

MỤC LỤC

CHƯƠNG 1 GIỚI THIỆU VỀ ASP.NET WEB API	3
1.1 Khái niệm API.....	3
1.2 Khái niệm Web API	3
1.2.1 Web API	3
1.2.2 Cấu trúc của 1 request.....	4
1.2.3 Cấu trúc của 1 Response.....	5
1.3 REST và RESTful API.....	5
1.4 APS.NET Web API.....	6
1.4.1 APS.NET Web API là gì	6
1.4.2 Đặc điểm của ASP.NET Web API	6
1.4.3 Các phiên bản của ASP.NET Web API.....	7
1.4.4 So sánh ASP.NET Web API và WCF	7
1.4.5 1.4.5. Khi nào nên chọn ASP.NET Web API?	7
1.5 Tạo ứng dụng Hello Web API.....	8
1.5.1 Tạo dự án ASP.NET Web API sử dụng Visual Studio	8
1.5.2 Cấu trúc của một ứng dụng ASP.NET Web API.....	9
CHƯƠNG 2 KIỂM TRA WEB API	15
2.1 Sử dụng Swagger để kiểm tra Web API.....	15
2.1.1 Swagger là gì?.....	15
2.1.2 Cài đặt Swagger	15
2.1.3 Cấu hình Swagger	15
2.1.4 Cách sử dụng Swagger	17
2.2 Sử dụng Postman để kiểm tra Web API.....	23
2.2.1 Postman là gì?	23
2.2.2 Cài đặt Postman	24
2.2.3 Cách sử dụng Postman.....	24
2.3 Sử dụng Fiddler để kiểm tra Web API	28
2.3.1 Fiddler là gì?	29
2.3.2 Cài đặt Fiddler	29
2.3.3 Cách sử dụng Fiddler	30
CHƯƠNG 3 ASP.NET WEB API SỬ DỤNG SQL SERVER.....	37
3.1 Tạo cơ sở dữ liệu	37
3.2 Tạo một Dự án Web API ASP.NET mới	38
3.3 Kết nối với ADO.NET Entity Framework	41
3.4 Thêm Web API Controller	48
CHƯƠNG 4 TẠO WEB API VỚI CRUD	52
4.1 Giới thiệu về CRUD	52
4.2 Triển khai phương thức GET	53
4.3 Triển khai phương thức POST	55
4.4 Triển khai phương thức PUT.....	58
4.5 Triển khai phương thức DELETE	62
4.6 Tùy chỉnh tên phương thức trong Web API controller	64
4.7 Tham số ràng buộc trong Web API.....	71
4.8 Thảo luận	77

CHƯƠNG 5 ĐỊNH TUYẾN WEB API	78
5.1 Giới thiệu về Định tuyến Web API ASP.NET	78
5.1.1 Bảng định tuyến trong Web API ASP.NET	78
5.1.2 WEB API framework xử lý yêu cầu HTTP request đến như thế nào?.....	78
5.2 Một số biến định tuyến trong WEB API	80
5.2.1 Các động từ HTTP Verb trong Web API	80
5.2.2 Định tuyến trong Web API theo tên phương thức.....	81
5.3 Định tuyến và phương thức thực thi trong Web API	82
5.3.1 Tuyến đường mẫu mặc định	82
5.3.2 Từ điển tuyến đường Route	83
5.3.3 Chọn bộ điều khiển	84
5.3.4 Lựa chọn phương thức.....	85
5.4 Định tuyến thuộc tính Web API	88
5.4.1 Giới thiệu	88
5.4.2 Tại sao chúng ta cần Định tuyến thuộc tính Web API?.....	89
CHƯƠNG 6 PHÁT TRIỂN ÚNG DỤNG SỬ DỤNG WEB API.....	94
6.1 Giới thiệu	94
6.2 Triển khai một Trang web với phương thức GET.....	94
6.3 Triển khai một Trang web với phương thức POST.....	111
6.4 Triển khai một Trang web với phương pháp PUT	119
6.5 Triển khai một Trang web với phương thức DELETE	128
CHƯƠNG 7 BẢO MẬT WEB API.....	132
7.1 Xác thực và Ủy quyền trong Web API.....	132
7.2 Xác thực cơ bản Web API ASP.NET	132
7.2.1 Tại sao chúng ta cần Xác thực trong Web API?.....	132
7.2.2 Xác thực cơ bản hoạt động như thế nào trong Web API?	132
7.2.3 Tạo mô hình	135
7.2.4 Tạo bộ lọc xác thực cơ bản trong Web API	139
7.2.5 Kiểm tra xác thực cơ bản Web API bằng Postman	144
7.3 Xác thực cơ bản dựa trên vai trò trong Web API	145
7.3.1 Tại sao chúng ta cần Xác thực Dựa trên Vai trò.....	146
7.3.2 Triển khai xác thực cơ bản dựa trên vai trò trong Web API	146
7.3.3 Kiểm tra Xác thực cơ bản dựa trên vai trò.....	155
7.4 Xác thực dựa trên mã thông báo trong Web API	156
7.4.1 Tại sao chúng ta cần Xác thực dựa trên mã thông báo trong Web API?..	156
7.4.2 Xác thực dựa trên mã thông báo hoạt động như thế nào?	157
7.4.3 Triển khai xác thực dựa trên mã thông báo trong Web API.....	157
7.4.4 Advantages of using Token Based Authentication.....	170
CHƯƠNG 8 ASP.NET CORE WEB API	172
8.1 Giới thiệu	172
8.2 Tại sao dùng ASP.NET Core?	172
8.3 Xây dựng các API và giao diện (UI) Web dùng ASP.NET Core MVC	173
8.4 Phát triển phía client	174

CHƯƠNG 1 GIỚI THIỆU VỀ ASP.NET WEB API

Hiện nay có rất nhiều lựa chọn ngôn ngữ để lập trình WebAPI như: Java, .NET (C#, VB), Ruby, Python, Perl, JavaScript (Node.js), Go, C++. Để lựa chọn ngôn ngữ lập trình trong những ngôn ngữ lập trình này, người ta thường cân nhắc nhiều khía cạnh như: đặc điểm cú pháp của các ngôn ngữ thì khả năng mở rộng, việc sử dụng các loại cơ sở dữ liệu khác nhau, , các yêu cầu về khả năng chịu lỗi, kích thước dữ liệu lớn, sự phổ biến của ngôn ngữ ưu nhược điểm của mỗi ngôn ngữ,... Trên thực tế một trong những ngôn ngữ được sử dụng nhiều nhất hiện nay .NET.

1.1 Khái niệm API

Theo Wikipedia API là “**An application programming interface (API)** is a computing interface which defines interactions between multiple software intermediaries.

API là từ viết tắt của “Application Programming Interface” với nghĩa là một giao diện lập trình ứng dụng. Đây là phần mềm trung gian cho phép kết nối 2 ứng dụng với nhau.

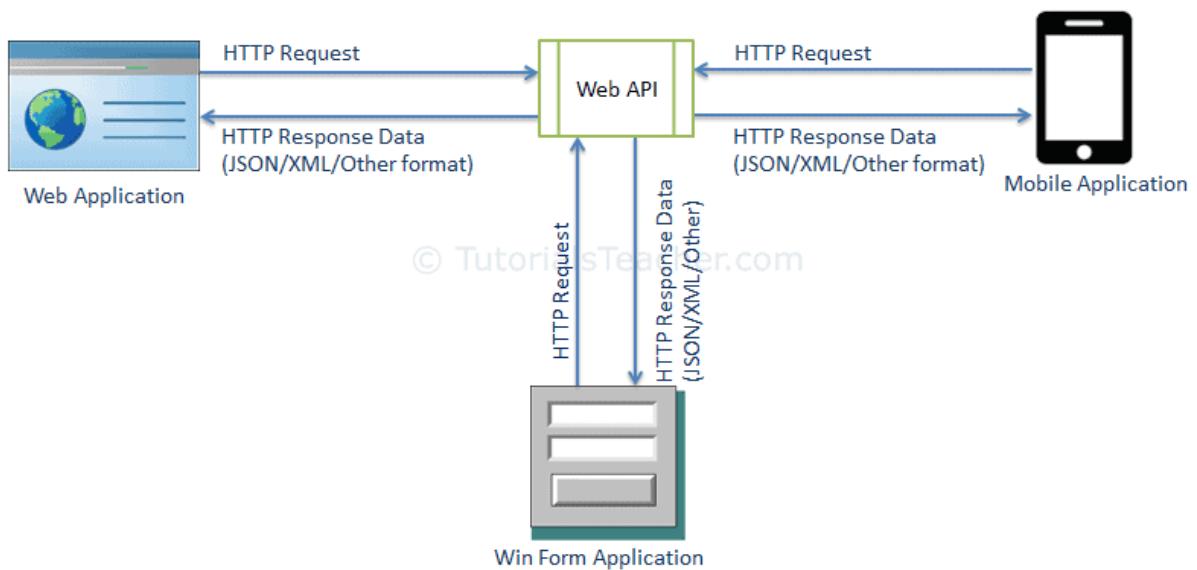
1.2 Khái niệm Web API

1.2.1 Web API

ASP.NET Web API là một khuôn khổ có thể mở rộng để xây dựng các dịch vụ dựa trên HTTP có thể được truy cập trong các ứng dụng khác nhau trên các nền tảng khác nhau như web, windows, thiết bị di động, v.v. Nó hoạt động ít nhiều giống với ứng dụng web ASP.NET MVC ngoại trừ rằng nó sẽ gửi dữ liệu dưới dạng phản hồi thay vì dạng xem html. Nó giống như một dịch vụ webservice hoặc WCF nhưng ngoại lệ là nó chỉ hỗ trợ giao thức HTTP.

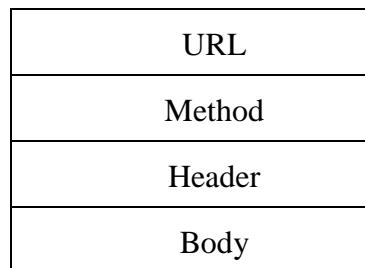
Web API là một API trên web có thể được truy cập bằng giao thức HTTP, HTTP có thể được truy cập trong các ứng dụng khác nhau trên các nền tảng khác nhau như web, windows, thiết bị di động, v.v. WebAPI hoạt động có nhiều điểm giống với ứng dụng web ASP.NET MVC, nó khác MVC ở chỗ nó sẽ gửi dữ liệu dưới dạng phản hồi thay vì dạng html. WebAPI giống như một dịch vụ webservice hoặc WCF nhưng ngoại lệ là nó chỉ hỗ trợ giao thức HTTP. Nó là một khái niệm và không phải là một công nghệ.

Các dịch vụ Web API có thể được sử dụng bởi nhiều loại clients khác nhau thông qua các request. Ví dụ: Browsers, Mobile applications, Desktop applications, IOTs, ...



Hình 1. Hình 1.1 Web API

1.2.2 Cấu trúc của 1 request



Hình 2. Hình 1.2 Cấu trúc request

Trong đó:

URL là 1 cái địa chỉ duy nhất

Method: là cái hành động client muốn tác động lên “resources”, và nó thường là động từ. Có 4 loại Method (Create – POST, Read – GET, Update – PUT, Delete – DELETE) được dùng:

– GET: Yêu cầu server đưa lại data

– POST: Yêu cầu server cho tạo ra 1 resource mới.– PUT: Yêu cầu server cho sửa / thêm vào resource đã có trên hệ thống.

– DELETE: Yêu cầu server cho xóa 1 resource.

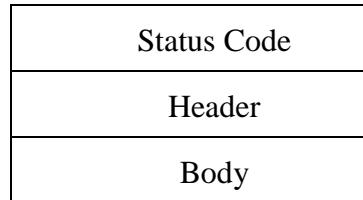
Header: nơi chứa các thông tin cần thiết của 1 request nhưng end-users không biết có sự tồn tại của nó. Ví dụ: độ dài của request body, thời gian gửi request, loại thiết bị đang sử dụng, loại định dạng cái response mà client có đọc được...

Body: nơi chứa thông tin mà client sẽ điền. Giả sử bạn đặt 1 cái bánh pizza, thì thông tin ở phần body sẽ là: Loại bánh pizza, kích cỡ, số lượng đặt.

Response:

Sau khi nhận được request từ phía client, server sẽ xử lý cái request đó và gửi ngược lại cho client 1 cái response.

1.2.3 Cấu trúc của 1 Response



Hình 3. Hình 1.3 Cấu trúc Response

Trong đó

Status code là những con số có 3 chữ số và có duy nhất 1 ý nghĩa. Chắc các bạn cũng không còn lạ lẫm với những Error “404 Not Found” hoặc “503 Service Unavailable”. Full list có ở đây.

Header và **Body** tương đối giống với request.

Một số thuộc tính của Header như sau

Content-type: Giá trị content-type header trong request được gửi từ Client tới Web API Server, có 1 vài định dạng giá trị như sau

Content-Type: application/xml trả về kiểu dữ liệu dạng XML

Content-Type: application/json trả về kiểu dữ liệu dạng JSON

Content-Type: Text trả về kiểu dữ liệu dạng văn bản thông thường text

Content-Type: Html trả về kiểu dữ liệu dạng văn bản thông thường Html

Content-Type: JavaScript trả về kiểu dữ liệu dạng văn bản thông thường JavaScript

Accept: Giá trị Accept header xác định media types, nó được chấp nhận cho phản hồi từ Web API Server về Client, có 1 vài định dạng giá trị như content-type header

Accept-Charset: Accept-Charset header xác định các tập ký tự nào được chấp nhận. Ví dụ: UTF-8 or ISO 8859-1.

Accept-Encoding: Accept-Encoding header xác định việc mã hóa nội dung nào được chấp nhận, ví dụ như gzip.

Accept-Language: Accept-Language header xác định việc ngôn ngữ tự nhiên hỗ trợ như en-us

1.3 REST và RESTful API

REST (REpresentational State Transfer) về cơ bản là một kiểu kiến trúc để phát triển ứng dụng dịch vụ web. REST phổ biến do tính đơn giản của nó và thực tế là nó được xây dựng dựa trên các

hệ thống và tính năng hiện có của Giao thức truyền siêu văn bản trên internet (HTTP) để đạt được các mục tiêu của nó, thay vì tạo ra các tiêu chuẩn, khuôn khổ và công nghệ mới.

RESTful API là một kiểu kiến trúc gọi REST sử dụng các yêu cầu GET, PUT, POST và DELETE tương ứng với thao tác với dữ liệu là Read, Update, Create và Delete.

REST có một số nguyên tắc như sau:

- REST cho phép các máy khách truy cập máy chủ API và thực hiện các lệnh lấy về, chỉnh sửa hay xóa dữ liệu từ external server. Các lập trình viên có thể thoải mái truy xuất, chỉnh sửa dữ liệu từ máy chủ mà không cần biết hệ thống hoạt động như thế nào.
- Giao thức chính của REST sử dụng là HTTP, một giao thức phổ biến với hầu hết các ứng dụng hay dịch vụ web hiện nay. Nó đem lại hiệu quả nhanh chóng trong bối cảnh đường truyền mạnh mẽ và khiến cho REST kiến trúc tốc độ nhanh hơn.
- REST cũng có ưu điểm khi sử dụng giao thức stateless (không trạng thái). Hệ thống này không sử dụng session, cookie, không cần biết những thông tin đó trong mỗi lần request đến máy chủ ngoài. Điều này giúp REST giảm tải cho máy chủ ngoài, nâng cao hiệu suất làm việc.
- REST ban đầu được thiết kế để sử dụng cho các dịch vụ web. Tuy nhiên, bất cứ phần mềm nào cũng có thể ứng dụng REST làm cầu nối giao tiếp với các ứng dụng khác tốt và hiệu quả hơn. Đó cũng là lý do giúp cho REST trở thành tiêu chuẩn mặc định cho hầu hết các giao thức. Và những API được thiết kế theo cấu trúc REST được gọi là RESTful API.
- Nó sẽ chỉ trả về kết quả dưới dạng JSON hoặc XML, nguyên tử, OData, v.v. (dữ liệu nhẹ)

Trong đó

- REST based services Thực hiện theo một số nguyên tắc trên và không phải tất cả
- RESTFUL services có nghĩa là nó tuân theo tất cả các nguyên tắc trên.

1.4 APS.NET Web API

1.4.1 APS.NET Web API là gì

The **ASP.NET Web API** là một framework có khả năng mở rộng để xây dựng các dịch vụ dựa trên HTTP, nó có thể truy cập trong các ứng dụng khác nhau trên các nền tảng khác nhau như web, windows, mobile, v.v.

Cách thức hoạt động của **ASP.NET Web API** hầu như giống như cách thức ứng dụng web ASP.NET MVC, **ASP.NET Web API** hoạt động khác ASP.NET MVC ở chỗ nó gửi dữ liệu dưới dạng các respond (phản hồi) thay vì trả lại các html view.

ASP.NET Web API giống như một webservice hoặc WCF service nhưng nó khác là nó chỉ hỗ trợ giao thức HTTP.

Các dịch vụ Web API có thể được sử dụng bởi nhiều loại clients khác nhau thông qua các request. Ví dụ: Browsers, Mobile applications, Desktop applications, IOTs, ...

1.4.2 Đặc điểm của ASP.NET Web API

- Là platform lý tưởng cho việc xây dựng RESTful services.

- Kiến trúc lý tưởng cho các thiết bị có băng thông giới hạn như các thiết bị di động.
- Giúp cho việc xây dựng các HTTP service rất đơn giản, nhanh chóng, maps các HTTPGet/Post/Delete với tên của phương thức
- Mã nguồn mở (Open Source) và có thể được sử dụng bởi bất kỳ client nào hỗ trợ XML, JSON.
- Hỗ trợ đầy đủ các thành phần HTTP: URI, request/response headers, caching, versioning, content formats.
- Có thể host trong ứng dụng hoặc trên IIS, Self-hosted hoặc web server (yêu cầu .NET 4.0 trở lên)
- Hỗ trợ nhiều định dạng dữ liệu khác nhau như có thể là JSON, XML, BSON hoặc một kiểu dữ liệu bất kỳ.

1.4.3 Các phiên bản của ASP.NET Web API

Web API Version	Supported .NET Framework	Coincides with	Supported in
Web API 1.0	.NET Framework 4.0	ASP.NET MVC 4	VS 2010
Web API 2 - Current	.NET Framework 4.5	ASP.NET MVC 5	VS 2012, 2013

1.4.4 So sánh ASP.NET Web API và WCF

Bảng 1.1 So sánh ASP.NET Web API và WCF

Web API	WCF
Mã nguồn mở và gắn liền với .NET framework	Gắn liền với .NET framework
Chỉ hỗ trợ giao thức HTTP	Hỗ trợ nhiều giao thức như: HTTP, TCP, UDP,...
Map các http verbs tới tên phương thức	Sử dụng thuộc tính dựa vào mô hình lập trình
Sử dụng định tuyến và controller tương tự như ASP.NET MVC	Sử dụng Service, Operation and Data contracts.
Không hỗ trợ Nhắn tin và giao dịch tin cậy	Hỗ trợ Nhắn tin và giao dịch tin cậy
Có thể được định cấu hình bằng lớp HttpConfiguration nhưng không phải dịch vụ trong web.config	Sử dụng web.config và các thuộc tính để định cấu hình
Lý tưởng để xây dựng các dịch vụ RESTful	Hỗ trợ giới hạn các dịch vụ RESTful

1.4.5 Khi nào nên chọn ASP.NET Web API?

- Sử dụng .NET framework 4.0 trở lên.

- Xây dựng webservice chỉ hỗ trợ giao thức HTTP protocol.
- Xây dựng webservice RESTful HTTP based services.
- Biết về ASP.NET MVC

1.5 Tạo ứng dụng Hello Web API

1.5.1 Tạo dự án ASP.NET Web API sử dụng Visual Studio

Các bước xây dựng một dự án ASP.NET Web API như sau

Bước 1: Đầu tiên mở ứng dụng Visual Studio

Khởi động Visual Studio → tạo một project ASP.NET Web Application và chọn template Web API.

Bước 2:

Tạo Data Model sử dụng Entity Framework để Web API service có thể tương tác CRUD (Create, Read, Update, Delete) dữ liệu được (không tạo cũng được). Mỗi bảng là 1 Controller

Bước 3: Tạo ra các Web API

Nhấp chuột phải vào thư mục Controllers và chọn thêm controller.

Web API 2 Controller Empty: tự viết các phương thức từ đầu.

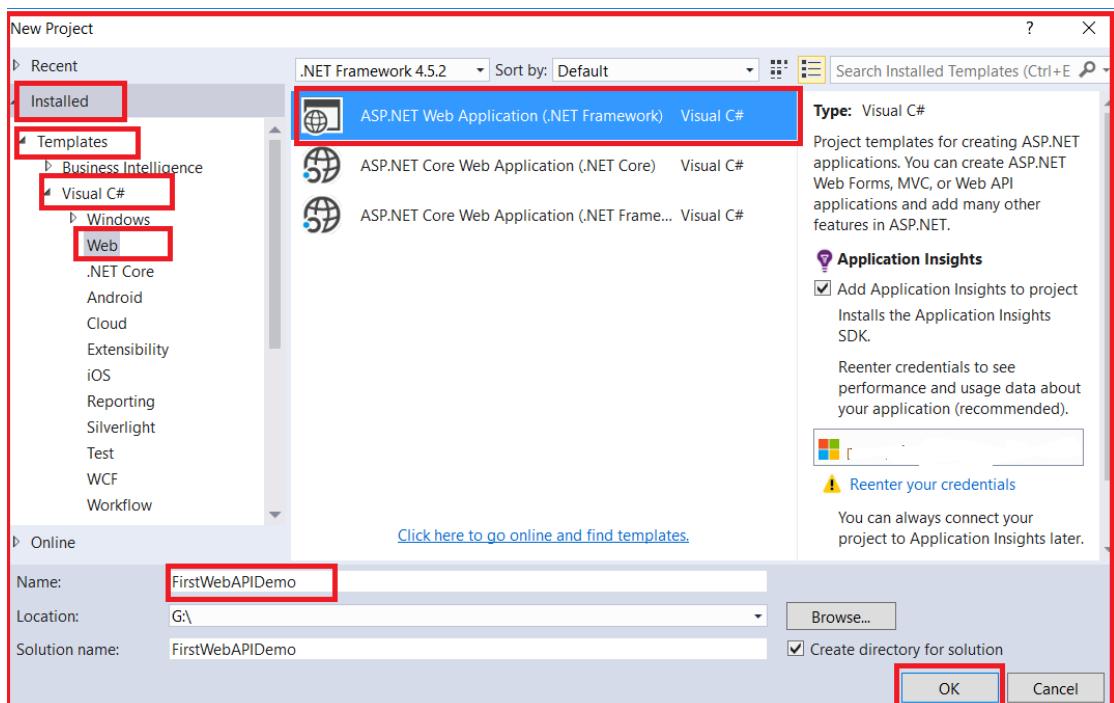
Web API 2 Controller with read/write actions: phát sinh các phương thức ví dụ để bạn có thể biết cách viết các service này.

Bước 4: Chạy thử và kiểm tra

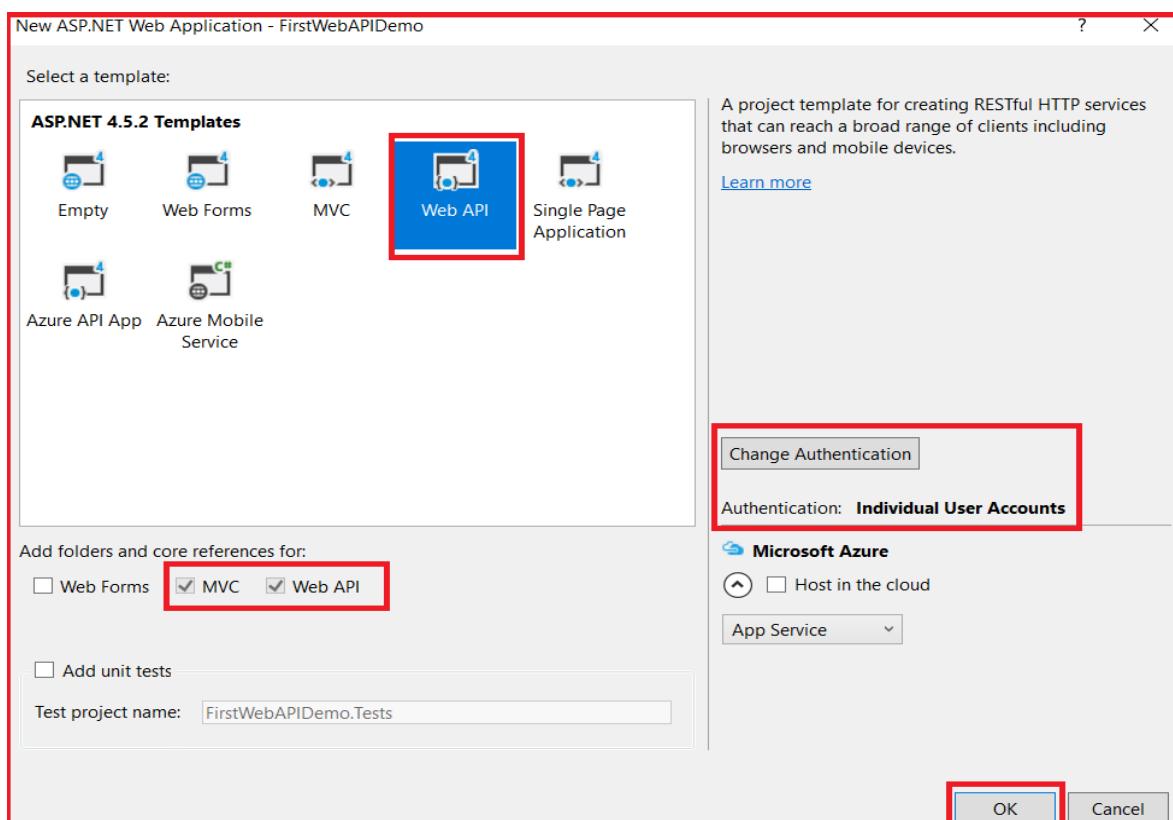
Ta thực hiện xây dựng ứng dụng Hello Web API như sau

1.5.1.1 Bước 1: Khởi động Visual Studio

Chọn Create a new project để tạo một project ASP.NET Web Application



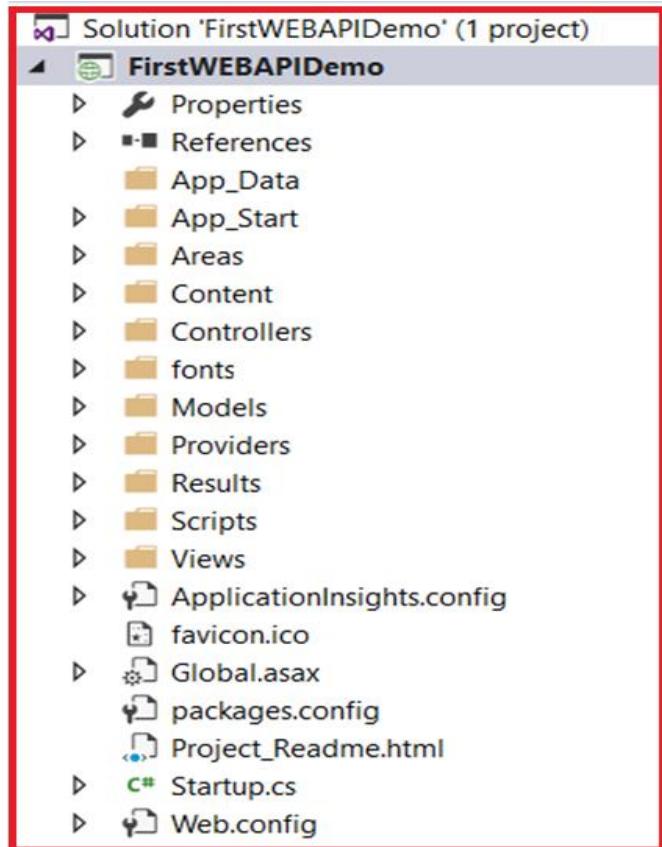
Sau khi bạn nhấp vào nút **OK** sau đó một cửa sổ mới sẽ mở ra với **Tên Dự án ASP.NET** Mới để chọn **Mẫu dự án** và từ cửa sổ đó chọn **mẫu dự án Web API** và nhấp vào nút **OK** như thể hiện trong hình dưới đây



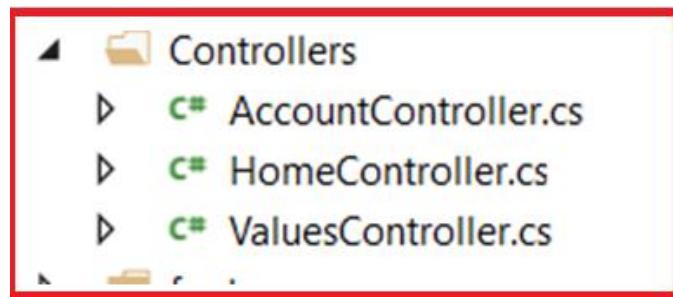
Sau khi bạn nhấp vào nút **OK**, sẽ mất một khoảng thời gian để tạo dự án.

1.5.2 Cấu trúc của một ứng dụng ASP.NET Web API

Nếu bạn đã làm việc với **ASP.NET MVC Framework**, thì cấu trúc thư mục dự án sẽ quen thuộc với bạn như hình dưới đây.



Trong ứng dụng WebAPI vừa tạo, có các thư mục riêng biệt (hình trên) như: View, Model, Controller. Trong đó, khi tạ project với sự lựa chọn API template thì có sẵn một số lớp HomeController và ValuesController



Điều quan trọng cần ghi nhớ là Controller Web API khác với Controller MVC. ValuesController là Controller WebAPI còn HomeController và Controller MVC

```
[Authorize]
public class ValuesController : ApiController
{
    // GET api/values
    public IEnumerable<string> Get() { ... }

    // GET api/values/5
    public string Get(int id) { ... }

    // POST api/values
    public void Post([FromBody]string value) { ... }

    // PUT api/values/5
    public void Put(int id, [FromBody]string value) { ... }

    // DELETE api/values/5
    public void Delete(int id) { ... }
}
```

Lớp **ValuesController** ở trên sẽ thấy rằng nó kế thừa từ lớp **ApiController** có trong tên **System.Web.Http**.

Hơn nữa, nếu bạn nhận thấy rằng lớp **HomeController** là một Controller MVC, được kế thừa từ lớp Controller có trong **System.Web.Mvc** như được hiển thị trong hình dưới đây.

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Title = "Home Page";

        return View();
    }
}
```

Ở đây ta sẽ tập trung vào Controller Web API (tức là **ValuesController**).

Lưu ý rằng trong **ValuesController** chúng ta có các phương thức như Get, Put, Post và Delete, ánh xạ tới các động từ HTTP GET, PUT, POST và DELETE tương ứng như trong hình bên dưới.

```

[Authorize]
public class ValuesController : ApiController
{
    // GET api/values
    public IEnumerable<string> Get()
    {
        return new string[] { "value1", "value2" };
    }

    // GET api/values/5
    public string Get(int id)
    {
        return "value";
    }

    // POST api/values
    public void Post([FromBody]string value)
    {
    }

    // PUT api/values/5
    public void Put(int id, [FromBody]string value)
    {
    }

    // DELETE api/values/5
    public void Delete(int id)
    {
    }
}

```

Có 2 phiên bản được nạp chồng của phương thức Get () - Một phương thức không có bất kỳ tham số nào và một phương thức còn lại có tham số id. Cả hai phương thức này đều phản hồi động từ GET HTTP tùy thuộc vào việc tham số id có được truyền hay không trong URL.

Bây giờ ta xem xét định tuyến (routing) cho dự án Web API. Phương **thức Application_Start()** trong tệp **Global.asax**. Phương thức này được thực thi khi ứng dụng khởi động lần đầu tiên. Trong phương thức **Application_Start()**, chúng ta có cấu hình **cho Bộ lọc, Gói, v.v.** như hình dưới đây.

```

public class WebApiApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        GlobalConfiguration.Configure(WebApiConfig.Register);
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
}

```

Cấu hình cho dự án Web API nằm trong phương thức **WebApiConfig.Register()**. Vì vậy, nhấp chuột phải vào phương thức **WebApiConfig.Register** và sau đó chọn “**Go To Definition**” đến phương **thức Register()** của lớp **WebApiConfig** như hình dưới đây.

```

public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services
        // Configure Web API to use only bearer token authentication.
        config.SuppressDefaultHostAuthentication();
        config.Filters.Add(new HostAuthenticationFilter(OAuthDefaults.AuthenticationType));

        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}

```

Lớp này nằm trong thư mục App_Start.

Định tuyến mặc định được định cấu hình trong phương thức Register () cho dự án Web API trong tệp RouteConfig.cs có trong thư mục App_Start. Lộ trình mặc định trong Web API bắt đầu bằng từ API sau là dấu / và sau đó là tên của bộ điều khiển và sau đó là một / và một tham số id tùy chọn như được hiển thị bên dưới.

“Api / {controller} / {id}”

Chú ý! Nếu sử dụng URI sau trong trình duyệt, bạn sẽ nhận được thông báo lỗi - Ủy quyền đã bị từ chối cho yêu cầu này.

<http://localhost:xxxxx/api/values>

Để loại bỏ lỗi này, hãy bao khai báo thuộc tính Authorize trên lớp ValuesController. Điều này liên quan đến bảo mật mà chúng ta sẽ thảo luận trong phần sau.

Bây giờ nếu bạn phát hành URI <http://localhost:xxxxx/api/values> thì ta sẽ thấy kết quả là XML sau

```

▼<ArrayOfstring xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <string>value1</string>
  <string>value2</string>
</ArrayOfstring>

```

Hãy để chúng tôi hiểu những gì đang xảy ra ở đây.

Tên của controller là "values". Vì vậy, nếu ta sử dụng URI <http://localhost:portnumber/api/values>, thì Web API Framework sẽ tìm kiếm Controller có tên là Values, tức là ValuesController trong ứng dụng.

Vì vậy, nếu ta đã chỉ định các Controller trong URI, nó sẽ tìm kiếm Controller mà ta chỉ định. Ví dụ, nếu chỉ định Values thì nó sẽ tìm kiếm ValuesController, nếu chỉ định Products thì nó sẽ tìm kiếm ProductsController.

Trong một ứng dụng thế giới thực, đây có thể là tên miền, trình duyệt đang đưa ra yêu cầu GET ánh xạ tới phương thức Get () trong lớp ValuesController. GET () trong bộ điều khiển giá trị đang trả về giá value1 và giá value2 là những gì chúng ta thấy trong trình duyệt.

Chúng ta có một phương thức GET () trong ValuesController nhận tham số Id. Nếu ta để định tuyến mặc định trong tệp WebApiConfig, thì tham số id là tùy chọn. Đây là lý do tại sao chúng ta có thể gọi phương thức GET có hoặc không có tham số Id. Vì vậy, nếu chỉ định tham số id trong URI, thì phương thức Get () với tham số id trong ValuesController sẽ được gọi.

Nếu một Controller có tên được chỉ định không được tìm thấy bởi Web API FrameWork, thì FrameWork sẽ bị lỗi. Ví dụ: trong ứng dụng nếu không tồn tại lớp “ValuesController” trong dự án nhưng vẫn sử dụng URI/api/values thì bạn sẽ gặp lỗi sau

Không tìm thấy tài nguyên HTTP nào phù hợp với URI yêu cầu ‘http://localhost:15648/api/values’. Không tìm thấy loại nào phù hợp với bộ điều khiển có tên 'giá trị'.

CHƯƠNG 2 KIỂM TRA WEB API

2.1 Sử dụng Swagger để kiểm tra Web API

Trong phần này, chúng ta sẽ thảo luận về cách thêm Swagger trong Ứng dụng Web API vào tài liệu và thử nghiệm các dịch vụ Web API còn lại. Như một phần của phần này, chúng ta sẽ thảo luận về các điểm sau..

2.1.1 Swagger là gì?

Swagger là một phần mềm mã nguồn mở được sử dụng để phát triển, thiết kế, xây dựng và làm tài liệu cho các RESTful Web Service.

Swagger vừa help vừa test cho web API xây dựng, với Swagger ta có thể biết rõ được có những API nào, method và url tương ứng của nó

Với mỗi api ta lại biết được chi tiết input và output của như trường nào bắt buộc gửi lên, kết quả trả về có thể nhận những status nào...

2.1.2 Cài đặt Swagger

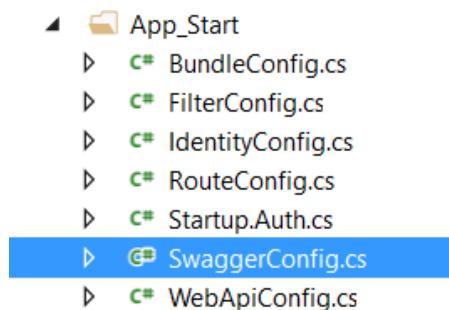
Thường cài Swagger cho ứng dụng WebAPI bằng 2 cách

Cách 1

Để thêm **Swagger** vào dự án **ASP.NET Web API** của bạn, bạn cần cài đặt một dự án mã nguồn mở có tên **Swashbuckle** thông qua **NuGet** như hình dưới đây.

```
PM> Install-Package Swashbuckle -Version 5.2.1
```

Khi gói được cài đặt thành công, hãy điều hướng đến thư mục **App_Start** trong **Solution Explorer**. Bạn sẽ tìm thấy một tệp mới có tên **SwaggerConfig.cs**. Đây là tệp mà **Swagger** được bật và mọi tùy chọn cấu hình nên được đặt ở đây..



2.1.3 Cấu hình Swagger

Để bật giao diện người dùng **Swagger** và **Swagger**, hãy sửa đổi lớp **SwaggerConfig** như được hiển thị bên dưới

```
SwaggerConfig.cs
```

```
namespace FirstWebAPIDemo
```

```
{
```

```
    public class SwaggerConfig
```

```
{
```

```
    public static void Register()
```

```
{
```

```
        var thisAssembly = typeof(SwaggerConfig).Assembly;
```

```
        GlobalConfiguration.Configuration
```

```
            .EnableSwagger(c => c.SingleApiVersion("v1", "First WEB API Demo"))
```

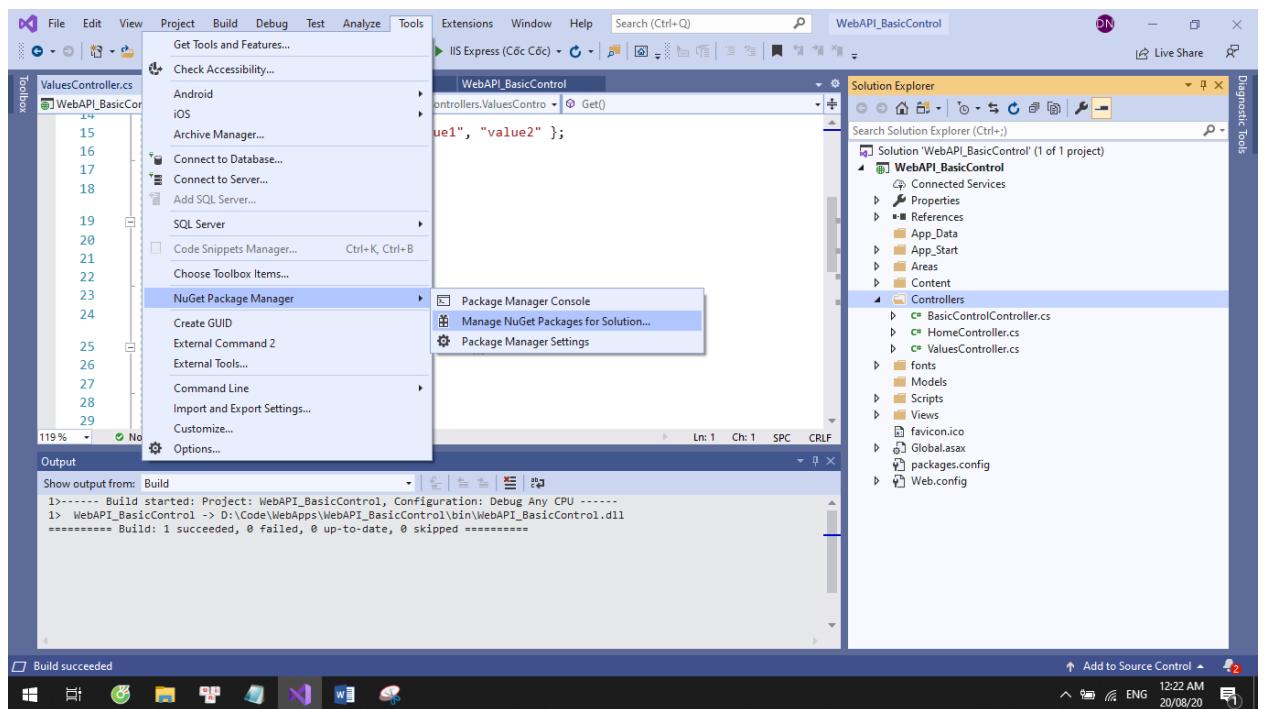
```
            .EnableSwaggerUi();
```

```
}
```

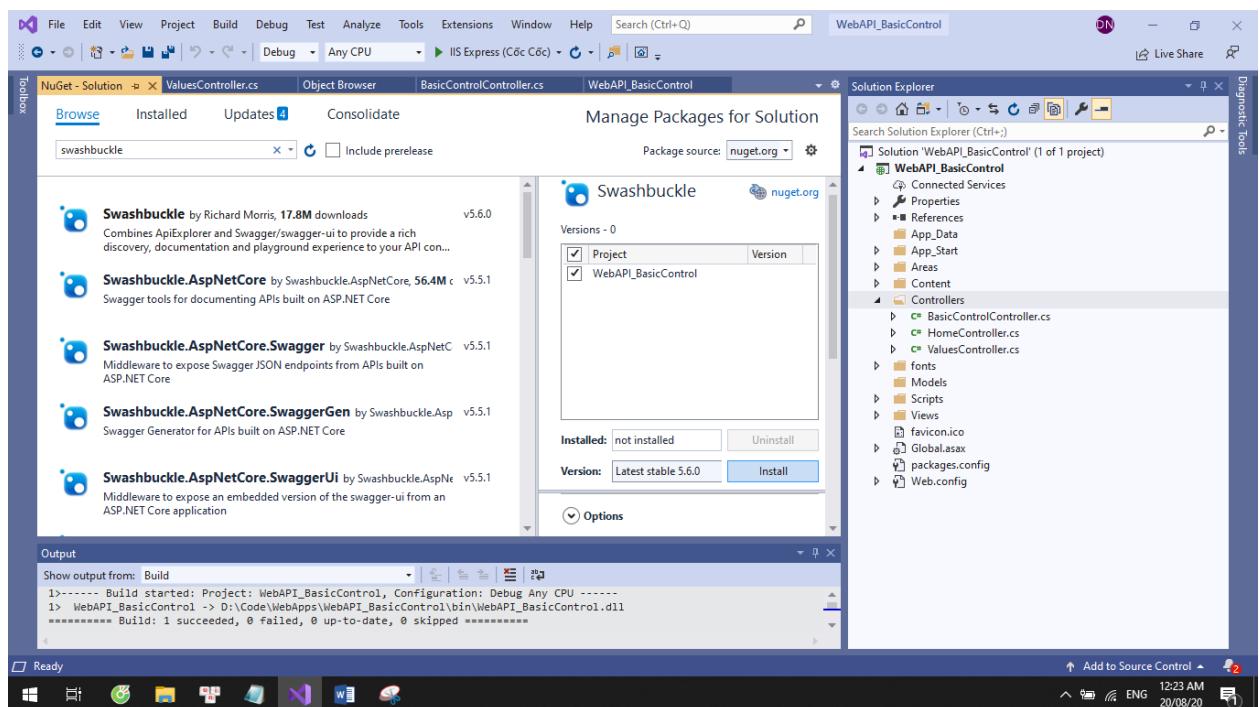
```
}
```

```
}
```

Cách 2 Mở Manage NuGet Package



Search package swashbuckler



Chọn Install



Hiện biểu tượng như trên là việc cài package đã hoàn tất

Với cách này, Swagger sẽ tự tích hợp vào ứng dụng và không cần cấu hình.

2.1.4 Cách sử dụng Swagger

Bắt đầu phiên gõ lỗi mới bằng cách nhấn phím **F5** và điều hướng đến **http://localhost:[PORT_NUM]/swagger** và sau đó bạn sẽ thấy các trang trợ giúp cho các **API** của mình.

The screenshot shows the Swagger UI homepage for a 'First WEB API Demo'. At the top, there's a header with the URL 'localhost:59281/swagger/ui/index'. Below the header, the word 'swagger' is displayed in a green bar along with the URL 'http://localhost:59281/swagger/docs/v1', an 'api_key' input field, and a 'Explore' button. The main content area is titled 'First WEB API Demo' and contains two sections: 'Account' and 'Values'. The 'Values' section lists several operations: GET /api/Values (blue), POST /api/Values (green), DELETE /api/Values/{id} (red), GET /api/Values/{id} (blue), and PUT /api/Values/{id} (orange). Each operation has a 'Show/Hide' link, a 'List Operations' link, and an 'Expand Operations' link.

First WEB API Demo

Account

Show/Hide | List Operations | Expand Operations

Values

Show/Hide | List Operations | Expand Operations

GET /api/Values

POST /api/Values

DELETE /api/Values/{id}

GET /api/Values/{id}

PUT /api/Values/{id}

[BASE URL: , API VERSION: V1]

Đồng ý. Thật tuyệt. Bây giờ hãy mở rộng một API và sau đó nhấp vào nút “**Try it out!**” nút này sẽ thực hiện cuộc gọi đến API cụ thể đó và trả về kết quả như được hiển thị trong hình ảnh bên dưới.

The screenshot shows the 'Values' endpoint page from the Swagger UI. At the top, there's a header with the URL 'localhost:59281/swagger/ui/index'. Below the header, the word 'swagger' is displayed in a green bar along with the URL 'http://localhost:59281/swagger/docs/v1', an 'api_key' input field, and a 'Explore' button. The main content area is titled 'First WEB API Demo' and contains two sections: 'Account' and 'Values'. The 'Values' section lists the 'GET /api/Values' operation. Below this, there's a 'Response Class (Status 200)' section with tabs for 'Model' (selected) and 'Model Schema'. The 'Model Schema' tab shows a JSON schema: ["string"]. Further down, there's a 'Response Content Type' dropdown set to 'application/json' and a 'Try it out!' button.

First WEB API Demo

Account

Show/Hide | List Operations | Expand Operations

Values

Show/Hide | List Operations | Expand Operations

GET /api/Values

Response Class (Status 200)

Model | Model Schema

```
[  
    "string"  
]
```

Response Content Type application/json ▾

Try it out!

Here click on the Try it out Button which will display the result as shown below.

Request URL

```
http://localhost:59281/api/Values
```

Response Body

```
[  
    "value1",  
    "value2"  
]
```

Response Code

```
200
```

Response Headers

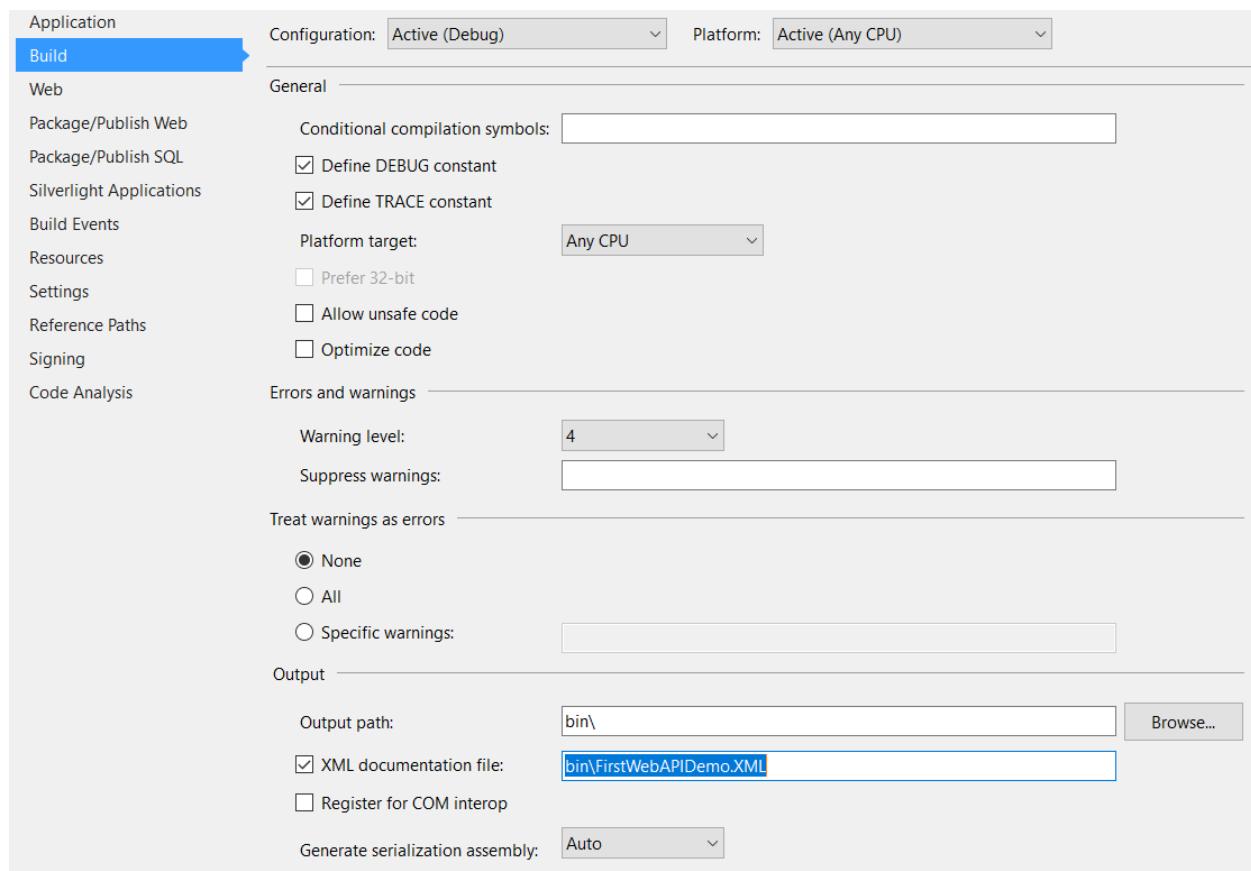
```
{  
    "pragma": "no-cache",  
    "date": "Sat, 21 Jul 2018 04:38:48 GMT",  
    "server": "Microsoft-IIS/10.0",  
    "x-aspNet-version": "4.0.30319",  
    "x-powered-by": "ASP.NET",  
    "content-type": "application/json; charset=utf-8",  
    "cache-control": "no-cache",  
    "x-sourcefiles": "=?UTF-8?B?RzpcRmlyc3RXZWJBUE1EZW1vXEZpcnN0V2ViQVBjRGVtb1xhcGlcVmFsdWVz?=",  
    "content-length": "19",  
    "expires": "-1"  
}
```

Theo cách tương tự, bạn có thể kiểm tra tất cả các phương pháp khác.

Làm cách nào để bật Swagger sử dụng Nhận xét XML trong Ứng dụng Web API ASP.NET?

Tô thè thiết lập **Swashbuckle** sử dụng các nhận xét XML tùy chỉnh để thêm các chi tiết khác về các API vào **Swagger** metadata.

Đầu tiên, cần kích hoạt tính năng tạo tệp tài liệu XML trong quá trình xây dựng. Kích chuột phải vào Solution Explorer trong the Web API project, nhấp vào tab Properties hướng đến Output. Đảm bảo rằng tệp tài liệu XML đã được kiểm tra, đường dẫn tệp mặc định. Trong trường hợp của chúng tôi, **bin \ FirstWebAPIDemo.XML** của nó như được hiển thị bên dưới



Tiếp theo, chúng ta cần yêu cầu **Swashbuckle** đưa các nhận xét XML của chúng ta vào siêu dữ liệu Swagger. Để làm điều này, chúng ta cần thêm dòng sau vào **SwaggerConfig.cs**. Đảm bảo thay đổi đường dẫn tệp thành đường dẫn tệp tài liệu XML.

```
c.IncludeXmlComments(string.Format(@"{0}\bin\FirstWebAPIDemo.XML",
System.AppDomain.CurrentDomain.BaseDirectory));
```

SwaggerConfig.cs

```
namespace FirstWebAPIDemo
{
    public class SwaggerConfig
    {
        public static void Register()
        {
            var thisAssembly = typeof(SwaggerConfig).Assembly;
            GlobalConfiguration.Configuration
                .EnableSwagger(c =>
            {
                c.SingleApiVersion("v1", "First WEB API Demo");
                c.IncludeXmlComments(string.Format(@"{0}\bin\FirstWebAPIDemo.XML",

```

```
System.AppDomain.CurrentDomain.BaseDirectory));  
    })  
    .EnableSwaggerUi();  
}  
}  
}
```

Hãy thêm một số tài liệu XML vào các phương thức API của chúng tôi như được hiển thị bên dưới. Ở đây chúng tôi đang thêm Tài liệu XML vào phương thức get. Sửa đổi phương thức Get như hình dưới đây.

Adding XML document to Get method

```
/// <summary>  
/// Get All the Values  
/// </summary>  
/// <remarks>  
/// Get All the String Values  
/// </remarks>  
/// <returns></returns>  
public IEnumerable<string> Get()  
{  
    return new string[] { "value1", "value2" };  
}
```

Chạy ứng dụng và điều hướng trở lại / **swagger**. Bạn sẽ thấy thêm chi tiết được thêm vào tài liệu API của mình như được hiển thị bên dưới.

First WEB API Demo

The screenshot shows the Swagger UI interface for a Web API. At the top, there's a navigation bar with 'Account' and buttons for 'Show/Hide', 'List Operations', and 'Expand Operations'. Below this, under 'Values', there's another set of buttons for 'Show/Hide', 'List Operations', and 'Expand Operations'. A 'GET' button is highlighted, pointing to the URL '/api/Values'. To the right of the URL is a link 'Get All the Values'. Below the URL, there's a section titled 'Implementation Notes' with the sub-section 'Get All the String Values'. Underneath that is a 'Response Class (Status 200)' section. It contains tabs for 'Model' and 'Model Schema'. The 'Model Schema' tab is active, showing a JSON schema:

```
[{"type": "string"}]
```

. Below the schema is a 'Response Content Type' dropdown set to 'application/json'. At the bottom left is a 'Try it out!' button.

Ví dụ Một ví dụ test WebAPI khác với swagger như sau

The screenshot shows a browser window displaying the Swagger UI for a Web API named 'WebAPI_BasicControl'. The title bar says 'cốc cốc' and the address bar shows 'localhost:44321/swagger/ui/index#/'. The main content area has a heading 'BasicControl'. Under 'BasicControl', there are three buttons: 'GET /api/BasicControl/{id}' (blue), 'GET /api/BasicControl' (blue), and 'PUT /api/BasicControl' (orange). Below this is a section titled 'Values' with the following buttons: 'GET /api/Values' (blue), 'POST /api/Values' (green), 'DELETE /api/Values/{id}' (red), 'GET /api/Values/{id}' (blue), and 'PUT /api/Values/{id}' (orange). At the bottom of the page, there's a note '[BASE URL: , API VERSION: v1]' and a taskbar with various icons.

Nhập giá trị input là các tham số Parameter

Chọn Try it out để gửi request tới webAPI

Để lấy kết quả đầu ra ta xem phản hồi response

Mở vào từng mục để test ứng dụng Web API

Tham khảo thêm <http://petstore.swagger.io/>

2.2 Sử dụng Postman để kiểm tra Web API

Trong phần này, chúng ta sẽ thảo luận về cách sử dụng POSTMAN để kiểm tra Dịch vụ Web API với các ví dụ. Chúng tôi sẽ làm việc với cùng một ví dụ mà chúng tôi đã tạo trong phần Tạo ứng dụng web API đầu tiên, vì vậy vui lòng đọc phần đó trước khi tiếp tục phần này. Như một phần của phần này, chúng ta sẽ thảo luận về các điểm sau..

2.2.1 Postman là gì?

Postman là công cụ để test API của công ty Postdot Technologies được bắt đầu phát triển từ năm 2012. Hiện tại Postman có 3 phiên bản: Postman, Postman Pro (2016) và Postman Enterprise (2017).

Postman là ứng dụng phổ biến và mạnh mẽ để thử nghiệm các dịch vụ web. Postman giúp dễ dàng kiểm tra các API Restful Web, cũng như phát triển và tạo tài liệu cho các API Restful bằng cách cho phép người dùng tập hợp các HTTP đơn giản và phức tạp.

Ưu điểm:

- Dễ sử dụng, hỗ trợ cả chạy bằng UI và non-UI.
- Hỗ trợ viết code cho assert tự động bằng Javascript.
- Hỗ trợ cả RESTful services và SOAP services.
- Có chức năng tạo API document.

Nhược điểm:

- Những bản tính phí mới hỗ trợ những tính năng advance: Làm việc theo team, upport trực tiếp...

2.2.2 Cài đặt Postman

Phiên bản ứng dụng đóng gói của **Postman** cung cấp nhiều tính năng nâng cao bao gồm hỗ trợ **OAuth 2.0**. Phiên bản trong trình duyệt bao gồm một số tính năng, chẳng hạn như hỗ trợ cookie phiên, chưa có sẵn trong phiên bản ứng dụng đóng gói.

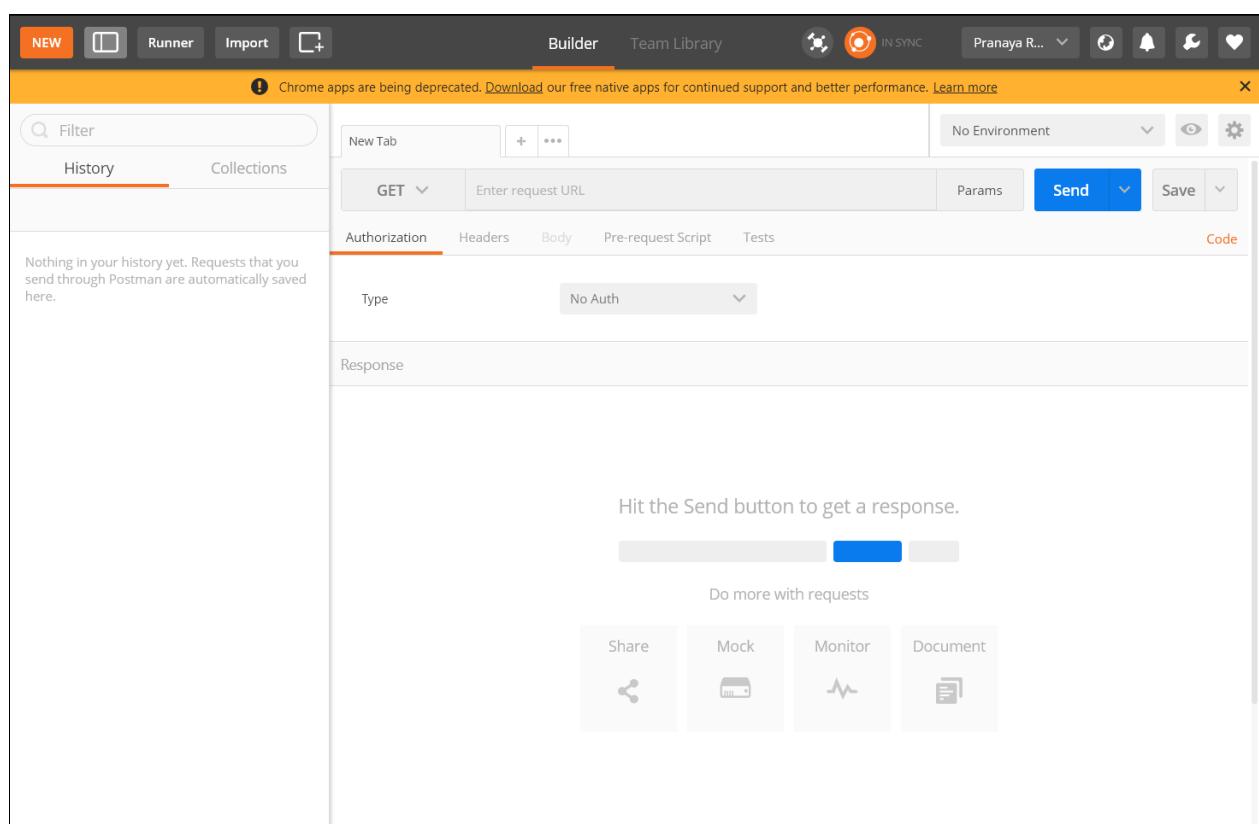
Sử dụng **Postman** để gửi một yêu cầu HTTP đến Dịch vụ **Web API ASP.NET** local và kiểm tra phản hồi.

Bước 1: Tải xuống và cài đặt **Postman**

Download tại địa chỉ:

[Hoặc https://www.postman.com/downloads/](https://www.getpostman.com/)

Bước 2: Sau khi đã cài đặt thành công **Postman**, hãy mở **Postman**. Nó sẽ giống như hình bên dưới.



Có thể đăng nhập vào tài khoản postman bằng tài khoản google or facebook

Xem thêm Link hướng dẫn

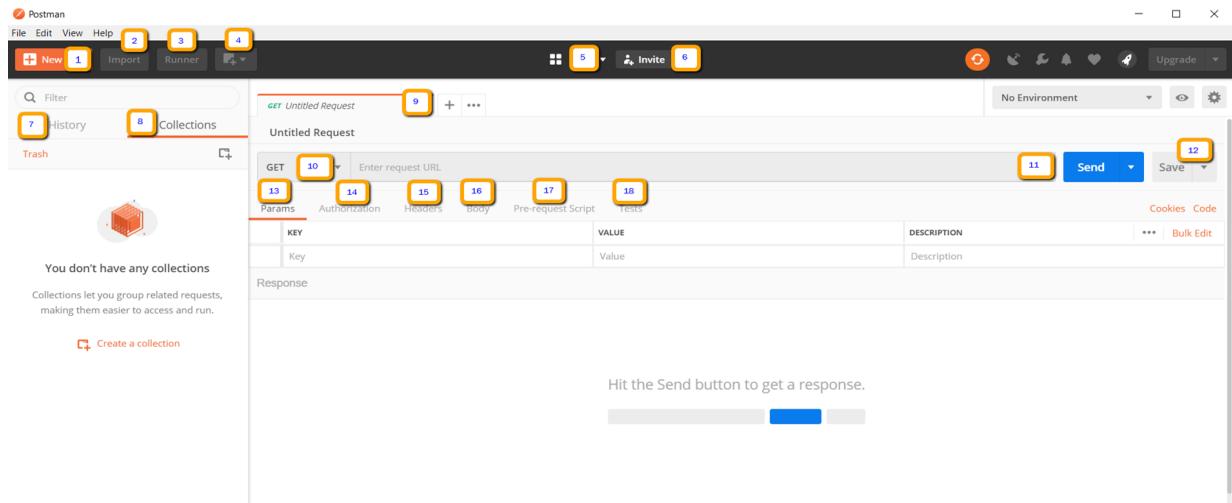
<https://www.guru99.com/postman-tutorial.html>

2.2.3 Cách sử dụng Postman

Bước 1 Tạo Collections

Bước 2 Tạo các requests trong Collections

Bước 3: Viết request tương ứng Http method, rồi nhấn send để gửi request, rồi chờ để nhận response từ web API



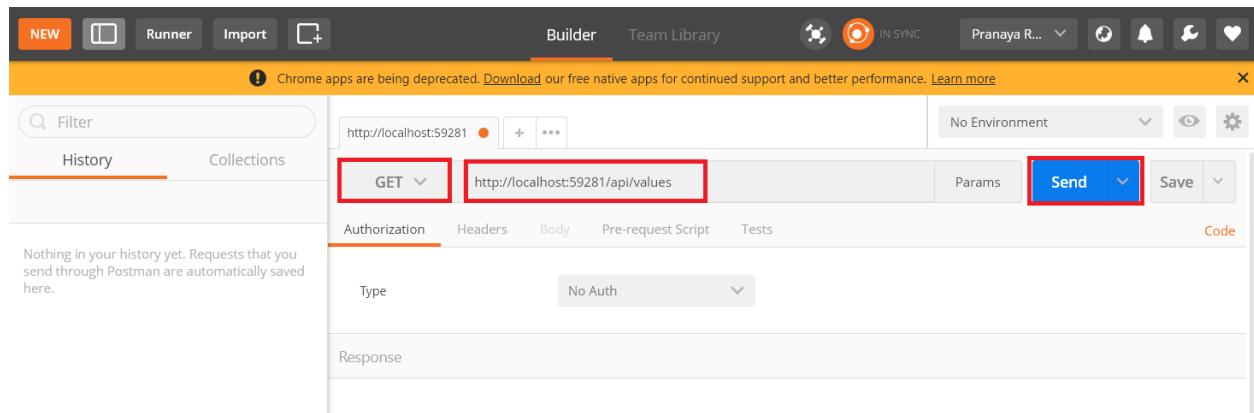
Giao diện Postman

1. **New** – Đây là nơi bạn sẽ tạo request, collection hoặc environment mới.
2. **Import** – Được sử dụng để import collection hoặc environment. Có các tùy chọn để import từ file, folder, link hoặc paste từ text thuần.
3. **Runner** – Kiểm tra tự động hóa có thể được thực hiện thông qua Runner cả collection. Điều này sẽ được thảo luận thêm trong bài học tiếp theo.
4. **Open New** – Mở một tab mới, cửa sổ Postman hoặc cửa sổ Runner bằng việc kích trên nút này.
5. **My Workspace** – Bạn có thể tạo sổ làm việc riêng hoặc như cho một nhóm.
6. **Invite** – Làm việc cộng tác với nhiều thành viên bằng việc mời các thành viên.
7. **History** – Các request đã thực hiện mà bạn đã thực hiện sẽ được hiển thị trong History. Giúp bạn có thể lùi theo các hành động bạn đã làm.
8. **Collections** – Tổ chức bộ thử nghiệm của bạn bằng cách tạo collection. Mỗi collection có thể có các thư mục con và nhiều yêu cầu. Request hoặc thư mục cũng có thể được trùng lặp.
9. **Tab Request** – Hiển thị tiêu đề của request mà bạn đang làm việc. Mặc định “Untitled Request” sẽ được hiển thị cho các request không có tiêu đề.
10. **HTTP Request** – Click vào đây sẽ hiển thị danh sách thả xuống với các request khác nhau như GET, POST, COPY, DELETE, v.v. Trong thử nghiệm, các yêu cầu được sử dụng phổ biến nhất là GET và POST.
11. **Request URL** – Còn được gọi là điểm cuối (endpoint), đây là nơi bạn sẽ xác định liên kết đến nơi API sẽ giao tiếp.
12. **Save** – Nếu có thay đổi đối với request, nhập vào Save là bắt buộc để những thay đổi mới sẽ không bị mất hoặc bị ghi đè.
13. **Params** – Đây là nơi bạn sẽ viết các tham số cần thiết cho một request, ví dụ như các cặp key – value.

14. **Authorization** – Để truy cập API, cần được cấp quyền. Nó có thể ở dạng tên người dùng và mật khẩu, bearer token, v.v.
15. **Headers** – Bạn có thể thiết lập các header như nội dung kiểu JSON tùy theo cách tổ chức của bạn.
16. **Body** – Đây là nơi chúng ta có thể tùy chỉnh chi tiết trong request thường được sử dụng trong request POST.
17. **Pre-request Script** – Đây là các tập lệnh sẽ được thực thi trước request. Thông thường, script tiền request (pre-request) cho cài đặt môi trường được sử dụng để đảm bảo các kiểm tra sẽ được chạy trong môi trường chính xác.
18. **Tests** – Đây là các script được thực thi khi request. Điều quan trọng là phải có các thử nghiệm như thiết lập các điểm checkpoint để kiểm tra trạng thái là ok, dữ liệu nhận được có như mong đợi không và các thử nghiệm khác.

2.2.3.1 Yêu cầu GET bằng Postman:

1. Thiết lập request HTTP của bạn là GET.
2. Trong trường URL yêu cầu, nhập vào link
3. Kích nút **Send**
4. Bạn sẽ nhìn thấy message **200 OK**
5. Sẽ hiển thị kết quả 10 người dùng trong phần Body của bạn



Chọn Phương thức HTTP là “GET” và nhập URL của Web API của bạn như được hiển thị trong hình ảnh bên dưới..

The screenshot shows the Postman interface with a successful HTTP request. The URL is set to `http://localhost:59281/api/values`. The response status is `200 OK` with a time of `571 ms`. The response body is displayed in JSON format: `[\"value0\", \"value1\", \"value2\"]`.

Như bạn có thể thấy trong hình trên, phản hồi HTTP hiển thị dữ liệu và trạng thái phản hồi. Do đó, bạn cũng có thể sử dụng Postman để kiểm tra Web API của mình.

The screenshot shows the Postman interface with a successful HTTP request. The URL is set to `https://jsonplaceholder.typicode.com/users`. The response status is `200 OK` with a time of `784 ms`. The response body is displayed in JSON format, showing a list of users:

```

1 [
2   {
3     "id": 1,
4     "name": "Leanne Graham",
5     "username": "Bret",
6     "email": "Sincere@april.biz",
7     "address": {
8       "street": "Kulas Light",
9       "suite": "Apt. 556",
10      "city": "Gwenborough",
11      "zipcode": "92998-3874",
12      "geo": {
13        "lat": "-37.3159",
14        "lng": "81.1496"
15      }
16    },
17    "phone": "1-770-736-8031 x56442",
18    "website": "hildegard.org",
19    "company": {
20      "name": "Romaguera-Crona",
21      "catchPhrase": "Multi-layered client server neural-net",
22      "bs": "harness real-time e-markets"
23    }
24  },
25  {
26    "id": 2,
27    "name": "Ervin Howell",
28    "username": "Antonette"
29  }
]

```

Chú ý: Có thể có nhiều trường hợp request GET không thành công. Nó có thể do URL của request không hợp lệ hoặc do chứng thực không thành công

2.2.3.2 POST Request với Postman:

- Chọn động từ HTTP là POST
- Đặt URL
- Đặt Content-Type trong **application/json**. Để thực hiện việc này, hãy nhập vào tab Header và cung cấp giá trị khóa như được hiển thị trong hình ảnh bên dưới

The screenshot shows the Postman interface. At the top, there's a toolbar with 'NEW', 'Runner', 'Import', etc. Below it is a banner about Chrome apps being deprecated. The main area has a search bar and tabs for 'History' and 'Collections'. A red box highlights the 'POST' method and the URL 'http://localhost:59281/api/values'. Another red box highlights the 'Headers (1)' tab, which contains a table with a row for 'Content-Type' set to 'application/json'. A third red box highlights the 'Body' tab, which is currently selected. Under 'Body', there are options for 'form-data', 'x-www-form-urlencoded', 'raw', 'binary', and 'JSON (application/json)'. The 'raw' option is selected and highlighted with a red box. Below these options, a code editor shows the JSON body: '1 "new string value"'.

Tiếp theo, chúng ta cần cung cấp giá trị chuỗi mà chúng ta muốn thêm vào mảng chuỗi. Chúng tôi cần cung cấp giá trị chuỗi trong phần nội dung yêu cầu. Để làm như vậy, hãy nhấp vào tab body và cung cấp giá trị chuỗi như hình dưới đây.

This screenshot shows the same Postman interface as the previous one, but with the 'Body' tab selected instead of 'Headers'. A red box highlights the 'Send' button at the top right. Another red box highlights the 'raw' selection under the body type dropdown. The code editor shows the JSON body again: '1 "new string value"'.

Khi bạn đã cung cấp giá trị chuỗi trong nội dung yêu cầu, hãy nhấp vào nút gửi, nút này sẽ đưa ra yêu cầu đăng lên Web API. Theo cách tương tự, bạn có thể kiểm tra các Yêu cầu PUT và DELETE.

Ví dụ test WebAPI template như sau

Với url <https://localhost:44321/api/values>

This screenshot shows a different Postman session. It's a GET request to 'https://localhost:44321/api/values'. The 'Body' tab is selected, with the 'raw' option highlighted. The response status is shown as 'Status: 200 OK Time: 39 ms Size: 371 B'. The response body is displayed in a JSON viewer, showing an array with two elements: 'value1' and 'value2'.

2.3 Sử dụng Fiddler để kiểm tra Web API

Trong phần này, chúng ta sẽ thảo luận về cách sử dụng Fiddler để kiểm tra các dịch vụ Web API. Vui lòng Đọc phần tạo ứng dụng Web API đầu tiên của chúng tôi, nơi chúng tôi đã thảo luận về quy trình từng bước để tạo ứng dụng Web API ASP.NET trước khi tiếp tục phần này vì chúng ta sẽ đến cùng một ví dụ. Như một phần của phần này, chúng ta sẽ thảo luận về các điểm sau.

2.3.1 Fiddler là gì?

Fiddler là một công cụ proxy gỡ lỗi miễn phí được sử dụng để kiểm tra các dịch vụ web ổn định. Chúng tôi có thể sử dụng Fiddler để soạn và thực thi các yêu cầu HTTP khác nhau tới Web API của chúng tôi và kiểm tra phản hồi HTTP.

2.3.2 Cài đặt Fiddler

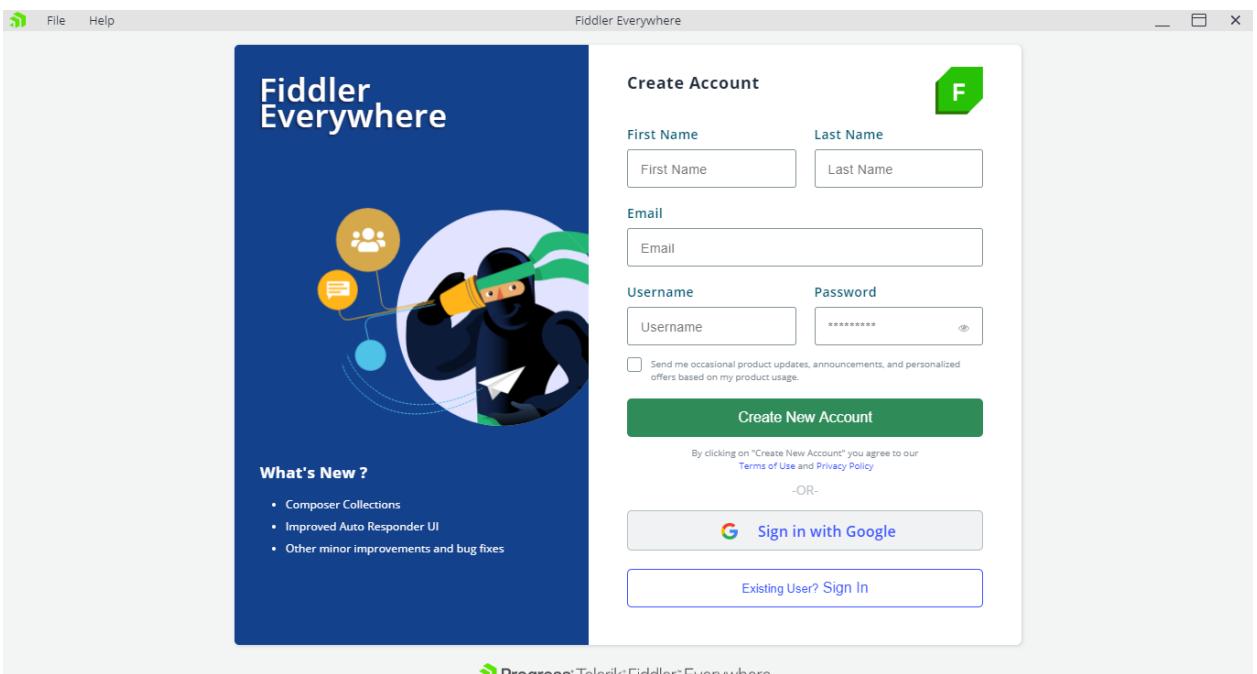
Hãy xem cách sử dụng Fiddler để gửi yêu cầu HTTP đến Dịch vụ Web API ASP.NET cục bộ của chúng tôi và kiểm tra phản hồi.

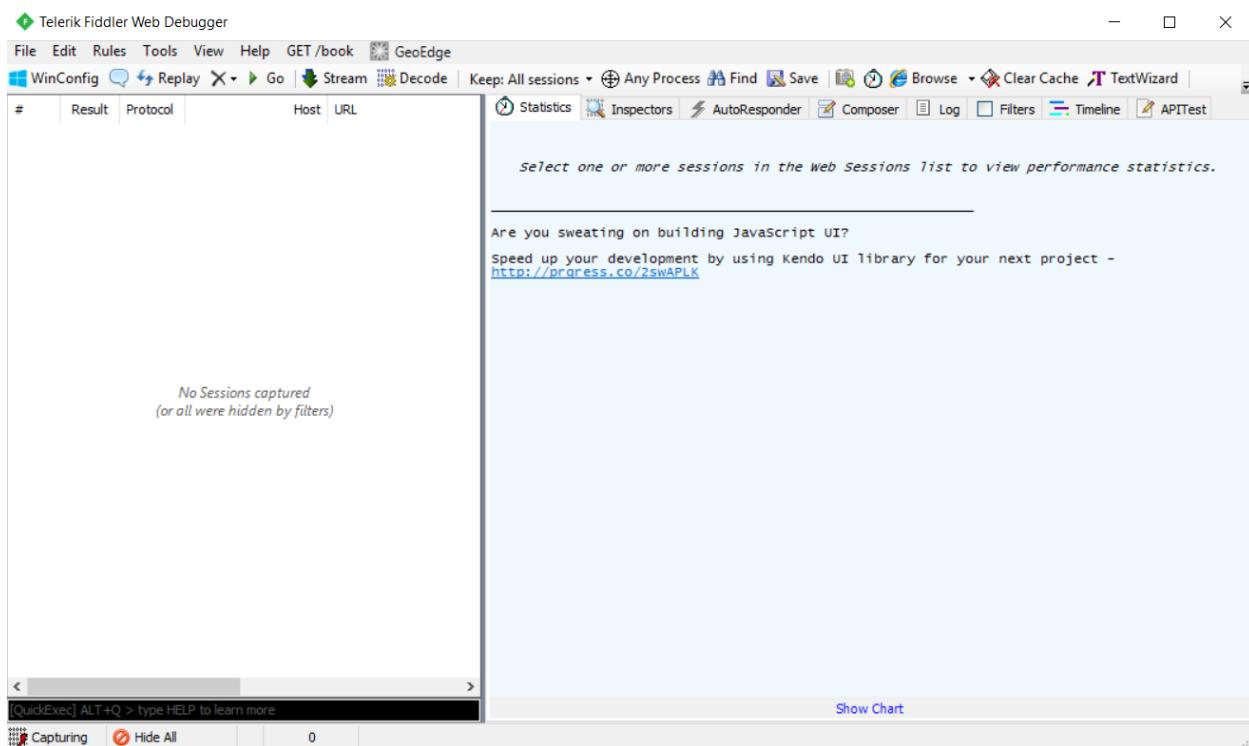
Bước 1: Tải xuống và cài đặt Fiddler <http://www.telerik.com/fiddler>

Bước 2: Sau khi Fiddler được cài đặt thành công, hãy nhấp vào Fiddler.exe để mở Fiddler. Nó sẽ giống như hình bên dưới.

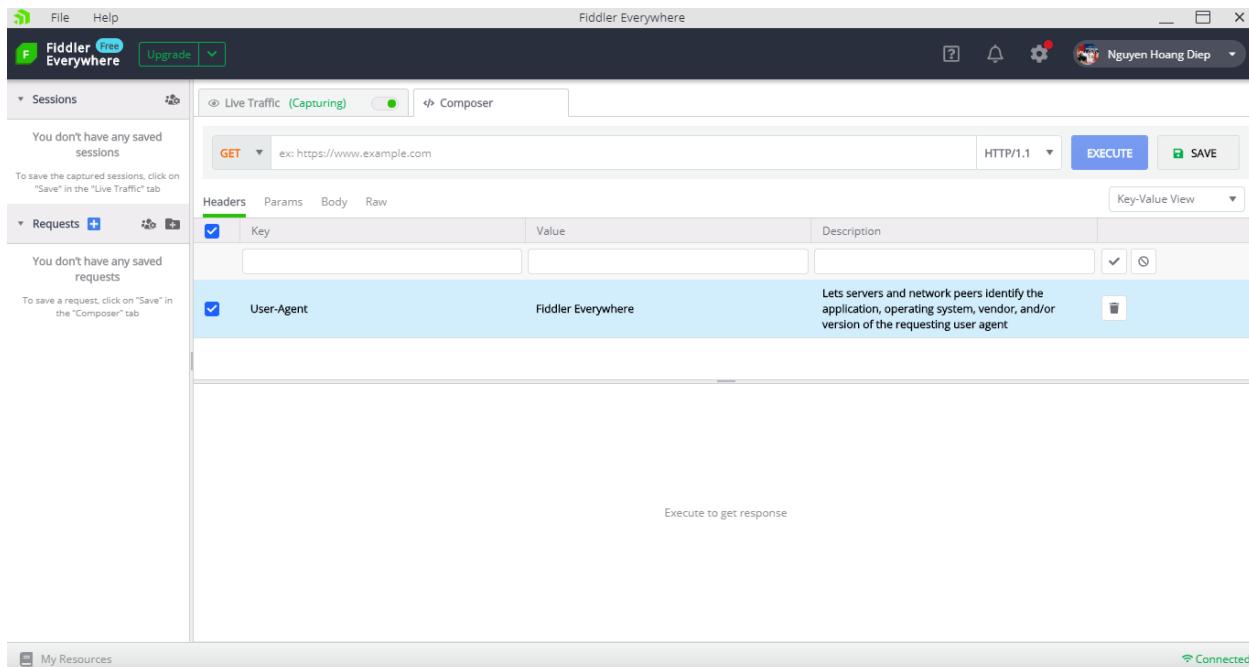
Có thể đăng nhập vào tài khoản postman bằng tài khoản google or facebook

Cài đặt xong nó có giao diện thế này



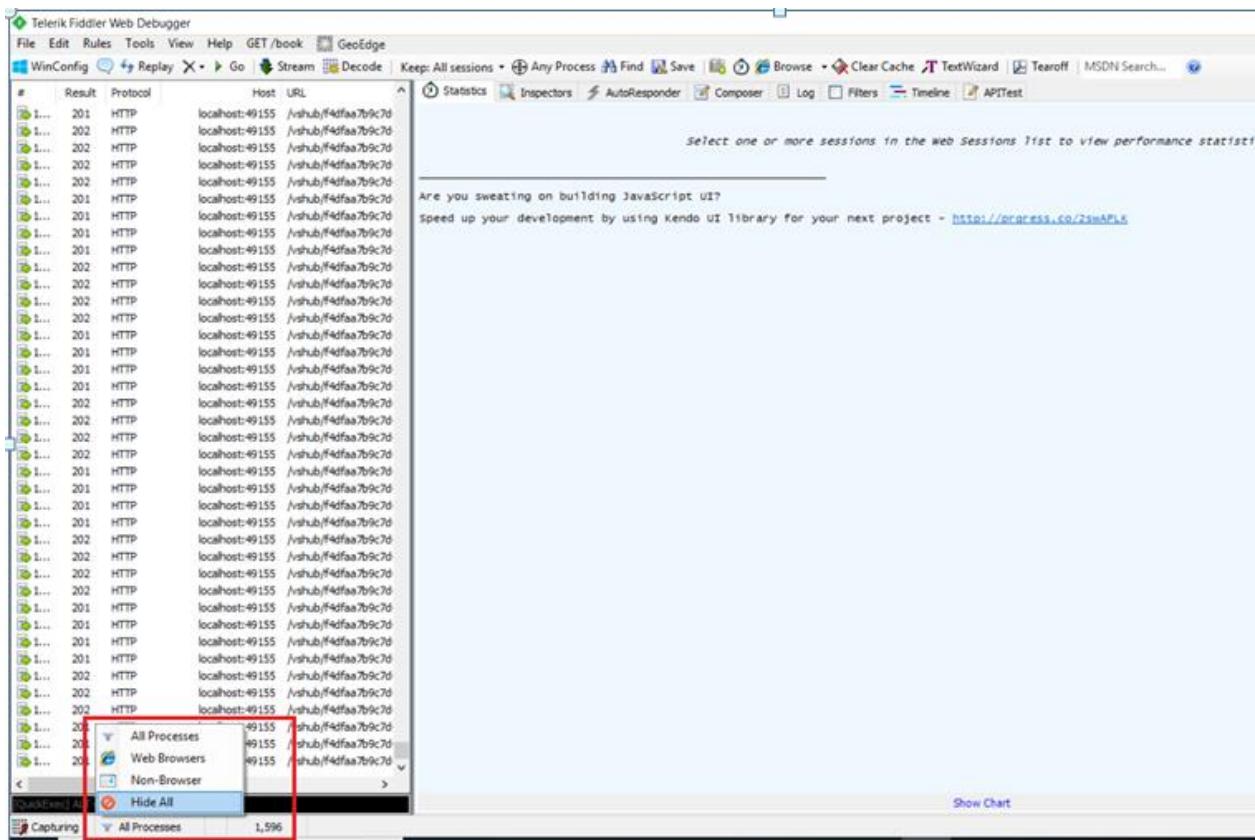


Hoặc giao diện với phiên bản mới hơn như thế này

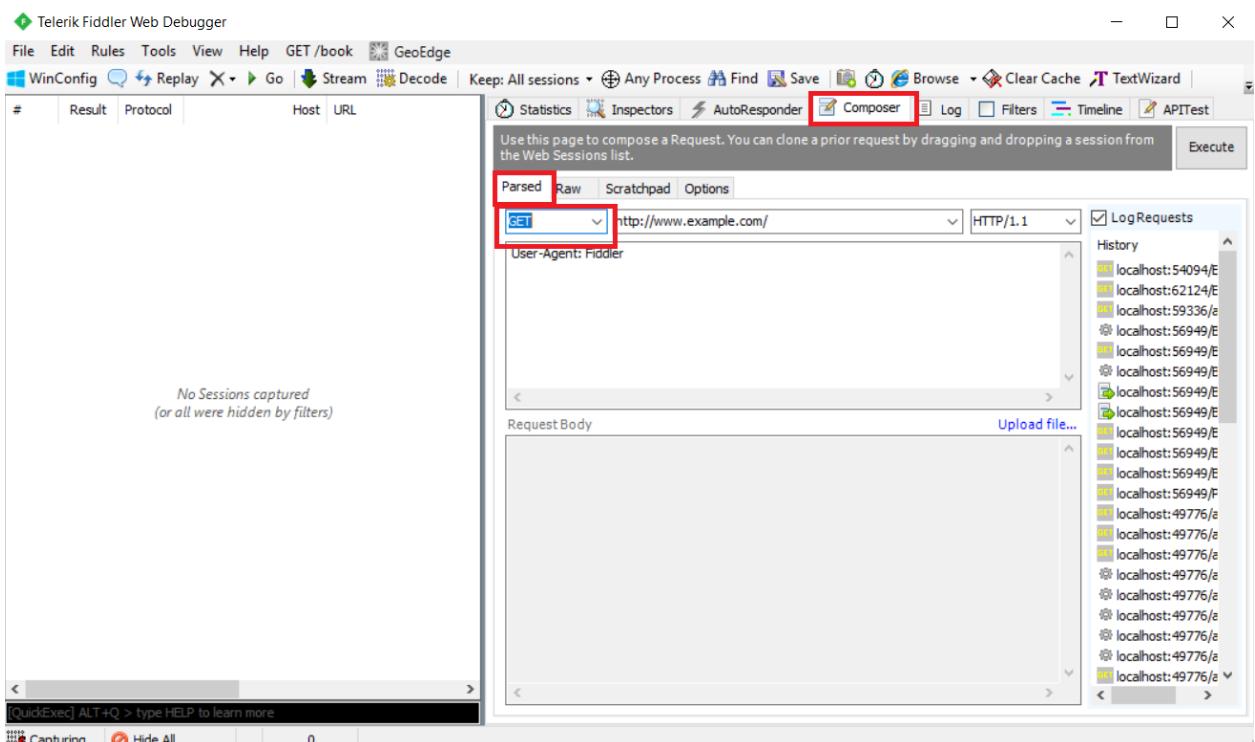


2.3.3 Cách sử dụng Fiddler

Fiddler theo mặc định nắm bắt tất cả các quy trình. Nhưng ở đây chúng tôi không quan tâm đến tất cả các quy trình đã xử lý, chúng tôi chỉ quan tâm đến việc nắm bắt các quy trình cục bộ của chúng tôi, tức là nắm bắt các **Tài nguyên HTTP của WEB API**. Vì vậy, hãy nhấp vào nút **All Processes** ở góc dưới cùng bên trái và chọn **Hide All** như thể hiện trong hình dưới đây. Giữ sự tập trung vào phần Được đánh dấu.



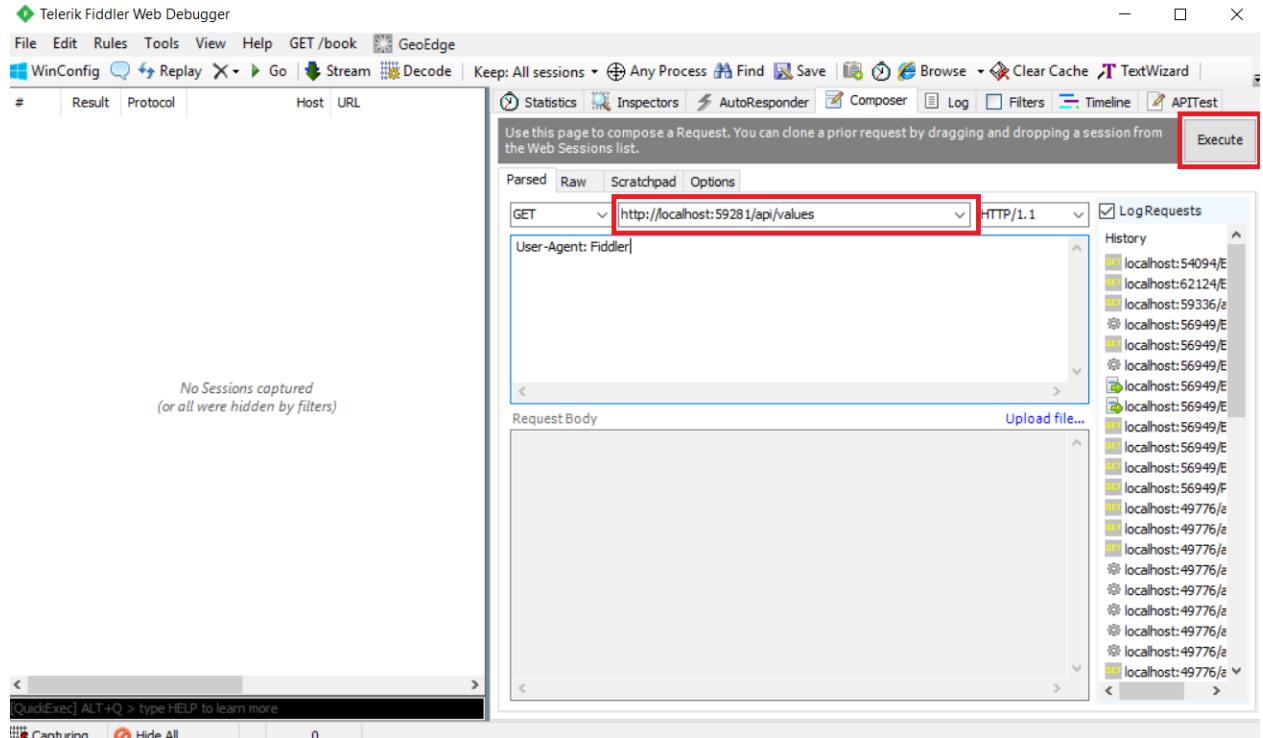
Trong bước tiếp theo, chúng ta cần nhấp vào tab Composer. Tab đầu tiên trong tab Composer là tab Parsed nơi chúng ta có thể cấu hình các yêu cầu HTTP và thực thi nó. Trình đơn thả xuống đầu tiên bao gồm tất cả các Phương thức HTTP. Chọn một phương thức HTTP cụ thể cho yêu cầu bạn muốn thực hiện. Tại đây, chúng ta sẽ chọn động từ GET HTTP để thực hiện yêu cầu HTTP GET như hình bên dưới.



Bây giờ hãy chạy ứng dụng Web API và tìm ra URL.

Nhập URL và Thực thi: Bây giờ, chúng ta cần nhập URL của một yêu cầu vào hộp văn bản. Tại đây, chúng ta sẽ thực hiện yêu cầu HTTP `http://localhost: xxxx / api / values` tới Web API mà chúng ta đã tạo ở phần trước như hình dưới đây.

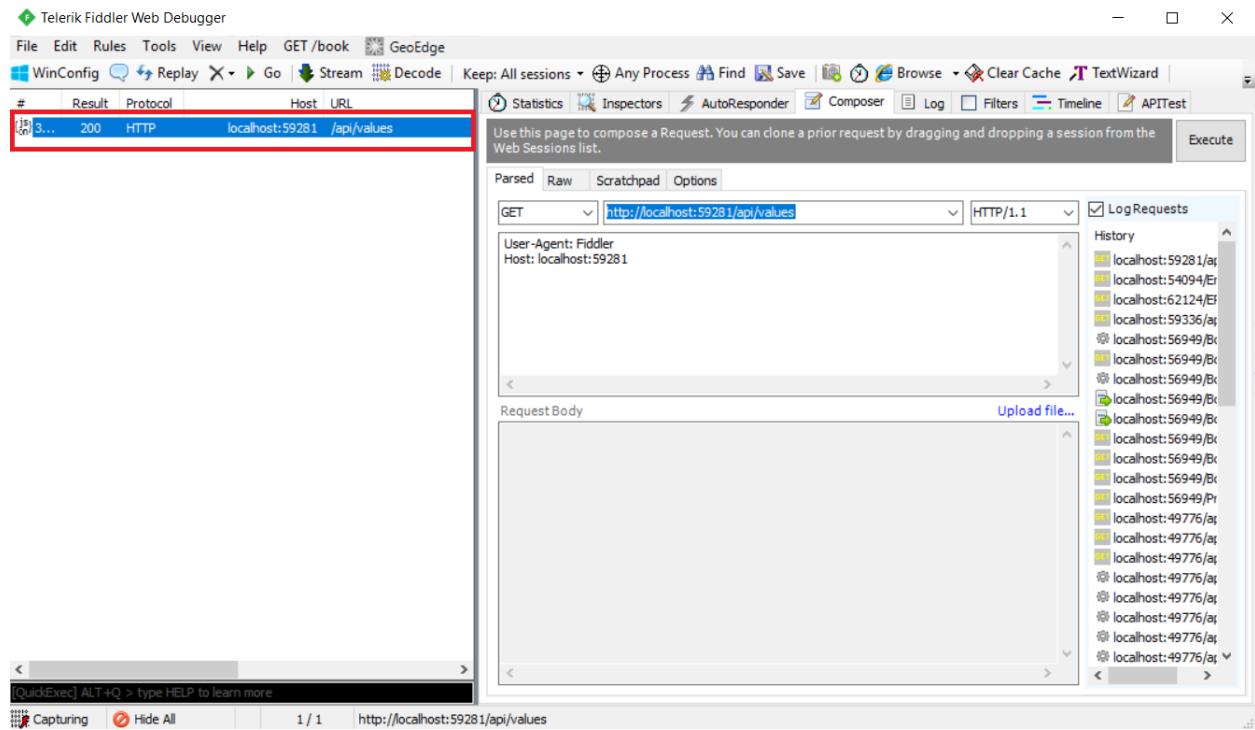
Lưu ý: Bạn cần thay đổi Số hiệu cổng (PORT Number) CÔNG.



Phản hồi trong Fiddler:

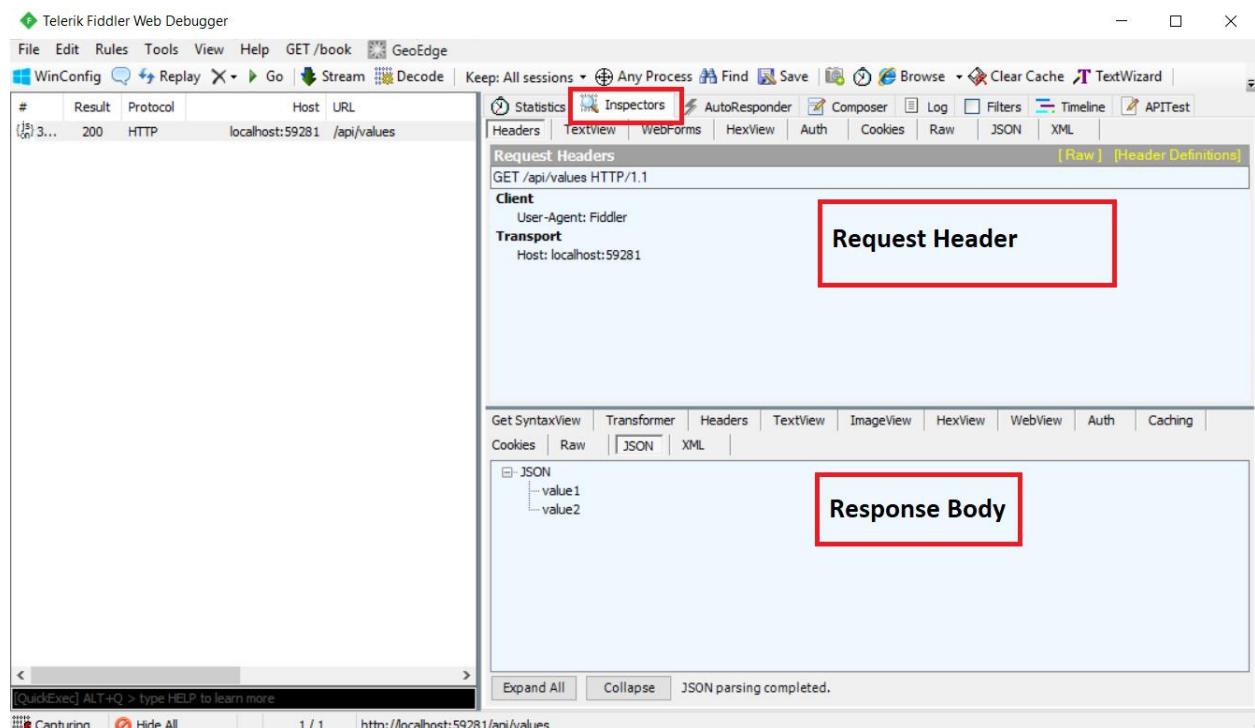
Sau khi bạn nhập URL, hãy nhấp vào nút Thực thi để gửi yêu cầu HTTP này và ngay lập tức nó sẽ hiển thị phản hồi trong khung bên trái như thể hiện trong hình dưới

đây.



Hiểu yêu cầu và phản hồi của Fiddler:

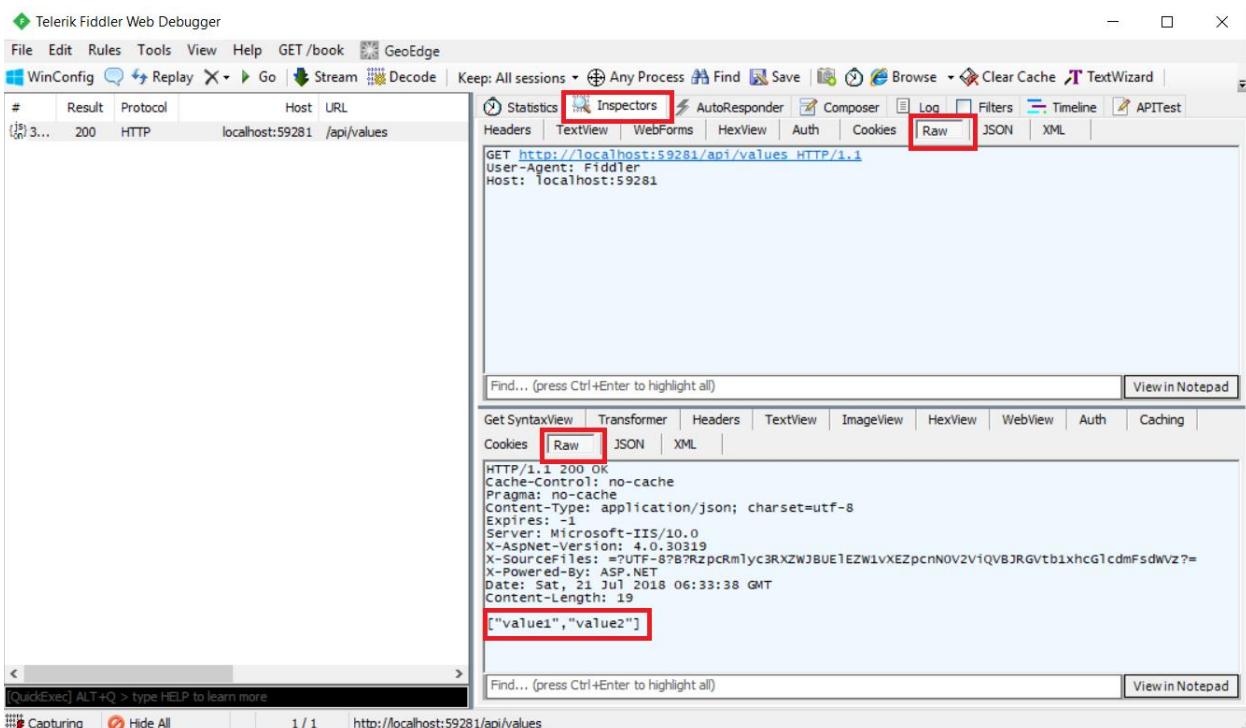
Nhấp đúp vào hàng kết quả ở trên (được đánh dấu bằng hình chữ nhật màu đỏ) để mở tab Thanh tra cho yêu cầu như hình dưới đây.



Như bạn có thể thấy trong hình trên, bảng trên cùng hiển thị tiêu đề Yêu cầu và bảng dưới cùng hiển thị nội dung phản hồi.

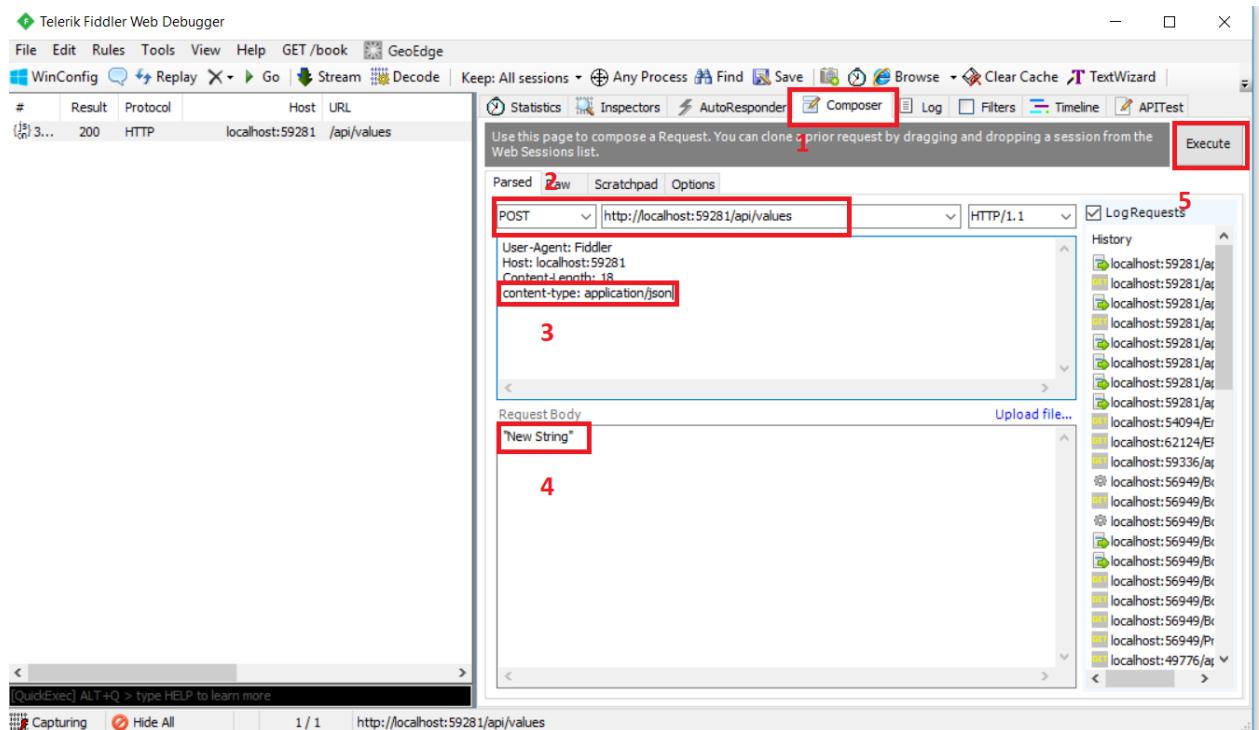
Yêu cầu và phản hồi của Fiddler ở định dạng Raw:

Có thể xem trong tiêu đề Request Raw và xem nội dung phản hồi bằng cách nhấp vào tab request và response như được hiển thị trong hình ảnh bên dưới.



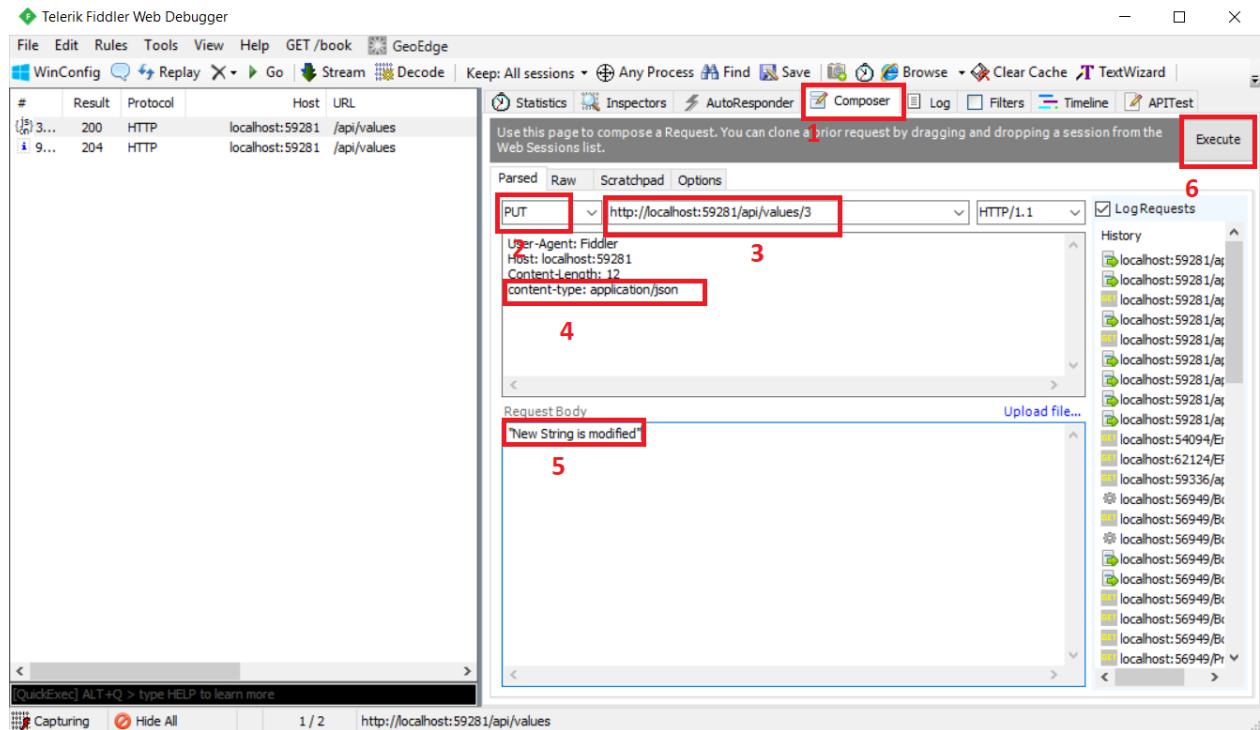
2.3.3.1 Kiểm tra Yêu cầu POST bằng Fiddler:

1. Chọn Tab Compose,
2. Sau đó chọn động từ HTTP là POST
3. Đặt Content-Type application/JSON.
4. Trong phần nội dung yêu cầu the request body, cung cấp giá trị chuỗi mà bạn muốn thêm vào mảng chuỗi.
5. Nhấn vào nút Execute như hình dưới đây.



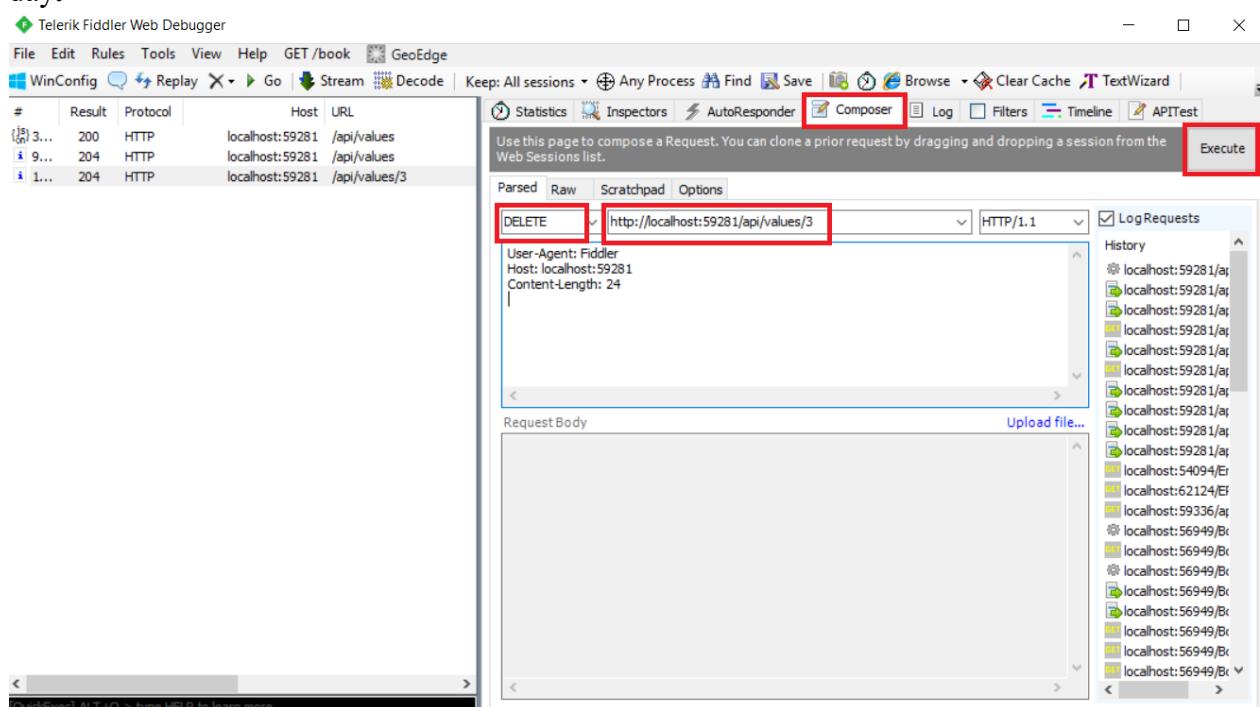
2.3.3.2 Kiểm tra Yêu cầu PUT bằng Fiddler

1. Chọn Tab Compose,
2. Sau đó chọn động từ HTTP là PUT
3. Trong URL, cung cấp chỉ mục của phần tử mảng có giá trị mà bạn muốn cập nhật.
4. Đặt Content-Type là application / json.
5. Trong phần Request body, cung cấp giá trị chuỗi đã cập nhật mà bạn muốn cập nhật vào mảng chuỗi.
6. Cuối cùng, bấm vào nút Execute như hình dưới đây.



2.3.3.3 Kiểm tra Yêu cầu DELETE bằng Fiddler

1. Chọn Tab Compose,
2. Sau đó chọn động từ HTTP là DELETE
3. Trong URL, cung cấp chỉ mục của phần tử mảng có giá trị mà bạn muốn xóa
4. Cuối cùng, bấm vào nút Execute như hình dưới đây.



CHƯƠNG 3 ASP.NET WEB API SỬ DỤNG SQL SERVER

Trong chương này, chúng ta sẽ thảo luận về ASP.NET Web API sử dụng cơ sở dữ liệu SQL Server. Vì vậy, ở đây chúng tôi sẽ tạo Dịch vụ Web API sẽ thực hiện hoạt động CRUD trên cơ sở dữ liệu SQL Server. Chúng tôi sẽ sử dụng dịch vụ này làm cơ sở để hiểu nhiều khái niệm về Web API ASP.NET mà chúng tôi sẽ thảo luận trong các phần sắp tới của chúng tôi.

Web API EmployeeService mà chúng ta sẽ xây dựng sẽ truy xuất dữ liệu từ bảng cơ sở dữ liệu Nhân viên. Chúng tôi sẽ sử dụng Entity Framework để lấy dữ liệu từ cơ sở dữ liệu máy chủ SQL. Bạn có thể sử dụng bất kỳ công nghệ nào bạn chọn để lấy dữ liệu từ cơ sở dữ liệu. Ví dụ, bạn thậm chí có thể sử dụng ADO.NET thô.

3.1 Tạo cơ sở dữ liệu

Bài này sử dụng CSDL chứa bảng Nhân viên để tạo dịch vụ Web API thực hiện hoạt động CRUD bằng cơ sở dữ liệu SQL Server.

Trước tiên cần tạo CSDL và dữ liệu cho ứng dụng

ID	FirstName	LastName	Gender	Salary
1	Pranaya	Rout	Male	60000
2	Anurag	Mohanty	Male	45000
3	Preety	Tiwari	Female	45000
4	Sambit	Mohanty	Male	70000
5	Shushanta	Jena	Male	45000
6	Priyanka	Dewangan	Female	30000
7	Sandeep	Kiran	Male	45000
8	Shudhanshu	Nayak	Male	30000
9	Hina	Sharma	Female	35000
10	Preetiranjan	Sahoo	Male	80000

Tạo cơ sở dữ liệu có tên WEBAPI_DB

Sau đó, tạo bảng Nhân viên và điền vào nó một số dữ liệu thử nghiệm.

Có thể sử dụng SQL Script như sau

Script.sql

```
Update Employees set DepartmentID='11' Where EmployeeID='01048'
```

```
Select * from Departments
```

```
CREATE DATABASE WEBAPI_DB
```

```
GO
```

```
USE WEBAPI_DB
```

```

GO
CREATE TABLE Employees
(
ID int primary key identity,
FirstName nvarchar(50),
LastName nvarchar(50),
Gender nvarchar(50),
Salary int
)
GO
INSERT INTO Employees VALUES ('Pranaya', 'Rout', 'Male', 60000)
INSERT INTO Employees VALUES ('Anurag', 'Mohanty', 'Male', 45000)
INSERT INTO Employees VALUES ('Preety', 'Tiwari', 'Female', 45000)
INSERT INTO Employees VALUES ('Sambit', 'Mohanty', 'Male', 70000)
INSERT INTO Employees VALUES ('Shushanta', 'Jena', 'Male', 45000)
INSERT INTO Employees VALUES ('Priyanka', 'Dewangan', 'Female', 30000)
INSERT INTO Employees VALUES ('Sandeep', 'Kiran', 'Male', 45000)
INSERT INTO Employees VALUES ('Shudhanshu', 'Nayak', 'Male', 30000)
INSERT INTO Employees VALUES ('Hina', 'Sharma', 'Female', 35000)
INSERT INTO Employees VALUES ('Preetiranjan', 'Sahoo', 'Male', 80000)
GO

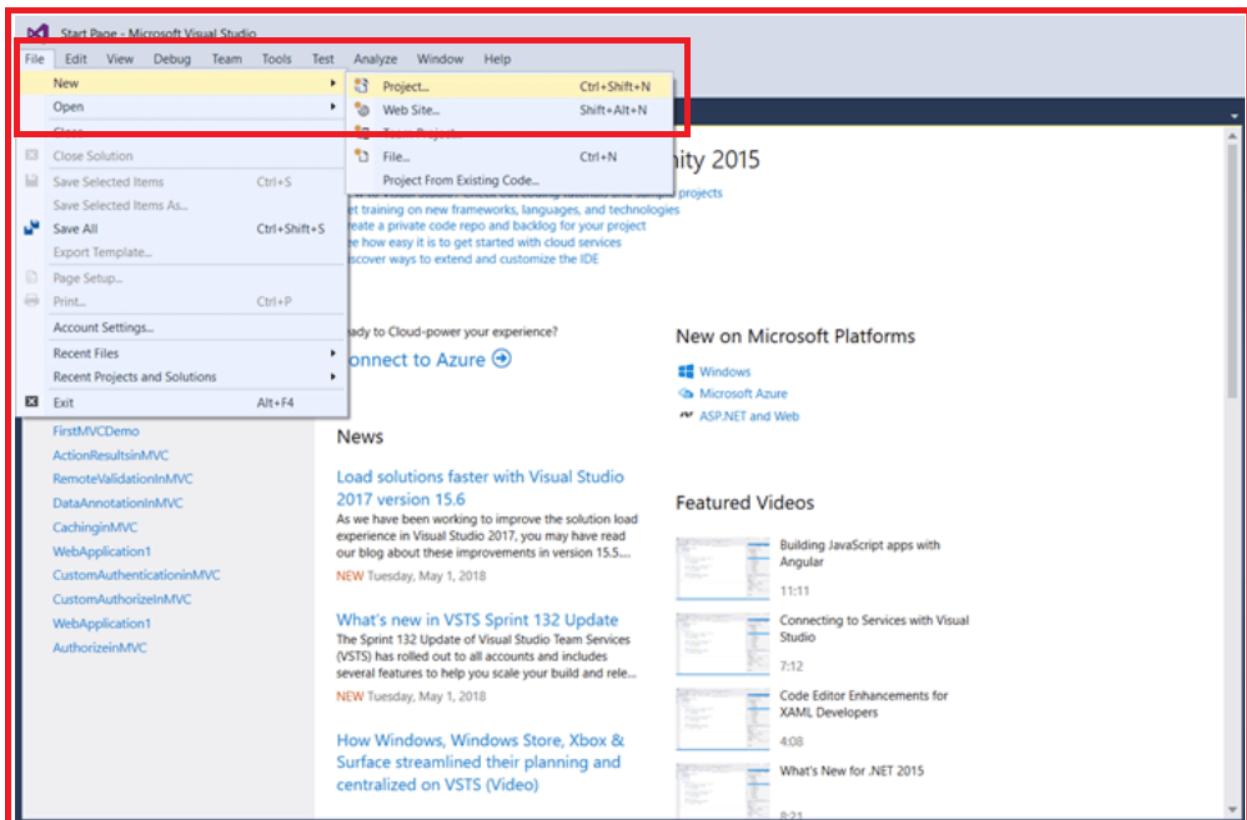
```

3.2 Tạo một Dự án Web API ASP.NET mới

Tạo một dự án ASP.NET Web API theo các bước sau

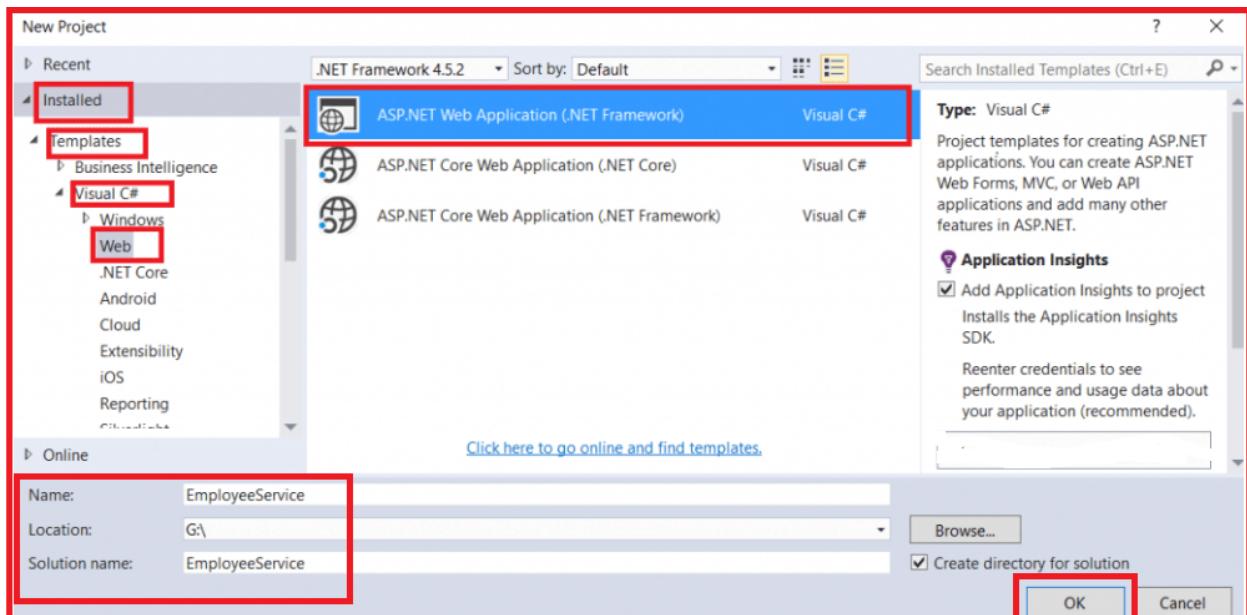
1. Mở Visual Studio
2. Tạo một project ASP.NET Web Application
3. Đặt tên cho ứng dụng, chọn nơi lưu trữ và FrameWork rồi chọn Create
4. Chọn template Web API

Mở Visual Studio và chọn **File – New – Project** như hình bên dưới



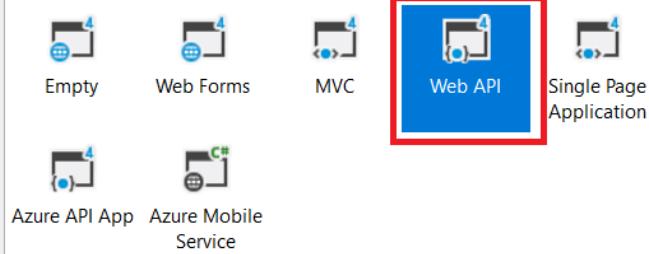
Từ cửa sổ “Dự án mới”, chọn “Visual C #” trong phần “Đã cài đặt - Mẫu”. Một lần nữa từ ngăn giữa, bạn cần chọn “Ứng dụng web ASP.NET” và đặt tên dự án là “EmployeeService”. Cuối cùng, nhấp vào nút “OK” như thể hiện trong hình ảnh bên dưới.

Chọn “New Project”, chọn “**Visual C#**” chọn “**Installed – Templates**”. Sau đó “**ASP.NET Web Application**”. Có thể đặt tên ứng dụng là “**EmployeeService**“. Kích vào “**OK**” để hoàn tất việc tạo ứng dụng webAPI.



Sau khi bạn nhấp vào nút OK. Một cửa sổ hộp thoại mới sẽ mở ra với Tên “**New ASP.NET Project**” để chọn các Mẫu dự án (project Templates) theo yêu cầu

Select a template:

ASP.NET 4.5.2 Templates

Add folders and core references for:

 Web Forms MVC Web API Add unit tests

Test project name: EmployeeService.Tests

Change AuthenticationAuthentication: **Individual User Accounts**

Microsoft Azure

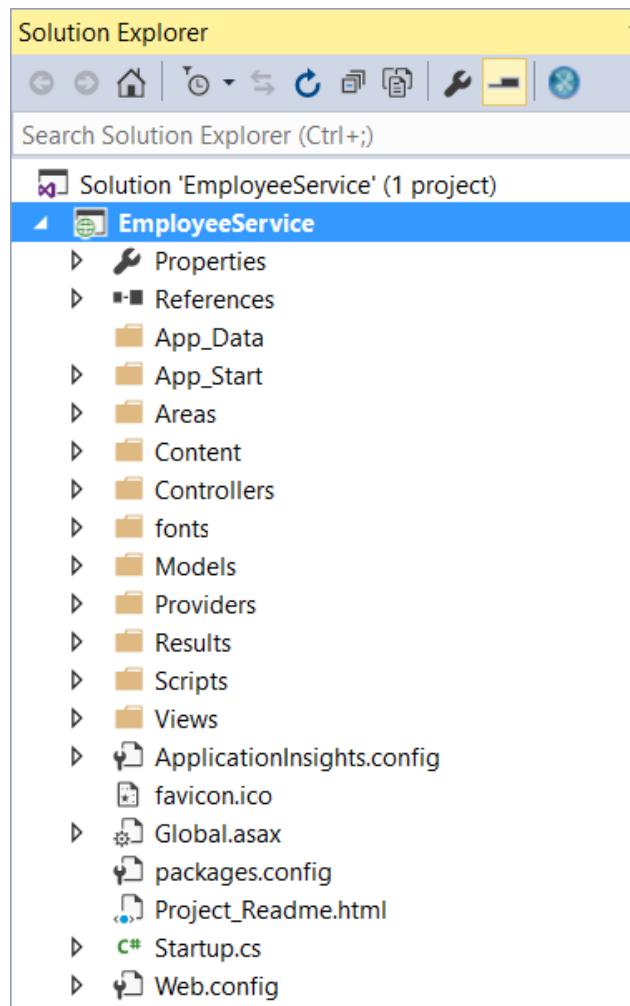
 Host in the cloud

App Service

OK

Cancel

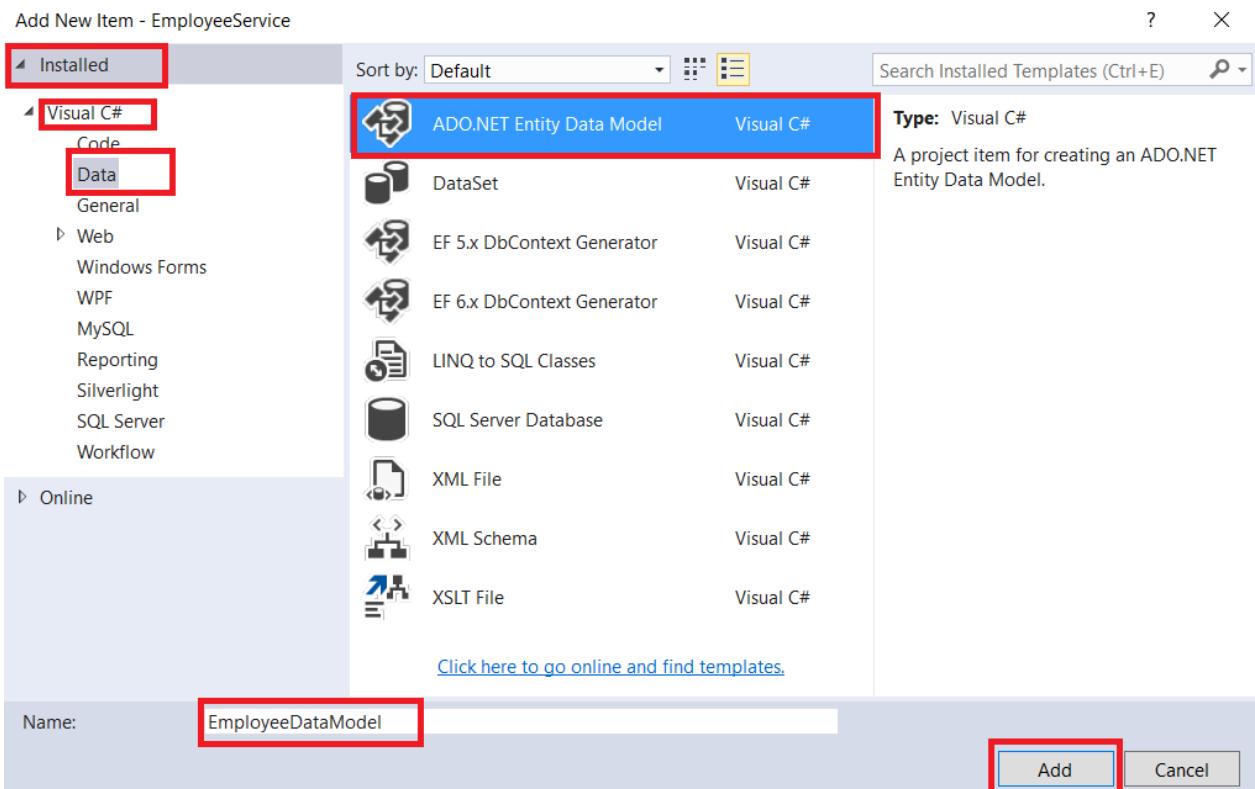
Trong hộp thoại này, chọn WEB API với project Templates và nhập vào nút OK. Tại thời điểm này, dự án Web API với cấu trúc sau.



3.3 Kết nối với ADO.NET Entity Framework

Nhấp chuột phải vào thư mục Models, sau đó chọn **Add – New Item** mới và từ cửa sổ **New Item**, chọn **Data** từ vùng bên trái và sau đó chọn Mô hình dữ liệu thực thể **ADO.NET (ADO.NET Entity Data Model)** từ vùng ở giữa

Trong hộp văn bản Tên, nhập **EmployeeDataModel** và nhấp vào nút **Add** như thể hiện trong hình dưới đây.



Trên Entity Data Model Wizard, chọn **EF Designer from database**” rồi chọn **next**



Choose Model Contents

What should the model contain?

EF Designer
from
databaseEmpty EF
Designer
modelEmpty Code
First modelCode First
from
database

Creates a model in the EF Designer based on an existing database. You can choose the database connection, settings for the model, and database objects to include in the model. The classes your application will interact with are generated from the model.

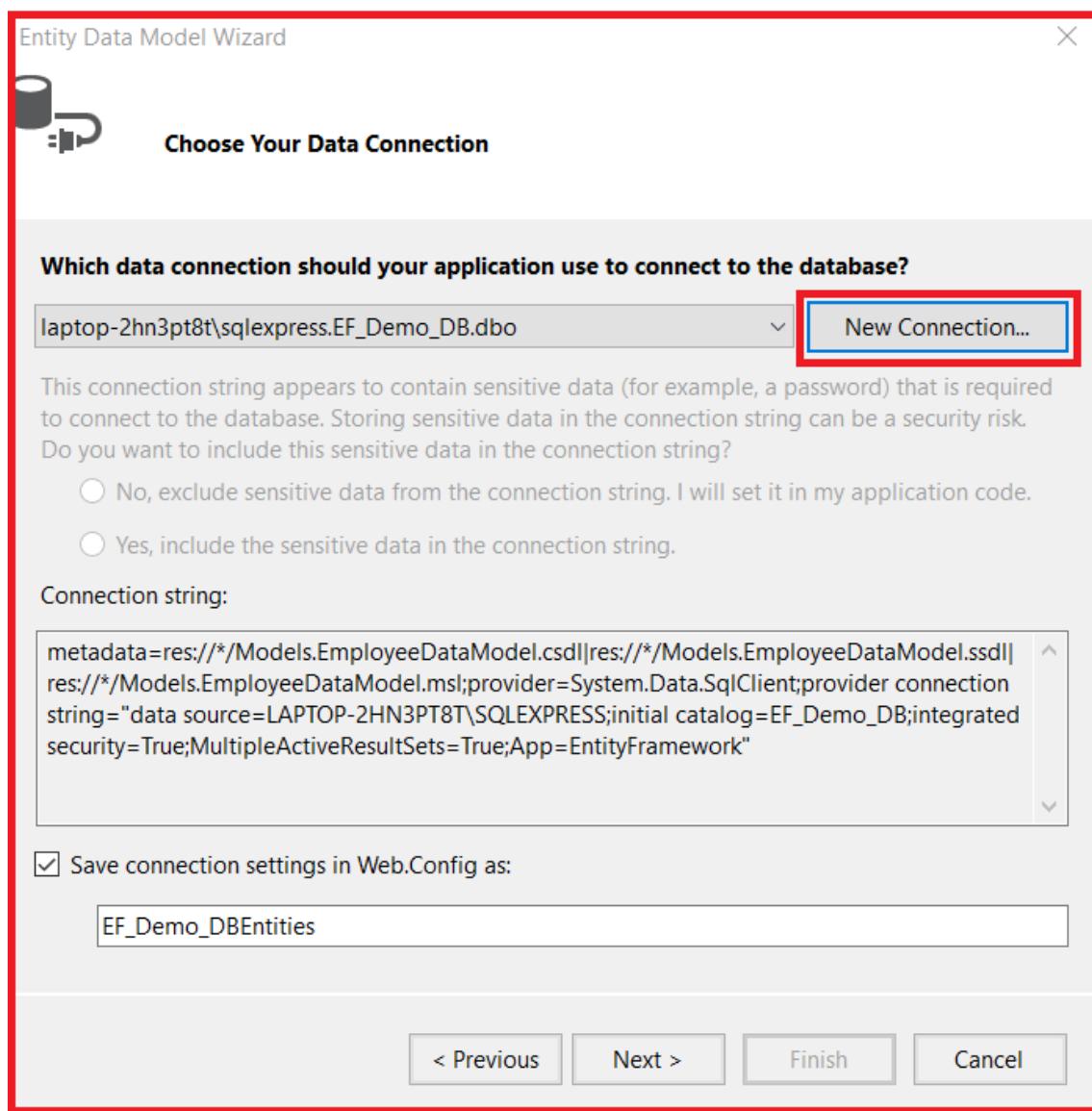
< Previous

Next >

Finish

Cancel

Trên màn hình tiếp theo, hãy nhấp vào “**New Connection**” như thể hiện trong hình ảnh bên dưới



Khi bạn nhập vào New Connection Mới, nó sẽ mở ra cửa sổ Thuộc tính Kết nối “**Connection Properties**”, thiết lập

1. Server Name = tên máy chủ
2. Authentication = Chọn loại xác thực
3. Chọn hoặc nhập một cửa sổ, đặt tên cơ sở dữ liệu = WEBAPI_DB
4. Cuối cùng bạn bấm vào nút **OK** như hình bên dưới.

Connection Properties

? X

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:

Microsoft SQL Server (SqlClient)

Change...

Server name:

LAPTOP-2HN3PT8T\SQLEXPRESS

Refresh

Log on to the server

Authentication: Windows Authentication

User name:

Password:

 Save my password

Connect to a database

 Select or enter a database name:

WEBAPI_DB

 Attach a database file:

Browse...

Logical name:

Advanced...

Test Connection

OK

Cancel

Khi bạn nhấp vào nút OK, nó sẽ điều hướng trở lại Choose Your Data Connection wizard. Tại đây Sửa đổi chuỗi kết nối Connection String thành **EmployeeDbContext** và nhấp vào nút Next như trong hình dưới đây.



Choose Your Data Connection

Which data connection should your application use to connect to the database?

laptop-2hn3pt8t\sqlexpress.WEBAPI_DB.dbo

New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

- No, exclude sensitive data from the connection string. I will set it in my application code.
- Yes, include the sensitive data in the connection string.

Connection string:

```
metadata=res://*/Models.EmployeeDataModel.csdl|res://*/Models.EmployeeDataModel.ssdl||  
res://*/Models.EmployeeDataModel.msl;provider=System.Data.SqlClient;provider connection  
string="data source=LAPTOP-2HN3PT8T\SQLEXPRESS;initial catalog=WEBAPI_DB;integrated  
security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

 Save connection settings in Web.Config as:

EmployeeDBContext

< Previous

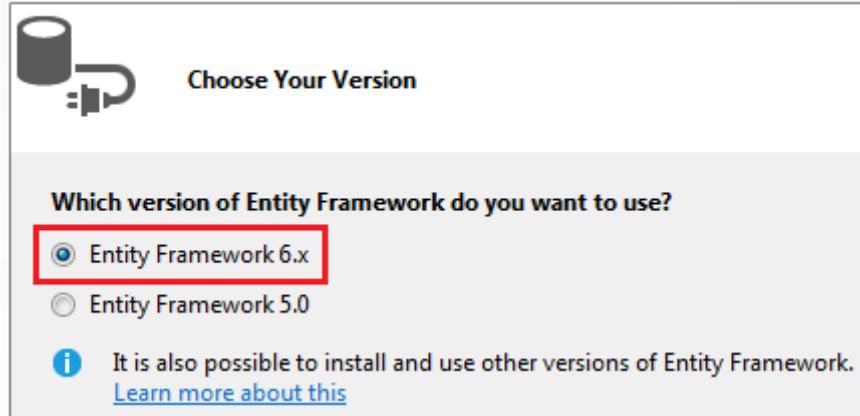
Next >

Finish

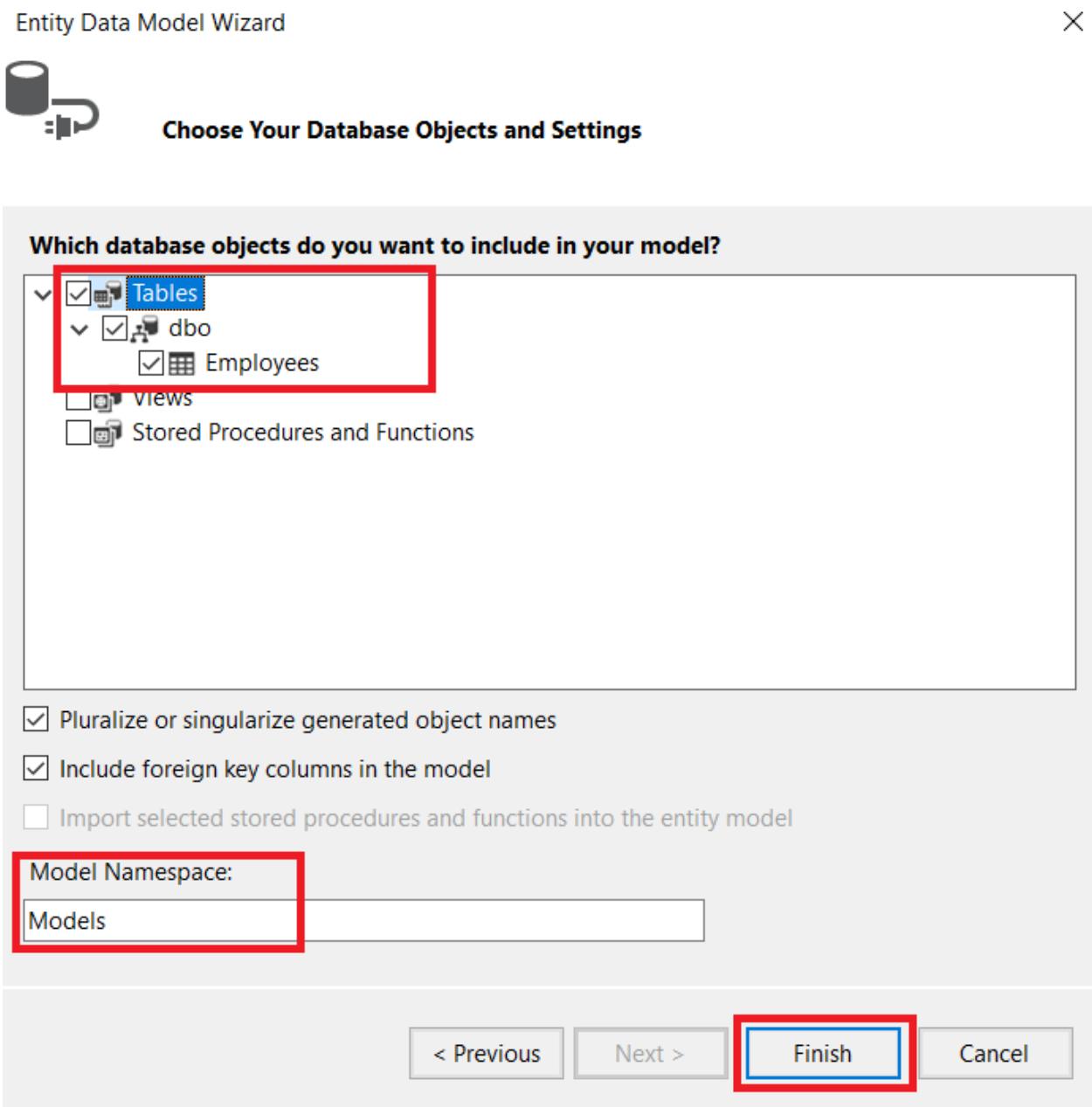
Cancel

Trên màn hình tiếp theo, chọn Entity Framework 6.x như thể hiện trong hình dưới đây. Bước này có thể là tùy chọn nếu bạn đang sử dụng phiên bản visual studio cao hơn.

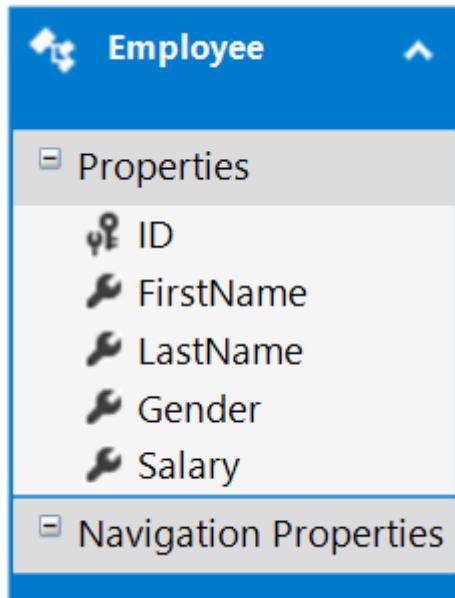
Entity Data Model Wizard



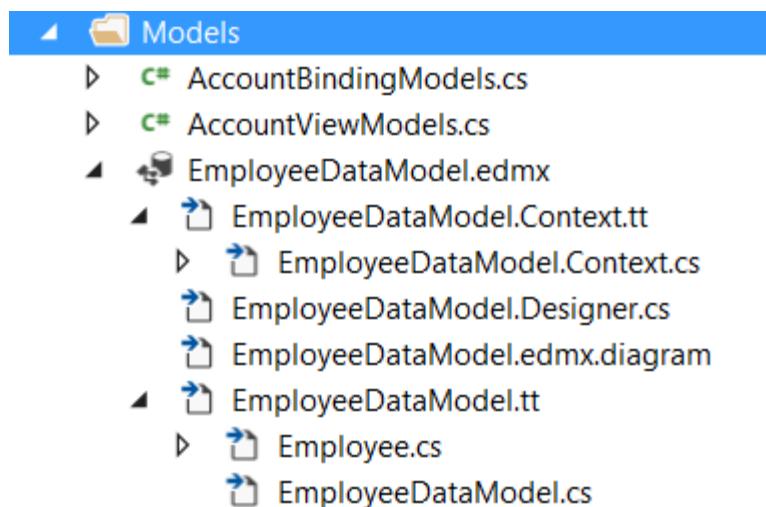
Trên màn hình Chọn đối tượng và cài đặt cơ sở dữ liệu của bạn, hãy chọn “Employees” bảng, cung cấp tên không gian tên mô hình và nhấp vào nút Kết thúc như hình dưới đây.



Khi bạn nhấp vào nút Kết thúc, tệp edmx sau đây sẽ tạo ra.

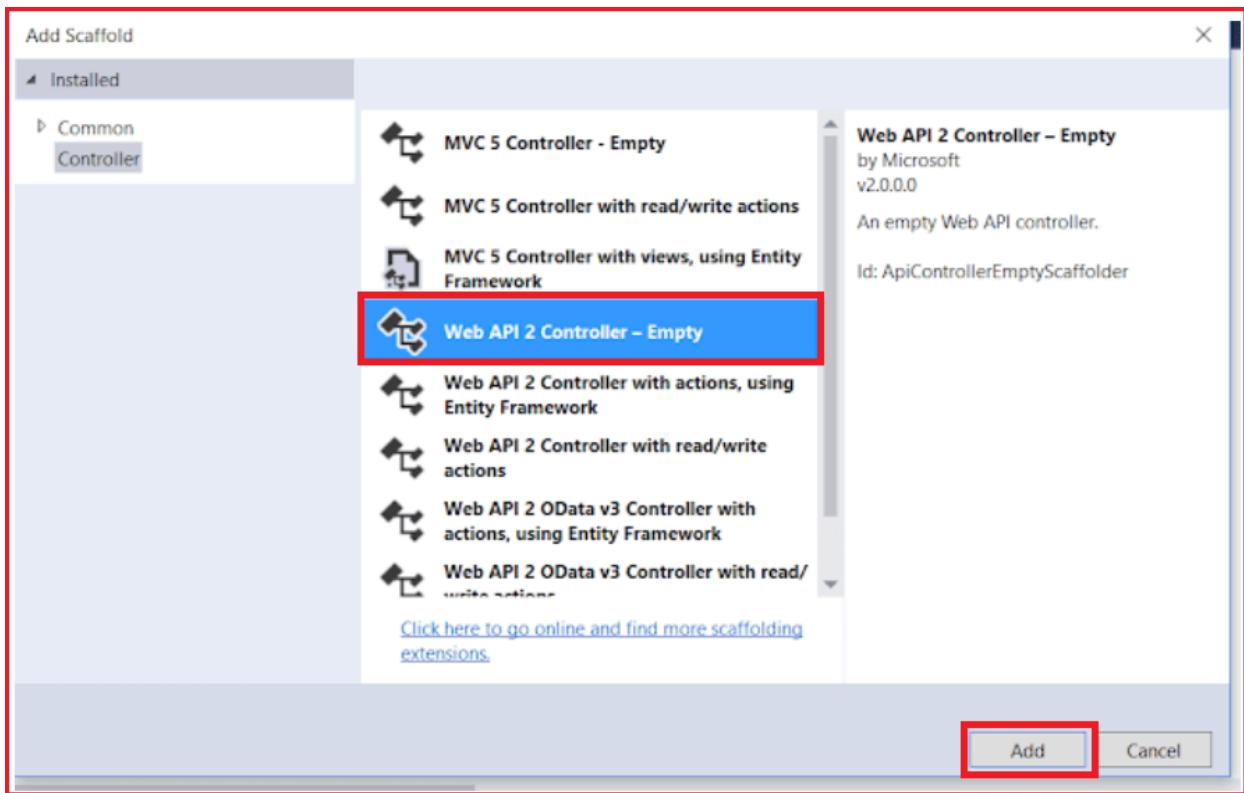


Tệp edmx được tạo trong thư mục Mô hình như hình dưới đây.

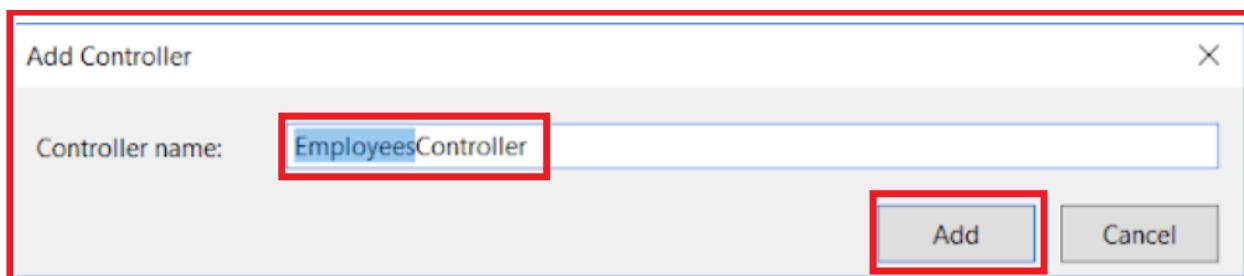


3.4 Thêm Web API Controller

1. Nhấp chuột phải vào thư mục Controller trong dự án **EmployeeService** và chọn **Add – Controller**
2. Sau đó, bạn cần chọn “**Web API 2 Controller – Empty**” và sau đó nhấp vào “**Add**” như thể hiện trong hình ảnh dưới đây.



Trên màn hình tiếp theo được đặt, Controller Name là **EmployeesController** sau đó kích vào **Add** như được hiển thị trong hình ảnh dưới đây.



Sao chép và dán mã sau vào EmployeesController.cs

EmployeesController.cs

```
namespace EmployeeService.Controllers
{
    public class EmployeesController : ApiController
    {
        public IEnumerable<Employee> Get()
        {
            using (EmployeeDBContext dbContext = new EmployeeDBContext())
            {
                return dbContext.Employees.ToList();
            }
        }
    }
}
```

```
    }

    public Employee Get(int id)
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            return dbContext.Employees.FirstOrDefault(e => e.ID == id);
        }
    }
}
```

Khai báo như sau

using EmployeeService.Models;

Tại thời điểm này khi điều hướng đến **/api/employees** sẽ thấy thông tin tất cả nhân viên, còn khi điều hướng đến **/api/employees/1** chúng ta sẽ thấy tất cả các chi tiết của nhân viên có Id=1.

CHƯƠNG 4 TẠO WEB API VỚI CRUD

Hiện nay có rất nhiều lựa chọn ngôn ngữ để lập trình WebAPI như: Java, .NET (C#, VB), Ruby, Python, Perl, JavaScript (Node.js), Go, C++.

Để lựa chọn ngôn ngữ lập trình trong những ngôn ngữ lập trình này, người ta thường cân nhắc nhiều khía cạnh như: đặc điểm cú pháp của các ngôn ngữ thì khả năng mở rộng, việc sử dụng các loại cơ sở dữ liệu khác nhau, , các yêu cầu về khả năng chịu lỗi, kích thước dữ liệu lớn, sự phổ biến của ngôn ngữ ưu nhược điểm của mỗi ngôn ngữ,... Trên thực tế vẫn là hai ngôn ngữ được sử dụng nhiều nhất hiện nay là Java và .NET.

4.1 Giới thiệu về CRUD

Để các web api tuân thủ theo chuẩn restful chúng ta cần sử dụng đúng các HTTP verb tương ứng với ý nghĩa của chúng. Đơn giản có thể hiểu từng HTTP verb tương ứng với một thuật ngữ rất quen thuộc đó là CRUD viết tắt của:

- POST – Create: Tạo dữ liệu mới
- GET – Read: Lấy dữ liệu về
- PUT – Update: Cập nhật dữ liệu
- DELETE – Delete: Xóa dữ liệu

Trong 4 HTTP verb trên mặc dù POST có thể thực hiện tất cả các action nhưng với RESTful service thì cần sử dụng tất cả các verb trên bởi vì:

3 verb (GET, PUT, DELETE) được gọi là các phương thức không thay đổi giá trị (idempotent), tức là bạn có thể gọi GET/PUT/DELETE nhiều lần cũng không có lỗi hay gây bất kỳ ảnh hưởng nào đến ứng dụng. Còn POST lại là một phương thức làm thay đổi giá trị, tức là nếu gọi POST nhiều lần thì sẽ tạo ra nhiều dữ liệu giống nhau.

Bốn hành động thực thi trên CSDL (**Read, Create, Update and Delete**) được xây dựng bởi các phương thức tương ứng là: **GET, POST, PUT and DELETE** tương ứng.

Request (GET,PUT, POST và DELETE)

Mô tả hành động thực thi với tài nguyên. GET là thực hiện việc đọc lấy dữ liệu, PUT là thực hiện việc sửa đổi lại dữ liệu đã có, POST là thực hiện việc tạo mới dữ liệu, DELETE là thực hiện việc xóa dữ liệu.

Request Header

Khi một máy khách gửi một yêu cầu đến máy chủ, yêu cầu sẽ chứa phần đầu và phần thân. Phương thức yêu cầu chứa thông tin bổ sung, chẳng hạn như - loại phản hồi nào được yêu cầu. Ví dụ: Phản hồi phải ở định dạng XML, JSON hoặc một số định dạng khác.

Request Body

Nội dung yêu cầu chứa dữ liệu mà chúng tôi muốn gửi đến máy chủ. Ví dụ: một yêu cầu bài đăng chứa dữ liệu cho một mục mới mà chúng tôi muốn tạo. Định dạng dữ liệu có thể ở dạng XML hoặc JSON

Response Body

Nội dung phản hồi chứa dữ liệu của phản hồi từ máy chủ. Ví dụ: nếu yêu cầu dành cho một nhân viên cụ thể, phần thân phản hồi sẽ bao gồm thông tin chi tiết của nhân viên trong XML, JSON, v.v. Phần nội dung phản hồi chứa dữ liệu của phản hồi từ máy chủ. Ví dụ: nếu yêu cầu dành cho một nhân viên cụ thể, nội dung phản hồi sẽ bao gồm thông tin chi tiết về nhân viên trong XML, JSON, v.v.

Response Status Codes

Mã Trạng thái phản hồi không là gì ngoài mã trạng thái HTTP cung cấp cụ thể trạng thái phản hồi cho máy khách. Một số mã trạng thái phổ biến là 404 không tìm thấy, 200 OK, 204 Không có nội dung, 500 Lỗi máy chủ nội bộ, v.v..

4.2 Triển khai phương thức GET

Sử dụng dự án trước (đã tạo ở Chương 3)

Xét đoạn chương trình sau:

EmployeesController.cs

```
public Employee Get(int id)
{
    using (EmployeeDbContext dbContext = new EmployeeDbContext())
    {
        return dbContext.Employees.FirstOrDefault(e => e.ID == id);
    }
}
```

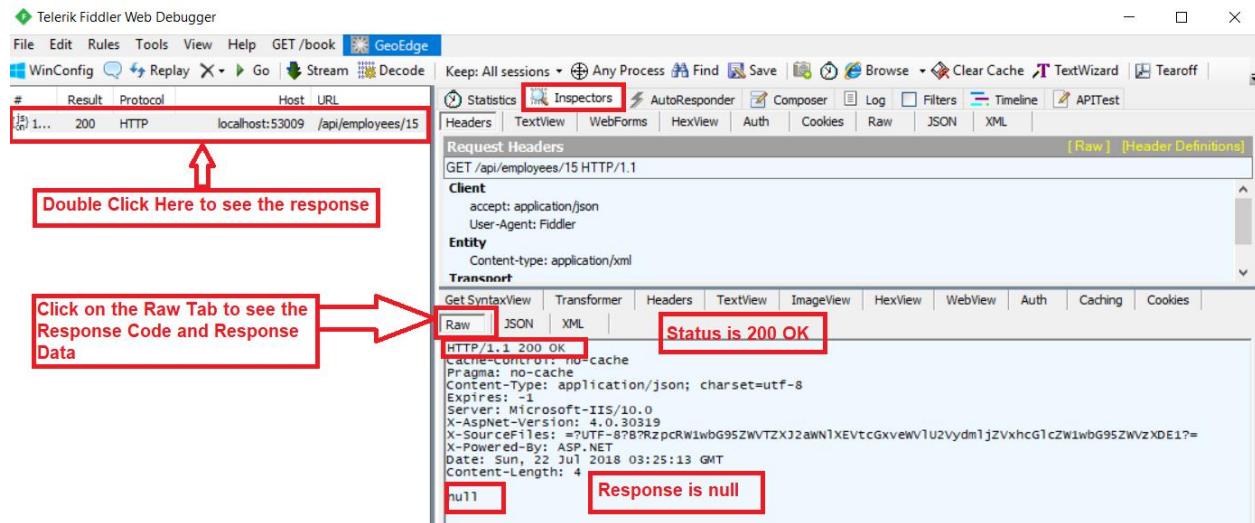
Nếu đoạn mã trên điều hướng đến **/api/employees** thì ta sẽ có thông tin tất cả nhân viên, còn khi điều hướng đến **/api/employees/1** thì ta sẽ có thông tin chi tiết của nhân viên có Id = 1

Có hai phương thức get thông dụng là: Phương thức GET không có tham số và phương thức GET có tham số. Bất cứ khi nào chúng ta truyền tham số id thì nó sẽ trả về nhân viên có id mà chúng ta đã truyền và nếu chúng ta không truyền giá trị tham số id thì nó sẽ trả về tất cả nhân viên.

Xét yêu cầu lấy thông tin của một nhân viên có id không tồn tại như sau:

The screenshot shows the Fiddler application's interface. The top navigation bar includes Statistics, Inspectors, AutoResponder, Composer (which is highlighted with a red box), Log, Filters, Timeline, and APITest. Below the bar, a message says "Use this page to compose a Request. You can clone a prior request by dragging and dropping a session from the Web Sessions list." On the right, there is an "Execute" button with a red box around it. The main area has tabs for Parsed, Raw, Scratchpad, and Options. The "Raw" tab is selected and shows a GET request to "http://localhost:53009/api/employees/15". The "User-Agent" field shows "Fiddler". The response message in the scratchpad area is: "Here we passing the id value as 15 for which there is no employee exists in the database". A checkbox for "Log Requests" is checked. To the right, there is a "History" pane.

Ở đây ta đang cố gắng lấy thông tin chi tiết của một nhân viên có id không tồn tại trong cơ sở dữ liệu. Vì vậy, khi ta nhấp vào nút execute, nó sẽ trả cho ta phản hồi như sau:



Hình trên cho thấy rằng ta sẽ nhận được phản hồi rõ ràng và mã trạng thái là 200 OK. Theo tiêu chuẩn của REST, khi một mục không được tìm thấy thì nó sẽ trả về 404 Not Found. Để đạt được điều này, chúng ta cần sửa đổi EmployeesController như hình dưới đây.

EmployeesController.cs

```
public class EmployeesController : ApiController
{
    public HttpResponseMessage Get()
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var Employees = dbContext.Employees.ToList();
            return Request.CreateResponse(HttpStatusCode.OK, Employees);
        }
    }

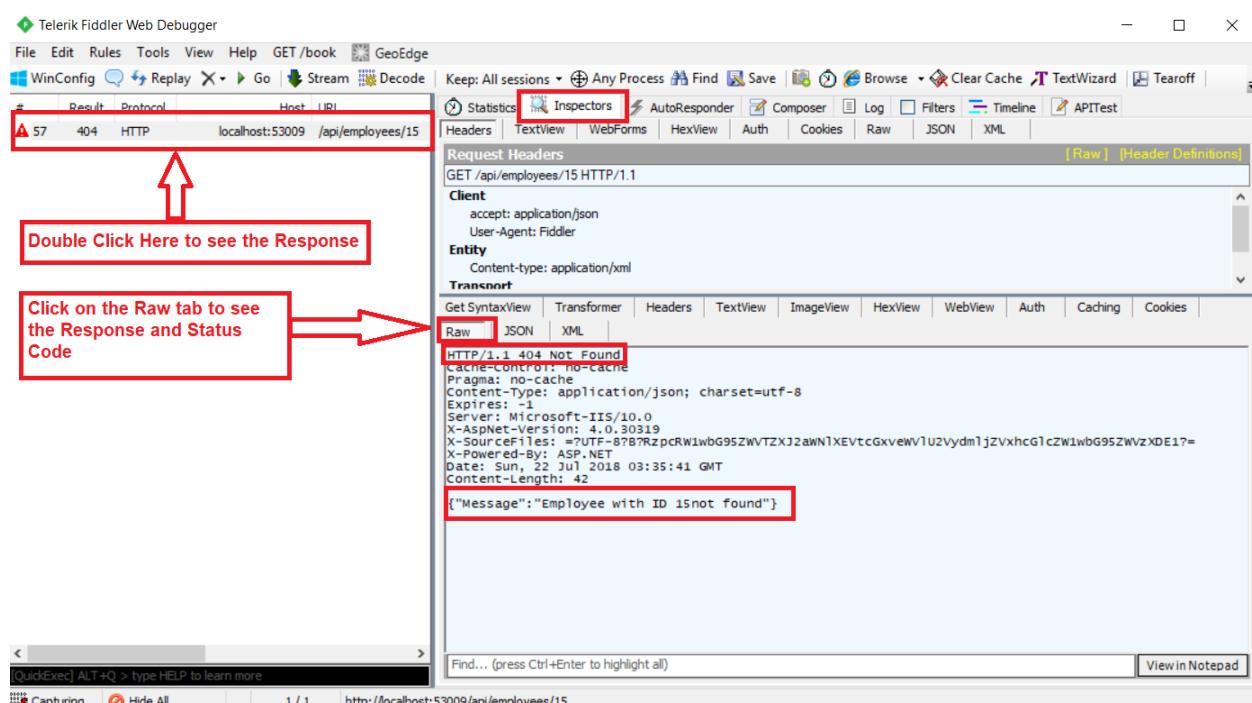
    public HttpResponseMessage Get(int id)
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var entity = dbContext.Employees.FirstOrDefault(e => e.ID == id);
            if (entity != null)
            {
                return Request.CreateResponse(HttpStatusCode.OK, entity);
            }
        }
    }
}
```

```

    }
    else
    {
        return Request.CreateErrorResponse(HttpStatusCode.NotFound,
            "Employee with ID " + id.ToString() + "not found");
    }
}
}
}
}

```

Lúc này, khi ta đưa ra một yêu cầu với thm số truyền vào cho một nhân viên có ID = 15 không tồn tại, ta nhận được mã phản hồi là 404 cùng với thông báo “**Employee with Id 15 not found**” Như hình bên dưới.



4.3 Triển khai phương thức POST

Phương pháp POST trong ứng dụng Web API cho phép chúng tôi tạo mới. Ở đây, ta muốn thêm một Nhân viên mới vào bảng Nhân viên. Đầu tiên, hãy đưa phương thức Post() sau vào bên trong EmploymentController. Lưu ý rằng đối tượng Employee đang được truyền dưới dạng tham số cho phương thức Post.

Hiện tại thuộc tính Employee [FromBody]. Chúng ta sẽ thảo luận chi tiết về thuộc tính [FromBody] trong phần sau tham số e được trang trí bằng cách hiểu thuộc tính [FromBody] này yêu cầu Web API lấy dữ liệu nhân viên từ phần thân yêu cầu.

EmployeesController.cs

```

public class EmployeesController : ApiController
{
    public void Post([FromBody] Employee employee)
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            dbContext.Employees.Add(employee);
            dbContext.SaveChanges();
        }
    }
}

```

1. Tại thời điểm này, hãy xây dựng giải pháp. Chạy ứng dụng và khởi động Fiddler và đưa ra yêu cầu POST Set the HTTP verb to POST
2. Content-Type: application/json. Điều này cho biết rằng chúng tôi đang gửi dữ liệu có định dạng JSON đến máy chủ
3. Trong Nội dung phần thân yêu cầu Request Body, hãy bao gồm đối tượng nhân viên mà chúng tôi muốn thêm vào bảng cơ sở dữ liệu Nhân viên employee ở định dạng JSON
4. Cuối cùng, nhấp vào nút Execute như trong hình dưới đây

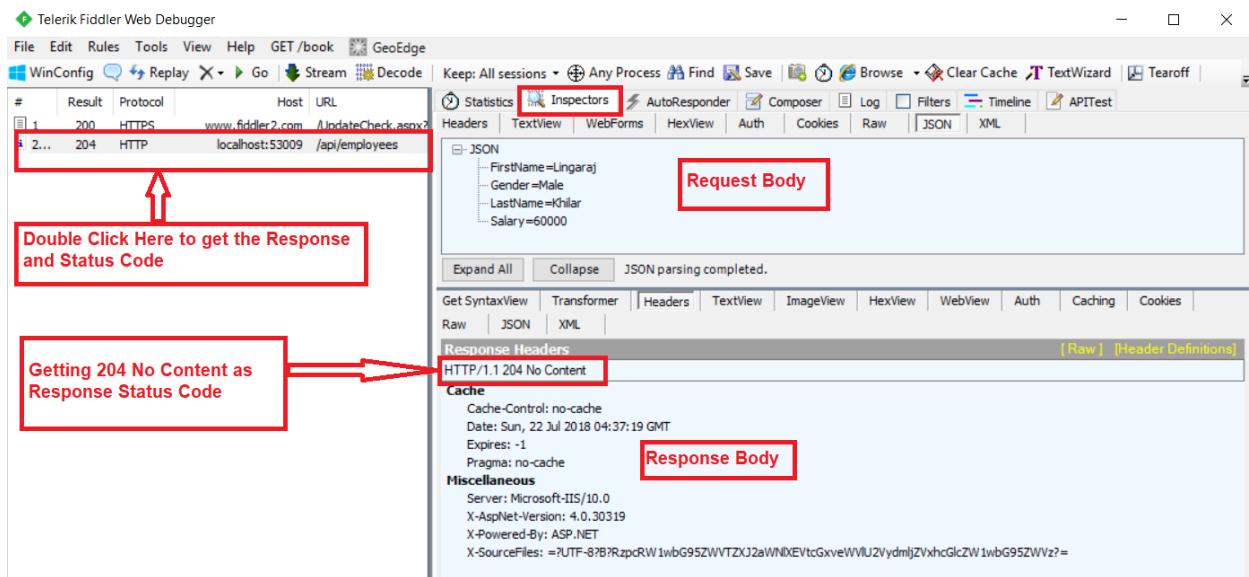
The screenshot shows the Fiddler application's Composer tab. The request method is set to POST, the URL is `http://localhost:53009/api/employees`, and the protocol is HTTP/1.1. The Headers section includes `User-Agent: Fiddler`, `Host: localhost:53009`, `Content-type: application/json`, and `accept: application/json`. The Request Body section contains the following JSON object:

```
{"FirstName": "Lingaraj", "LastName": "Khilar", "Gender": "Male", "Salary": 60000}
```

A red box highlights the JSON object in the Request Body. To the right of the main interface, there is a sidebar with a checkbox for "Log Requests" and a history list containing the entry `localhost:53009/api/e...`. A red box also highlights the "Execute" button at the top right of the Fiddler window.

Here we are passing the Employee Object in JSON Format

Khi chúng ta nhấp vào nút Execute, nó sẽ cung cấp cho chúng ta Phản hồi



Điều này hoạt động tốt và thêm nhân viên vào cơ sở dữ liệu như mong đợi. Vấn đề ở đây là vì kiểu trả về của phương thức Post là void nên chúng ta nhận được mã trạng thái *204 No Content*. Theo tiêu chuẩn REST, khi một mục mới được tạo, nó sẽ trả về mã trạng thái 201 Mục Đã tạo. Với mã trạng thái 201, chúng tôi cũng có thể bao gồm vị trí, tức là URI của mục mới được tạo.

Hãy xem cách đạt được điều này. Để đạt được điều này, chúng ta cần sửa đổi phương thức POST như hình dưới đây.

EmployeesController.cs

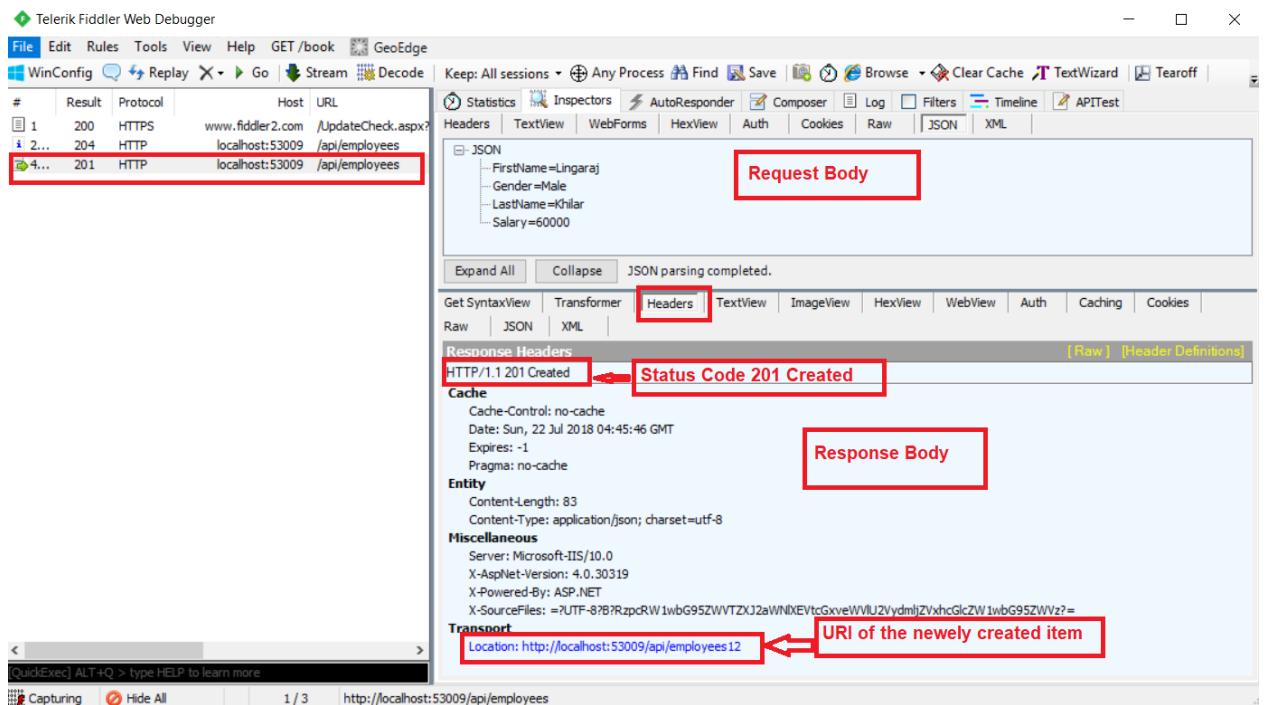
```
public class EmployeesController : ApiController
{
    public HttpResponseMessage Post([FromBody] Employee employee)
    {
        try
        {
            using (EmployeeDbContext dbContext = new EmployeeDbContext())
            {
                dbContext.Employees.Add(employee);
                dbContext.SaveChanges();
                var message = Request.CreateResponse(HttpStatusCode.Created, employee);
                message.Headers.Location = new Uri(Request.RequestUri +
                    employee.ID.ToString());
                return message;
            }
        }
        catch (Exception ex)
    }
}
```

```

    {
        return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);
    }
}
}
}

```

Tại thời điểm này, hãy đưa ra một yêu cầu Post request từ Fiddler. Lưu ý trong tiêu đề phản hồi, chúng tôi có mã trạng thái 201 Đã tạo và cũng có vị trí tức là URI của mục mới được tạo như thể hiện trong hình ảnh bên dưới.



Những điểm cần nhớ khi làm việc với POST method trong Web API:

1. Nếu kiểu trả về phương thức bị vô hiệu trong Dịch vụ Web API thì theo mặc định, Dịch vụ Web API sẽ trả về mã trạng thái 204 Không có Nội dung.
2. Khi một mục mới được tạo, chúng ta sẽ trả về mã trạng thái 201 Mục Đã tạo.
3. Với mã trạng thái 201, chúng ta cũng nên bao gồm vị trí, tức là URI của mục mới được tạo.
4. Khi không tìm thấy một mục, thay vì trả về NULL và mã trạng thái 200 OK, hãy trả về mã trạng thái 404 Not Found cùng với một thông báo có ý nghĩa như “Không tìm thấy nhân viên có Id = 15”

4.4 Triển khai phương thức PUT

Phương thức PUT trong Web API cho phép cập nhật một item. Giả sử ta muốn cập nhật nhân viên theo Id. Bao gồm phương thức PUT sau đây trong EmployeesController. Lưu ý rằng id của nhân viên mà chúng ta muốn cập nhật và đối tượng Employee mà chúng ta muốn cập nhật đang

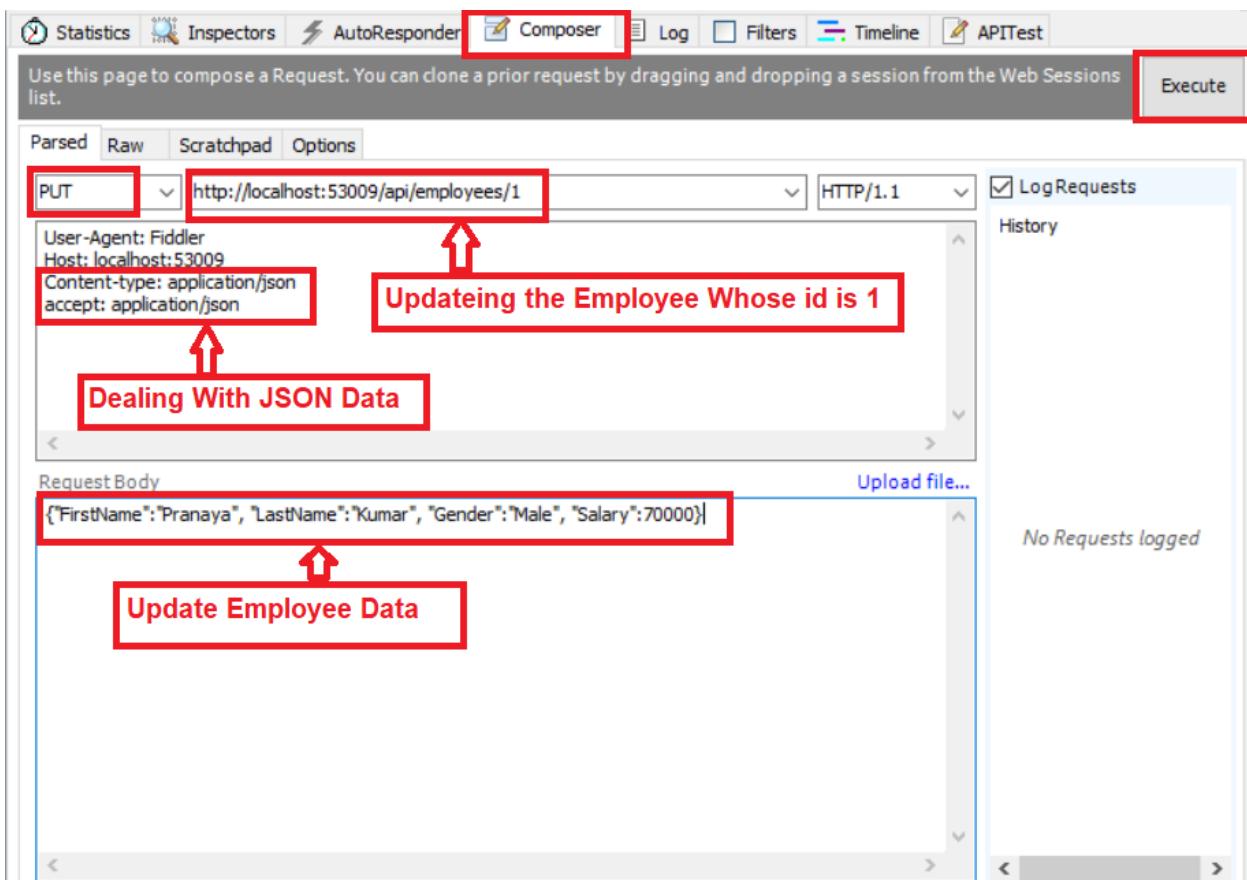
được truyền dưới dạng tham số cho phương thức Post. Tham số Nhân viên Employee được trang trí bằng thuộc tính [FromBody]. Điều này yêu cầu Web API lấy dữ liệu nhân viên từ request body

EmployeesController.cs

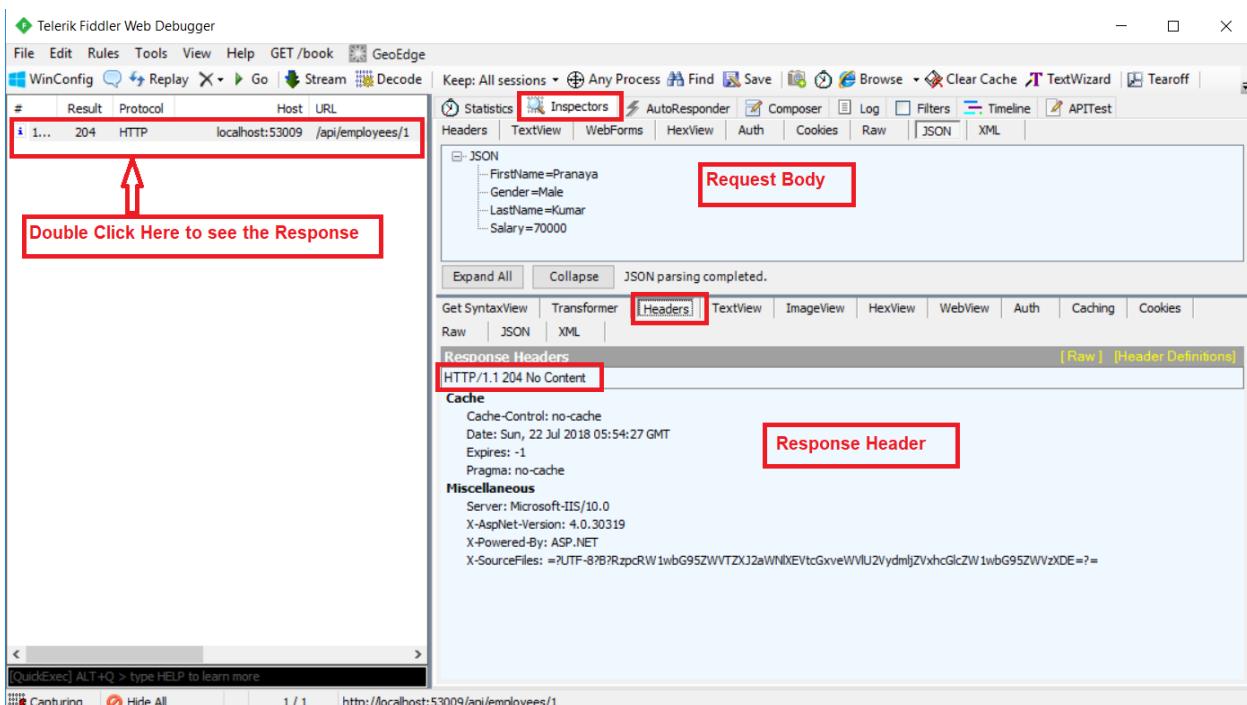
```
public class EmployeesController : ApiController
{
    public void Put(int id, [FromBody]Employee employee)
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var entity = dbContext.Employees.FirstOrDefault(e => e.ID == id);
            entity.FirstName = employee.FirstName;
            entity.LastName = employee.LastName;
            entity.Gender = employee.Gender;
            entity.Salary = employee.Salary;
            dbContext.SaveChanges();
        }
    }
}
```

Tại thời điểm này, hãy xây dựng giải pháp, chạy ứng dụng và kích hoạt Fiddler và đưa ra PUT request.

1. Chọn giao thức HTTP là PUT
2. Content-Type: application / json. Điều này cho biết rằng ta đang gửi dữ liệu có định dạng JSON đến máy chủ
3. Trong Request Body, hãy bao gồm đối tượng nhân viên được cập nhật mà bạn muốn cập nhật
2. Cuối cùng, click the execute button as shown below



Khi chúng ta nhấp vào nút execute, nó sẽ cho chúng ta phản hồi bên dưới



Điều này hoạt động tốt và cập nhật hồ sơ nhân viên trong cơ sở dữ liệu như mong đợi. Vấn đề ở đây là vì kiểu trả về của phương thức Put là void nên chúng ta nhận được mã trạng thái 204 No Content. Khi cập nhật thành công, chúng tôi muốn trả về mã trạng thái 200 OK cho biết cập nhật thành công.

Ngoài ra, khi ta muốn cập nhật nhân viên có Id không tồn tại, chúng tôi nhận lại mã trạng thái HTTP 500 Lỗi máy chủ nội bộ. ta nhận được mã trạng thái 500, do ngoại lệ tham chiếu NULL. Để khắc phục cả hai vấn đề này, hãy sửa đổi mã trong phương thức PUT như được hiển thị bên dưới.

EmployeesController.cs

```
public class EmployeesController : ApiController
{
    public HttpResponseMessage Put(int id, [FromBody]Employee employee)
    {
        try
        {
            using (EmployeeDBContext dbContext = new EmployeeDBContext())
            {
                var entity = dbContext.Employees.FirstOrDefault(e => e.ID == id);
                if (entity == null)
                {
                    return Request.CreateErrorResponse(HttpStatusCode.NotFound,
                        "Employee with Id " + id.ToString() + " not found to update");
                }
                else
                {
                    entity.FirstName = employee.FirstName;
                    entity.LastName = employee.LastName;
                    entity.Gender = employee.Gender;
                    entity.Salary = employee.Salary;
                    dbContext.SaveChanges();
                    return Request.CreateResponse(HttpStatusCode.OK, entity);
                }
            }
        }
        catch (Exception ex)
        {
            return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);
        }
    }
}
```

```
}
```

Thực thi một Put request from Fiddler. Lưu ý trong tiêu đề response header phản hồi có mã trạng thái 200 OK. Ngoài ra, khi chúng tôi cố gắng cập nhật nhân viên có id không tồn tại, chúng tôi nhận được mã trạng thái 404 Not Found thay vào lỗi 500 Internal Server Error

4.5 Triển khai phương thức DELETE

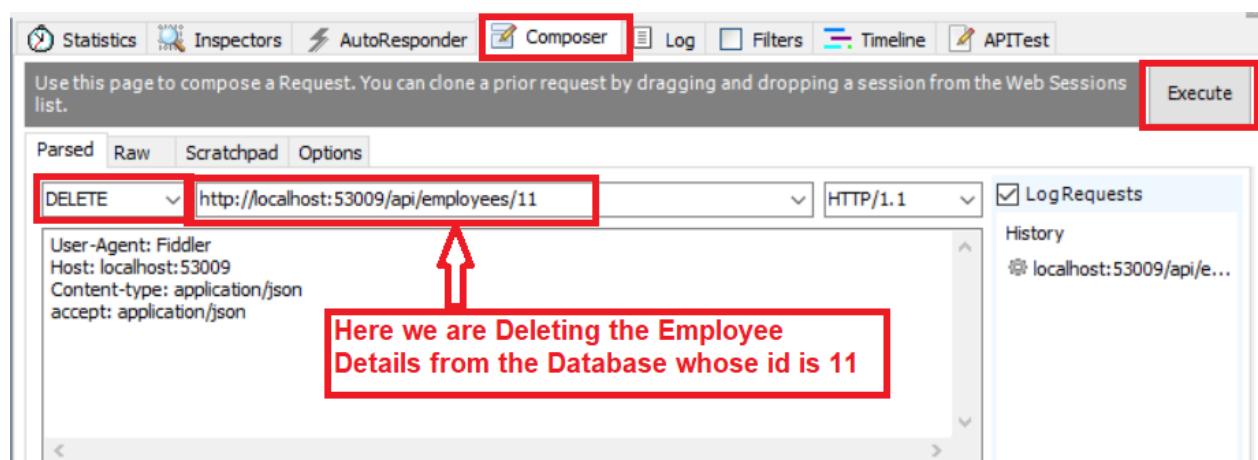
Phương thức DELETE trong Web API cho phép xóa một item. Muốn xóa một nhân viên được chỉ định khỏi bảng cơ sở dữ liệu Nhân viênxây dựng phương thức DELETE như sau trong EmployeesController

EmployeesController.cs

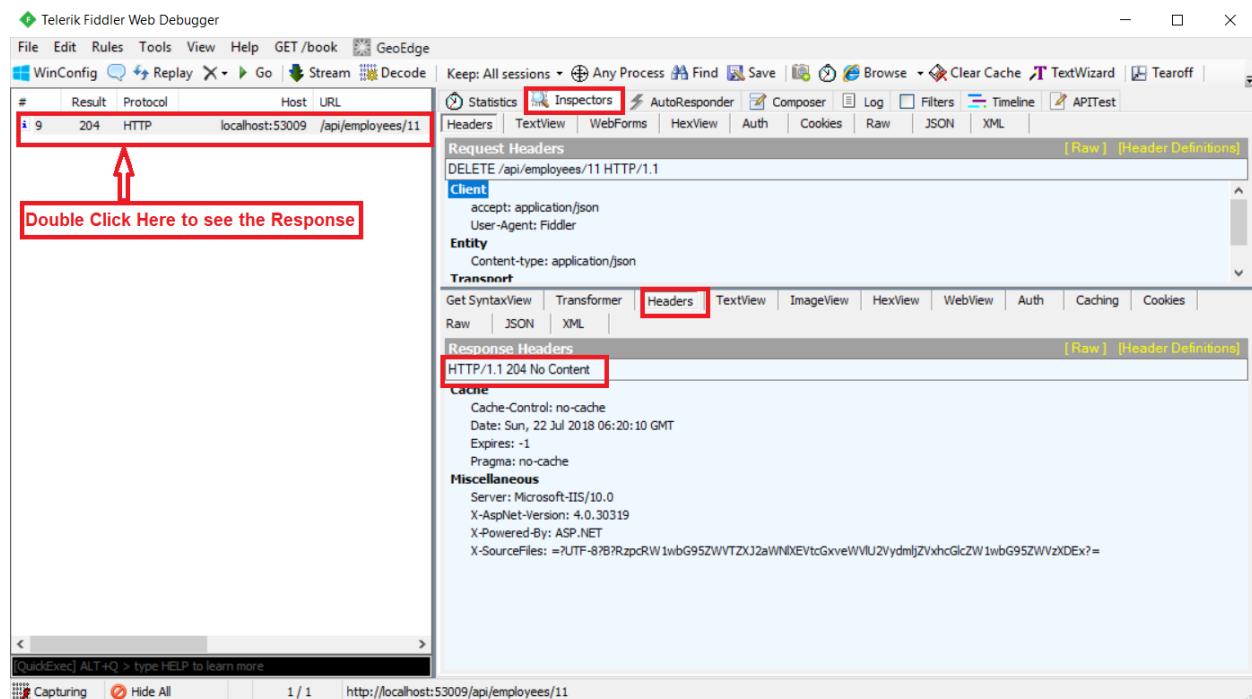
```
public class EmployeesController : ApiController
{
    public void Delete(int id)
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            dbContext.Employees.Remove(dbContext.Employees.FirstOrDefault(e => e.ID == id));
            dbContext.SaveChanges();
        }
    }
}
```

Kiểm tra phương thức DELETE: Chạy ứng dụng và kích hoạt Fiddler và đưa ra request DELETE.

1. Chọn giao thức HTTP là DELETE
2. Content-Type: application/json.. Điều này cho biết rằng chúng tôi đang gửi dữ liệu có định dạng JSON đến máy chủ
3. Cuối cùng bấm vào nút execute như hình bên dưới



Khi chúng ta nhấp vào nút execute, nó sẽ cho chúng ta phản hồi bên dưới



Trường hợp hoạt động tốt và xóa nhân viên khỏi cơ sở dữ liệu như mong đợi. Vấn đề ở đây là vì kiểu trả về của phương thức Delete là void nên chúng ta nhận được mã trạng thái 204 No Content. Khi Xóa thành công, ta muốn trả về mã trạng thái 200 OK cho biết rằng việc xóa thành công.

Ngoài ra, khi ta cố gắng xóa một nhân viên có Id không tồn tại, chúng tôi nhận lại mã trạng thái status code là “500 Internal Server Error”. Ta nhận mã trạng thái là 500 là do ngoại lệ tham chiếu NULL. Nếu một mục không được tìm thấy, thì chúng tôi cần trả lại mã trạng thái 404 Không tìm thấy.

=> Làm thế nào để khắc phục các vấn đề trên?

Để khắc phục cả hai vấn đề này, ta sửa đổi mã trong DELETE method như sau:

EmployeesController.cs

```
public class EmployeesController : ApiController
{
    public HttpResponseMessage Delete(int id)
    {
        try
        {
            using (EmployeeDBContext dbContext = new EmployeeDBContext())
            {
                var entity = dbContext.Employees.FirstOrDefault(e => e.ID == id);
                if (entity == null)
```

```
        {
            return Request.CreateErrorResponse(HttpStatusCode.NotFound,
                "Employee with Id = " + id.ToString() + " not found to delete");
        }
    else
    {
        dbContext.Employees.Remove(entity);
        dbContext.SaveChanges();
        return Request.CreateResponse(HttpStatusCode.OK);
    }
}
}

catch (Exception ex)
{
    return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);
}
}
}
```

Tại thời điểm này, đưa ra một yêu cầu XÓA khác từ Fiddler. Lưu ý trong tiêu đề phản hồi chúng ta có mã trạng thái 200 OK. Ngoài ra, khi chúng tôi cố gắng xóa một nhân viên có id không tồn tại, chúng tôi nhận được mã trạng thái 404 Not Found thay vì 500 Internal Server Error

Kiểm tra một request DELETE bằng Fiddler. Lưu ý trong the response header ta có mã trạng thái 200 là thành công OK. Ngoài ra, khi chúng tôi cố gắng xóa một nhân viên có id không tồn tại, ta nhận được mã trạng thái 404 Not Found thay vì 500 Internal Server Error.

4.6 Tùy chỉnh tên phương thức trong Web API controller

Theo quy ước mặc định được sử dụng bởi ASP.NET Web API để ánh xạ các động từ HTTP GET, PUT, POST và DELETE với các phương thức trong controller với một ví dụ.

Theo mặc định, động từ HTTP GET được ánh xạ tới một phương thức trong bộ điều khiển có tên Get () hoặc bắt đầu bằng từ Get.

Trong những điều sau đây **EmployeesController**, tên phương pháp là **Get()** vì vậy theo quy ước mặc định, điều này được ánh xạ tới động từ HTTP GET. Ngay cả khi chúng tôi đổi tên nó thành **GetEmployees()** hoặc **GetSomething()** nó sẽ vẫn được ánh xạ tới động từ HTTP GET miễn là tên của phương thức được bắt đầu bằng từ Get. Từ Get không phân biệt chữ hoa chữ thường. Nó có thể là chữ thường, chữ hoa hoặc kết hợp cả hai.

EmployeesController.cs

```

public class EmployeesController : ApiController
{
    public HttpResponseMessage Get()
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var Employees = dbContext.Employees.ToList();
            return Request.CreateResponse(HttpStatusCode.OK, Employees);
        }
    }
}

```

Nếu phương thức không được đặt tên là GET hoặc không bắt đầu bằng từ GET thì Web API không biết tên phương thức, yêu cầu GET phải được ánh xạ và yêu cầu không thành công với thông báo lỗi nêu rõ **The requested resource does not support HTTP method ‘GET’ with the status code 405 Method Not Allowed.**

Trong ví dụ sau, ta đã đổi tên phương thức Get () thành LoadAllEmployees (). Khi đó, nếu ta đưa ra một yêu cầu GET thì yêu cầu sẽ không thành công vì ASP.NET Web API không biết nó phải ánh xạ yêu cầu GET tới phương thức này.

EmployeesController.cs

```

public class EmployeesController : ApiController
{
    public HttpResponseMessage LoadAllEmployees()
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var Employees = dbContext.Employees.ToList();
            return Request.CreateResponse(HttpStatusCode.OK, Employees);
        }
    }
}

```

Để hướng dẫn Web API ánh xạ động từ HTTP sang phương thức LoadAllEmployees (), cần khai báo phương thức với thuộc tính [HttpGet] như dưới đây.

EmployeesController.cs

```

public class EmployeesController : ApiController

```

```

{
    [HttpGet]
    public HttpResponseMessage LoadAllEmployees()
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var Employees = dbContext.Employees.ToList();
            return Request.CreateResponse(HttpStatusCode.OK, Employees);
        }
    }
}

```

Giả sử ta có hai phương thức trong EmployeesController

EmployeesController.cs

```

public class EmployeesController : ApiController
{
    [HttpGet]
    public HttpResponseMessage LoadAllEmployees()
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var Employees = dbContext.Employees.ToList();
            return Request.CreateResponse(HttpStatusCode.OK, Employees);
        }
    }

    public HttpResponseMessage LoadEmployeeByID(int id)
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var entity = dbContext.Employees.FirstOrDefault(e => e.ID == id);
            if (entity != null)
            {
                return Request.CreateResponse(HttpStatusCode.OK, entity);
            }
        }
    }
}

```

```

else
{
    return Request.CreateErrorResponse(HttpStatusCode.NotFound,
    "Employee with ID " + id.ToString() + "not found");
}
}
}
}

```

Khi đó, ta điều hướng đến URI là **/api/employees/1** ta nhận tất cả Nhân viên, thay vì chỉ Nhân viên có Id = 1. Điều này là do trong trường hợp này, yêu cầu GET được ánh xạ tới **LoadAllEmployees()** và không thực hiện phương thức **LoadEmployeeById(int id)**.

Nếu muốn yêu cầu GET được ánh xạ tới **LoadEmployeeById(int id)** khi tham số id được chỉ định trong URI, thì khi xây dựng phương thức **LoadEmployeeById(int id)** khai báo thuộc tính [HttpGet] như bên dưới.

EmployeesController.cs

```

public class EmployeesController : ApiController
{
    [HttpGet]
    public HttpResponseMessage LoadAllEmployees()
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var Employees = dbContext.Employees.ToList();
            return Request.CreateResponse(HttpStatusCode.OK, Employees);
        }
    }
    [HttpGet]
    public HttpResponseMessage LoadEmployeeByID(int id)
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var entity = dbContext.Employees.FirstOrDefault(e => e.ID == id);
            if (entity != null)

```

```

{
    return Request.CreateResponse(HttpStatusCode.OK, entity);
}
else
{
    return Request.CreateErrorResponse(HttpStatusCode.NotFound,
    "Employee with ID " + id.ToString() + "not found");
}
}
}
}
}

```

Trong thời gian thực, ta có thể phải triển khai nhiều phương thức get hoặc post hoặc put trong một ApiController duy nhất. Ví dụ, chúng ta có hai phương pháp như sau:

```

public class EmployeesController : ApiController
{
    [HttpGet]
    public HttpResponseMessage LoadAllEmployees()
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var Employees = dbContext.Employees.ToList();
            return Request.CreateResponse(HttpStatusCode.OK, Employees);
        }
    }

    [HttpGet]
    public HttpResponseMessage LoadAllMaleEmployees()
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var Employees = dbContext.Employees.Where(x => x.Gender ==
"Male").ToList();
            return Request.CreateResponse(HttpStatusCode.OK, Employees);
        }
    }
}

```

```

        }
    }
}
```

Bây giờ khi chúng tôi thực hiện yêu cầu với URL / api / staff Chúng tôi sẽ gặp lỗi bên dưới

```

▼<Error>
  <Message>An error has occurred.</Message>
  ▼<ExceptionMessage>
    Multiple actions were found that match the request: System.Collections.Generic.IEnumerable`1[EmployeeDataAccess.Employee] LoadAllEmployees() on type EmployeeService.Controllers.EmployeesController System.Collections.Generic.IEnumerable`1[EmployeeDataAccess.Employee] LoadAllMaleEmployees() on type EmployeeService.Controllers.EmployeesController
  </ExceptionMessage>
  <ExceptionType>System.InvalidOperationException</ExceptionType>
  ▼<StackTrace>
    at System.Web.Http.Controllers.ApiControllerActionSelector.ActionSelectorCacheItem.SelectAction(HttpContext controllerContext) at System.Web.Http.Controllers.ApiControllerActionSelector.SelectAction(HttpContext controllerContext) at System.Web.Http.ApiController.Execute(controllerContext, CancellationToken cancellationToken) at System.Web.Http.Dispatcher.HttpControllerDispatcher.SendAsyncCore(HttpRequestMessage request, CancellationToken cancellationToken) at System.Web.Http.Dispatcher.HttpControllerDispatcher.<SendAsync>d__0.MoveNext()
  </StackTrace>
</Error>
```

? Câu hỏi đặt ra là làm thế nào để truy cập hai phương thức này cùng với cách chúng tôi sẽ cung cấp một URL duy nhất cho mỗi tài nguyên.

Trước tiên, hãy xem cách triển khai mặc định của lớp WebApiConfig có trong Thư mục App_Start.

WebApiConfig

```

public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        config.MapHttpAttributeRoutes();
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

Tuyến mặc định chỉ định tuyến URL là tên miền Theo sau là API và tên bộ điều khiển. Trong ví dụ của chúng tôi, nó sẽ là http:// localhost:xxxxx / api / architects trong đó "Employee" là tên controller của EmployeesController

Triển khai tên phương thức tùy chỉnh trong Web API:

Để triển khai tên phương thức tùy chỉnh trong Web API, trước tiên hãy thay đổi cách triển khai mặc định của lớp WebApiConfig như được hiển thị bên dưới

WebApiConfig

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{action}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

Bây giờ hãy thực hiện một yêu cầu với cùng một URL / api / staff Nó sẽ cho chúng ta lỗi sau



Server Error in '/' Application.

The resource cannot be found.

Description: HTTP 404. The resource you are looking for (or one of its dependencies) could have been removed, has changed, or is temporarily unavailable.

Requested URL: /api/employees

Version Information: Microsoft .NET Framework Version:4.0.30319; ASP.NET Version:4.6.1586.0

Vì vậy, hãy thay đổi URL khi chúng ta cần đưa tên hành động vào URL khi chúng ta thực hiện các thay đổi trong lớp WebApiConfig.

/api/employees/LoadAllEmployees
/api/employees/LoadAllMaleEmployees

4.7 Tham số ràng buộc trong Web API

Tham số ràng buộc trong ASP.NET Web API có nghĩa là cách Khung Web API liên kết dữ liệu yêu cầu HTTP đến với các tham số của phương thức hành động của bộ điều khiển Web API.

Các phương thức hành động của ASP.NET Web API có thể nhận một hoặc nhiều tham số thuộc các kiểu khác nhau. Các tham số của phương thức hành động có thể thuộc kiểu phức tạp hoặc kiểu nguyên thủy. Khung Web API liên kết các tham số phương thức hành động với chuỗi truy vấn của URL hoặc từ nội dung yêu cầu của Yêu cầu HTTP đến dựa trên loại tham số.

Liên kết tham số mặc định trong Web API

Theo mặc định, nếu kiểu tham số là kiểu nguyên thủy như int, bool, double, string, GUID, DateTime, decimal hoặc bất kỳ kiểu nào khác có thể được chuyển đổi từ kiểu chuỗi thì Web API Framework sẽ đặt giá trị tham số phương thức hành động từ chuỗi truy vấn. Và nếu kiểu tham số của phương thức hành động là kiểu phức tạp thì Web API Framework sẽ cố gắng lấy giá trị từ phần thân của yêu cầu và đây là bản chất mặc định của Tham số ràng buộc trong Web API Framework.

Bảng sau liệt kê các quy tắc mặc định cho Liên kết tham số Web API.

HTTP Method	Query String	Request Body
GET	Primitive Type, Complex Type	NA
POST	Primitive Type	Complex Type
PUT	Primitive Type	Complex Type
PATCH	Primitive Type	Complex Type
DELETE	Primitive Type, Complex Type	NA

Hiểu liên kết tham số trong ASP.NET Web API với một ví dụ.

Sao chép và dán mã sau vào EmployeesController.

```
public class EmployeesController : ApiController
{
    public HttpResponseMessage GET()
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var Employees = dbContext.Employees.ToList();
            return Request.CreateResponse(HttpStatusCode.OK, Employees);
        }
    }
}
```

```
}
```

Tùy thuộc vào giá trị mà chúng tôi chỉ định cho giới tính tham số chuỗi truy vấn, phương thức Get () sẽ trả về dữ liệu.

/api/employees?gender=all trả lại All Employees

/api/employees?gender=Male trả lại All Male Employees

/api/employees?gender=Female trả lại All Female Employees

Nếu giá trị cho giới tính không phải là Nam, Nữ hoặc Tất cả, thì dịch vụ sẽ trả về mã trạng thái **400 Bad Request**. Ví dụ: nếu chúng tôi chỉ định ABC làm giá trị cho giới tính, thì dịch vụ sẽ trả về mã trạng thái **400 Bad Request** với tin nhắn sau.

Giá trị cho giới tính phải là Nam, Nữ hoặc Tất cả. ABC không hợp lệ.

Dưới đây là phương thức Get () đã được sửa đổi

Giới tính đang được truyền dưới dạng tham số cho phương thức Get (). Giá trị mặc định là "All". Giá trị mặc định làm cho tham số là tùy chọn. Tham số giới tính của phương thức Get () được ánh xạ tới tham số giới tính được gửi trong chuỗi truy vấn

```
public class EmployeesController : ApiController
{
    public HttpResponseMessage Get(string gender = "All")
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            switch (gender.ToLower())
            {
                case "all":
                    return Request.CreateResponse(HttpStatusCode.OK,
                        dbContext.Employees.ToList());
                case "male":
                    return Request.CreateResponse(HttpStatusCode.OK,
                        dbContext.Employees.Where(e => e.Gender.ToLower() == "male").ToList());
                case "female":
                    return Request.CreateResponse(HttpStatusCode.OK,
                        dbContext.Employees.Where(e => e.Gender.ToLower() ==
"female").ToList());
                default:
                    return Request.CreateErrorResponse(HttpStatusCode.BadRequest,
                        new { message = "Gender must be all, male or female" });
            }
        }
    }
}
```

```

        "Value for gender must be Male, Female or All. " + gender + " is invalid.");
    }
}
}
}

```

Thuộc tính FromBody và FromUri

Hãy để chúng tôi hiểu công dụng của chúng với một ví dụ. Hãy xem xét phương thức Put () sau đây. Phương pháp này cập nhật chi tiết Nhân viên được chỉ định. Thêm phương thức PUT sau trong Employees Controller

```

public HttpResponseMessage Put(int id, Employee employee)
{
    try
    {
        using (EmployeeDBContext dbContext = new EmployeeDBContext())
        {
            var entity = dbContext.Employees.FirstOrDefault(e => e.ID == id);
            if (entity == null)
            {
                return Request.CreateErrorResponse(HttpStatusCode.NotFound,
                    "Employee with Id " + id.ToString() + " not found to update");
            }
            else
            {
                entity.FirstName = employee.FirstName;
                entity.LastName = employee.LastName;
                entity.Gender = employee.Gender;
                entity.Salary = employee.Salary;
                dbContext.SaveChanges();
                return Request.CreateResponse(HttpStatusCode.OK, entity);
            }
        }
    }
}

```

```

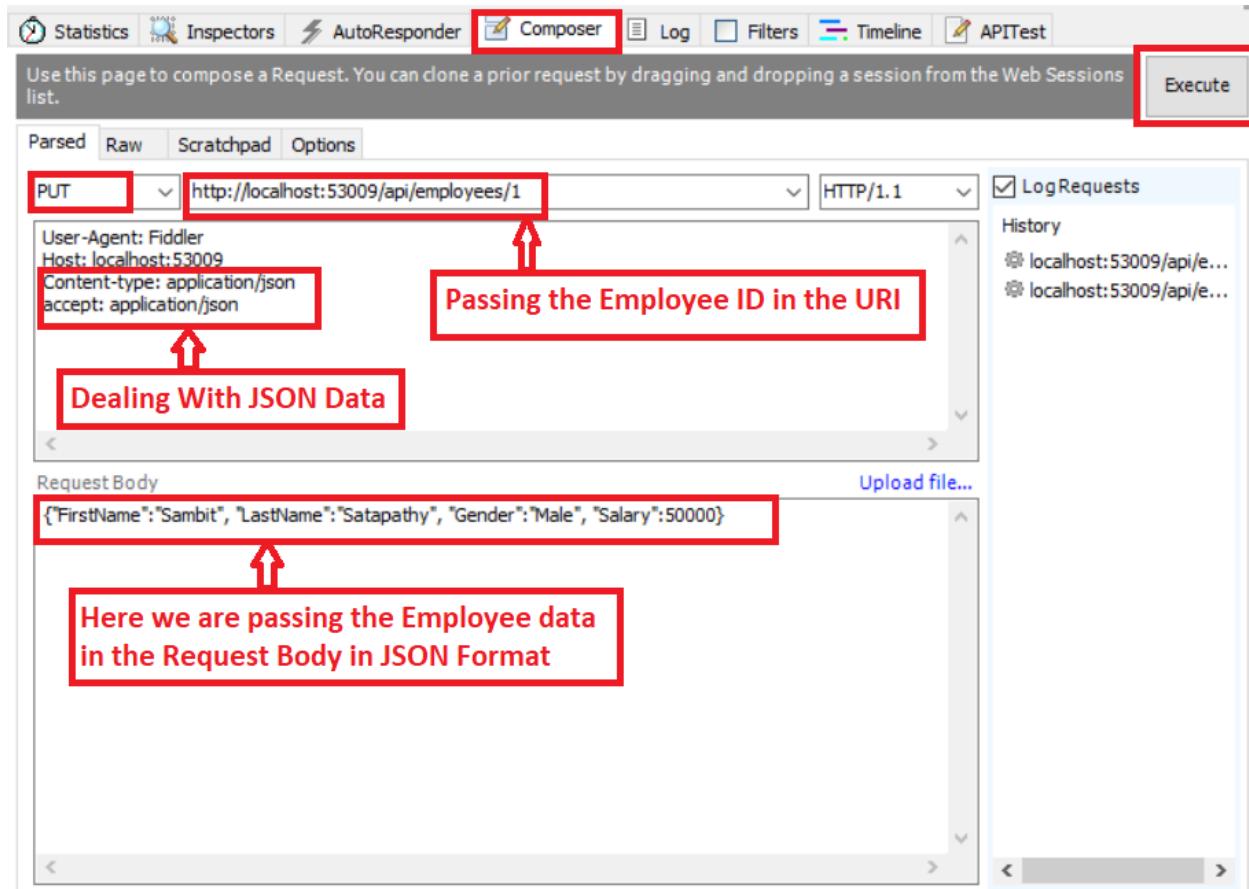
    }
    catch (Exception ex)
    {
        return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);
    }
}

```

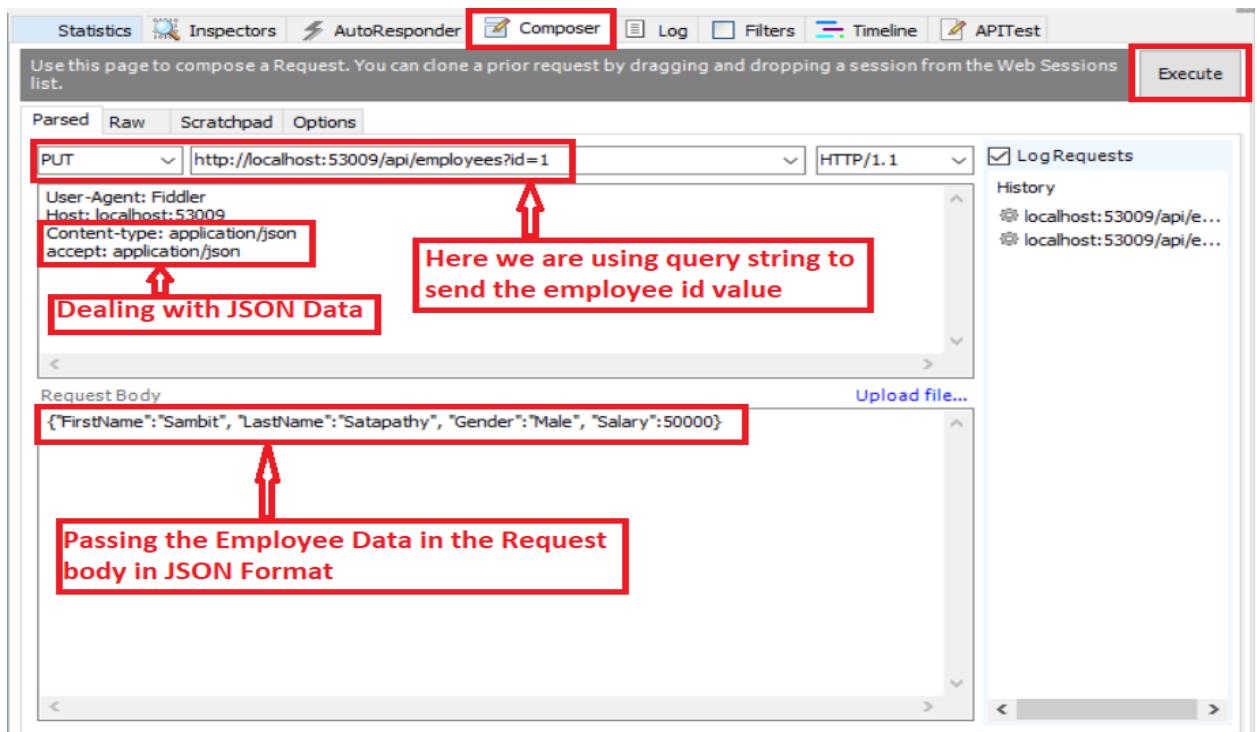
Để cập nhật thông tin chi tiết về nhân viên có Id là 1, chúng tôi đưa ra yêu cầu Đặt cho URI / api / architects / 1

Lưu ý: Tên tham số chuỗi truy vấn phải khớp với tên của tham số phương thức hành động. Tuy nhiên, chúng có thể theo một thứ tự khác.

Nếu bạn đang sử dụng Fiddler, yêu cầu PUT như hình dưới đây. Lưu ý rằng Id của nhân viên nằm trong URI và dữ liệu của nhân viên nằm trong phần thân yêu cầu.



Tại thời điểm này, nếu bạn thực hiện yêu cầu, dữ liệu nhân viên được cập nhật như mong đợi. Nay giờ, hãy bao gồm Id dưới dạng tham số chuỗi truy vấn. Trong yêu cầu đầu tiên, Id được chỉ định như một phần của dữ liệu truyền đường. Lưu ý trong Fiddler, chúng tôi đã bao gồm tham số id dưới dạng một chuỗi truy vấn.



Khi chúng tôi thực hiện yêu cầu này, bản cập nhật sẽ thành công như mong đợi. Khi một yêu cầu PUT được đưa ra, Web API ánh xạ dữ liệu trong yêu cầu tới các tham số của phương thức PUT trong Bộ điều khiển nhân viên. Quá trình này được gọi là Tham số ràng buộc.

Quy ước mặc định được sử dụng bởi Web API cho liên kết tham số.

Nếu tham số là một loại đơn giản như int, bool, double, v.v., Web API sẽ cố gắng lấy giá trị từ URI (Từ dữ liệu truyền hoặc từ Chuỗi truy vấn) trong khi nếu tham số là một loại phức tạp như Khách hàng, Nhân viên , v.v., sau đó Khung Web API cố gắng lấy giá trị từ phần thân yêu cầu.

Vì vậy, trong trường hợp của chúng tôi, tham số id là một loại đơn giản, vì vậy Web API cố gắng lấy giá trị từ URI yêu cầu. Tham số nhân viên là một loại phức tạp, vì vậy Web API nhận giá trị từ phần thân yêu cầu.

Lưu ý: Tên của thuộc tính kiểu phức tạp và tham số chuỗi truy vấn phải khớp với nhau.

Chúng ta có thể thay đổi hành vi mặc định này của quá trình ràng buộc tham số bằng cách sử dụng các thuộc tính [FromBody] và [FromUri]. Lưu ý trong ví dụ dưới đây, chúng tôi đã trang trí tham số id bằng thuộc tính [FromBody], điều này sẽ buộc Khung Web API lấy nó từ phần thân yêu cầu. Chúng tôi cũng đã trang trí đối tượng nhân viên bằng thuộc tính [FromUri], thuộc tính này sẽ buộc Khung Web API lấy dữ liệu nhân viên từ URI (tức là dữ liệu truyền đường hoặc chuỗi truy vấn)

```
public HttpResponseMessage Put([FromBody] int id, [FromUri] Employee employee)
{
    try
    {
        using (EmployeeDbContext dbContext = new EmployeeDbContext())
        {
```

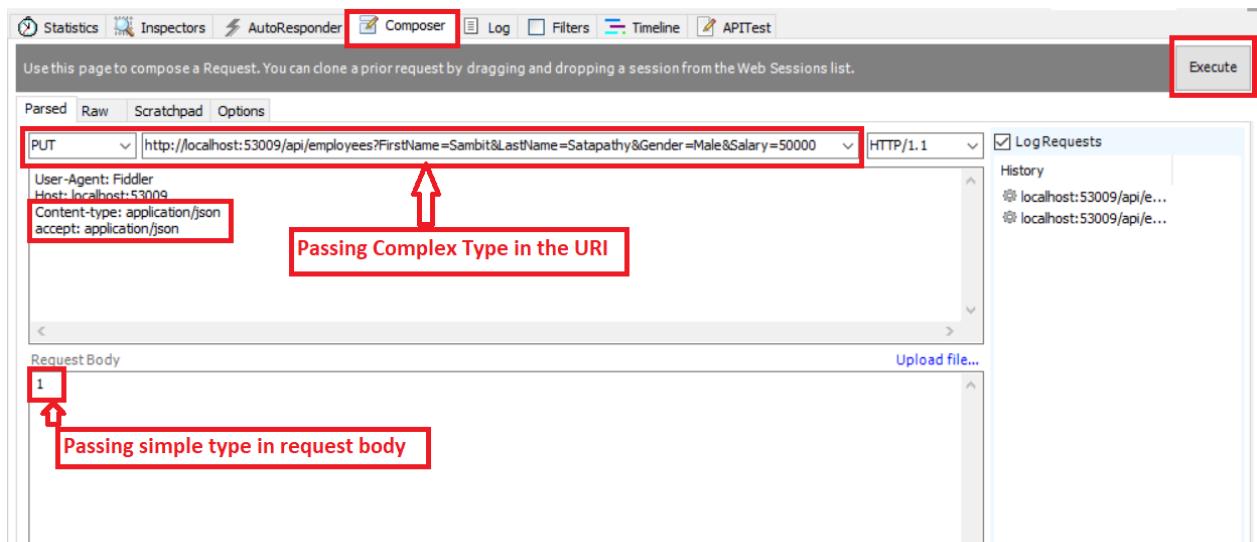
```

var entity = dbContext.Employees.FirstOrDefault(e => e.ID == id);
if (entity == null)
{
    return Request.CreateErrorResponse(HttpStatusCode.NotFound,
    "Employee with Id " + id.ToString() + " not found to update");
}
else
{
    entity.FirstName = employee.FirstName;
    entity.LastName = employee.LastName;
    entity.Gender = employee.Gender;
    entity.Salary = employee.Salary;
    dbContext.SaveChanges();
    return Request.CreateResponse(HttpStatusCode.OK, entity);
}
}
catch (Exception ex)
{
    return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);
}

```

Đây là yêu cầu từ Fiddler

1. Dữ liệu nhân viên được chỉ định trong URI bằng cách sử dụng các tham số chuỗi truy vấn
2. Id được chỉ định trong phần thân yêu cầu



Khi chúng ta thực hiện yêu cầu, cập nhật thành công như mong đợi

4.8 Thảo luận

CHƯƠNG 5 ĐỊNH TUYẾN WEB API

5.1 Giới thiệu về Định tuyến Web API ASP.NET

Định tuyến Web API thực hiện ánh xạ các HTTP request gửi đến với một phương thức thực thi trong controller xác định. Định tuyến Web API khá giống với định tuyến Web MVC nên biết lập trình Web MVC là một lợi thế khi tiếp cận lập trình định tuyến Web API.

Sự khác biệt chính giữa hai cơ chế định tuyến Web API và Web MVC là: Web API sử dụng phương thức HTTP, không phải đường dẫn URI, để lựa chọn phương thức. Bạn cũng có thể sử dụng định tuyến kiểu MVC trong Web API mà chúng ta sẽ thảo luận trong các phần sau.

5.1.1 Bảng định tuyến trong Web API ASP.NET

Trong ứng dụng Web API, một bộ điều khiển controller là một lớp chứa các phương thức thực thi thực sự xử lý các Yêu cầu HTTP request đến. Các phương thức public của lớp controller được gọi là các phương thức thực thi hoặc đơn giản là các phương thức. Khi Web API Framework nhận được một Yêu cầu HTTP request, nó sẽ định tuyến yêu cầu HTTP request đó đến một phương thức thực thi của bộ điều khiển.

Để xác định phương thức thực thi nào để chọn hoặc gọi cho một Yêu cầu HTTP request cụ thể, WEB API framework sử dụng bảng Định tuyến. Khi tạo ứng dụng WEB API, theo mặc định, Visual Studio sẽ tạo một tuyến mặc định cho ứng dụng như thể hiện trong hình ảnh bên dưới.

```
config.Routes.MapHttpRoute(  
    name: "DefaultApi",  
    routeTemplate: "api/{controller}/{id}",  
    defaults: new { id = RouteParameter.Optional }  
)
```

Định tuyến routing như ở trên được xác định trong tệp WebApiConfig.cs, có trong thư mục App_Start.

Bảng định tuyến trong Web API chứa mỗi và mọi mẫu tuyến đường mà ta xác định trong tệp WebApiConfig. Mẫu tuyến mặc định cho ứng dụng Web API là “api/{controller}/{id}”. Thuật ngữ “api” là một đoạn đường dẫn theo nghĩa đen và {controller} và {id} là các biến giữ chỗ sẽ được thay thế bằng giá trị cụ thể.

5.1.2 WEB API framework xử lý yêu cầu HTTP request đến như thế nào?

Khi ASP.NET Web API Framework nhận được Yêu cầu HTTP request, nó sẽ cố gắng so khớp URI với một trong các mẫu tuyến có sẵn trong bảng định tuyến. Nếu không có mẫu tuyến nào phù hợp với URI, thì Web API Framework sẽ trả về lỗi 404 cho máy khách đưa ra yêu cầu.

Ví dụ: các URI sau phù hợp với mẫu tuyến đường mặc định

1. /api/student
2. /api/students/1
3. /api/products/prd1

Tuy nhiên, URI sau không khớp vì nó thiếu "api":

/products/1
/students/1

Ghi chú:

Lý do sử dụng “api” trong tuyến là để tránh xung đột giữa định tuyến Web API và MVC. Vì vậy, bạn có thể có “/ products” đi tới bộ điều khiển MVC controller và “/api/products” đi tới bộ điều khiển Web API controller. Tất nhiên, nếu bạn không thích quy ước này, bạn có thể thay đổi bảng lô trình mặc định (phần này sẽ thảo luận sau).

Sau khi tìm thấy một tuyến đường phù hợp trong mẫu tuyến đường mặc định đã định nghĩa, WEB API framework sẽ chọn bộ điều khiển Controller và thực thi phương thức tương ứng. Để tìm bộ điều khiển, WEB API framework thêm Bộ điều khiển “Controller” vào giá trị của biến {controller}. Để tìm phương thức, WEB API framework xem xét phương thức HTTP và sau đó tìm phương thức thực thi có tên bắt đầu bằng tên phương thức HTTP đó.

Ví dụ: với yêu cầu GET, WEB API framework tìm kiếm một phương thức bắt đầu bằng “Get”, chẳng hạn như “GetProduct” hoặc “GetAllProducts”. Quy ước này chỉ áp dụng cho các phương thức GET, POST, PUT và DELETE. Có thể kích hoạt các phương thức HTTP khác bằng cách sử dụng các thuộc tính trên bộ điều khiển của mình mà chúng ta sẽ thảo luận trong phần sắp tới.

Các biến tham gia trong mẫu đường dẫn định tuyến như {id} sẽ được ánh xạ tới các tham số trong phương thức thực thi trong controller.

Hãy xét một ví dụ với bộ điều khiển controller là Student như sau:

```
public class StudentController : ApiController
{
    public IEnumerable<Student> GetAllStudents() ...
    public Student GetStudentById(int id) ...
    public HttpResponseMessage DeleteStudent(int id) ...
    public HttpResponseMessage PutStudent(Student student) ...
}
```

Dưới đây là một số Yêu cầu HTTP request có thể cùng với phương thức được gọi cho mỗi yêu cầu:

HTTP Method	URI Path	Action	Parameter
GET	api/student	GetAllStudents	(none)
GET	api/student/2	GetStudentById	4
DELETE	api/student/2	DeleteStudent	4
POST	api/student	(no match)	
PUT	api/student	PutStudent	Student

Lưu ý rằng phân đoạn {**id**} của URI (nếu có) được ánh xạ tới tham số id của phương thức thực thi tương ứng. Trong ví dụ bộ điều khiển controller Student định nghĩa hai phương thức GET, một phương thức có tham số id và một phương thức **không có tham số**. Nó cũng định nghĩa một phương thức PUT nhận một tham số kiểu sinh viên từ phần thân yêu cầu body request.

Trong ví dụ trên, yêu cầu POST sẽ không thành công vì bộ điều khiển controller không có định nghĩa bất kỳ một phương thức "POST" nào.

5.2 Một số biến định tuyến trong WEB API

5.2.1 Các động từ HTTP Verb trong Web API

Trong các phần trước, ta đã thảo luận về việc sử dụng quy ước đặt tên mặc định do Web API Framework cung cấp. Thay vì sử dụng quy ước đặt tên mặc định cho các phương thức HTTP, bạn cũng có thể chỉ định rõ ràng phương thức HTTP cho phương thức bằng cách trang trí phương thức thực thi với thuộc tính `HttpGet`, `HttpPut`, `HttpPost` hoặc `HttpDelete`.

Xét ví dụ bộ điều khiển sau

```
public class StudentController : ApiController
{
    [HttpGet]
    public IEnumerable<Student> GetAllStudents()...
    [HttpGet]
    public Student FindStudentById(int id)...
    [HttpDelete]
    public HttpResponseMessage RemoveStudent(int id)...
    [HttpPost]
    public HttpResponseMessage AddStudent(Student student)...
}
```

Trong ví dụ trên, các phương thức `FindAllStudents` và `FindStudentById` được ánh xạ tới yêu cầu GET, trong khi phương thức `RemoveStudent` được ánh xạ tới Yêu cầu DELETE và phương thức `AddStudent` được ánh xạ tới Yêu cầu POST.

Nếu bạn muốn cho phép nhiều động từ HTTP trong một phương thức thực thi hoặc nếu bạn cho phép các phương thức HTTP khác với GET, PUT, POST và DELETE, thì bạn cần sử dụng thuộc tính `AcceptVerbs`, thuộc tính này có danh sách các phương thức HTTP như được hiển thị trong hình ảnh dưới đây

```
public class StudentController : ApiController
{
    [AcceptVerbs("GET", "HEAD")]
    public Student FindStudentById(int id)...
}
```

Trong ví dụ trên, phương thức thực thi `FindStudentById` được ánh xạ tới cả GET và HEAD HTTP Request.

5.2.2 Định tuyến trong Web API theo tên phương thức

Với mẫu định tuyến mặc định, WEB API framework sử dụng phương thức HTTP để chọn phương thức. Tuy nhiên, nếu muốn, bạn cũng có thể tạo tuyến đường của riêng mình trong đó tên phương thức được bao gồm như một phần của URI như được hiển thị trong hình dưới đây.

```
config.Routes.MapHttpRoute(  
    name: "ActionApi",  
    routeTemplate: "api/{controller}/{action}/{id}",  
    defaults: new { id = RouteParameter.Optional }  
) ;
```

Trong mẫu tuyến trên, tham số {action} đặt tên cho phương thức thực thi trên bộ điều khiển. Với kiểu định tuyến này, bạn cần sử dụng các thuộc tính để chỉ định các phương thức HTTP được phép.

Xét bộ điều khiển sau đây.

```
public class StudentController : ApiController  
{  
    [HttpGet]  
    public IEnumerable<Student> GetAllStudents()...  
    [HttpGet]  
    public Student FindStudentById(int id)...  
    [HttpDelete]  
    public HttpResponseMessage RemoveStudent(int id)...  
    [HttpPost]  
    public HttpResponseMessage AddStudent(Student student)...  
}
```

Trong ví dụ trên, một yêu cầu GET cho “api/Student/FindAllStudents” sẽ ánh xạ tới phương thức thực thi GetAllStudents.

Trong ASP.NET Web API bằng cách sử dụng thuộc tính ActionName, bạn cũng có thể ghi đè tên phương thức thực thi. Trong lớp Kiểm soát sinh viên bên dưới, chúng ta có hai phương thức ánh xạ tới “api/student/Hình ảnh”. Một phương thức thực thi hỗ trợ yêu cầu GET trong khi phương thức kia hỗ trợ yêu cầu POST HTTP.

```
public class StudentController : ApiController  
{  
    [HttpGet]  
    [ActionName("Image")]  
    public IEnumerable<Student> FetchStudentImage()...  
  
    [HttpPost]  
    [ActionName("Image")]  
    public HttpResponseMessage AddStudentImage(Student student)...  
}
```

Hiểu thuộc tính NonAction trong ASP.NET Web API:

Nếu muốn ngăn một phương thức thực thi được gọi dưới dạng phản hồi cho một Yêu cầu HTTP request, thì cần khai báo cho phương thức đó bằng thuộc tính NonAction. Điều này cho WEB API framework biết rằng phương thức không phải là một phương thức NonAction ngay cả khi nó phù hợp với các quy tắc định tuyến.

```
public class StudentController : ApiController
{
    [NonAction]
    public IEnumerable<Student> SomePrivateMethod() ...
}
```

Hiện tại, chúng ta đã thảo luận về tổng quan cấp cao về Định tuyến Web API ASP.NET.

5.3 Định tuyến và phương thức thực thi trong Web API

Mô-đun Định tuyến có ba giai đoạn chính:

1. Đối sánh URI với một mẫu tuyến đường.
2. Lựa chọn bộ điều khiển.
3. Chọn một phương thức thực thi.

5.3.1 Tuyến đường mẫu mặc định

Mẫu đường dẫn trong WEB API khá giống với đường dẫn URI, nhưng nó có thể có các giá trị giữ chỗ được chỉ định bằng dấu ngoặc nhọn như được hiển thị trong hình ảnh bên dưới.

```
routeTemplate: "api/{controller}/public/{category}/{id}",
```

Khi tạo một tuyến đường, cũng có thể cung cấp các giá trị mặc định cho một số hoặc tất cả các trình giữ chỗ như thể hiện trong hình dưới đây.

```
defaults: new { category = "all",
               id = RouteParameter.Optional
             }
```

Ta cũng có thể cung cấp một số ràng buộc sẽ hạn chế cách phân đoạn URI có thể khớp với trình giữ chỗ như được hiển thị bên dưới.

```
// Only matches if the "id" is one or more digits.
constraints: new { id = @"\d+" }
```

WEB API framework cố gắng khớp các phân đoạn trong đường dẫn URI với mẫu tuyến đường có trong bảng Tuyến đường, phải khớp chính xác từng ký tự tức là khớp với bất kỳ giá trị nào trừ khi chỉ định một số ràng buộc. WEB API framework không khớp với các phần khác của URI như tên máy chủ hoặc các tham số truy vấn, framework luôn chọn tuyến đường đầu tiên trong bảng định tuyến phù hợp với URI.

Có hai phần đặc biệt được sử dụng trong định tuyến WEB API là “{controller}” và “{action}”.

Phân đoạn tên trong URI là “{controller}” cung cấp tên của bộ điều khiển.

Tương tự, Phân đoạn tên trong URI là “{action}” cung cấp tên của phương thức. Trong Web API, quy ước thông thường là bỏ qua phân đoạn “{action}”. Đó là lý do tại sao khi bạn tạo ứng dụng WEB API mới và sau đó, bạn có thể thấy rằng mẫu tuyến đường mặc định được tạo bởi khuôn khổ không chứa phân đoạn tên trong URI.

Mặc định

Nếu giá trị mặc định được định nghĩa cho thì tuyến đường sẽ khớp với URI bị thiếu các phân đoạn đó. Ví dụ:

```
config.Routes.MapHttpRoute(  
    name: "ActionApi",  
    routeTemplate: "api/{controller}/public/{category}",  
    defaults: new { category = "all" }  
)
```

URI “http://localhost/api/student/public” khớp với tuyến này. Phân đoạn “{category}” được gán giá trị mặc định là “all”.

5.3.2 Từ điển tuyến đường Route

Khi WEB API framework tìm thấy kết quả phù hợp với URI, thì khung này sẽ tạo một từ điển chứa giá trị cho từng trình giữ chỗ. Như chúng ta biết, từ điển chứa dữ liệu dưới dạng một cặp khóa-giá trị. Ở đây, các khóa không là gì ngoài tên trình giữ chỗ nhưng loại trừ dấu ngoặc nhọn và các giá trị được lấy từ đường dẫn URI hoặc từ giá trị mặc định. Từ điển được lưu trữ trong đối tượng IHttpRouteData như hình dưới đây.

```
namespace System.Web.Http.Routing  
{  
    public interface IHttpRouteData  
    {  
        IHttpRoute Route { get; }  
        IDictionary<string, object> Values { get; }  
    }  
}
```

Trong giai đoạn đối sánh tuyến đường, các trình giữ chỗ đặc biệt như “{controller}” và “{action}” được xử lý giống như bất kỳ trình giữ chỗ nào khác. Chúng chỉ đơn giản là được lưu trữ trong từ điển với các giá trị khác.

Mặc định có thể có một giá trị đặc biệt RouteParameter.Optional. Nếu một trình giữ chỗ được chỉ định với giá trị này, thì giá trị đó sẽ không được thêm vào từ điển tuyến đường. Ví dụ:

```

config.Routes.MapHttpRoute(
    name: "ActionApi",
    routeTemplate: "api/{controller}/public/{category}/{id}",
    defaults: new {
        category = "all",
        id = RouteParameter.Optional
    }
);

```

Đối với đường dẫn URI “api/student/public”, từ điển tuyến đường sẽ chứa hai phần tử như:

1. controller: “student”
2. category: “all”

Đối với đường dẫn URI “api/student/public/cse/101”, từ điển lô trình sẽ chứa ba phần tử như:

1. controller: “student”
2. category: “all”
3. id: “101”

Giá trị mặc định cũng có thể bao gồm một giá trị không xuất hiện ở bất kỳ đâu trong mẫu tuyến đường. Nếu tuyến đường khớp, giá trị đó cũng được lưu trữ trong từ điển. Ví dụ:

```

config.Routes.MapHttpRoute(
    name: "Root",
    routeTemplate: "api/root/{id}",
    defaults: new {
        controller = "Employee",
        id = RouteParameter.Optional
    }
);

```

Nếu đường dẫn URI là "api/root/101", thì từ điển sẽ chứa hai phần tử như:

1. controller: “Employee”
2. id: “101”

5.3.3 Chọn bộ điều khiển

Lựa chọn Bộ điều khiển trong WEB API được xử lý bởi phương thức IHttpControllerSelector.SelectController.

```

namespace System.Web.Http.Dispatcher
{
    public interface IHttpControllerSelector
    {
        IDictionary<string, HttpControllerDescriptor> GetControllerMapping();
        HttpControllerDescriptor SelectController(HttpRequestMessage request);
    }
}

```

Như thể hiện trong hình trên, phương thức SelectController lấy một cá thể HttpRequestMessage làm tham số và trả về một HttpControllerDescriptor. Việc triển khai mặc định cho phương thức SelectController ở trên được cung cấp bởi lớp DefaultHttpControllerSelector như thể hiện trong hình bên dưới.

```

namespace System.Web.Http.Dispatcher
{
    public class DefaultHttpControllerSelector : IHttpControllerSelector
    {
        public static readonly string ControllerSuffix;

        public DefaultHttpControllerSelector(HttpConfiguration configuration);

        public virtual IDictionary<string, HttpControllerDescriptor> GetControllerMapping();
        public virtual string GetControllerName(HttpRequestMessage request);
        public virtual HttpControllerDescriptor SelectController(HttpRequestMessage request);
    }
}

```

Lớp trên sử dụng một thuật toán đơn giản để tìm bộ điều khiển như:

1. Đầu tiên, nó sẽ xem xét bộ sưu tập từ điển tuyến đường cho khóa “controller”.
2. Thứ hai, nó lấy giá trị cho phím “controller” và nối chuỗi “Controller” để lấy tên loại bộ điều khiển.
3. Cuối cùng, nó tìm kiếm bộ điều khiển Web API với loại tên này.

Ví dụ: nếu từ điển tuyến đường chứa cặp khóa-giá trị “controller” = “Student”, thì loại bộ điều khiển là “StudentController”; “controller” = “Students”, thì loại bộ điều khiển là “StudentsController”; “controller” = “Employee”, thì loại bộ điều khiển là “EmployeeController”.

Nếu không tìm thấy loại đối sánh hoặc tìm thấy nhiều đối sánh, thì ASP.NET WEB API framework chỉ trả về một lỗi cho máy khách (đưa ra yêu cầu).

5.3.4 Lựa chọn phương thức

Sau khi chọn bộ điều khiển, tiếp theo, WEB API framework chọn phương thức bằng cách gọi phương thức IHttpActionSelector.SelectAction. Phương thức này lấy HttpControllerContext làm tham số và trả về một HttpActionDescriptor như thể hiện trong hình dưới đây.

```

namespace System.Web.Http.Controllers
{
    public interface IHttpActionSelector
    {
        ILookup<string, HttpActionDescriptor> GetActionMapping(HttpControllerDescriptor controllerDescriptor);
        HttpActionDescriptor SelectAction(HttpContext controllerContext);
    }
}

```

Việc triển khai mặc định cho SelectAction được cung cấp bởi lớp ApiControllerActionSelector như thể hiện trong hình bên dưới.

```
namespace System.Web.Http.Controllers
{
    public class ApiControllerActionSelector : IHttpActionSelector
    {
        public ApiControllerActionSelector()
        {
        }

        public virtual ILookup<string, HttpActionDescriptor> GetActionMapping(HttpControllerDescriptor controllerDescriptor);
        public virtual HttpActionDescriptor SelectAction(HttpContext controllerContext);
    }
}
```

Để chọn một phương thức, nó sẽ xem xét thuật toán sau:

1. Phương thức HTTP của yêu cầu.
2. “{action}” trong mẫu định tuyến (nếu có)
3. Các tham số của các phương thức trên bộ điều khiển.

Trước khi xem xét thuật toán lựa chọn, trước tiên, chúng ta cần hiểu phương thức nào trên lớp controller được coi là phương thức “phương thức”?

Khi chọn một phương thức, WEB API Framework chỉ xem xét các phương thức công khai của bộ điều khiển, ngoại trừ hàm tạo, sự kiện, quá tải toán tử, v.v. và các phương thức được kế thừa từ lớp ApiController.

Phương thức HTTP:

WEB API framework chỉ chọn các phương thức thực thi phù hợp với phương thức HTTP của yêu cầu đến, được xác định như sau:

1. Các phương thức được khai báo bằng thuộc tính HTTP như AcceptVerbs, HttpDelete, HttpGet, HttpHead, HttpOptions, HttpPatch, HttpPost hoặc HttpPut.
2. Nếu tên phương thức của bộ điều khiển bắt đầu bằng “Get”, “Post”, “Put”, “Delete”, “Head”, “Options” hoặc “Patch”, thì theo quy ước phương thức hỗ trợ phương thức HTTP đó.

Tham số ràng buộc:

Liên kết tham số là cách Web API tạo giá trị cho một tham số. Đây là quy tắc mặc định cho liên kết tham số:

1. Các kiểu đơn giản được lấy từ URI.
2. Các kiểu phức tạp được lấy từ phần thân yêu cầu (body request).
3. Có thể thay đổi ràng buộc tham số mặc định trong WEB API.

Với nền đó, hãy xem thuật toán lựa chọn phương thức.

1. Tạo danh sách tất cả các phương thức trên bộ điều khiển phù hợp với phương thức Yêu cầu HTTP request.
2. Nếu từ điển tuyến đường có mục nhập “phương thức”, hãy xóa các phương thức có tên không khớp với giá trị này.
3. Cố gắng khớp các thông số phương thức với URI, như sau:
4. Đối với mỗi phương thức, lấy danh sách các tham số là kiểu đơn giản, trong đó ràng buộc lấy tham số từ URI. Loại trừ các thông số tùy chọn.

5. Từ danh sách này, hãy cố gắng tìm một kết quả phù hợp cho từng tên tham số, trong từ điển tuyến đường hoặc trong chuỗi truy vấn URI. Các so khớp không phân biệt chữ hoa chữ thường và không phụ thuộc vào thứ tự tham số.

6. Chọn một phương thức trong đó mọi tham số trong danh sách đều khớp trong URI.

7. Nếu nhiều phương thức đáp ứng các tiêu chí này, hãy chọn phương thức có nhiều thông số phù hợp nhất.

8. Bỏ qua các phương thức với thuộc tính [NonAction].

Bước 3 có lẽ là bước khó hiểu nhất. Ý tưởng cơ bản là một tham số có thể nhận giá trị của nó từ URI hoặc từ phần thân yêu cầu hoặc từ một liên kết tùy chỉnh. Đối với các tham số đến từ URI, ta muốn đảm bảo rằng URI thực sự chứa một giá trị cho tham số đó, trong đường dẫn (qua từ điển tuyến đường) hoặc trong chuỗi truy vấn.

Ví dụ:

Chúng ta hãy xem xét các điểm trên với một ví dụ.

Các tuyến:

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "RootApi",
            routeTemplate: "api/root/{id}",
            defaults: new { controller = "Student", id = RouteParameter.Optional }
        );

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

Bộ điều khiển:

```
public class StudentController : ApiController
{
    public IEnumerable<Student> GetAllStudents()...
    public Student GetStudentById(int id, double version = 1.0)...
    [HttpGet]
    public void FindStudentsByName(string name) { }
    public void Post(Student value) { }
    public void Put(int id, Student value) { }
}
```

Yêu cầu HTTP request:

GET http://localhost:50470/api/student/1?version=2.1&details=1

Đối sánh tuyến đường

URI ở trên khớp với tuyến đường có tên “DefaultApi”. Từ điển tuyến đường chứa các phần tử sau:

1. controller: “Student”
2. id: “1”

Từ điển tuyến đường không chứa các tham số chuỗi truy vấn, “version” và “details”, nhưng chúng vẫn sẽ được xem xét trong quá trình lựa chọn phương thức.

Lựa chọn bộ điều khiển

Từ mục nhập bộ điều khiển “controller” trong từ điển tuyến đường, WEB API framework chọn loại bộ điều khiển là StudentController

Lựa chọn phương thức

Yêu cầu HTTP request ở trên là một yêu cầu GET. Các phương thức của bộ điều khiển hỗ trợ GET Request là GetAllStudents, GetStudentById và FindStudentsByName. Từ điển tuyến đường không chứa mục nhập cho "phương thức", vì vậy không cần phải khớp với tên phương thức.

Tiếp theo, chúng ta cần đổi sánh tên tham số cho các phương thức, chỉ xem xét các phương thức GET.

Action	Parameters to Match
GetAllStudents	none
GetStudentById	"id"
FindStudentsByName	"name"

Lưu ý rằng tham số phiên bản của GetStudentById không được xem xét vì nó là tham số tùy chọn.

Phương thức GetAllStudents phù hợp rất ít. Phương thức GetStudentById cũng phù hợp, vì từ điển tuyến đường có chứa “id”. Phương thức FindStudentsByName không khớp.

Phương thức GetStudentById thắng vì nó khớp với một tham số, so với không có tham số nào cho GetAllStudents. Phương thức được gọi với các giá trị tham số sau:

1. id = 1
2. version = 2.1

Lưu ý rằng mặc dù phiên bản không được sử dụng trong thuật toán lựa chọn, giá trị của tham số đến từ chuỗi truy vấn URI.

5.4 Định tuyến thuộc tính Web API

5.4.1 Giới thiệu

ASP.NET Web API 2 và ASP.NET MVC 5 hỗ trợ một kiểu định tuyến mới được gọi là **định tuyến thuộc tính**. Như tên của nó, định tuyến thuộc tính có nghĩa là các thuộc tính được sử dụng để

xác định các tuyến đường. Định tuyến thuộc tính cung cấp nhiều quyền kiểm soát hơn đối với các URI trong ứng dụng Web API của bạn bằng cách xác định các tuyến trực tiếp trên các phương thức và bộ điều khiển. Ví dụ: bạn có thể dễ dàng tạo URI mô tả phân cấp tài nguyên.

Kiểu định tuyến trước đó được gọi là định tuyến dựa trên quy ước vẫn được hỗ trợ đầy đủ bởi Web API. Trên thực tế, bạn có thể kết hợp cả hai cách tiếp cận trong cùng một dự án.

Trong phần này, chúng ta sẽ thảo luận về cách bật định tuyến thuộc tính trong Web API và mô tả các tùy chọn khác nhau để định tuyến thuộc tính.

5.4.2 Tại sao chúng ta cần Định tuyến thuộc tính Web API?

Bản phát hành đầu tiên của Web API sử dụng định tuyến dựa trên quy ước. Theo quy ước, chúng ta có thể xác định một hoặc nhiều mẫu tuyến đường trong tệp WebApiConfig, về cơ bản là các chuỗi được tham số hóa. Khi WEB API framework nhận được một Yêu cầu HTTP request, nó sẽ khớp với URI với mẫu tuyến có sẵn trong Bảng tuyến. Để biết thêm thông tin về định tuyến dựa trên quy ước, Vui lòng đọc các phần sau, nơi ta đã thảo luận về Định tuyến dựa trên quy ước trong Web API với các ví dụ.

Một ưu điểm của định tuyến dựa trên quy ước là tất cả các mẫu URI được xác định ở một nơi duy nhất và các quy tắc định tuyến được áp dụng nhất quán trên tất cả các bộ điều khiển.

Tuy nhiên, định tuyến dựa trên quy ước trong Web API khiến việc hỗ trợ một số mẫu URI thường gặp trong các API RESTful trở nên khó khăn. Ví dụ: các tài nguyên thường chứa các tài nguyên con: Khách hàng có đơn đặt hàng, phim có diễn viên, sách có tác giả, v.v. Việc tạo các URI phản ánh các mối quan hệ sau là điều tự nhiên:

Hãy hiểu điều này bằng một ví dụ.

Bước 1: Tạo một ứng dụng Web API mới. Đặt tên là AttributeRoutingInWEBAPI

Bước 2: Nhấp chuột phải vào thư mục “Mô hình” và thêm tệp lớp với tên Student.cs, sau đó sao chép và dán đoạn mã sau.

```
namespace AttributeRoutingInWEBAPI.Models
{
    public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

Bước 3: Bây giờ, thêm Bộ điều khiển học sinh Students Controller

Để làm như vậy Nhấp chuột phải vào thư mục Controllers và thêm bộ điều khiển Web API2 mới - Trống. Đặt tên cho nó là StudentsController.cs. Sao chép và dán đoạn mã sau.

```
namespace AttributeRoutingInWEBAPI.Controllers
{
}
```

```

public class StudentsController : ApiController
{
    static List<Student> students = new List<Student>()
    {
        new Student() { Id = 1, Name = "Pranaya" },
        new Student() { Id = 2, Name = "Priyanka" },
        new Student() { Id = 3, Name = "Anurag" },
        new Student() { Id = 4, Name = "Sambit" }
    };

    public IEnumerable<Student> Get()
    {
        return students;
    }

    public Student Get(int id)
    {
        return students.FirstOrDefault(s => s.Id == id);
    }

    public IEnumerable<string> GetStudentCourses(int id)
    {
        List<string> CourseList = new List<string>();
        if (id == 1)
            CourseList = new List<string>() { "ASP.NET", "C#.NET", "SQL Server" };
        else if (id == 2)
            CourseList = new List<string>() { "ASP.NET MVC", "C#.NET", "ADO.NET" };
        else if (id == 3)
            CourseList = new List<string>() { "ASP.NET WEB API", "C#.NET", "Entity Framework" };
        else
            CourseList = new List<string>() { "Bootstrap", "jQuery", "AngularJs" };
        return CourseList;
    }
}

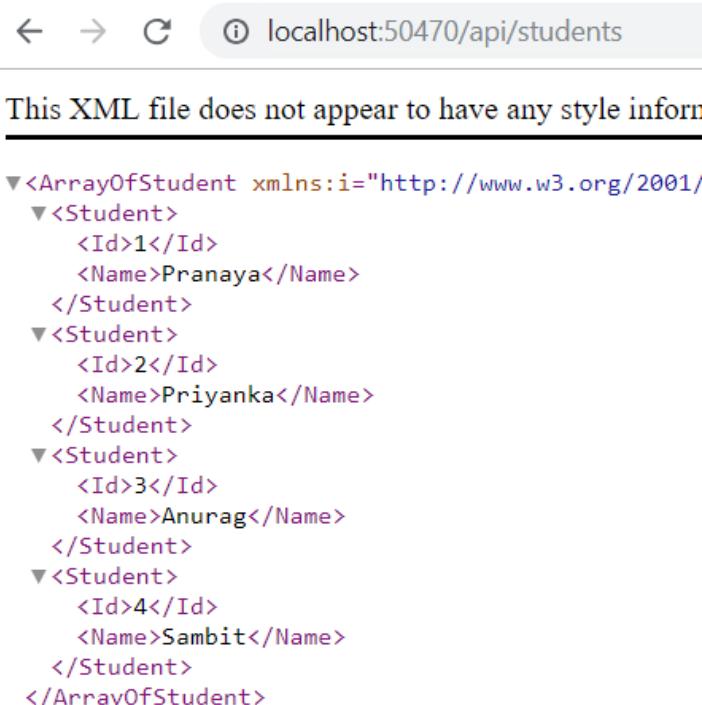
```

Trong Web API1, ta có định tuyến dựa trên quy ước xác định các tuyến bằng cách sử dụng các mẫu định tuyến. Khi ta tạo một dự án Web API mới, WEB API framework sẽ tạo một tuyến mặc định trong tệp **WebApiConfig.cs**. Tuyến đường mặc định được hiển thị bên dưới

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

Vì vậy, với tuyến đường mặc định ở trên và **StudentsController** trong địa chỉ **/api/students** được ánh xạ tới **Get()** phương thức của StudentsController như mong đợi như trong hình dưới đây.



Nhưng khi ta điều hướng đến `/api/student/1`, ta nhận được thông báo ngoại lệ sau Đã tìm thấy nhiều phương thức phù hợp với yêu cầu: Nhận trên loại AttributeRoutingInWEBAPI.Controllers.StudentsController GetStudentCourses trên loại AttributeRoutingInWEBAPI.Controllers.StudentsController. Điều này là do WEB API framework không biết phương pháp phương thức nào trong số 2 phương pháp phương thức sau đây để ánh xạ tới URI `/api/student/1`

Nhận (int id) GetStudentCourses (int id)

Điều này có thể rất dễ dàng đạt được bằng cách sử dụng Định tuyến thuộc tính. Đây là những gì ta muốn WEB API framework thực hiện.

URI/api/Students/1 nên được ánh xạ tới Get (int id). Phương thức này sẽ trả về sinh viên theo id.

URI /api/student/1/các khóa học phải được ánh xạ tới GetStudentCourses (int id). Phương thức này sẽ trả về các khóa học sinh viên theo id sinh viên.

Để đạt được những điều trên, chúng ta cần trang trí phương thức thực thi GetStudentCourses () với thuộc tính [Route] như trong hình dưới đây

```
[Route("api/students/{id}/courses")]
public IEnumerable<string> GetStudentCourses(int id)
{
    List<string> CourseList = new List<string>();

    if (id == 1)
        CourseList = new List<string>() { "ASP.NET", "C#.NET", "SQL Server" };
    else if (id == 2)
        CourseList = new List<string>() { "ASP.NET MVC", "C#.NET", "ADO.NET" };
    else if (id == 3)
        CourseList = new List<string>() { "ASP.NET WEB API", "C#.NET", "Entity Framework" };
    else
        CourseList = new List<string>() { "Bootstrap", "jQuery", "AngularJs" };

    return CourseList;
}
```

Tại thời điểm này, khi xây dựng giải pháp và điều hướng đến/api/Students/1. Lưu ý rằng, bây giờ ta sẽ nhận được chi tiết sinh viên có id = 1 và khi bạn điều hướng đến /api/student/1/course, bạn sẽ nhận được tất cả các khóa học mà sinh viên có id = 1 đã đăng ký.

Hãy xem một số ví dụ trong đó định tuyến thuộc tính giúp dễ dàng.

Phiên bản API

Trong ví dụ dưới đây, tuyến đường “/api/v1/student” sẽ được chuyển đến một bộ điều khiển khác với tuyến đường “/api/v2/student”.

/api/v1/student

/api/v2/student

Phân đoạn URI bị quá tải

Trong ví dụ này, “1” là số thứ tự, nhưng đang chờ xử lý “pending” ánh xạ tới một bộ sưu tập collection.

/order/1

/order/pending

Nhiều loại tham số

Trong ví dụ này, “1” là số đơn đặt hàng, nhưng “2013/06/16” chỉ định một ngày.

/order/1

/order/2013/06/16

Làm cách nào để bật Định tuyến thuộc tính Web API?

Trong Web API 2, Định tuyến thuộc tính được bật theo mặc định. Config.MapHttpAttributeRoutes(); mã có trong tệp WebApiConfig.cs cho phép Định tuyến thuộc tính Web API.

Ta cũng có thể kết hợp Định tuyến thuộc tính Web API với định tuyến dựa trên quy ước. Để xác định các tuyến đường dựa trên quy ước, hãy gọi phương thức MapHttpRoute như hình dưới đây.

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Attribute routing.
        config.MapHttpAttributeRoutes();

        // Convention-based routing.
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

Ta có thể sử dụng cả Định tuyến thuộc tính và Định tuyến dựa trên quy ước trong một dự án Web API không? Câu trả lời là có, ta có thể kết hợp cả hai cơ chế định tuyến trong một dự án ASP.NET Web API. Các phương thức thực thi của bộ điều khiển có thuộc tính [Lộ trình] sử dụng Định tuyến thuộc tính và các phương thức khác không có thuộc tính [Lộ trình] sử dụng định tuyến dựa trên quy ước.

Lưu ý: Bạn cần định cấu hình định tuyến Thuộc tính trước khi định tuyến dựa trên quy ước trong ASP.NET Web API.

Lợi ích của việc sử dụng Định tuyến thuộc tính Web API là gì?

1. Nó cho phép chúng ta kiểm soát nhiều hơn các URI so với định tuyến dựa trên quy ước. Việc tạo các mẫu URI như phân cấp tài nguyên (Ví dụ: sinh viên có khóa học, Phòng ban có nhân viên) là rất khó với định tuyến dựa trên quy ước.
2. Giảm khả năng xảy ra lỗi, nếu một tuyến đường được sửa đổi không chính xác trong RouteConfig.cs thì nó có thể ảnh hưởng đến quá trình định tuyến của toàn bộ ứng dụng.
3. Có thể tách hoàn toàn tên bộ điều khiển và tên phương thức khỏi tuyến đường.
4. Dễ dàng lập bản đồ hai tuyến đường chỉ đến một phương thức.

CHƯƠNG 6 PHÁT TRIỂN ỨNG DỤNG SỬ DỤNG WEB API

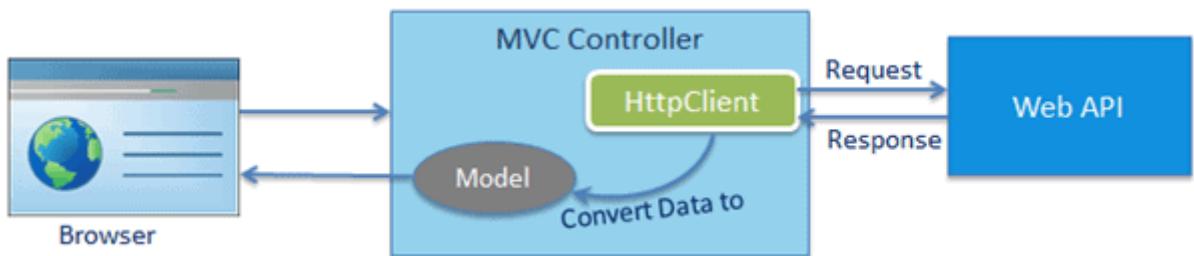
6.1 Giới thiệu

Trong các phần trước, chúng ta đã tạo Web API với các phương thức Get, Post, Put và Delete để xử lý các Yêu cầu HTTP request GET, POST, PUT và DELETE tương ứng. Ở đây, chúng ta sẽ xem cách sử dụng (truy cập) Web API cho hoạt động CRUD.

Web API có thể được truy cập bằng mã phía máy chủ trong .NET và cả phía máy khách bằng cách sử dụng các JavaScript FrameWork như jQuery, AngularJS, v.v.

Ở đây, ta sẽ sử dụng Web API của ta (đã tạo trong phần trước) trong ASP.NET MVC. Để sử dụng Web API ở phía máy chủ ASP.NET MVC, chúng ta có thể sử dụng HttpClient trong bộ điều khiển MVC. HttpClient gửi yêu cầu đến Web API và nhận được phản hồi. Sau đó, ta cần chuyển đổi dữ liệu phản hồi đến từ Web API thành một mô hình và sau đó hiển thị nó thành một chế độ xem.

Hình sau minh họa việc sử dụng Web API trong ASP.NET MVC.



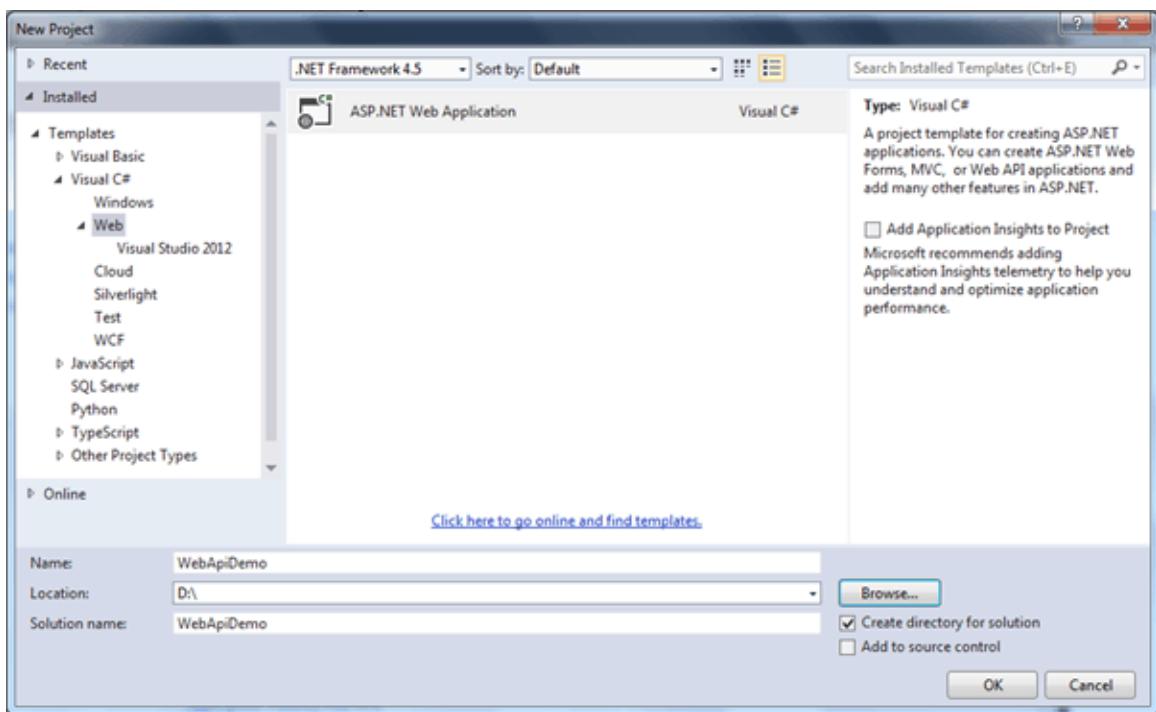
Lưu ý: AngularJS hoặc bất kỳ JavaScript FrameWork nào khác có thể được sử dụng trong chế độ xem MVC và có thể truy cập trực tiếp Web API từ chế độ xem bằng AJAX. Ta đã sử dụng ASP.NET MVC chỉ để trình bày cách truy cập Web API từ mã phía máy chủ trong trường hợp bạn không sử dụng bất kỳ JavaScript FrameWork nào.

6.2 Triển khai một Trang web với phương thức GET

Ở đây, ta sẽ tạo một dự án Web API mới và triển khai phương thức GET, POST, PUT và DELETE cho hoạt động CRUD bằng Entity Framework.

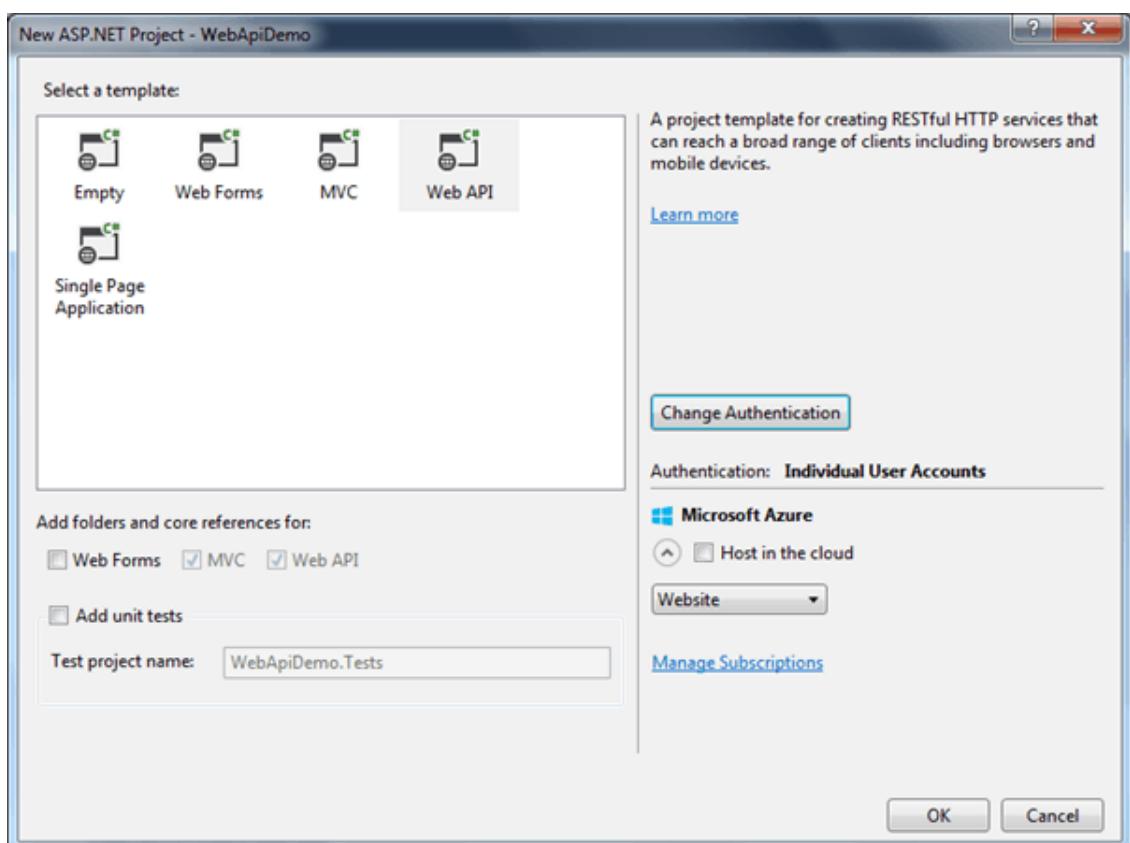
Đầu tiên, tạo một dự án Web API mới trong Visual Studio 2013 cho phiên bản Web express.

Mở Visual Studio 2013/2017 cho Web và nhấp vào **File** menu -> **New Project**. Thao tác này sẽ mở cửa sổ bật lên Dự án mới như hình dưới đây.



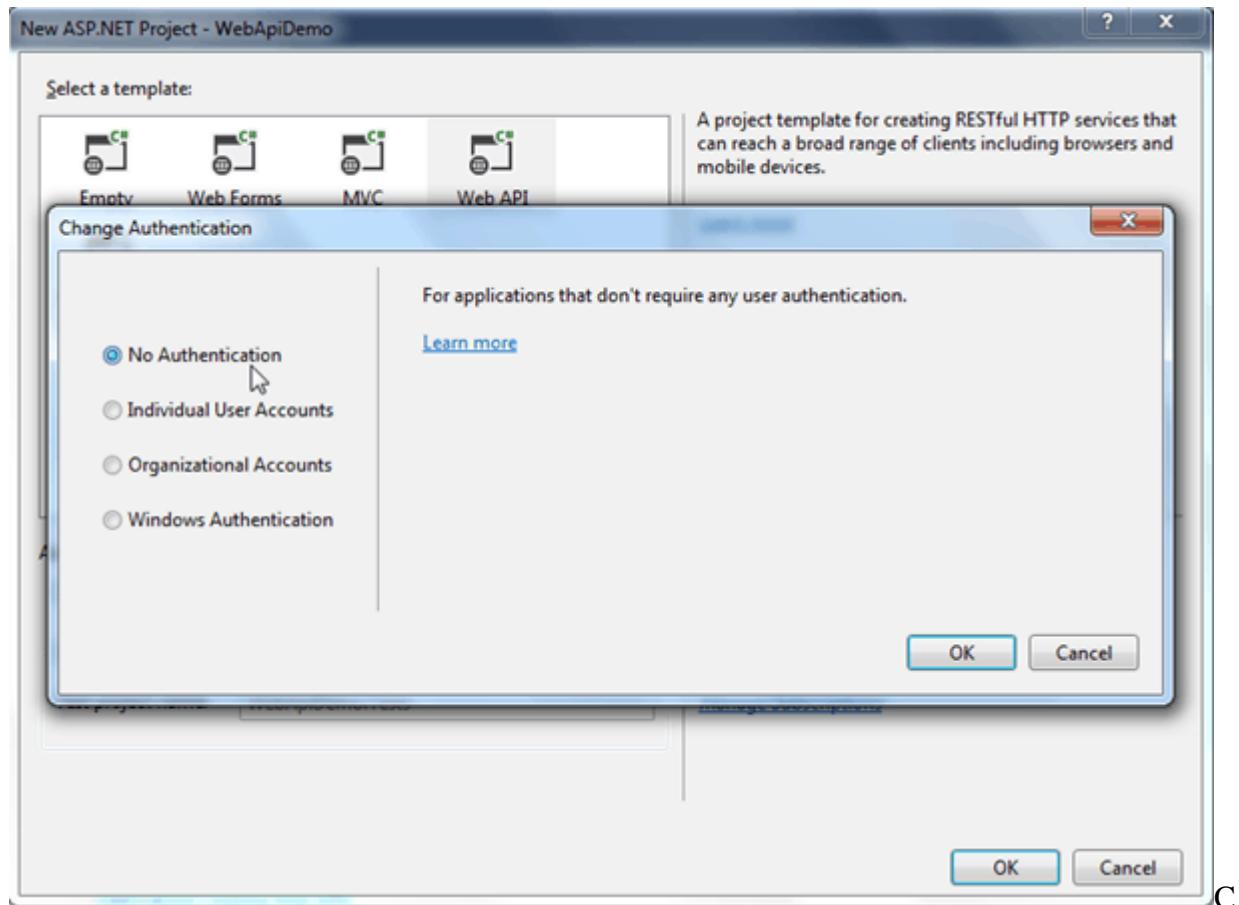
Hình 4. Tạo dự án Web API

Trong cửa sổ bật lên Dự án mới, chọn Mẫu web trong Trực quan C#. Nhập tên dự án **WebApiDemo** và vị trí bạn muốn tạo dự án. Nhấn **OK** để tiếp tục. Thao tác này sẽ mở một cửa sổ bật lên khác để chọn một mẫu dự án. Chọn dự án **Web API** như hình dưới đây.



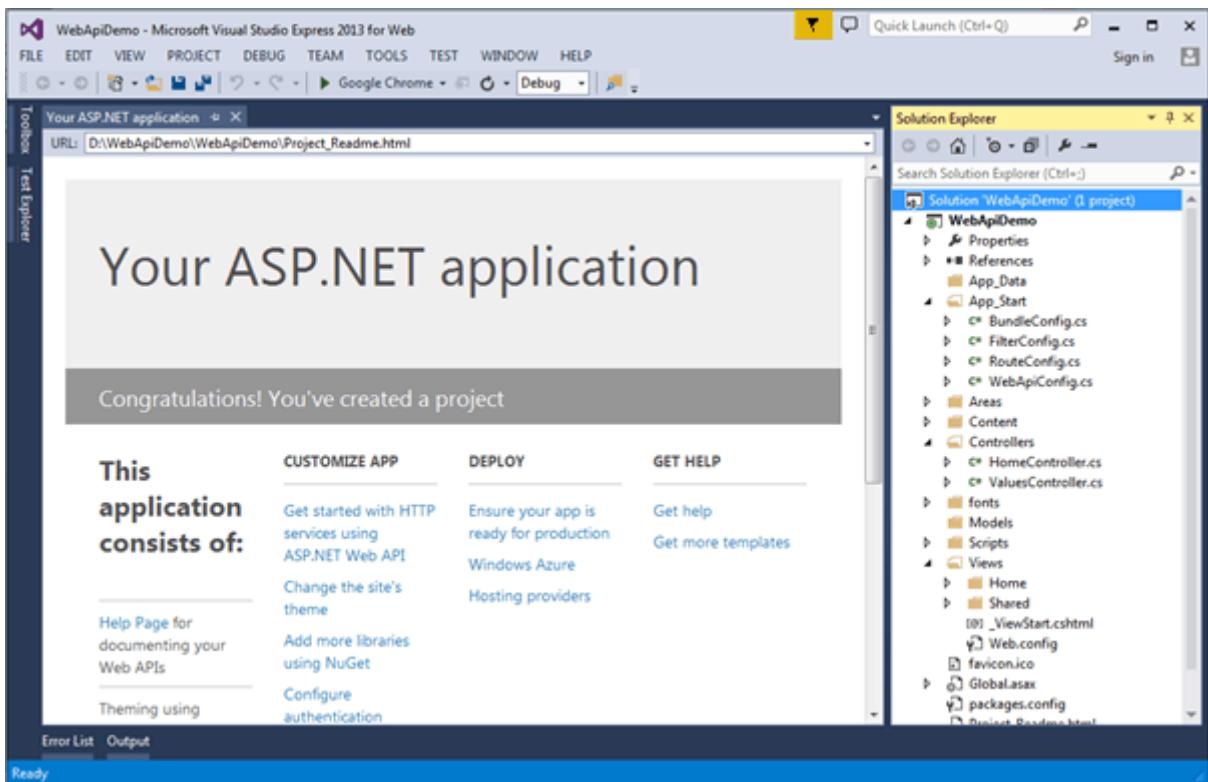
Hình 5. Chọn loại dự án Web API Project Template

Ở đây, ta sẽ không sử dụng bất kỳ xác thực nào trong dự án demo của chúng tôi. Vì vậy, hãy nhấp vào **Change Authentication** để mở cửa sổ bật lên Xác thực và chọn **No Authentication** nút radio và sau đó bấm OK như hình dưới đây.



Hình 6. Change Authentication

Bây giờ, nhấp vào **OK** trong cửa sổ bật lên Dự án **ASP.NET** Mới để tạo một dự án như hình dưới đây.



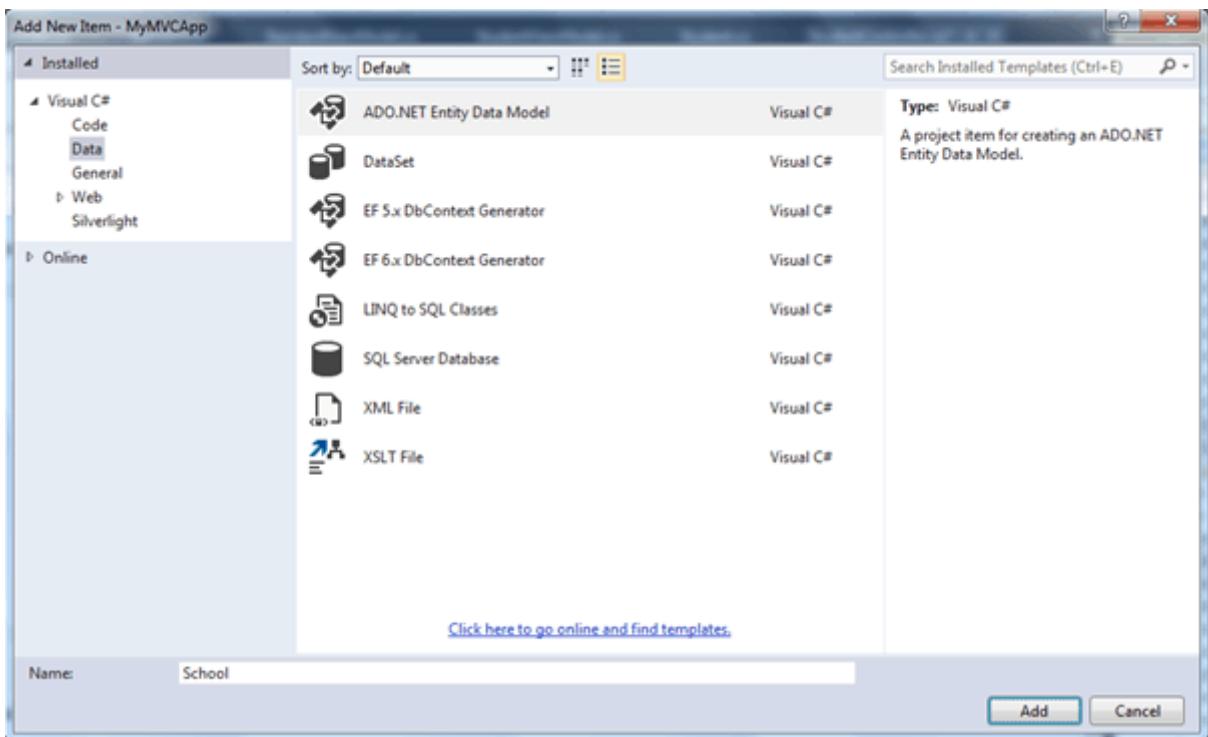
Hình 7. Dự án Web API

Như ta có thể thấy, một dự án WebApiDemo mới được tạo với tất cả các tệp cần thiết. Nó cũng đã thêm ValuesController mặc định. Vì ta sẽ thêm bộ điều khiển Web API mới của mình, ta có thể xóa các ValuesController mặc định.

Ở đây, chúng ta sẽ sử dụng phương pháp tiếp cận Entity Framework DB-First để truy cập cơ sở dữ liệu trường học hiện có. Vì vậy, hãy thêm mô hình dữ liệu EF cho cơ sở dữ liệu trường học bằng cách sử dụng phương pháp DB First.

Thêm Entity Framework Data Model

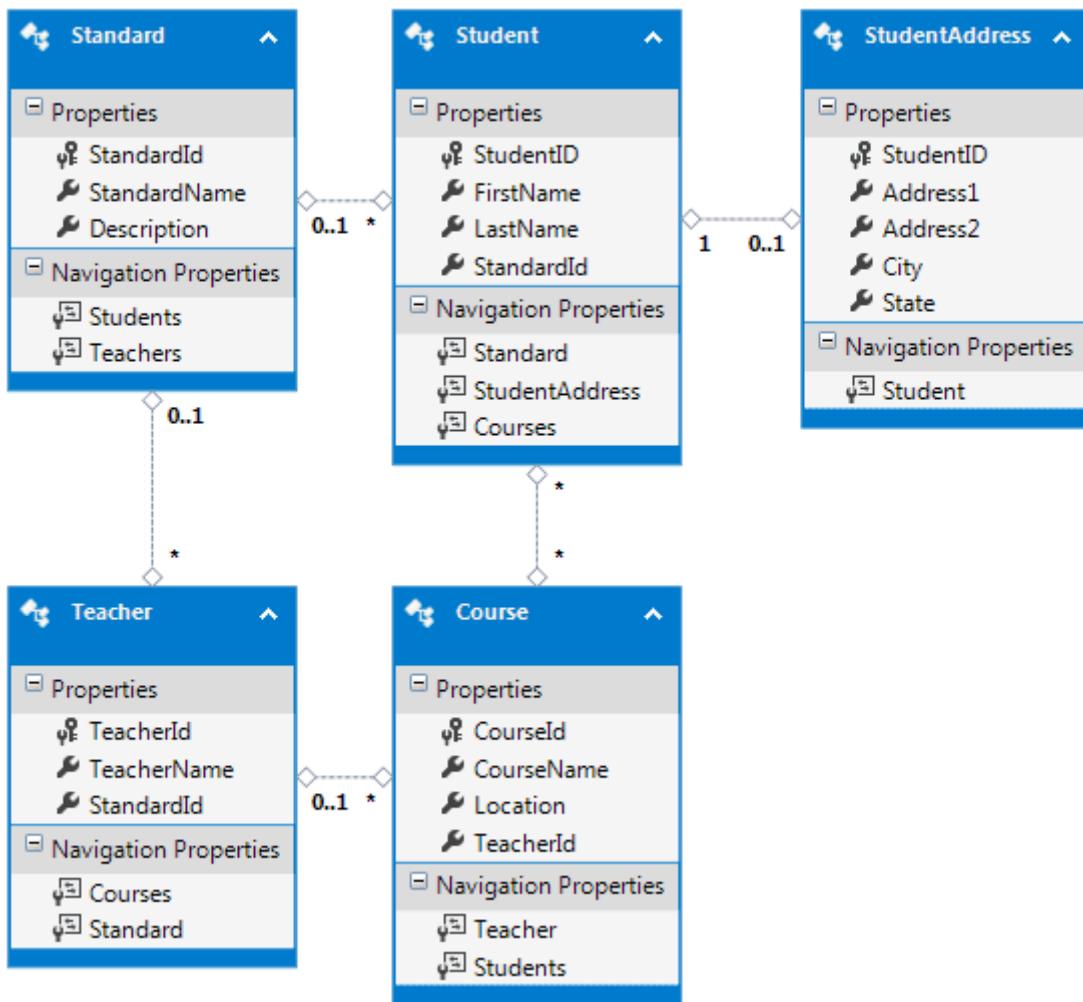
Để thêm mô hình dữ liệu EF bằng cách sử dụng phương pháp DB-First, nhấp chuột phải vào dự án của bạn -> nhấp **New Item..**. Thao tác này sẽ mở cửa sổ bật lên Thêm mục mới như hình dưới đây.



Hình 8. Tạo mô hình dữ liệu thực thể Entity Data Model

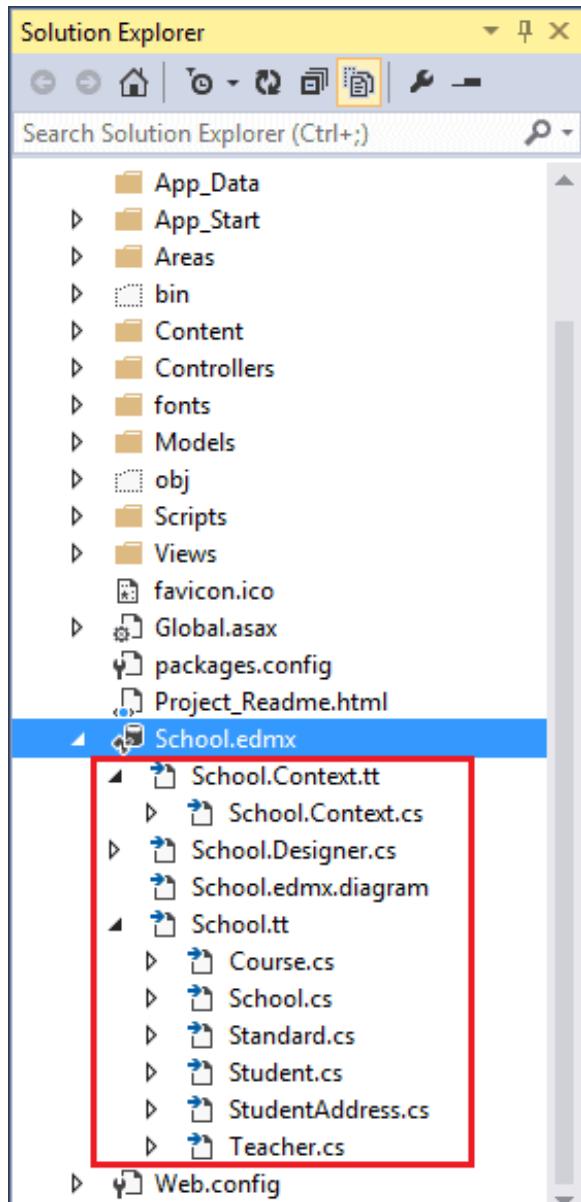
Lựa chọn **Data** trong ngăn bên trái và chọn **ADO.NET Entity Data Model** trong ngăn giữa và nhập tên của mô hình dữ liệu và nhấp vào **Thêm**. Thao tác này sẽ mở Trình hướng dẫn Mô hình Dữ liệu Thực thể bằng cách sử dụng mà bạn có thể tạo Mô hình Dữ liệu Thực thể cho cơ sở dữ liệu Trường học hiện có. Phạm vi của chủ đề chỉ giới hạn trong Web API nên ta chưa trình bày về cách tạo EDM.

Entity Framework sẽ tạo mô hình dữ liệu sau sau khi hoàn thành tất cả các bước của Entity Data Model Wizard.



Hình 9. Các thực thể đã tạo trong Trình thiết kế EDM

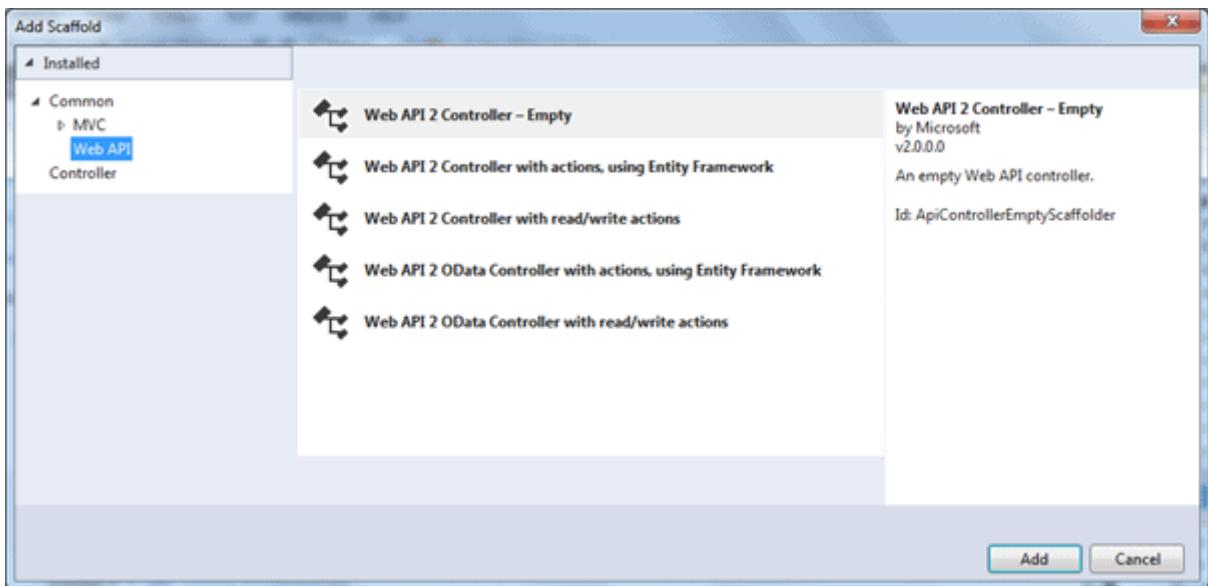
Entity Framework cũng tạo ra các thực thể và các lớp ngẫu cảnh như hình dưới đây.



Hình 10. .edmx in the Project

Từ bây giờ, ta sẽ triển khai hoạt động CRUD bằng Entity Framework trong dự án Web API của chúng tôi. Bây giờ, hãy thêm bộ điều khiển Web API vào dự án của chúng tôi. **Add Web API Controller**

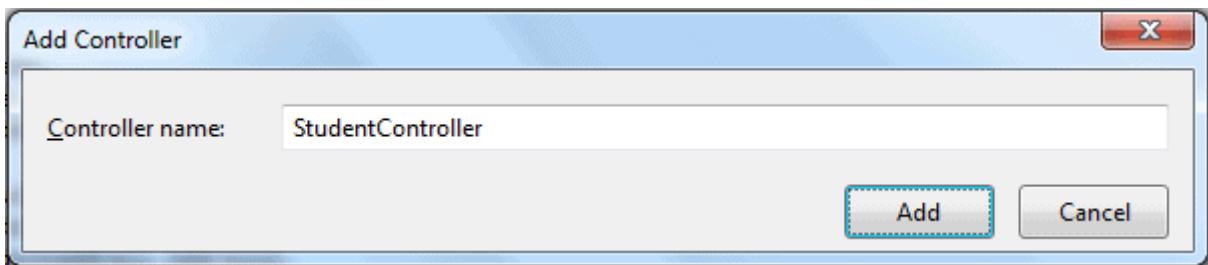
Để thêm bộ điều khiển Web API vào dự án MVC của bạn, hãy nhấp chuột phải vào **Controllers** thư mục hoặc một thư mục khác mà bạn muốn thêm Web API controller -> chọn **Add** -> chọn **Controller**. Điều này sẽ mở ra **Add Scaffold** bật lên như hình bên dưới.



Hình 11. Tạo bộ điều khiển Web API Controller

Trong cửa sổ bật lên Add Scaffold, hãy chọn **Web API** trong ngăn bên trái và chọn **Web API 2 Controller - Empty** trong ngăn giữa và nhấp vào **Add**. (Ta chọn Mẫu trống vì ta dự định thêm các phương pháp phương thức và Khung thực thẻ của chính mình.)

Điều này sẽ mở ra **Add Controller** Cửa sổ bật lên bộ điều khiển nơi bạn cần nhập tên bộ điều khiển của mình. Nhập "StudentController" làm tên bộ điều khiển và nhấp vào **Add** như hình bên dưới.



Hình 12. Tạo bộ điều khiển Web API

Điều này sẽ thêm trống **StudentController** lớp bắt nguồn từ **ApiController** như hình bên dưới.

Web API Controller
<pre>using System; using System.Collections.Generic; using System.Linq; using System.Net; using System.Net.Http; using System.Web.Http; namespace MyMVCApp.Controllers { </pre>

```

public class StudentController : ApiController
{
    public StudentController()
    {
    }

    public IHttpActionResult GetAllStudents(bool includeAddress = false)
    {
        IList<StudentViewModel> students = null;

        using (var ctx = new SchoolDBEntities())
        {
            students = ctx.Students.Include("StudentAddress").Select(s => new StudentViewModel()
            {
                Id = s.StudentID,
                FirstName = s.FirstName,
                LastName = s.LastName,
                Address = s.StudentAddress == null || includeAddress == false ? null : new
AddressViewModel()
                {
                    StudentId = s.StudentAddress.StudentID,
                    Address1 = s.StudentAddress.Address1,
                    Address2 = s.StudentAddress.Address2,
                    City = s.StudentAddress.City,
                    State = s.StudentAddress.State
                }
            }).ToList<StudentViewModel>();
        }

        if (students.Count == 0)
        {
            return NotFound();
        }
    }
}

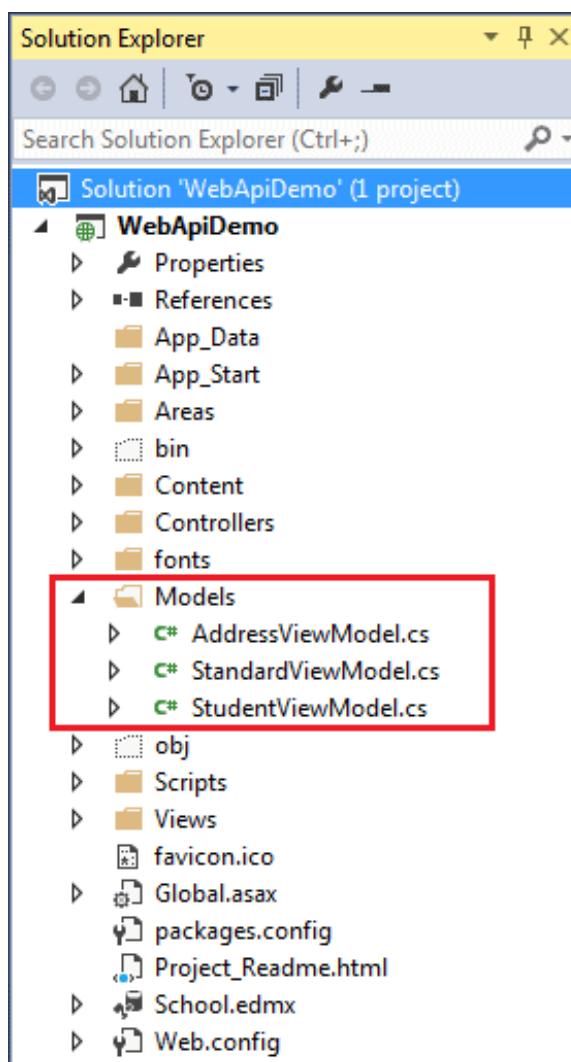
```

```
        return Ok(students);  
    }  
}  
}
```

Thêm mô hình

Ta sẽ truy cập cơ sở dữ liệu cơ bản bằng Entity Framework (EF). Như bạn đã thấy ở trên, EF tạo các lớp thực thể của riêng mình. Tốt nhất, chúng ta không nên trả về các đối tượng thực thể EF từ Web API. Bạn nên trả lại DTO (Đối tượng truyền dữ liệu) từ Web API. Vì ta đã tạo dự án Web API với MVC, ta cũng có thể sử dụng các lớp mô hình MVC sẽ được sử dụng trong cả MVC và Web API.

Tại đây, ta sẽ trả về Student, Address và Tiêu chuẩn từ Web API. Vì vậy, ta tạo **StudentViewModel**, **AddressViewModel** và **StandardViewModel** vào **Models** thư mục như hình dưới đây.



Hình 13. Models

Sau đây là các lớp mô hình model classes

Model Classes

```
public class StudentViewModel
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public AddressViewModel Address { get; set; }
    public StandardViewModel Standard { get; set; }
}

public class StandardViewModel
{
    public int StandardId { get; set; }
    public string Name { get; set; }

    public ICollection<StudentViewModel> Students { get; set; }
}

public class AddressViewModel
{
    public int StudentId { get; set; }
    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string City { get; set; }
    public string State { get; set; }
}
```

Lưu ý: Các lớp ViewModel hoặc các lớp DTO chỉ để truyền dữ liệu từ bộ điều khiển Web API tới các máy khách. Bạn có thể đặt tên cho nó theo sự lựa chọn của bạn.

Hãy sử dụng Web API trên vào ứng dụng ASP.NET MVC từng bước.

Bước 1:

Trước hết, hãy tạo lớp điều khiển MVC được gọi là StudentController trong thư mục Controllers dưới dạng shown below. Right click on the Controllers folder > Add..> select Controller.

MVC Controller

```
public class StudentController : Controller
{
    // GET: Student
    public ActionResult Index()
    {
        return View();
    }
}
```

Bước 2:

Chúng ta cần truy cập Web API trong phương thức thực thi Index () bằng HttpClient như hình dưới đây

MVC Controller

```
public class StudentController : Controller
{
    // GET: Student
    public ActionResult Index()
    {
        I Enumerable<StudentViewModel> students = null;

        using (var client = new HttpClient())
        {
            client.BaseAddress = new Uri("http://localhost:64189/api/");
            //HTTP GET
            var responseTask = client.GetAsync("student");
            responseTask.Wait();

            var result = responseTask.Result;
            if (result.IsSuccessStatusCode)
```

```

{
    var readTask = result.Content.ReadAsAsync<IList<StudentViewModel>>();
    readTask.Wait();

    students = readTask.Result;
}

else //web api sent error response
{
    //log response status here..

    students = Enumerable.Empty<StudentViewModel>();

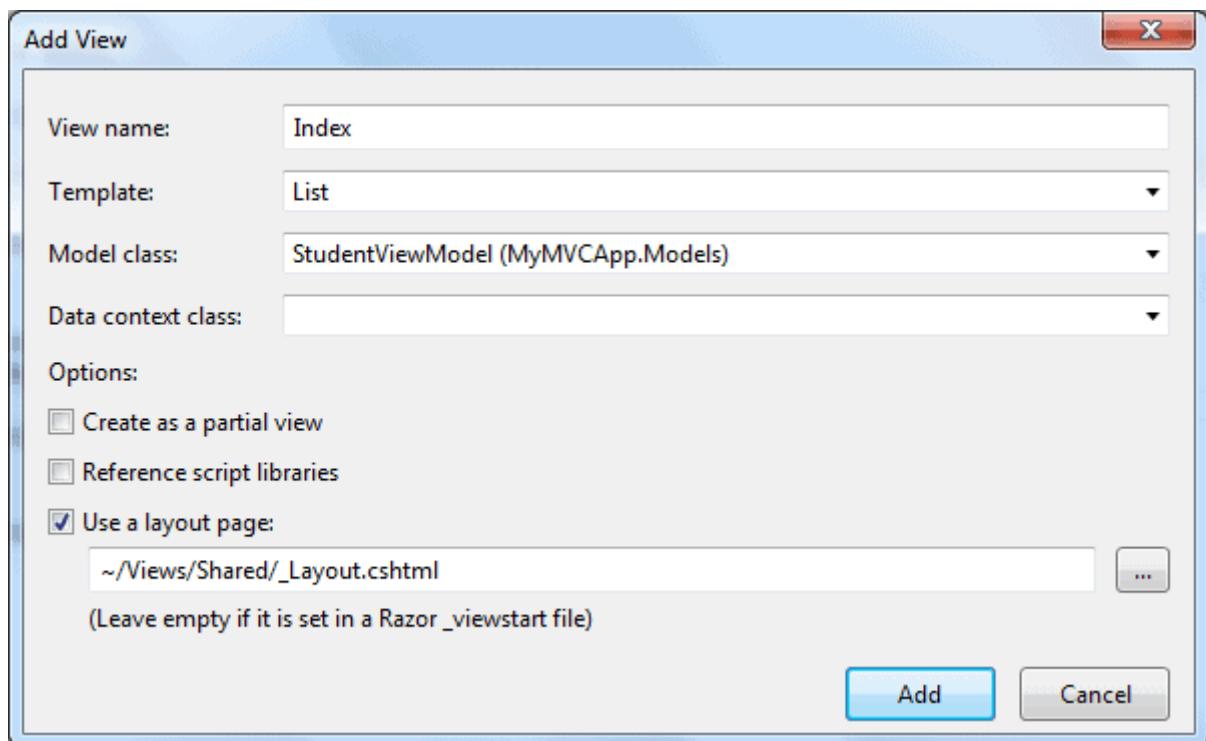
    ModelState.AddModelError(string.Empty, "Server error. Please contact administrator.");
}

return View(students);
}
}
}

```

Bước 3:

Bây giờ, chúng ta cần thêm chế độ xem Chỉ mục. Nhấp chuột phải vào phương pháp phương thức Chỉ mục và chọn **Add View..** Lựa chọn. Điều này sẽ mở ra **Add View** popup như hình dưới đây. Bây giờ, hãy chọn Danh sách dưới dạng mẫu và **StudentViewModel** như lớp Model như bên dưới (chúng ta đã tạo **StudentViewModel** trong phần trước).



Hình 14. Thêm chế độ xem trong ASP.NET MVC

Bấm **ADD** để thêm chế độ xem Chỉ mục trong thư mục **VIEW**. Điều này sẽ tạo ra **Index.cshtml** sau.

```
Index.cshtml
@model IEnumerable<WebAPIDemo.Models.StudentViewModel>

@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.FirstName)
        </th>
    </tr>
```

```

</th>
<th>
    @Html.DisplayNameFor(model => model.LastName)
</th>
<th></th>
</tr>

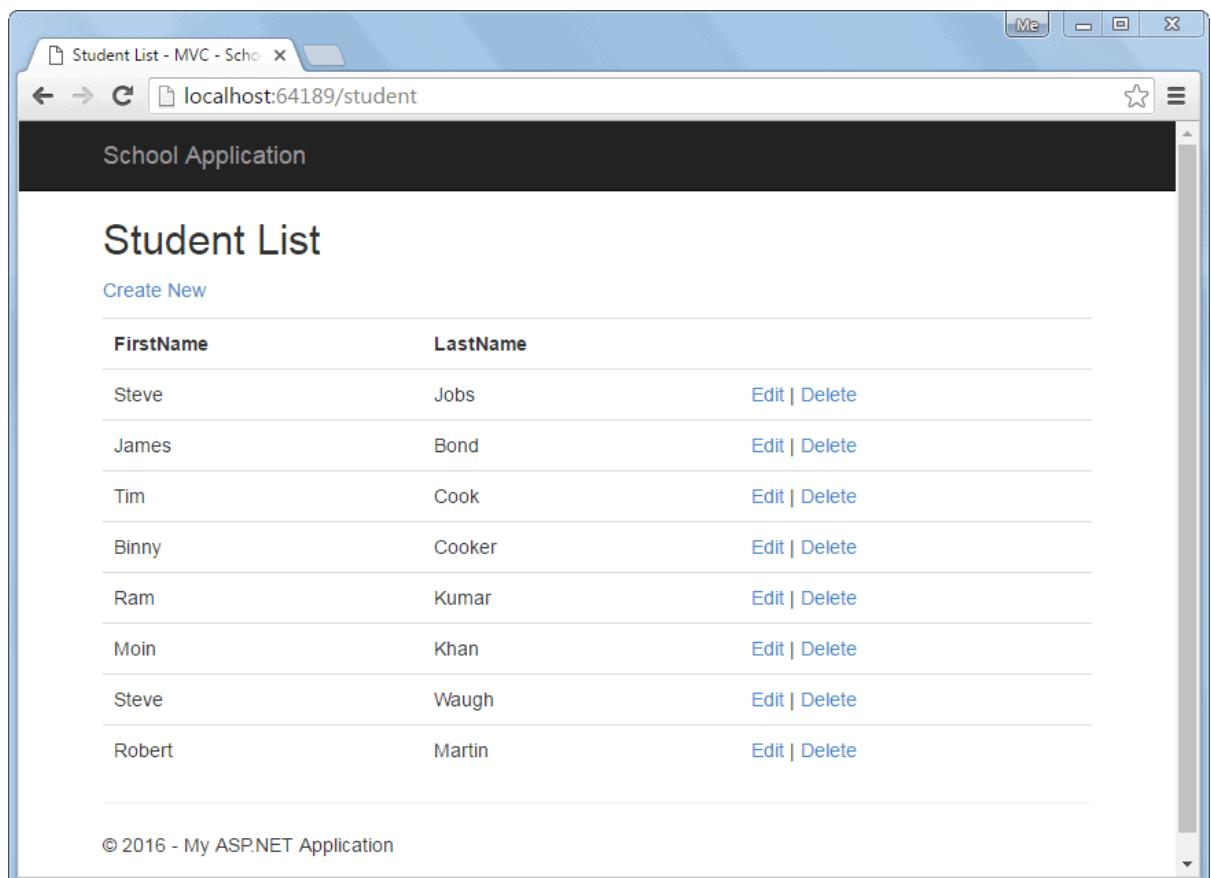
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.FirstName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.LastName)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
            @Html.ActionLink("Details", "Details", new { id=item.Id }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.Id })
        </td>
    </tr>
}

</table>

```

Xóa liên kết Chi tiết khỏi Ché độ xem vì ta sẽ không tạo trang Chi tiết tại đây.

Bây giờ, chạy ứng dụng và bạn sẽ thấy danh sách sinh viên trên trình duyệt như hình dưới đây.



Hình 15. Xem danh sách sinh viên

Lỗi hiển thị

Ta đã hiển thị thành công các bản ghi trong chế độ xem ở trên nhưng nếu Web API trả về phản hồi lỗi thì sao?

Để hiển thị thông báo lỗi thích hợp trong dạng xem MVC, hãy thêm ValidationSummary () như hình dưới đây.

Index.cshtml

```
@model IEnumerable<WebAPIDemo.Models.StudentViewModel>

@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")

```

```

</p>
<table class="table">
<tr>
<th>
    @Html.DisplayNameFor(model => model.FirstName)
</th>
<th>
    @Html.DisplayNameFor(model => model.LastName)
</th>
<th></th>
</tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.FirstName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.LastName)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.Id })
        </td>
    </tr>
}

<tr>
<td>
    @Html.ValidationSummary(true, "", new { @class = "text-danger" })
</td>
</tr>

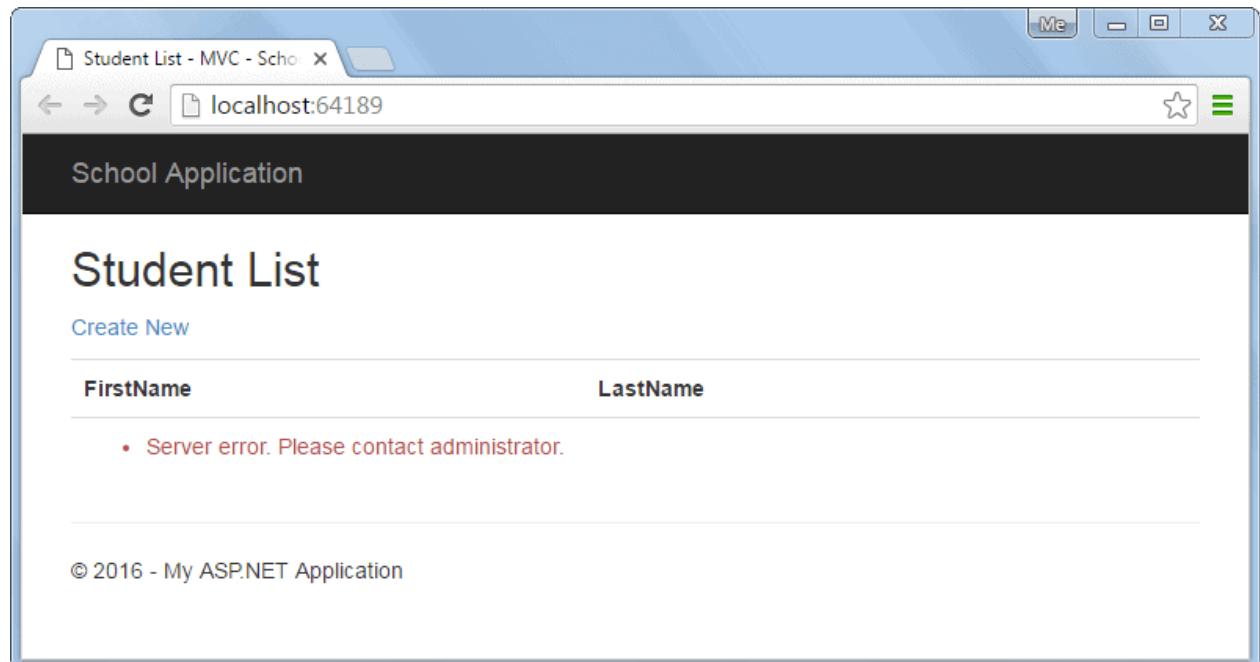
</table>

```

Ở chế độ xem trên, ta đã thêm @ Html.ValidationSummary (true, "", new {@class = "text-risk"}) vào hàng cuối cùng của bảng. Đây là để hiển thị thông báo lỗi nếu Web API trả về phản hồi lỗi với trạng thái khác 200 OK.

Xin lưu ý rằng ta đã thêm lỗi mô hình trong phương thức thực thi Index () trong lớp StudentController được tạo ở bước 2 nếu Web API phản hồi với mã trạng thái khác 200 OK.

Vì vậy, bây giờ, nếu Web API trả về bất kỳ loại lỗi nào thì chế độ xem Danh sách sinh viên sẽ hiển thị thông báo bên dưới.



Hình 16. Hiển thị thông báo lỗi

Trong phần tiếp theo, chúng ta sẽ sử dụng phương pháp Đăng để tạo bản ghi mới trong nguồn dữ liệu cơ bản bằng cách nhấp vào liên kết Tạo Mới ở giao diện trên.

6.3 Triển khai một Trang web với phương thức POST

Trong phần trước, chúng ta đã học cách sử dụng phương thức Get Web API và hiển thị các bản ghi trong Dạng xem ASP.NET. Ở đây, chúng ta sẽ thấy cách sử dụng phương thức Post của Web API để tạo một bản ghi mới trong nguồn dữ liệu.

Chúng ta sẽ tạo Web API với phương thức Post như bên dưới.

Sample Web API with Post Method

```
public class StudentController : ApiController
{
    public StudentController()
    {
    }
```

```

//Get action methods of the previous section
public IHttpActionResult PostNewStudent(StudentViewModel student)
{
    if (!ModelState.IsValid)
        return BadRequest("Not a valid model");

    using (var ctx = new SchoolDBEntities())
    {
        ctx.Students.Add(new Student()
        {
            StudentID = student.Id,
            FirstName = student.FirstName,
            LastName = student.LastName
        });

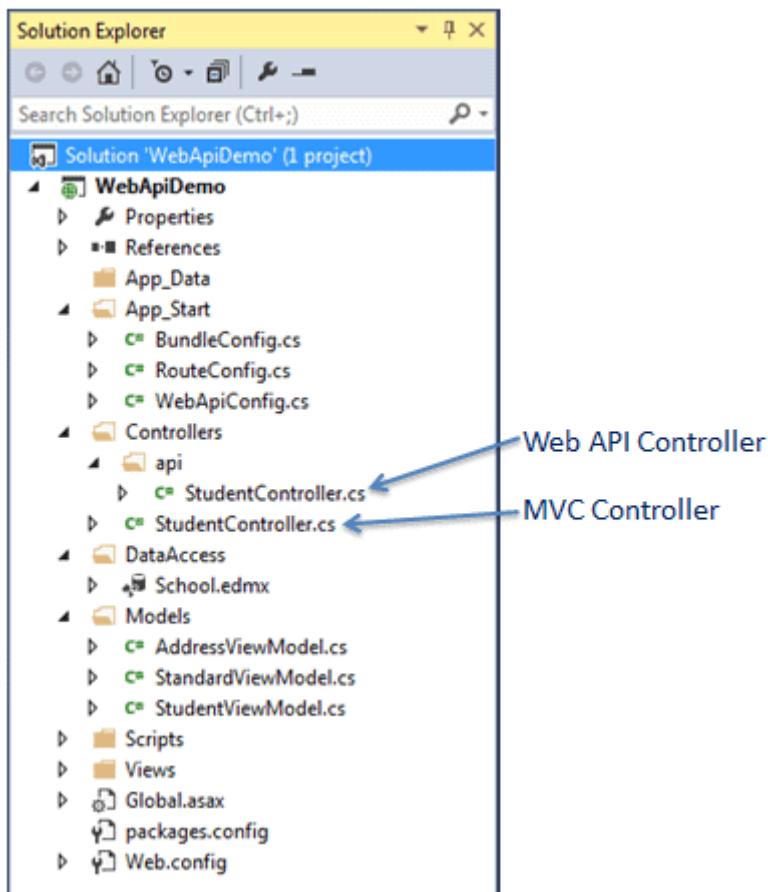
        ctx.SaveChanges();
    }

    return Ok();
}

```

Trong Web API ở trên, phương thức **PostNewStudent** sẽ xử lý yêu cầu HTTP request POST <http://localhost:64189/api/student>. Nó sẽ chèn bản ghi mới vào cơ sở dữ liệu bằng Entity Framework và sẽ trả về trạng thái phản hồi 200 OK.

Sau đây là cấu trúc dự án **Web API + MVC** được tạo trong các phần trước. Ta sẽ thêm các lớp cần thiết trong dự án này.



Hình 17. Dự án Web API

Bây giờ, hãy tạo chế độ xem MVC để tạo bản ghi mới bằng cách sử dụng phương thức Bài đăng Web API ở trên.

Bước 1:

Đầu tiên, chúng ta cần thêm phương thức thực thi "create" sẽ hiển thị chế độ xem "Tạo sinh viên mới" nơi người dùng có thể nhập dữ liệu và gửi nó. Chúng ta đã tạo lớp StudentController trong phần trước để hiển thị dạng xem danh sách sinh viên. Tại đây, thêm phương thức thực thi "tạo" để hiển thị chế độ xem "Tạo sinh viên mới" được hiển thị bên dưới.

MVC Controller

```
public class StudentController : Controller
{
    public ActionResult Index()
    {
        //consume Web API Get method here..

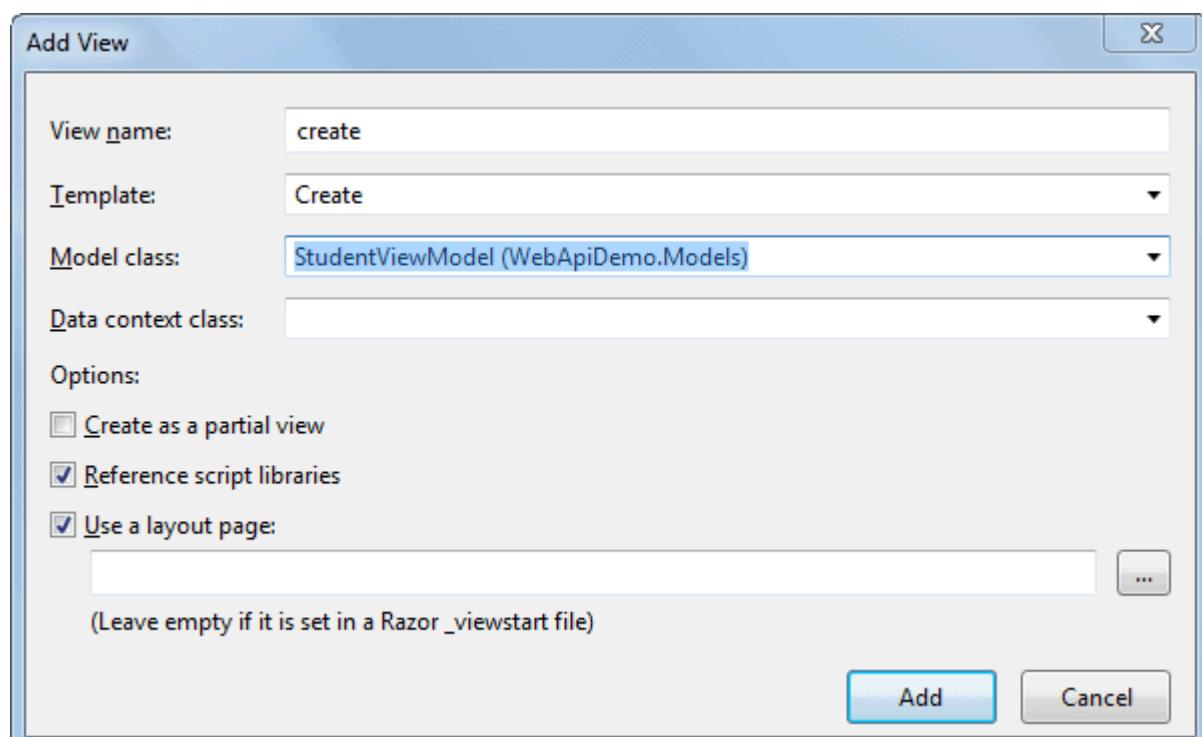
        return View();
    }
}
```

```

public ActionResult create()
{
    return View();
}

```

Bây giờ, nhấp chuột phải vào phương pháp phương thức ở trên và chọn **Add View..**. Thao tác này sẽ mở sau cửa sổ bật lên **Thêm Chế độ xem**.



Hình 18. Thêm chế độ xem trong ASP.NET MVC

Bây giờ, hãy chọn **Create Template**, StudentViewModel class as a model và click vào nút Add như hình trên. Điều này sẽ tạo ra create.cshtml trong Views > Student thư mục như bên dưới.

create.cshtml

```

@model WebApiDemo.Models.StudentViewModel

 @{
    ViewBag.Title = "Create New Student - MVC";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Create New Student</h2>

```

```

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.FirstName, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.FirstName, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.FirstName, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.LastName, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.LastName, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.LastName, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
}

```

```

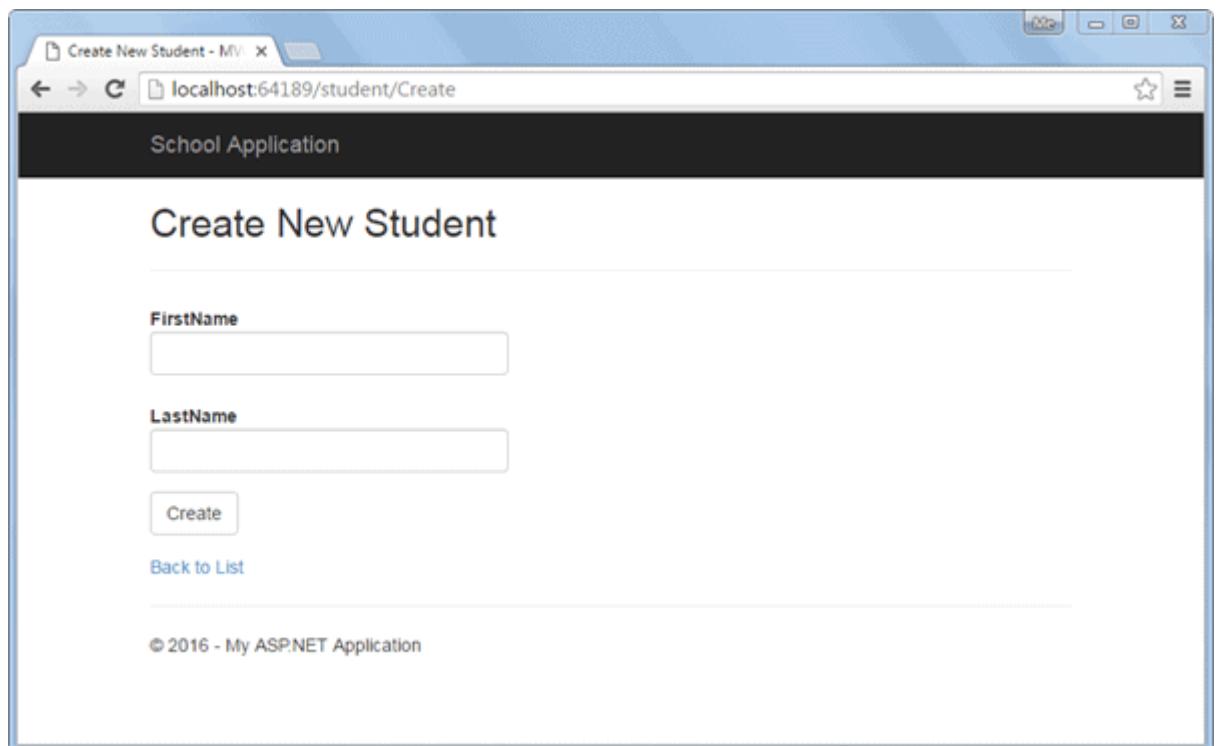
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

```

Trong giao diện trên, `Html.BeginForm()` tạo thẻ biểu mẫu HTML `<form>` `action = "/Student/Create"` `method = "post"` `</form>` sẽ gửi yêu cầu đăng bài khi người dùng nhấp vào nút tạo.

Bây giờ, hãy chạy dự án và điều hướng đến `http://localhost:64189/student/create`. Nó sẽ hiển thị chế độ xem nhập dữ liệu đơn giản như hình dưới đây.



Hình 19. Create New Student View

Ngay sau khi người dùng nhập dữ liệu sinh viên và nhấp vào **Create** trong giao diện trên, nó sẽ gửi yêu cầu Đăng đến Student MVC bộ điều khiển. Để xử lý bài đăng này, yêu cầu thêm `HttpPost` phương pháp phương thức "create" như hình bên dưới.

Post Method in MVC Controller

```

public class StudentController : Controller
{
    public ActionResult Index()
    {
        //consume Web API Get method here..
    }
}

```

```

        return View();
    }

    public ActionResult create()
    {
        return View();
    }

    [HttpPost]
    public ActionResult create(StudentViewModel student)
    {
        using (var client = new HttpClient())
        {
            client.BaseAddress = new Uri("http://localhost:64189/api/student");

            //HTTP POST
            var postTask = client.PostAsJsonAsync<StudentViewModel>("student", student);
            postTask.Wait();

            var result = postTask.Result;
            if (result.IsSuccessStatusCode)
            {
                return RedirectToAction("Index");
            }
        }

        ModelState.AddModelError(string.Empty, "Server Error. Please contact administrator.");

        return View(student);
    }
}

```

Như bạn có thể thấy trong phương thức thực thi create () của HttpPost ở trên, nó sử dụng HttpClient để gửi yêu cầu HTTP request POST đến Web API với đối tượng StudentViewModel.

Nếu phản hồi trả về trạng thái thành công thì nó sẽ chuyển hướng đến chế độ xem danh sách. Truy cập phần HttpClient để tìm hiểu thêm về nó.

Bây giờ, chạy dự án và điều hướng đến <http://localhost:64189/student/create>, nhập thông tin sinh viên như hình bên dưới.

Create New Student

FirstName
Steve

LastName
Jobs

Create

Back to List

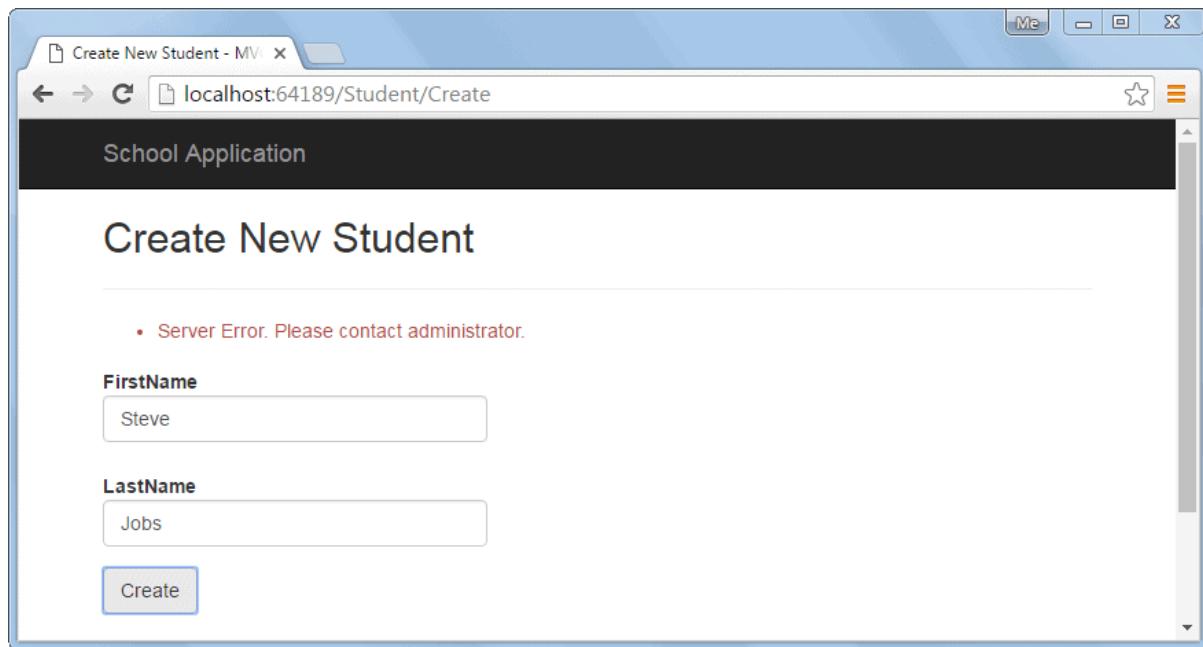
Hình 20. Tạo một sinh viên mới

Bây giờ, khi nhấp vào nút tạo ở trên, nó sẽ chèn một bản ghi mới trong DB và chuyển hướng đến dạng xem danh sách như hình dưới đây.

FirstName	LastName	
Steve	Jobs	Edit Delete

Hình 21. Chuyển hướng đến Chế độ xem danh sách sinh viên

Ngoài ra, chế độ xem tạo ở trên sẽ hiển thị thông báo lỗi nếu Web API gửi phản hồi lỗi như hình dưới đây.



Hình 22. Hiển thị thông báo lỗi

Vì vậy, theo cách này, chúng ta có thể sử dụng phương thức Post của Web API để thực hiện Yêu cầu HTTP request POST để tạo một bản ghi mới.

Tiếp theo, sử dụng phương thức Put của Web API để chỉnh sửa bản ghi hiện có.

6.4 Triển khai một Trang web với phương pháp PUT

Trong hai phần trước, chúng ta đã học cách sử dụng các phương thức Nhận và Đăng Web API trong Dạng xem ASP.NET. Ở đây, chúng ta sẽ thấy cách sử dụng phương thức Put của Web API để cập nhật một bản ghi hiện có.

Ta sẽ tạo một Web API với phương thức Put xử lý Yêu cầu HTTP request PUT trong như bên dưới.

Sample Web API with Put method

```
public class StudentController : ApiController
{
    public StudentController()
    {
    }

    public IHttpActionResult Put(StudentViewModel student)
    {
        if (!ModelState.IsValid)
            return BadRequest("Not a valid data");
    }
}
```

```

using (var ctx = new SchoolDBEntities())
{
    var existingStudent = ctx.Students.Where(s => s.StudentID == student.Id).FirstOrDefault<Student>();

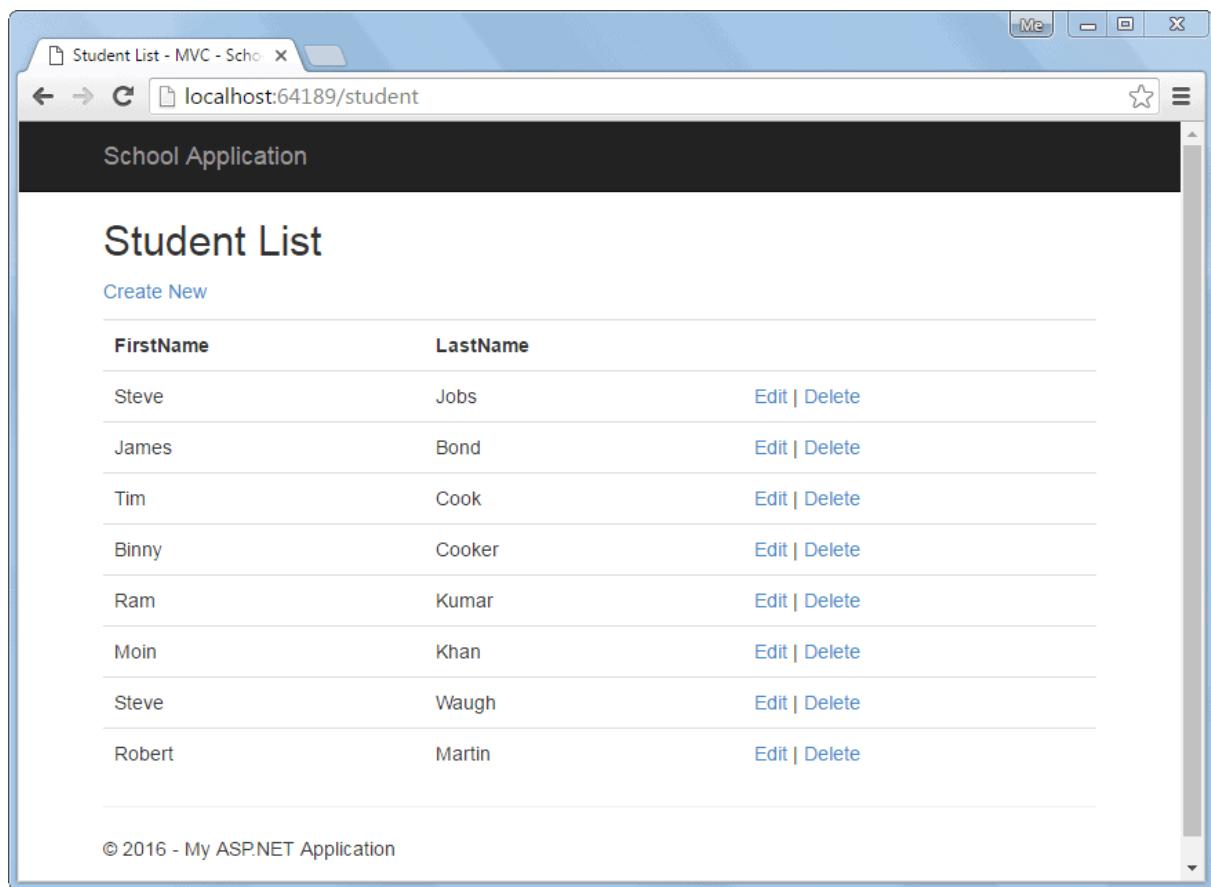
    if (existingStudent != null)
    {
        existingStudent.FirstName = student.FirstName;
        existingStudent.LastName = student.LastName;

        ctx.SaveChanges();
    }
    else
    {
        return NotFound();
    }
}

return Ok();
}
}

```

Ta tạo ra **Student List** xem trong phần trước như bên dưới. Trong dạng xem bên dưới có một liên kết chỉnh sửa cho mỗi bản ghi để chỉnh sửa bản ghi cụ thể đó. Ta sẽ xử lý chức năng chỉnh sửa trong phần này.



Hình 23. Xem danh sách sinh viên

Vì vậy, hãy sử dụng phương pháp Đặt Web API bằng cách triển khai chức năng chỉnh sửa.

Bước 1:

Trong chế độ xem Danh sách sinh viên ở trên, khi người dùng nhấp vào liên kết Chính sửa, nó sẽ gửi Yêu cầu HTTP request GET `http://localhost:64189/student/edit/{id}` đến bộ điều khiển MVC. Vì vậy, chúng ta cần thêm phương thức thực thi `HttpGet "Chỉnh sửa"` trong `StudentController` để hiển thị dạng xem chỉnh sửa như hình dưới đây.

Implement Edit Action Method

```
public class StudentController : Controller
{
    public ActionResult Index()
    {
        //consume Web API Get method here..

        return View();
    }

    public ActionResult Edit(int id)
```

```

{
    StudentViewModel student = null;

    using (var client = new HttpClient())
    {
        client.BaseAddress = new Uri("http://localhost:64189/api/");
        //HTTP GET
        var responseTask = client.GetAsync("student?id=" + id.ToString());
        responseTask.Wait();

        var result = responseTask.Result;
        if (result.IsSuccessStatusCode)
        {
            var readTask = result.Content.ReadAsAsync<StudentViewModel>();
            readTask.Wait();

            student = readTask.Result;
        }
    }

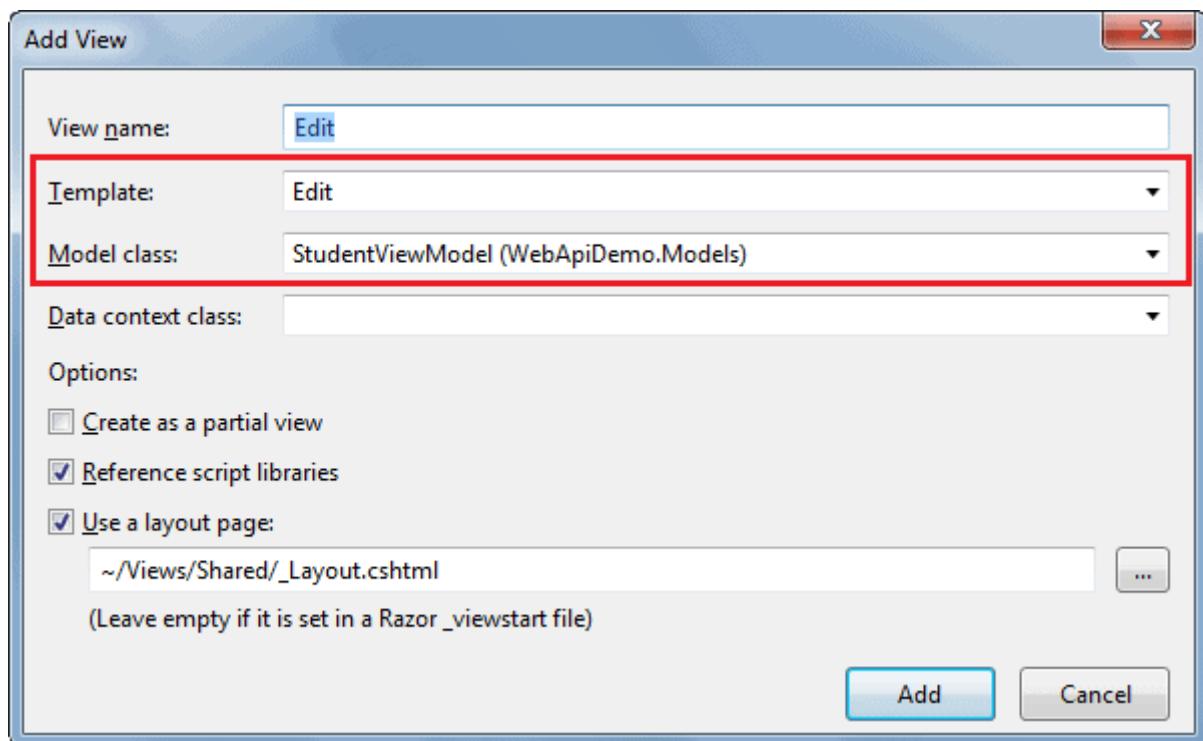
    return View(student);
}
}

```

Như bạn có thể thấy ở trên, phương thức thực thi Edit () bao gồm tham số id. Tham số id này sẽ được liên kết với tham số id chuỗi truy vấn. Ta sử dụng id này để lấy hồ sơ học sinh từ cơ sở dữ liệu bằng HttpClient và chuyển hồ sơ học sinh vào dạng xem chỉnh sửa. Truy cập phần HttpClient để biết thêm về nó.

Bước 2:

Tạo chế độ xem chỉnh sửa bằng cách nhấp chuột phải vào phương pháp phương thức Chính sửa ở trên và chọn Thêm Chế độ xem .. Thao tác này sẽ mở cửa sổ bật lên Thêm Chế độ xem như hình dưới đây.



Hình 24. Thêm chế độ xem trong ASP.NET MVC

Trong cửa sổ bật lên Thêm Chế độ xem, chọn Chính sửa mẫu và StudentViewModel làm lớp mô hình như được hiển thị ở trên. Nhập vào nút Thêm để tạo chế độ xem Edit.cshtml trong thư mục Chế độ xem> Sinh viên như hình dưới đây.

```
Edit.cshtml

@model WebApiDemo.Models.StudentViewModel

{@{
    ViewBag.Title = "Edit Student - MVC";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Edit Student</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <hr />
```

```

@Html.ValidationSummary(true, "", new { @class = "text-danger" })
@Html.HiddenFor(model => model.Id)

<div class="form-group">
    @Html.LabelFor(model => model.FirstName, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.FirstName, new { htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.FirstName, "", new { @class = "text-danger" })
    </div>
</div>

<div class="form-group">
    @Html.LabelFor(model => model.LastName, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.LastName, new { htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.LastName, "", new { @class = "text-danger" })
    </div>
</div>

<div class="form-group">
    <div class="col-md-offset-2 col-md-10">
        <input type="submit" value="Save" class="btn btn-default" />
    </div>
</div>
</div>
}

<div>
    @Html.ActionLink("Back to List", "Index")

```

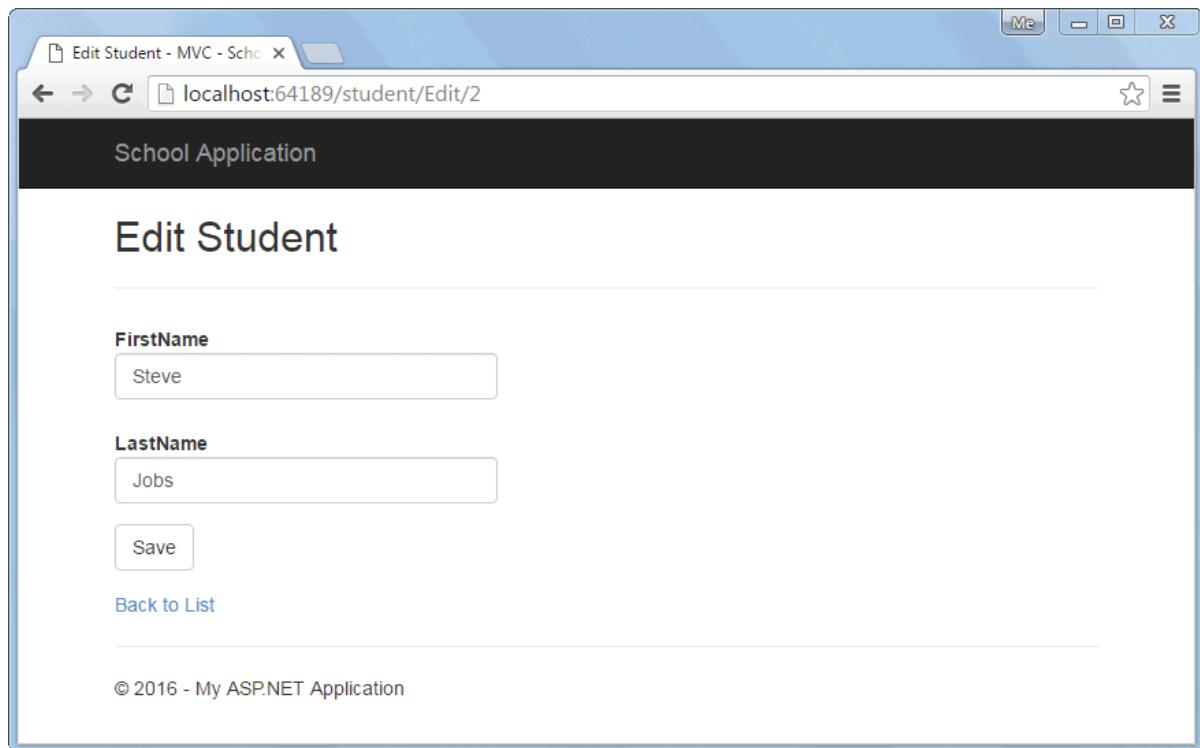
```
</div>
```

Theo quan điểm trên, `Html.BeginForm()` tạo thẻ biểu mẫu HTML `<form>` `action="/Student/edit"` `method="post"` `</form>` sẽ gửi yêu cầu đăng khi người dùng nhập vào nút lưu.

Bây giờ, nó sẽ hiển thị dạng xem StudentList sau khi bạn chạy dự án bằng cách nhấn Ctrl + F5.

FirstName	LastName	
Steve	Jobs	Edit Delete
James	Bond	Edit Delete
Tim	Cook	Edit Delete
Binny	Cooker	Edit Delete
Ram	Kumar	Edit Delete
Moin	Khan	Edit Delete
Steve	Waugh	Edit Delete
Robert	Martin	Edit Delete

Nó sẽ hiển thị chế độ xem chỉnh sửa sau khi bạn nhập vào liên kết Chỉnh sửa trong chế độ xem ở trên.



Hình 25. Chỉnh sửa chế độ xem

Bây giờ, hãy triển khai phương thức thực thi `HttpPost Edit` sẽ được thực thi khi người dùng nhập vào nút Lưu ở trên.

Bước 3:

Thêm phương thức thực thi `HttpPost` trong `StudentController` của MVC sẽ gửi Yêu cầu HTTP request `PUT` tới Web API để cập nhật bản ghi hiện tại.

Implement `HttpPut` Action Method

```
public class StudentController : Controller
{
    public ActionResult Edit(int id)
    {
        StudentViewModel student = null;

        using (var client = new HttpClient())
        {
            client.BaseAddress = new Uri("http://localhost:64189/api/");
            //HTTP GET
            var responseTask = client.GetAsync("student?id=" + id.ToString());
            responseTask.Wait();
        }
    }
}
```

```

        var result = responseTask.Result;
        if (result.IsSuccessStatusCode)
        {
            var readTask = result.Content.ReadAsAsync<StudentViewModel>();
            readTask.Wait();

            student = readTask.Result;
        }
    }

    return View(student);
}

[HttpPost]
public ActionResult Edit(StudentViewModel student)
{
    using (var client = new HttpClient())
    {
        client.BaseAddress = new Uri("http://localhost:64189/api/student");

        //HTTP PUT
        var putTask = client.PutAsJsonAsync<StudentViewModel>("student", student);
        putTask.Wait();

        var result = putTask.Result;
        if (result.IsSuccessStatusCode)
        {
            return RedirectToAction("Index");
        }
    }

    return View(student);
}

```

Như bạn thấy ở trên, `HttpPut` Chính sửa cách sử dụng phương pháp phương thức `HttpClient` để gửi Yêu cầu HTTP request `PUT` tới Web API với hồ sơ sinh viên được cập nhật. Kích chọn `HttpClient` phần để tìm hiểu thêm về nó.

Vì vậy, theo cách này, chúng ta có thể sử dụng phương thức `Put` của Web API để thực hiện Yêu cầu HTTP request `PUT` để chỉnh sửa bản ghi hiện có.

Tiếp theo, sử dụng phương thức Xóa của Web API để xóa bản ghi trong nguồn dữ liệu.

6.5 Triển khai một Trang web với phương thức DELETE

Trong các phần trước, ta đã giới thiệu các phương pháp `Get`, `Post` và `Put` của Web API. Ở đây, chúng ta sẽ sử dụng phương thức `Delete` của Web API trong ASP.NET MVC để xóa một bản ghi.

Ta sẽ tạo một Web API với phương thức `Delete` để xử lý Yêu cầu HTTP request `DELETE` như bên dưới.

Sample Web API with Delete Method

```
public class StudentController : ApiController
{
    public StudentController()
    {

    }

    public IHttpActionResult Delete(int id)
    {
        if (id <= 0)
            return BadRequest("Not a valid student id");

        using (var ctx = new SchoolDBEntities())
        {
            var student = ctx.Students
                .Where(s => s.StudentID == id)
                .FirstOrDefault();

            ctx.Entry(student).State = System.Data.Entity.EntityState.Deleted;
            ctx.SaveChanges();
        }
    }
}
```

```

        return Ok();
    }
}

```

Sau đây là dạng xem danh sách Sinh viên được tạo trong phương pháp Tiêu thụ lấy trong phần MVC. Ở đây, ta sẽ triển khai chức năng xóa khi người dùng nhập vào liên kết Xóa trong giao diện người dùng sau.

FirstName	LastName	
James	Bond	Edit Delete
Tim	Cook	Edit Delete
Binny	Cooker	Edit Delete
Ram	Kumar	Edit Delete
Moin	Khan	Edit Delete
Steve	Waugh	Edit Delete
Robert	Martin	Edit Delete
Steve	Jobs	Edit Delete

© 2016 - My ASP.NET Application

Hình 26. Xem danh sách sinh viên

Khi người dùng nhập vào liên kết Xóa trong giao diện người dùng ở trên, nó sẽ gửi HTTP Get request `http://localhost:64189/student/delete/{id}` tới Student controller với thông số id hiện tại. Vì vậy, hãy triển khai chức năng xóa bằng cách sử dụng phương pháp Xóa Web API.

Bước 1:

Tạo phương thức thực thi `HttpGet` Xóa bằng tham số id trong MVC `StudentController` như hình bên dưới.

Implement <code>HttpGet Delete</code> method
--

<pre> public class StudentController : Controller { // GET: Student </pre>
--

```

public ActionResult Index()
{
    IList<StudentViewModel> students = null;

    using (var client = new HttpClient())
    {
        client.BaseAddress = new Uri("http://localhost:64189/api/student");
        //HTTP GET
        var responseTask = client.GetAsync("student");
        responseTask.Wait();

        var result = responseTask.Result;
        if (result.IsSuccessStatusCode)
        {
            var readTask = result.Content.ReadAsAsync<IList<StudentViewModel>>();
            readTask.Wait();

            students = readTask.Result;
        }
    }

    return View(students);
}

public ActionResult Delete(int id)
{
    using (var client = new HttpClient())
    {
        client.BaseAddress = new Uri("http://localhost:64189/api/");

        //HTTP DELETE
        var deleteTask = client.DeleteAsync("student/" + id.ToString());
        deleteTask.Wait();
    }
}

```

```
var result = deleteTask.Result;
if (result.IsSuccessStatusCode)
{
    return RedirectToAction("Index");
}

return RedirectToAction("Index");
}
```

Như bạn có thể thấy, phương thức thực thi Delete () ở trên sử dụng HttpClient để gửi Yêu cầu HTTP request DELETE với tham số id hiện tại. Bộ điều khiển Web API được hiển thị trong ví dụ mã đầu tiên, sẽ xử lý yêu cầu DELETE này và xóa bản ghi khỏi nguồn dữ liệu. Truy cập phần HttpClient để tìm hiểu thêm về nó.

Vì vậy, theo cách này, bạn có thể sử dụng phương thức Xóa của Web API trong ASP.NET MVC.

CHƯƠNG 7 BẢO MẬT WEB API

7.1 Xác thực và Ủy quyền trong Web API

Trong phần này, chúng ta sẽ thảo luận về Xác thực và Ủy quyền trong Web API. Ở đây, ta sẽ cung cấp cho bạn cái nhìn tổng quan về Xác thực và Ủy quyền trong Web API và từ phần tiếp theo trở đi, ta sẽ thảo luận về việc triển khai thực tế Xác thực và Ủy quyền trong Web API ASP.NET với các ví dụ.

Khi bạn tạo Dịch vụ Web API, điều quan trọng nhất bạn cần quan tâm là bảo mật, nghĩa là bạn cần kiểm soát quyền truy cập vào Dịch vụ Web API của mình. Vì vậy, hãy bắt đầu cuộc thảo luận với định nghĩa về Xác thực và Ủy quyền

Xác thực là quá trình xác định người dùng. Ví dụ: một người dùng giả sử James đăng nhập bằng tên người dùng và mật khẩu của anh ấy và máy chủ sử dụng tên người dùng và mật khẩu của anh ấy để xác thực James.

Ủy quyền là quá trình quyết định xem người dùng được xác thực có được phép thực hiện một phương thức trên một tài nguyên cụ thể (Tài nguyên Web API) hay không. Ví dụ, James (người dùng được xác thực) có quyền lấy tài nguyên nhưng không có quyền tạo tài nguyên.

7.2 Xác thực cơ bản Web API ASP.NET

7.2.1 Tại sao chúng ta cần Xác thực trong Web API?

Hãy bắt đầu cuộc thảo luận với một trong các Ràng buộc còn lại, tức là Ràng buộc không quốc tịch. Ràng buộc Không trạng thái là một trong những Ràng buộc Hạn chế nói rằng giao tiếp giữa máy khách và máy chủ phải là không trạng thái giữa các yêu cầu. Điều này có nghĩa là ta không nên lưu trữ thông tin máy khách trên máy chủ cần thiết để xử lý yêu cầu. Yêu cầu đến từ máy khách phải chứa tất cả các thông tin cần thiết được máy chủ yêu cầu để xử lý yêu cầu đó. Điều này đảm bảo rằng mỗi yêu cầu đến từ máy khách có thể được máy chủ xử lý độc lập.

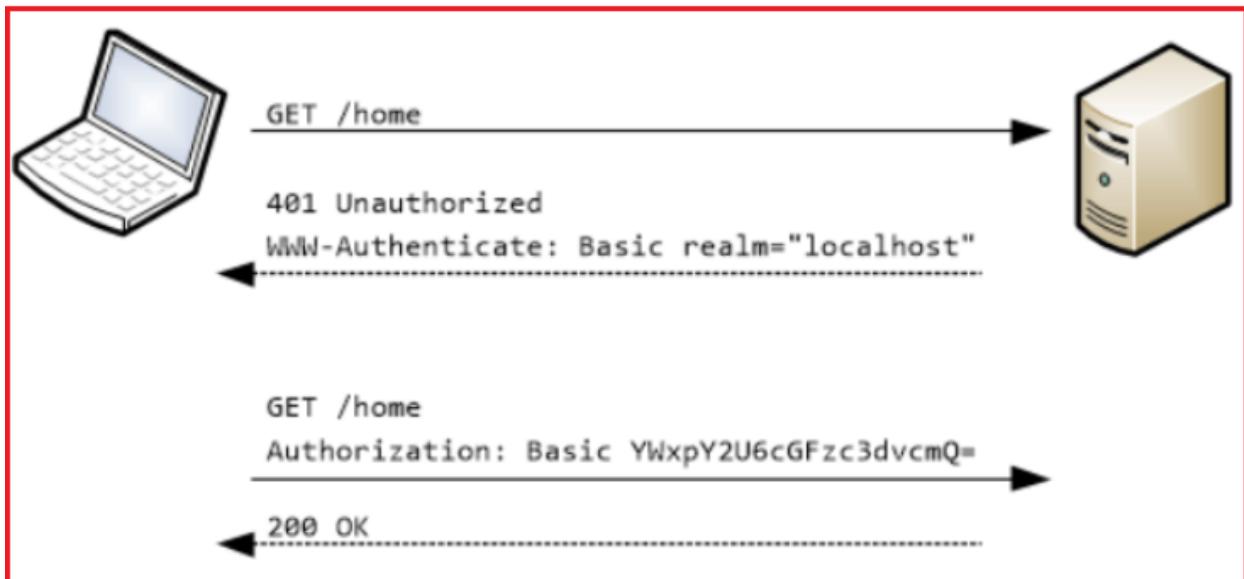
Cách tiếp cận trên là tốt và ưu điểm là chúng ta có thể tách máy khách hoặc máy chủ tại bất kỳ thời điểm nào mà không ảnh hưởng đến người khác. Ở đây, máy khách có thể là bất kỳ loại ứng dụng nào bao gồm JavaScript hoặc bất kỳ ngôn ngữ lập trình nào khác như Java hoặc C#. Máy chủ không nhớ máy khách khi yêu cầu đã được xử lý, vì vậy mỗi và mọi yêu cầu đến từ máy khách đều mới đối với máy chủ và máy chủ cần kiểm tra yêu cầu (hầu hết thời gian là tiêu đề HTTP) để xác định người dùng.

Vì vậy, để xử lý yêu cầu của máy chủ, máy khách cần chuyển thông tin đăng nhập của nó với mỗi và mọi yêu cầu và sau đó máy chủ sẽ kiểm tra và khớp thông tin đăng nhập với bất kỳ bộ nhớ lâu dài nào (hầu hết thời gian nó có thể là cơ sở dữ liệu). Nếu thông tin đăng nhập được tìm thấy trong bộ nhớ liên tục thì máy chủ sẽ coi Yêu cầu HTTP request đó là một yêu cầu hợp lệ và xử lý nó nếu nó chỉ trả về lỗi trái phép cho máy khách.

Ta có thể triển khai Xác thực và Ủy quyền theo nhiều cách trong một ứng dụng. Ở đây, trong phần này, chúng ta sẽ thảo luận về cách triển khai **ASP.NET Web API Basic Authentication**.

7.2.2 Xác thực cơ bản hoạt động như thế nào trong Web API?

Vui lòng xem sơ đồ sau.



Hình 27. Hoạt động Basic Authentication trong Web API

Nếu một yêu cầu yêu cầu xác thực và nếu khách hàng không gửi thông tin xác thực trong tiêu đề (hầu hết thời gian đó là tiêu đề Ủy quyền), thì máy chủ sẽ trả về 401 (Không được ủy quyền). Phản hồi cũng sẽ bao gồm tiêu đề WWW-Authenticate, cho biết rằng máy chủ hỗ trợ Basic Authentication.

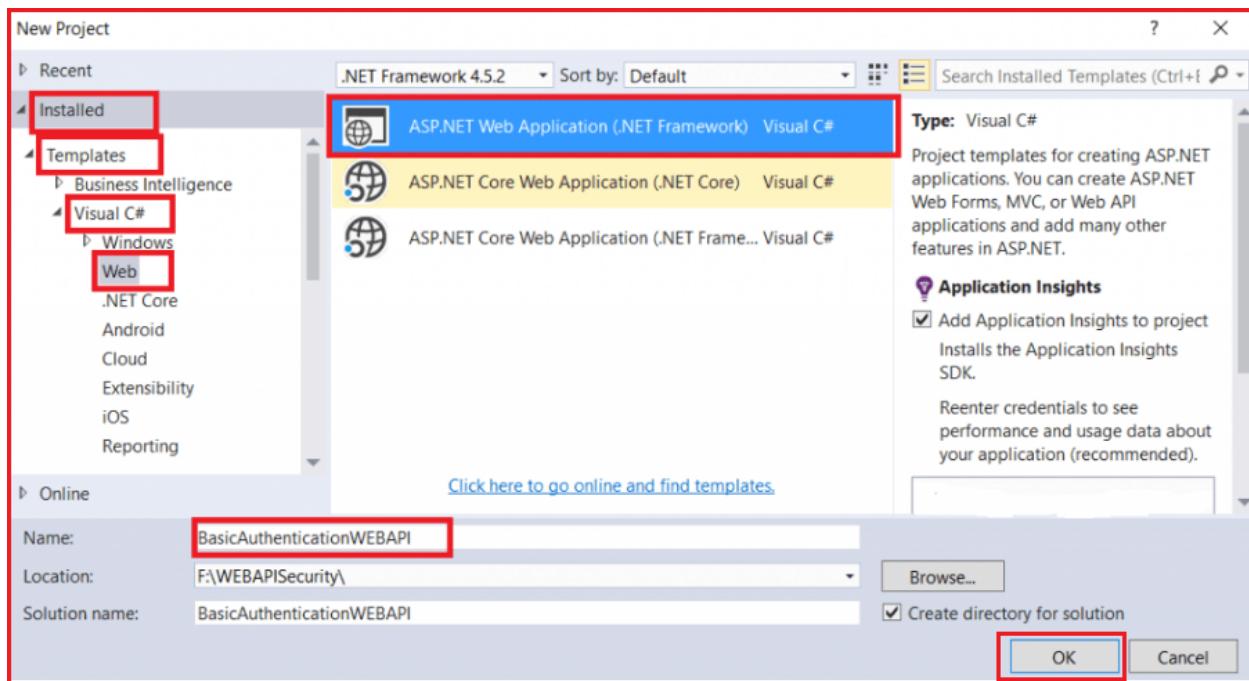
Máy khách gửi một yêu cầu khác đến máy chủ, với thông tin xác thực của máy khách trong tiêu đề Ủy quyền. Nói chung, thông tin xác thực của khách hàng được định dạng dưới dạng chuỗi “tên: mật khẩu”, định dạng được mã hóa base64.

Khi ta đính kèm dữ liệu nhạy cảm (i, e. Tên người dùng và mật khẩu) trong mỗi và mọi Yêu cầu HTTP request, nó phải được chuyển ở định dạng được mã hóa và giao thức phải là HTTPS, khi đó chỉ ta mới có thể bảo vệ dữ liệu của mình qua internet.

Các **ASP.NET Web API Basic Authentication** được thực hiện trong bối cảnh của một “realm.” Máy chủ bao gồm tên của khu vực trong **WWW-Authenticate** đầu trang. Thông tin đăng nhập của người dùng hợp lệ trong lĩnh vực đó. Phạm vi chính xác của một lĩnh vực được xác định bởi máy chủ. Ví dụ, bạn có thể xác định một số lĩnh vực để phân vùng tài nguyên.

Triển khai xác thực cơ bản trong ASP.NET Web API

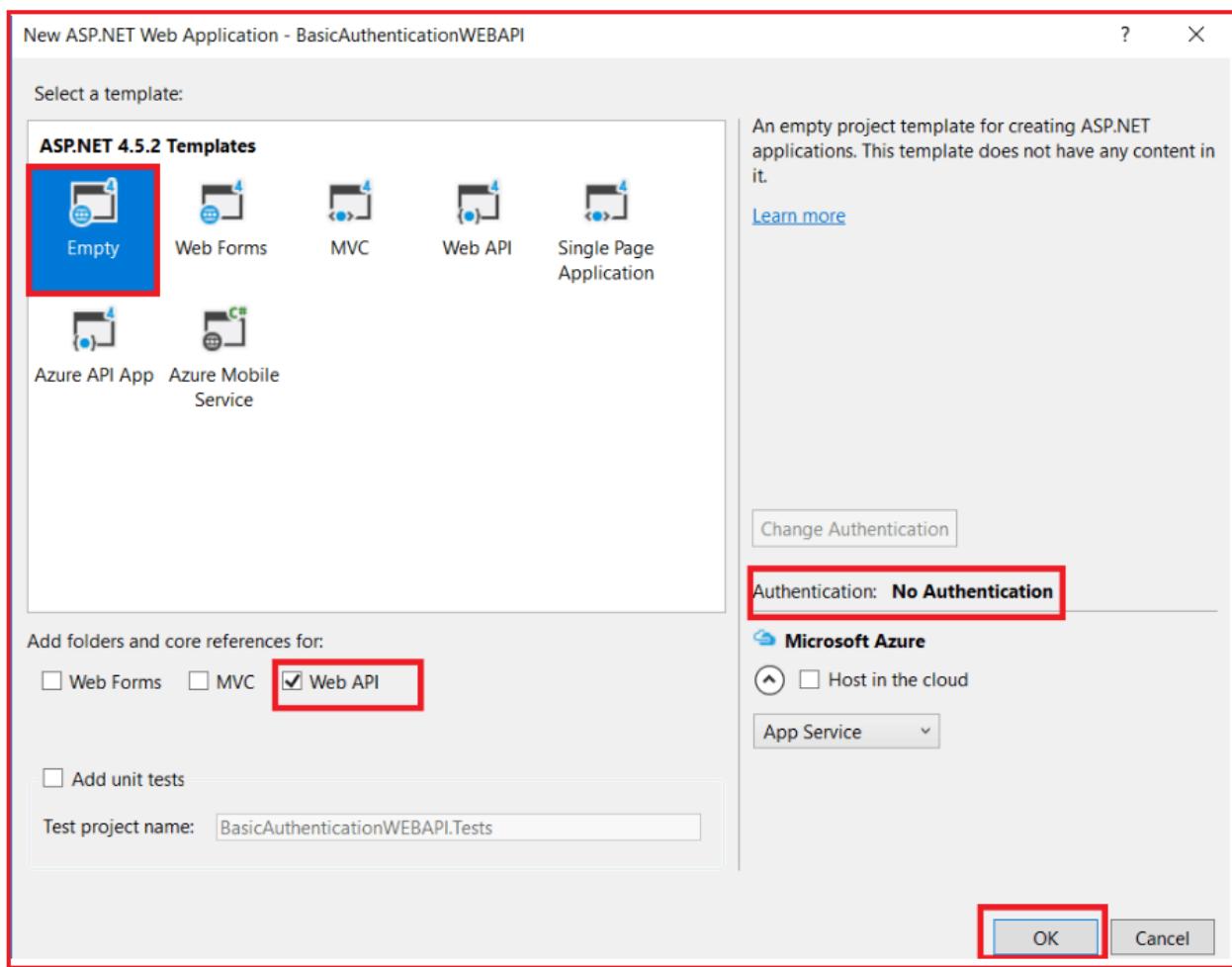
Đầu tiên, tạo một Ứng dụng Web ASP.NET với tên BasicAuthenticationWEBAPI (bạn có thể đặt bất kỳ tên nào) như trong hình dưới đây.



Sau khi bạn nhập vào nút OK, nó sẽ mở ra cửa sổ “Chọn mẫu”. Từ cửa sổ “Chọn mẫu”, hãy chọn

1. Empty template
2. Web API Checkbox
3. No Authentication

Và cuối cùng, bấm vào nút OK như hình dưới đây.



Sau khi bạn nhấp vào nút OK, sẽ mất một khoảng thời gian để tạo dự án cho chúng tôi.

7.2.3 Tạo mô hình

Bây giờ chúng ta cần tạo hai mô hình, tức là User and Employee. Vì vậy, hãy nhấp chuột phải vào thư mục Models và thêm một tệp lớp có tên là User, sau đó sao chép và dán đoạn mã dưới đây.

User.cs

```
namespace BasicAuthenticationWEBAPI.Models
{
    public class User
    {
        public int ID { get; set; }
        public string UserName { get; set; }
        public string Password { get; set; }
    }
}
```

Tương tự, nhấp chuột phải vào thư mục Models và thêm tệp lớp có tên là Employee, sau đó sao chép và dán đoạn mã dưới đây.

Employee.cs

```
namespace BasicAuthenticationWEBAPI.Models

{
    public class Employee
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public string Gender { get; set; }
        public string Dept { get; set; }
        public int Salary { get; set; }
    }
}
```

Bây giờ chúng ta cần tạo hai lớp sẽ trả về danh sách User and Employee. Nhấp chuột phải vào thư mục Models và thêm tệp lớp với Tên UserBL, sau đó sao chép và dán mã bên dưới.

UserBL.cs

```
namespace BasicAuthenticationWEBAPI.Models

{
    public class UsersBL
    {
        public List<User> GetUsers()
        {
            // In Real-time you need to get the data from any persistent storage
            // For Simplicity of this demo and to keep focus on Basic Authentication
            // Here we are hardcoded the data

            List<User> userList = new List<User>();

            userList.Add(new User()
            {
                ID = 101,
                UserName = "MaleUser",
                Password = "123456"
            });
        }
    }
}
```

```

});
```

```

userList.Add(new User()
{
    ID = 101,
    UserName = "FemaleUser",
    Password = "abcdef"
});
```

```

return userList;
}
```

```

}
```

```

}
}
```

Tương tự, nhập chuột phải vào thư mục Models và thêm tệp lớp với tên là EmployeeBL, sau đó sao chép và dán đoạn mã dưới đây.

EmployeeBL.cs

```

namespace BasicAuthenticationWEBAPI.Models
{
    public class EmployeeBL
    {
        public List<Employee> GetEmployees()
        {
            // In Real-time you need to get the data from any persistent storage
            // For Simplicity of this demo and to keep focus on Basic Authentication
            // Here we hardcoded the data

            List<Employee> empList = new List<Employee>();
            for (int i = 0; i < 10; i++)
            {
                if (i > 5)
                {
                    empList.Add(new Employee()
                    {
```

```

        ID = i,
        Name = "Name" + i,
        Dept = "IT",
        Salary = 1000 + i,
        Gender = "Male"
    );
}
else
{
    empList.Add(new Employee()
    {
        ID = i,
        Name = "Name" + i,
        Dept = "HR",
        Salary = 1000 + i,
        Gender = "Female"
    );
}
}

return empList;
}
}
}

```

Bây giờ, chúng ta cần tạo một lớp để kiểm tra tên người dùng và mật khẩu có hợp lệ hay không. Nhấp chuột phải vào thư mục Models và thêm một tệp lớp với Tên UserValidate, sau đó sao chép và dán đoạn mã sau.

UserValidate.cs

```

namespace BasicAuthenticationWEBAPI.Models
{
    public class UserValidate
    {

```

```

//This method is used to check the user credentials

public static bool Login(string username, string password)
{
    UsersBL userBL = new UsersBL();
    var UserLists = userBL.GetUsers();
    return UserLists.Any(user =>
        user.UserName.Equals(username, StringComparison.OrdinalIgnoreCase)
        && user.Password == password);
}
}
}

```

7.2.4 Tạo bộ lọc xác thực cơ bản trong Web API

Nhấp chuột phải vào thư mục Models và thêm tệp lớp với tên BasicAuthenticationAttribute, sau đó sao chép và dán mã sau vào đó

BasicAuthenticationAttribute.cs

```

using System;
using System.Net;
using System.Net.Http;
using System.Security.Principal;
using System.Text;
using System.Threading;
using System.Web;
using System.Web.Http.Controllers;
using System.Web.Http.Filters;

namespace BasicAuthenticationWEBAPI.Models
{
    public class BasicAuthenticationAttribute : AuthorizationFilterAttribute
    {
        private const string Realm = "My Realm";
        public override void OnAuthorization(HttpActionContext actionContext)
        {

```

```

//If the Authorization header is empty or null
//then return Unauthorized
if (actionContext.Request.Headers.Authorization == null)
{
    actionContext.Response = actionContext.Request
        .CreateResponse(HttpStatusCode.Unauthorized);

// If the request was unauthorized, add the WWW-Authenticate header
// to the response which indicates that it require basic authentication
if (actionContext.Response.StatusCode == HttpStatusCode.Unauthorized)
{
    actionContext.Response.Headers.Add("WWW-Authenticate",
        string.Format("Basic realm=\"{0}\"", Realm));
}
else
{
    //Get the authentication token from the request header
    string authenticationToken = actionContext.Request.Headers
        .Authorization.Parameter;

    //Decode the string
    string decodedAuthenticationToken = Encoding.UTF8.GetString(
        Convert.FromBase64String(authenticationToken));

    //Convert the string into an string array
    string[] usernamePasswordArray = decodedAuthenticationToken.Split(':');

    //First element of the array is the username
    string username = usernamePasswordArray[0];

    //Second element of the array is the password
    string password = usernamePasswordArray[1];
}

```

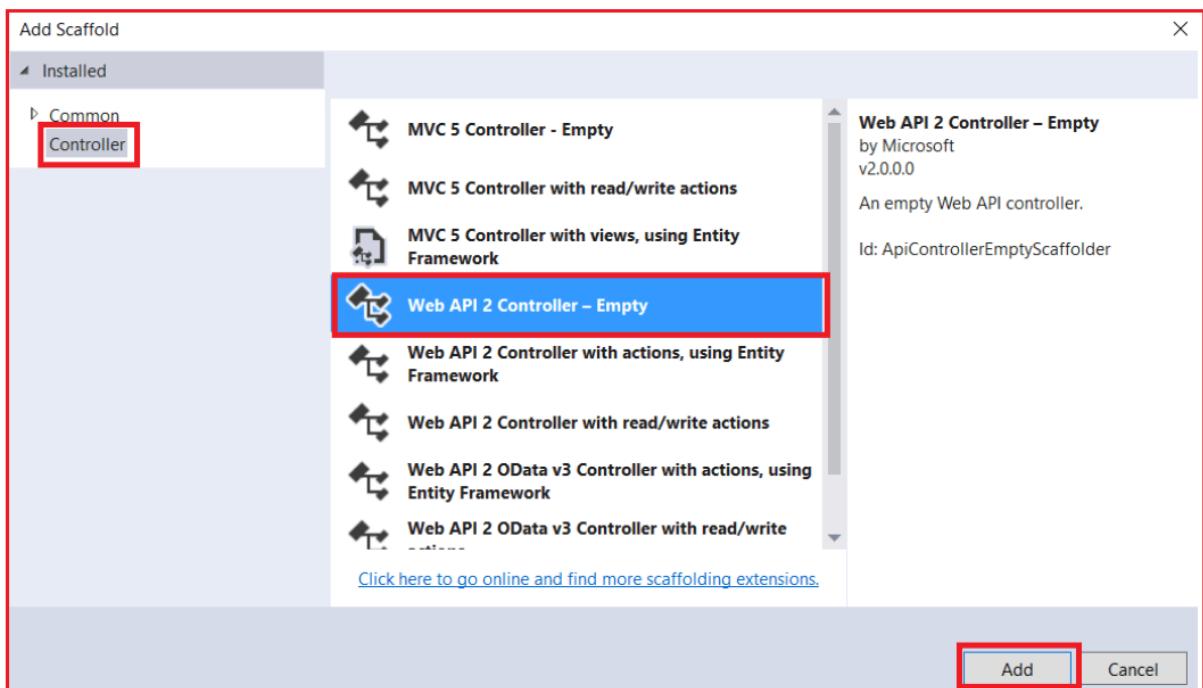
```
//call the login method to check the username and password
if (UserValidate.Login(username, password))
{
    var identity = new GenericIdentity(username);

    IPrincipal principal = new GenericPrincipal(identity, null);
    Thread.CurrentPrincipal = principal;

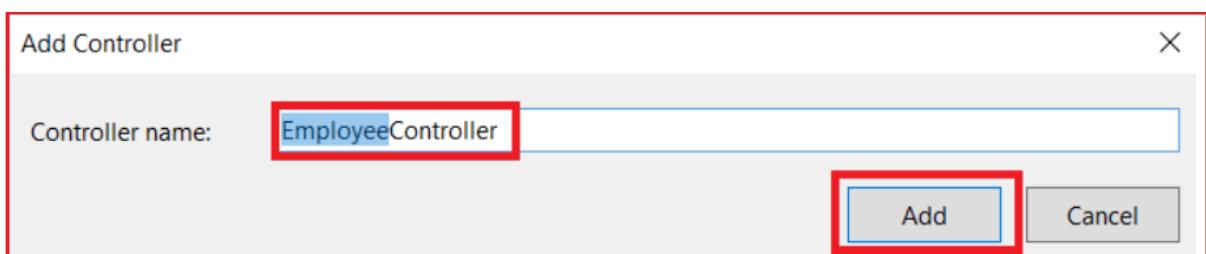
    if (HttpContext.Current != null)
    {
        HttpContext.Current.User = principal;
    }
}
else
{
    actionContext.Response = actionContext.Request
        .CreateResponse(HttpStatusCode.Unauthorized);
}
}
}
}
}
}
```

- **Thêm bộ điều khiển trong WebAPI2**

Nhấp chuột phải vào thư mục Controllers và chọn Add => Controller sẽ mở cửa sổ chọn bộ điều khiển như hình dưới đây.



Từ cửa sổ này, hãy chọn **Web API 2 Controller – Empty** và nhấp vào Nút Thêm **Add**, sẽ mở ra một cửa sổ khác để đặt tên cho bộ điều khiển của bạn như hình dưới đây



Cung cấp tên bộ điều khiển là Nhân viên và nhấp vào nút Thêm sẽ thêm EmployeeController trong thư mục bộ điều khiển.

Bật xác thực cơ bản Web API

Ta có thể kích hoạt xác thực cơ bản theo nhiều cách khác nhau bằng cách áp dụng `BasicAuthenticationAttribute`. Ta có thể áp dụng thuộc tính `BasicAuthenticationAttribute` trên bộ điều khiển cụ thể, phương thức cụ thể hoặc toàn cầu trên tất cả bộ điều khiển Web API.

Để kích hoạt xác thực cơ bản trên toàn bộ ứng dụng Web API, hãy đăng ký `BasicAuthenticationAttribute` làm bộ lọc bằng cách sử dụng phương thức `Register()` trong lớp `WebApiConfig` như thể hiện trong hình dưới đây.

```

public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        //To enable Basic Authentication for entire web application
        config.Filters.Add(new BasicAuthenticationAttribute());
    }

    // Web API routes
    config.MapHttpAttributeRoutes();

    config.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}/{id}",
        defaults: new { id = RouteParameter.Optional }
    );
}

```

Ta cũng có thể áp dụng thuộc tính BasicAuthenticationAttribute trên một bộ điều khiển cụ thể, điều này sẽ cho phép xác thực cơ bản cho tất cả các phương thức có trong bộ điều khiển đó như được hiển thị trong hình ảnh bên dưới.

```

[BasicAuthentication]
public class EmployeeController : ApiController
{
    public HttpResponseMessage GetEmployees() ...
}

```

Ta có thể kích hoạt basic authentication bằng cách khai báo tại action method (chỉ áp dụng cho particular action method which is decorated with the BasicAuthenticationAttribute).

Giả sử, ta có hai người dùng là MaleUser và FemaleUser và nếu người dùng đăng nhập bằng tên người dùng MaleUser, ta hiển thị tất cả nhân viên "nam" và nếu người dùng đăng nhập bằng tên người dùng FemaleUser, ta hiển thị tất cả nhân viên nữ.

Thêm phương thức thực thi sau trong bộ điều khiển Nhân viên **EmployeeController** như sau

EmployeeController.cs

namespace BasicAuthenticationWEBAPI.Controllers

```

{
    public class EmployeeController : ApiController
    {
        [BasicAuthentication]
        public HttpResponseMessage GetEmployees()
        {

```

```

string username = Thread.CurrentPrincipal.Identity.Name;

var EmpList = new EmployeeBL().GetEmployees();

switch (username.ToLower())
{
    case "maleuser":
        return Request.CreateResponse(HttpStatusCode.OK,
            EmpList.Where(e => e.Gender.ToLower() == "male").ToList());
    case "femaleuser":
        return Request.CreateResponse(HttpStatusCode.OK,
            EmpList.Where(e => e.Gender.ToLower() == "female").ToList());
    default:
        return Request.CreateResponse(HttpStatusCode.BadRequest);
}
}
}
}
}

```

7.2.5 Kiểm tra xác thực cơ bản Web API bằng Postman

Trước tiên, hãy thực hiện yêu cầu mà không cần chuyển tiêu đề ủy quyền authorization header. Đặt loại phương thức là **GET**, cung cấp URI yêu cầu và nhấp vào nút **Send** như thể hiện trong hình dưới đây.

The screenshot shows the Postman interface with a GET request to `http://localhost:63228/api/Employee`. The **Headers** tab is selected, displaying the following headers:

Key	Value	Description
New key	Value	Description

At the bottom right of the interface, the status bar displays `Status: 401 Unauthorized`.

Ở đây bạn có thể quan sát rằng bạn sẽ nhận được mã trạng thái *401* là Không được phép. Hãy thực hiện yêu cầu sử dụng tiêu đề Ủy quyền. Tên người dùng và mật khẩu phải được phân tách bằng dấu hai chấm (`:`) và phải được mã hóa *base64*. Để làm như vậy, chỉ cần sử dụng trang web sau

<https://www.base64encode.org/>

Nhập tên người dùng và mật khẩu được phân tách bằng dấu hai chấm (:) trong hộp văn bản “Encode to Base64 format”, sau đó nhấp vào nút Mã hóa “Encode” như được hiển thị trong sơ đồ dưới đây sẽ tạo ra giá trị được mã hóa Base64.

Encode to Base64 format

Simply use the form below

The screenshot shows the "Encode to Base64 format" page. A text input field contains "MaleUser:123456". Below it, a red box highlights the instruction "User Name and Password separated by colon". Underneath the input field, there are several configuration options: "UTF-8" (selected) for Destination charset, "LF (Unix)" for Newline separator, and a checkbox for Split lines into 76 character wide chunks. A radio button for "Live mode OFF" is selected, with a note that it encodes in real-time when typed or pasted. A green "ENCODE" button is highlighted with a red box. To the right, a text area displays the encoded string "TWFsZVVzZXI6MTIzMjU2", which is also highlighted with a red box. Next to it is the text "Base64 encoded string for the above username & password".

Sau khi bạn tạo chuỗi được mã hóa Base64, hãy xem cách sử dụng xác thực cơ bản trong tiêu đề để chuyển giá trị được mã hóa Base64. Ở đây chúng ta cần sử dụng tiêu đề `Authorization` và giá trị sẽ là chuỗi được mã hóa Base64 sau là “BASIC” như hình dưới đây.

Authorization: BASIC TWFsZVVzZXI6MTIzMjU2

The screenshot shows a Postman request configuration. The method is set to "GET" and the URL is "http://localhost:63228/api/Employee". The "Headers" tab is selected, showing one header named "Authorization" with the value "Basic TWFsZVVzZXI6MTIzMjU2". The "Send" button is highlighted with a red box. At the bottom, the status bar shows "Status: 200 OK" and "Time: 2258 ms".

Khi bạn nhấp vào nút Gửi, bạn có thể thấy rằng mã trạng thái là 200 như mong đợi.

7.3 Xác thực cơ bản dựa trên vai trò trong Web API

Trong phần này, chúng ta sẽ thảo luận về cách triển khai Xác thực Cơ bản Dựa trên Vai trò trong Ứng dụng Web API.

7.3.1 Tại sao chúng ta cần Xác thực Dựa trên Vai trò

Hãy để ta hiểu điều này với một ví dụ. Như được hiển thị trong hình ảnh dưới đây, ta có ba tài nguyên, tức là GetAllMaleEmployees, GetAllFemaleEmployees và GetAllEmployees trong dịch vụ của chúng tôi.

```
[Route("api/AllMaleEmployees")]
public HttpResponseMessage GetAllMaleEmployees()...
```



```
[Route("api/AllFemaleEmployees")]
public HttpResponseMessage GetAllFemaleEmployees()...
```



```
[Route("api/AllEmployees")]
public HttpResponseMessage GetAllEmployees()...
```

Trong ứng dụng của chúng tôi, ta có hai loại Vai trò, tức là Quản trị viên và Superadmin. Theo yêu cầu kinh doanh của chúng tôi,

1. Chỉ những người dùng có Vai trò Quản trị viên mới có thể truy cập duy nhất vào tài nguyên GetAllMaleEmployees.
2. Người dùng có Superadmin Vai trò chỉ có thể truy cập vào tài nguyên GetAllFemaleEmployees.
3. Cả tài nguyên Quản trị viên và Superadmin đều có thể truy cập tài nguyên GetAllEmployees.

Để đạt được điều này, chúng ta cần triển khai Xác thực dựa trên vai trò.

7.3.2 Triển khai xác thực cơ bản dựa trên vai trò trong Web API

Đầu tiên, tạo một ứng dụng Web API trống với tên RoleBasedBasicAuthenticationWEBAPI. Thêm mô hình User and Employee sau vào thư mục Models.

User.cs

```
namespace RoleBasedBasicAuthenticationWEBAPI.Models
{
    public class User
    {
        public int ID { get; set; }
        public string UserName { get; set; }
        public string Password { get; set; }
        public string Roles { get; set; }
        public string Email { get; set; }
    }
}
```

Employee.cs

```
namespace RoleBasedBasicAuthenticationWEBAPI.Models
{
    public class Employee
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public string Gender { get; set; }
        public string Dept { get; set; }
        public int Salary { get; set; }
    }
}
```

Bây giờ chúng ta cần thêm tệp lớp UserBL và EmployeeBL trong thư mục Models.

UserBL.cs

```
namespace RoleBasedBasicAuthenticationWEBAPI.Models
{
    public class UsersBL
    {
        public List<User> GetUsers()
        {
            // In Realtime you need to get the data from any persistent storage
            // For Simplicity of this demo and to keep focus on Basic Authentication
            // Here we are hardcoded the data
            List<User> userList = new List<User>();
            userList.Add(new User()
            {
                ID = 101,
                UserName = "AdminUser",
                Password = "123456",
                Roles = "Admin",
                Email = "Admin@a.com"
            });
        }
    }
}
```

```

userList.Add(new User()
{
    ID = 102,
    UserName = "BothUser",
    Password = "abcdef",
    Roles = "Admin,Superadmin",
    Email = "BothUser@a.com"
});

userList.Add(new User()
{
    ID = 103,
    UserName = "SuperadminUser",
    Password = "Password@123",
    Roles = "Superadmin",
    Email = "Superadmin@a.com"
});

return userList;
}
}
}

```

EmployeeBL.cs

```

namespace RoleBasedBasicAuthenticationWEBAPI.Models
{
    public class EmployeeBL
    {
        public List<Employee> GetEmployees()
        {
            // In Realtime you need to get the data from any persistent storage
            // For Simplicity of this demo and to keep focus on Basic Authentication
            // Here we hardcoded the data
            List<Employee> empList = new List<Employee>();
            for (int i = 0; i < 10; i++)

```

```

{
    if (i > 5)
    {
        empList.Add(new Employee()
        {
            ID = i,
            Name = "Name" + i,
            Dept = "IT",
            Salary = 1000 + i,
            Gender = "Male"
        });
    }
    else
    {
        empList.Add(new Employee()
        {
            ID = i,
            Name = "Name" + i,
            Dept = "HR",
            Salary = 1000 + i,
            Gender = "Female"
        });
    }
}
return empList;
}
}
}

```

Bây giờ, hãy thêm một lớp nữa với tên UserValidate và sao chép và dán đoạn mã sau.

UserValidate.cs

```

namespace RoleBasedBasicAuthenticationWEBAPI.Models
{
    public class UserValidate

```

```

{
    //This method is used to check the user credentials
    public static bool Login(string username, string password)
    {
        UsersBL userBL = new UsersBL();
        var UserLists = userBL.GetUsers();
        return UserLists.Any(user =>
            user.UserName.Equals(username, StringComparison.OrdinalIgnoreCase)
            && user.Password == password);
    }

    //This method is used to return the User Details
    public static User GetUserDetails(string username, string password)
    {
        UsersBL userBL = new UsersBL();
        return userBL.GetUsers().FirstOrDefault(user =>
            user.UserName.Equals(username, StringComparison.OrdinalIgnoreCase)
            && user.Password == password);
    }
}

```

Bây giờ tạo BasicAuthenticationAttribute sẽ triển khai AuthorizationFilterAttribute nơi chúng ta sẽ đặt logic cho xác thực cơ bản dựa trên vai trò.

BasicAuthenticationAttribute.cs

```

using System;
using System.Net;
using System.Net.Http;
using System.Security.Claims;
using System.Security.Principal;
using System.Text;
using System.Threading;
using System.Web;
using System.Web.Http.Controllers;

```

```

using System.Web.Http.Filters;

namespace RoleBasedBasicAuthenticationWEBAPI.Models
{
    public class BasicAuthenticationAttribute : AuthorizationFilterAttribute
    {
        private const string Realm = "My Realm";
        public override void OnAuthorization(HttpActionContext actionContext)
        {
            if (actionContext.Request.Headers.Authorization == null)
            {
                actionContext.Response =
                    actionContext.Request.CreateResponse(HttpStatusCode.Unauthorized);
            }
            if (actionContext.Response.StatusCode == HttpStatusCode.Unauthorized)
            {
                actionContext.Response.Headers.Add("WWW-Authenticate", string.Format("Basic realm=\"{0}\"", Realm));
            }
        }
        else
        {
            string authenticationToken = actionContext.Request.Headers.Authorization.Parameter;
            string decodedAuthenticationToken =
                Encoding.UTF8.GetString(Convert.FromBase64String(authenticationToken));
            string[] usernamePasswordArray = decodedAuthenticationToken.Split(':');
            string username = usernamePasswordArray[0];
            string password = usernamePasswordArray[1];
            if (UserValidate.Login(username, password))
            {
                var UserDetails = UserValidate.GetUserDetails(username, password);
                var identity = new GenericIdentity(username);
                identity.AddClaim(new Claim("Email", UserDetails.Email));
                identity.AddClaim(new Claim(ClaimTypes.Name, UserDetails.UserName));
                identity.AddClaim(new Claim("ID", Convert.ToString(UserDetails.ID)));
            }
        }
    }
}

```

```
IPrincipal principal = new GenericPrincipal(identity, UserDetails.Roles.Split(','));

Thread.CurrentPrincipal = principal;

if (HttpContext.Current != null)

{

    HttpContext.Current.User = principal;

}

else

{

    actionContext.Response = actionContext.Request

        .CreateResponse(HttpStatusCode.Unauthorized);

}

}

}

}

}
```

Bây giờ chúng ta sẽ tạo Thuộc tính ủy quyền tùy chỉnh sẽ kế thừa từ AuthorizeAttribute, nơi ta sẽ triển khai logic để trả về phản hồi thích hợp khi Cấp phép không thành công.

MyAuthorizeAttribute.cs

```
namespace RoleBasedBasicAuthenticationWEBAPI.Models
{
    public class MyAuthorizeAttribute : System.Web.Http.AuthorizeAttribute
    {
        // 401 (Unauthorized) - indicates that the request has not been applied because it lacks valid
        // authentication credentials for the target resource.
        // 403 (Forbidden) - when the user is authenticated but isn't authorized to perform the
        // requested
        // operation on the given resource.

        protected override void
HandleUnauthorizedRequest(System.Web.Http.Controllers.HttpActionContext actionContext)
{
    if (!HttpContext.Current.User.Identity.IsAuthenticated)
    {
        base.HandleUnauthorizedRequest(actionContext);
    }
}
```

```

    }
    else
    {
        actionContext.Response = new
System.Net.Http.HttpResponseMessage(System.Net.HttpStatusCode.Forbidden);
    }
}
}
}
}

```

Hãy tạo Bộ điều khiển trống cho Web API 2 với tên EmployeeController và sao chép và dán mã sau.

EmployeeController.cs

```

using RoleBasedBasicAuthenticationWEBAPI.Models;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Security.Claims;
using System.Web.Http;

namespace RoleBasedBasicAuthenticationWEBAPI.Controllers
{
    public class EmployeeController : ApiController
    {
        [BasicAuthentication]
        [MyAuthorize(Roles = "Admin")]
        [Route("api/AllMaleEmployees")]
        public HttpResponseMessage GetAllMaleEmployees()
        {
            var identity = (ClaimsIdentity)User.Identity;

            //Getting the ID value
            var ID = identity.Claims
                .FirstOrDefault(c => c.Type == "ID").Value;

```

```

//Getting the Email value
var Email = identity.Claims
    .FirstOrDefault(c => c.Type == "Email").Value;

//Getting the Username value
var username = identity.Name;

//Getting the Roles only if you set the roles in the claims
//var Roles = identity.Claims
//    .Where(c => c.Type == ClaimTypes.Role)
//    .Select(c => c.Value).ToArray();

var EmpList = new EmployeeBL().GetEmployees().Where(e => e.Gender.ToLower() ==
"male").ToList();

return Request.CreateResponse(HttpStatusCode.OK, EmpList);
}

[BasicAuthentication]
[MyAuthorize(Roles = "Superadmin")]
[Route("api/AllFemaleEmployees")]
public HttpResponseMessage GetAllFemaleEmployees()
{
    var EmpList = new EmployeeBL().GetEmployees().Where(e => e.Gender.ToLower() ==
"female").ToList();
    return Request.CreateResponse(HttpStatusCode.OK, EmpList);
}

[BasicAuthentication]
[MyAuthorize(Roles = "Admin,Superadmin")]
[Route("api/AllEmployees")]
public HttpResponseMessage GetAllEmployees()
{

```

```

        var EmpList = new EmployeeBL().GetEmployees();
        return Request.CreateResponse(HttpStatusCode.OK, EmpList);
    }
}
}

```

Đó là nó. Ta đã thực hiện với việc triển khai của chúng tôi.

7.3.3 Kiểm tra Xác thực cơ bản dựa trên vai trò

Ta cần chuyển tên người dùng và mật khẩu trong tiêu đề Ủy quyền. Tên người dùng và mật khẩu phải được phân tách bằng dấu hai chấm (:) và phải được mã hóa base64. Để làm như vậy, chỉ cần sử dụng trang web sau

<https://www.base64encode.org/>

Nhập tên người dùng và mật khẩu được phân tách bằng dấu hai chấm (:) trong hộp văn bản “Mã hóa thành định dạng Base64”, sau đó nhấp vào nút “Mã hóa” như được hiển thị trong sơ đồ dưới đây sẽ tạo ra giá trị được mã hóa Base64. Đầu tiên hãy tạo chuỗi được mã hóa Base64 cho người dùng AdminUser như thể hiện trong hình ảnh bên dưới

Encode to Base64 format
Simply use the form below

UserName and Password separated by colon

To encode binaries (like images, documents, etc.) upload your data via the [file encode form](#) below.

UTF-8

LF (Unix)

Split lines into 76 character wide chunks (useful for MIME).

Live mode OFF

> ENCODE <

Destination charset.
 Newline separator.
 Encodes in real-time when you type or paste (supports only unicode charsets).
 Encodes your data into the textarea below.

QWRtaW5Vc2VyOjEyMzQ1Ng==

Base64 encoded string for the above username and password

Sau khi bạn tạo chuỗi được mã hóa Base64, hãy xem cách sử dụng xác thực cơ bản trong tiêu đề để chuyển giá trị được mã hóa Base64. Ở đây chúng ta cần sử dụng tiêu đề Ủy quyền và giá trị sẽ là chuỗi được mã hóa Base64 theo sau là “CƠ BẢN” như hình dưới đây.

Authorization: BASIC TWFsZVVzZXI6MTIxNDU2

Vai trò Quản trị viên đã được chỉ định cho Người dùng quản trị. Vì vậy, anh ta chỉ có thể truy cập hai tài nguyên sau

/api/AllMaleEmployees

/api/AllEmployees

Nhưng anh ta không thể truy cập tài nguyên sau

/api/AllFemaleEmployees

- Để chứng minh điều này bằng Postman.

/api/AllMaleEmployees

The screenshot shows the Postman interface with a successful API call. The request method is GET, and the URL is http://localhost:63228/api/AllMaleEmployees. The Authorization header is set to 'Basic QWRtaW5vc2VyOjEyMzQ1Ng=='. The response status is 200 OK, and the time taken is 358 ms.

Ở đây ta nhận được phản hồi 200 OK.

/api/AllEmployees

The screenshot shows the Postman interface with a successful API call. The request method is GET, and the URL is http://localhost:63228/api/AllEmployees. The Authorization header is set to 'Basic QWRtaW5vc2VyOjEyMzQ1Ng=='. The response status is 200 OK, and the time taken is 73 ms.

Ở đây ta cũng nhận được phản hồi 200 OK như mong đợi.

/api/AllFemaleEmployees

The screenshot shows the Postman interface with a failed API call. The request method is GET, and the URL is http://localhost:63228/api/AllFemaleEmployees. The Authorization header is set to 'Basic QWRtaW5vc2VyOjEyMzQ1Ng=='. The response status is 403 Forbidden, and the time taken is 36 ms.

Như bạn có thể thấy, ở đây ta nhận được phản hồi là 403 Forbidden, có nghĩa là người dùng được xác thực nhưng không được phép truy cập tài nguyên trên. Tương tự, bạn có thể kiểm tra những người dùng khác.

7.4 Xác thực dựa trên mã thông báo trong Web API

Trong phần này, chúng ta sẽ thảo luận về cách triển khai Xác thực dựa trên mã thông báo trong Web API để bảo mật tài nguyên máy chủ với một ví dụ.

7.4.1 Tại sao chúng ta cần Xác thực dựa trên mã thông báo trong Web API?

ASP.NET Web API là một khuôn khổ lý tưởng, do Microsoft cung cấp, để xây dựng các dịch vụ dựa trên Web API, tức là dựa trên HTTP trên .NET Framework. Khi ta phát triển các dịch vụ sử dụng Web API thì các dịch vụ này sẽ được nhiều khách hàng sử dụng, chẳng hạn như

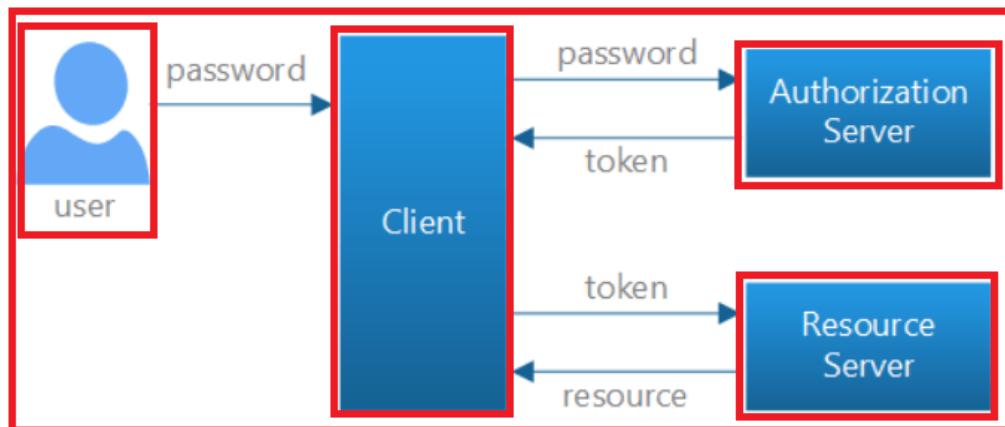
1. Trình duyệt
2. Ứng dụng di động
3. Ứng dụng máy tính để bàn
4. IoT, v.v.

Ngày nay, việc sử dụng WEB API đang gia tăng một cách nhanh chóng. Vì vậy, là một nhà phát triển, bạn nên biết cách phát triển các Web API. Chỉ phát triển các Web API là không đủ nếu không có bảo mật. Vì vậy, với tư cách là nhà phát triển, ta cũng rất cần triển khai bảo mật cho tất cả các loại khách hàng (chẳng hạn như Trình duyệt, Thiết bị di động, ứng dụng Máy tính để bàn và IoT) sẽ sử dụng dịch vụ Web API của chúng tôi.

Phương pháp được ưa thích nhất hiện nay để bảo mật tài nguyên Web API là xác thực người dùng trong máy chủ Web API bằng cách sử dụng mã thông báo đã ký (chứa đủ thông tin để xác định một người dùng cụ thể) cần được khách hàng gửi đến máy chủ với mỗi và mọi yêu cầu. Đây được gọi là phương pháp xác thực dựa trên mã thông báo.

7.4.2 Xác thực dựa trên mã thông báo hoạt động như thế nào?

Để hiểu cách xác thực dựa trên mã thông báo hoạt động, vui lòng xem sơ đồ sau.



1. Xác thực dựa trên mã thông báo hoạt động như sau: The user enters his credentials (i.e. the username and password) into the client (here client means the browser or mobile devices, etc).
2. Sau đó, khách hàng sẽ gửi các thông tin đăng nhập này (tức là tên người dùng và mật khẩu) đến Máy chủ Cấp quyền.
3. Sau đó, Máy chủ Ủy quyền xác thực thông tin xác thực của khách hàng (tức là tên người dùng và mật khẩu) và tạo và trả về mã thông báo truy cập. Mã thông báo truy cập này chứa đủ thông tin để xác định người dùng và cũng chứa thời gian hết hạn mã thông báo.
4. Ứng dụng khách sau đó bao gồm Mã thông báo truy cập trong tiêu đề Cấp quyền của Yêu cầu HTTP request để truy cập các tài nguyên bị hạn chế từ Máy chủ tài nguyên cho đến khi mã thông báo hết hạn.

7.4.3 Triển khai xác thực dựa trên mã thông báo trong Web API

Chúng ta hãy thảo luận về quy trình từng bước để triển khai Xác thực dựa trên mã thông báo trong Web API và sau đó chúng ta cũng sẽ hướng dẫn cách sử dụng xác thực dựa trên mã thông báo để truy cập các tài nguyên bị hạn chế bằng Postman và Fiddler.

Bước 1: Tạo cơ sở dữ liệu cần thiết

Ta sẽ sử dụng bảng UserMaster sau trong bản trình diễn này.

Ta sẽ sử dụng bảng UserMaster sau trong bản trình diễn này.

UserID	UserName	UserPassword	UserRoles	UserEmailID
101	Anurag	123456	Admin	Anurag@g.com
102	Priyanka	abcdef	User	Priyanka@g.com
103	Sambit	123pqr	SuperAdmin	Sambit@g.com
104	Pranaya	abc123	Admin, User	Pranaya@g.com

Vui lòng sử dụng SQL Script bên dưới để tạo và điền vào bảng UserMaster với dữ liệu mẫu bắt buộc.

Script.sql

```
CREATE DATABASE SECURITY_DB
```

```
GO
```

```
USE [SECURITY_DB]
```

```
CREATE TABLE UserMaster
```

```
(
```

```
    UserID INT PRIMARY KEY,
```

```
    UserName VARCHAR(50),
```

```
    UserPassword VARCHAR(50),
```

```
    UserRoles VARCHAR(500),
```

```
    UserEmailID VARCHAR(100),
```

```
)
```

```
GO
```

```
INSERT INTO UserMaster VALUES(101, 'Anurag', '123456', 'Admin', 'Anurag@g.com')
```

```
INSERT INTO UserMaster VALUES(102, 'Priyanka', 'abcdef', 'User', 'Priyanka@g.com')
```

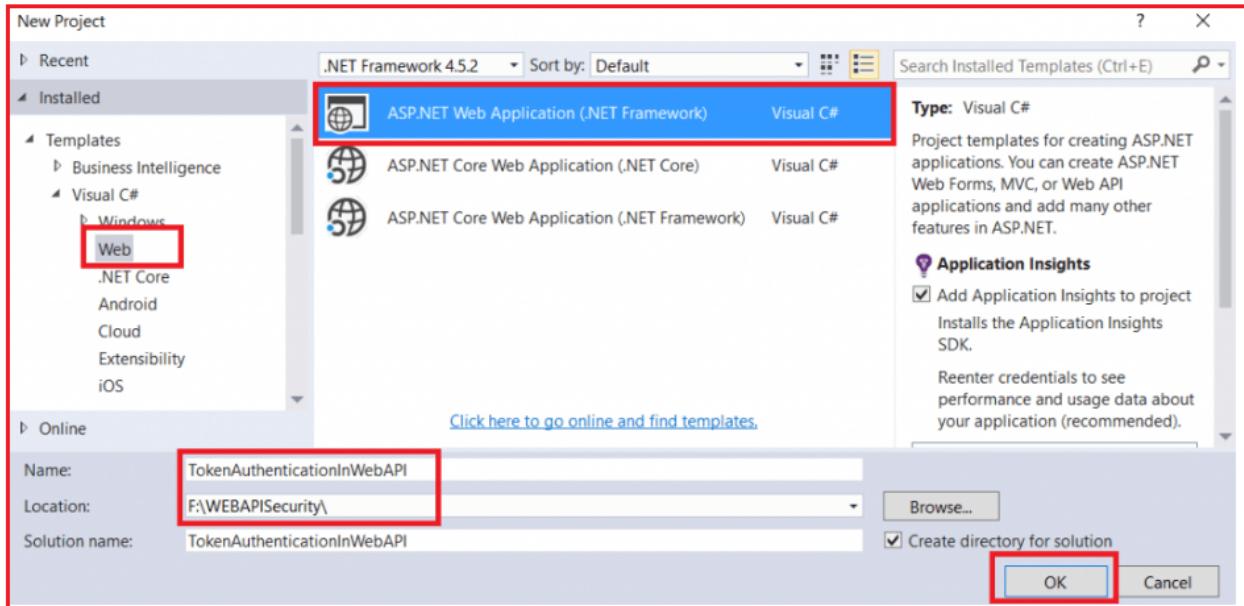
```
INSERT INTO UserMaster VALUES(103, 'Sambit', '123pqr', 'SuperAdmin', 'Sambit@g.com')
```

```
INSERT INTO UserMaster VALUES(104, 'Pranaya', 'abc123', 'Admin, User', 'Pranaya@g.com')
```

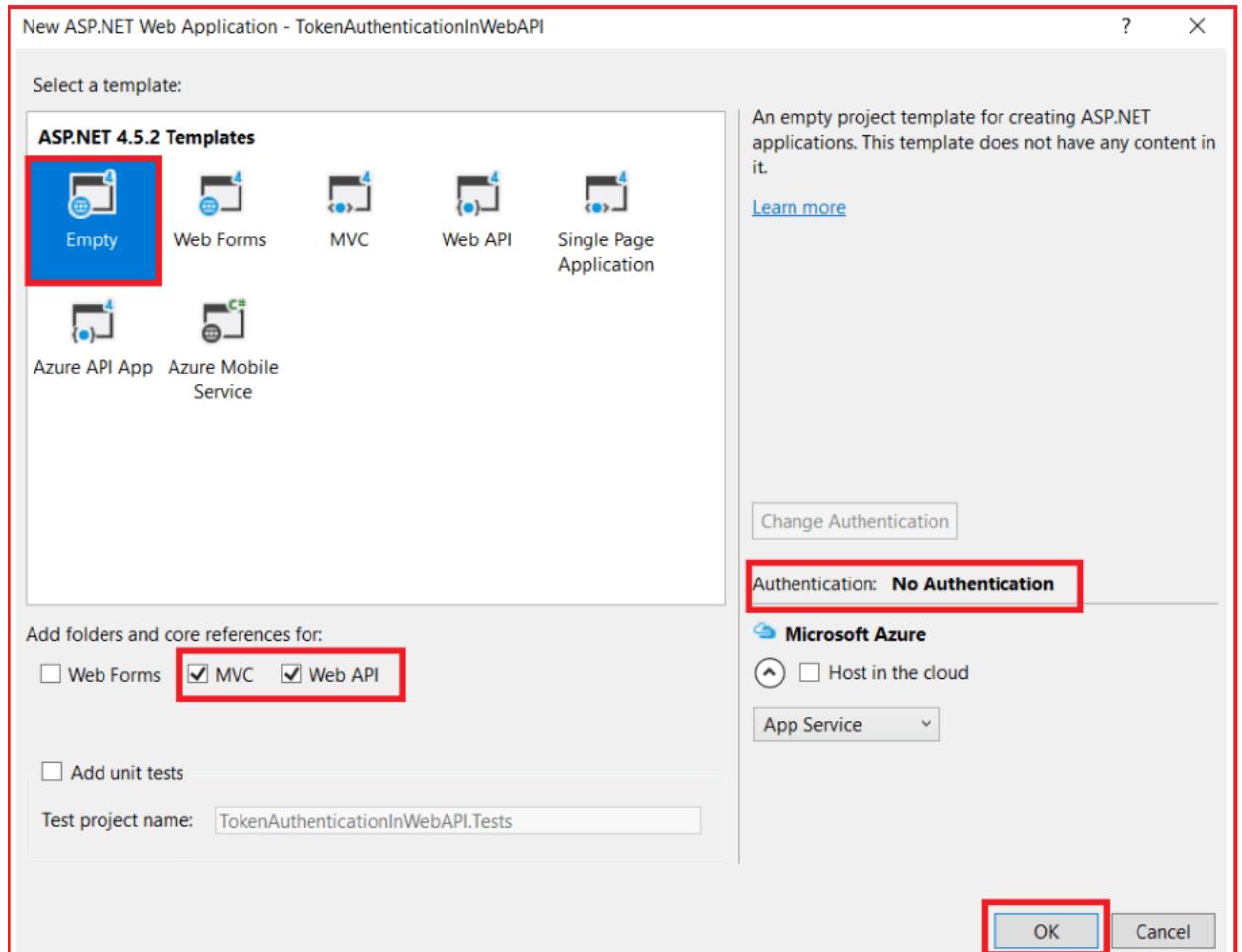
```
GO
```

Bước 2: Tạo một Dự án Web API trống với tên TokenAuthenticationWEBAPI

Đi đến **File menu > create > project** > tại đây chọn “**asp.net web application**” dưới web. Cung cấp tên ứng dụng là **TokenAuthenticationWEBAPI** và chọn vị trí dự án mà bạn muốn tạo dự án. Sau đó bấm vào nút **OK** như trong hình dưới đây.



Khi bạn nhập vào **OK**, sau đó một cửa sổ mới sẽ mở ra với Tên **New ASP.NET Web Application** để chọn **Project Templates** và từ cửa sổ này, bạn cần chọn **Empty** mẫu dự án vì ta sẽ làm mọi thứ từ đầu và sau đó kiểm tra MVC và Web API hộp kiểm từ **Add folder and core references for** và sau đó nhấp vào nút **OK** như hiển thị trong hình dưới đây.



Bước 3: Thêm các tham chiếu bắt buộc từ các gói NuGet vào ứng dụng của bạn.

Để triển khai xác thực dựa trên mã thông báo trong Web API ASP.NET, ta cần cài đặt các tham chiếu sau từ các gói NuGet. Phần sau của phần này, chúng ta sẽ thảo luận về việc sử dụng từng gói bên dưới.

Microsoft.Owin.Host.SystemWeb

Microsoft.Owin.Security.OAuth

Microsoft.Owin.Cors

Newtonsoft.json

Để thêm các tài liệu tham khảo trên từ NuGet, rồi tới Solution Explorer > Right Click trên References > click vào Manage NuGet Packages > Search cho Microsoft.Owin.Host.SystemWeb, Microsoft.Owin.Security.OAuth, Microsoft.Owin.Cors và Newtonsoft.json và cài đặt.

Lưu ý: Khi bạn cài đặt các gói trên, các tham chiếu phụ cũng được tự động cài đặt vào ứng dụng của bạn.

Bước 4: Tạo mô hình dữ liệu thực thể ADO.NET Entity Data Model

Ở đây, ta sẽ sử dụng Phương pháp tiếp cận đầu tiên của Khung thực thể DB để tạo Mô hình dữ liệu thực thể dựa trên cơ sở dữ liệu SECURITY_DB mà ta đã tạo và sau đó chọn bảng UserMaster từ cơ sở dữ liệu SECURITY_DB.

Bước 5: Tạo một lớp Kho lưu trữ Repository class

Bây giờ, bạn cần tạo một lớp với tên **UserMasterRepository** sẽ xác thực người dùng và cũng trả về thông tin người dùng. