không chọn bất kỳ nút nào trong số chúng thì Biểu thức chính quy sẽ không thành công và quá trình xác thực cũng vậy.

Thuộc tính Required chỉ hoạt động khi các giá trị rỗng hoặc trống, vì vậy đã không áp dụng thuộc tính Required cho Age và Salary. Lý do là, giả sử nếu người dùng không điền bất kỳ giá trị nào vào các mục này thì giá trị mặc định là 0 sẽ được áp dụng cho trường Age và Salary (giá trị mặc định của cả kiểu int và kiểu thập phân đều là 0).

Giá trị mặc định của char là ‘\0’ vì vậy thuộc tính Required cũng vô dụng đối với trường Sex.

Trường hợp của Department. Nó được áp dụng trên điều khiển chọn và vì vậy người dùng không thể thoát khỏi việc chọn một tùy chọn.

Để hiển thị các lỗi xác thực này trên View, thêm một điều khiển như 'span', 'label' hoặc 'div' bên cạnh mỗi điều khiển mà người dùng phải điền hoặc chọn. Cũng phải liên kết điều khiển này với thông báo lỗi xác thực bằng asp-validation-for tag helper.

Ví dụ: Đối với trường tên, hãy thêm phần tử span như thế này:

<p>

<label asp-for="Name">Your Name:</label>

<input asp-for="Name" />

<span asp-validation-for="Name"></span>

</p>

Làm điều này cho bất cứ nơi nào phải hiển thị thông báo xác thực. Vì vậy, cập nhật View - Các file Create.cshtml & Update.cshtml như được hiển thị bên dưới:

Create.cshtml

@model Employee

@{

ViewData["Title"] = "Create";

}

<a asp-action="Index">View All Employees</a>

<h2>Create</h2>

<form method="post">

<p>

<label asp-for="Name">Your Name:</label>

<input asp-for="Name" />

<span asp-validation-for="Name"></span>

</p>

<p>

<label asp-for="Age">Your Age:</label>

<input asp-for="Age" />

<span asp-validation-for="Age"></span>

</p>

<p>

<label asp-for="Salary">Your Salary:</label>

<input asp-for="Salary" />

<span asp-validation-for="Salary"></span>

</p>

<p>

<label asp-for="Department">Your Department:</label>

<select asp-for="Department">

<option value="Development">Development</option>

<option value="HR">HR</option>

<option value="Research">Research</option>

</select>

<span asp-validation-for="Department"></span>

</p>

<p>

<label asp-for="Sex">Your Sex:</label>

<input type="radio" asp-for="Sex" value="M" />Male

<input type="radio" asp-for="Sex" value="F" />Female

<span asp-validation-for="Sex"></span>

</p>

<p><button type="submit">Submit</button></p>

</form>

Update.cshtml

@model Employee

@{

ViewData["Title"] = "Update";

}

<a asp-action="Index">View All Employees</a>

<h2>Update</h2>

<form method="post">

<p>

<label asp-for="Name">Your Name:</label>

<input asp-for="Name" />

<span asp-validation-for="Name"></span>

</p>

<p>

<label asp-for="Age">Your Age:</label>

<input asp-for="Age" />

<span asp-validation-for="Age"></span>

</p>

<p>

<label asp-for="Salary">Your Salary:</label>

<input asp-for="Salary" />

<span asp-validation-for="Salary"></span>

</p>

<p>

<label asp-for="Department">Your Department:</label>

<select asp-for="Department">

<option value="Development">Development</option>

<option value="HR">HR</option>

<option value="Research">Research</option>

</select>

<span asp-validation-for="Department"></span>

</p>

<p>

<label asp-for="Sex">Your Sex:</label>

<input type="radio" asp-for="Sex" value="M" />Male

<input type="radio" asp-for="Sex" value="F" />Female

<span asp-validation-for="Sex"></span>

</p>

<p>

<button type="submit">Submit</button>

</p>

</form>

Cuối cùng phải thực hiện những thay đổi cần thiết trên Controller. Mở controller Employee và thay đổi các phiên bản HTTP POST của các Create và Update actions như được hiển thị bên dưới:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using FirstApp.Models;

using Microsoft.AspNetCore.Mvc;

namespace FirstApp.Controllers

{

public class EmployeeController : Controller

{

[HttpPost]

public IActionResult Create(Employee employee)

{

if (ModelState.IsValid)

{

Repository.Create(employee);

return View("Thanks", employee);

}

else

return View();

}

[HttpPost]

public IActionResult Update(Employee employee, string empname)

{

if (ModelState.IsValid)

{

Repository.AllEmpoyees.Where(e => e.Name == empname).FirstOrDefault().Age = employee.Age;

Repository.AllEmpoyees.Where(e => e.Name == empname).FirstOrDefault().Salary = employee.Salary;

Repository.AllEmpoyees.Where(e => e.Name == empname).FirstOrDefault().Department = employee.Department;

Repository.AllEmpoyees.Where(e => e.Name == empname).FirstOrDefault().Sex = employee.Sex;

Repository.AllEmpoyees.Where(e => e.Name == empname).FirstOrDefault().Name = employee.Name;

return RedirectToAction("Index");

}

else

return View();

}

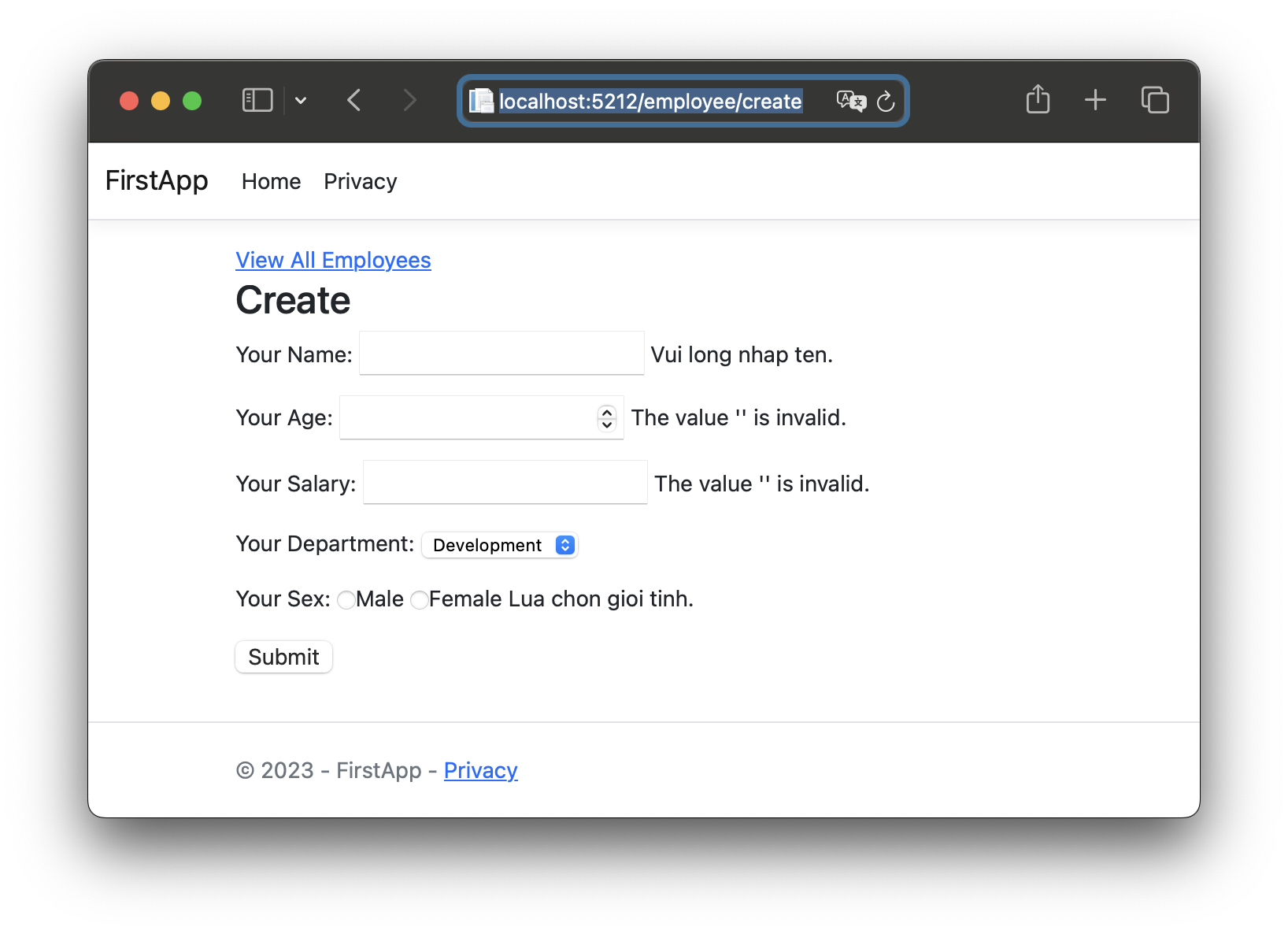
}

}

Trong code trên, đã thêm một điều kiện if else để kiểm tra xem ModelState có hợp lệ hay không – if (ModelState.IsValid). Trong ModelState MVC giữ các chi tiết kiểm tra xác thực, nếu tất cả các xác thực đều vượt qua thì chỉ MVC mới làm cho ModelState hợp lệ.

Do đó, nếu ModelState hợp lệ thì thêm hoặc cập nhật Employee trong Repository. Điều này hoàn thành phần xác thực của MVC.

Hãy thử gửi biểu mẫu nhân viên mà không cần điền hoặc chọn bất kỳ giá trị nào trên các điều khiển. Sẽ thấy các lỗi xác thực như trong hình bên dưới:



## Thêm Styles

Để làm cho giao diện của ứng dụng đẹp hơn, phải thêm một số kiểu CSS.

Mở file \_ViewStart.cshtml nằm trong thư mục Views. File này chứa bố cục của tất cả các View trong ứng dụng.

Code của file \_ViewStart.cshtml sẽ giống như sau:

@{

Layout = "\_Layout";

}

Chỉ định layout là \_Layout.cshtml và nằm bên trong thư mục Views 🡪 Shared.

Bootstrap là một gói quan trọng cần cài đặt trong mọi trang web, vì vậy hãy tìm hiểu Cách cài đặt gói Bootstrap trong ASP.NET Core trong Visual Studio

Code của file \_Layout.cshtml được cung cấp bên dưới:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"] - FirstApp</title>

<environment include="Development">

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />

<link rel="stylesheet" href="~/css/site.css" />

</environment>

<environment exclude="Development">

<link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"

asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"

asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-value="absolute" />

<link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />

</environment>

</head>

Cấu trúc HTML cung cấp thiết kế cần thiết cho tất cả các View. Xem bên trong khu vực đầu, sẽ tìm thấy 2 phần:

<environment include="Development">

</environment>

<environment exclude="Development">

</environment>

Khi chạy ứng dụng của mình trong localhost thì MVC chỉ bao gồm những file đó, nằm trong khối include="Development" và loại trừ những file nằm trong khối exclude="Development". Sẽ thấy có một liên kết đến file Style Sheet có tên là site.css. Nó nằm bên trong thư mục wwwroot. Vì vậy, bây giờ đã biết nơi phải đặt mã CSS.

Mở file site.css và thêm đoạn mã sau để tạo kiểu cho biểu mẫu và bảng.

container a {

font-family: 'Adobe Arabic';

font-size: 25px;

}

form p label {

width: 10%;

}

form p input[type="text"], form p input[type="number"], form p select {

width: 20%;

}

form p span {

width: 20%;

color: Red;

}

form p button, table button {

background-color: #006bff;

color: #FFF;

}

form p .input-validation-error {

border: 1px solid Red;

background-color: #fee;

}

table {

border: solid 1px #ff6a00;

}

table thead {

background: #b6ff00;

}

table td, table th {

padding: 10px;

}

ASP.NET Core MVC thêm class=”input-validation-error” vào các điều khiển khi quá trình xác thực không thành công. Thực hiện kiểm tra html của bất kỳ điều khiển nào bằng các công cụ dành cho nhà phát triển chrome để tìm lớp này. Xem html bên dưới của kiểm soát đầu vào cho trường Name.

<input type="text" class="input-validation-error" data-val="true" data-val-required="Please enter your name" id="Name" name="Name" value="">

Giả sử, thêm một đường viền màu đỏ cho chúng mặc dù mã CSS bên dưới:

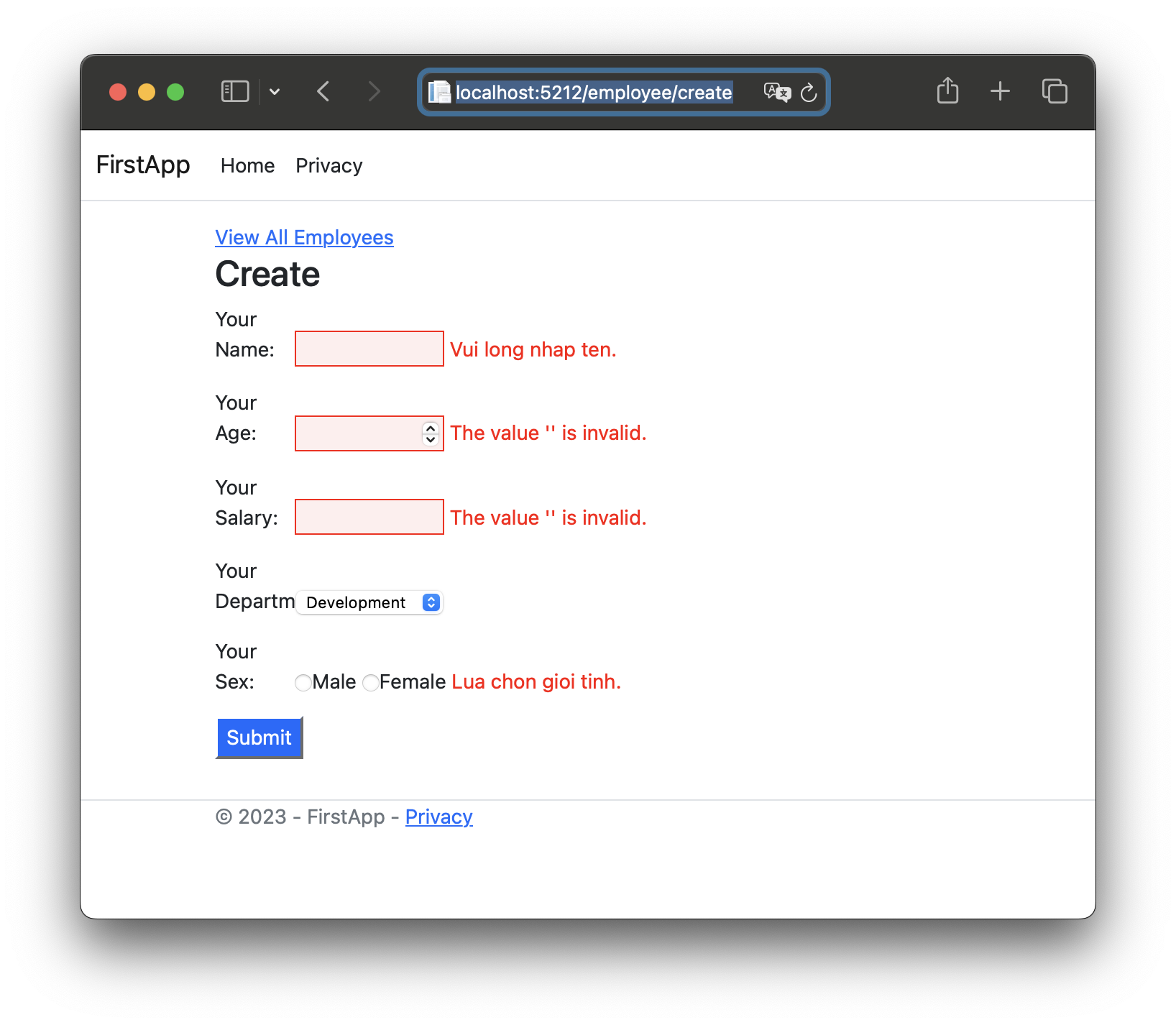
form p .input-validation-error {

border: 1px solid Red;

background-color: #fee;

}

Trang Create Employee sẽ như thế này:



Và tất cả các nhân viên được hiển thị trên bảng:

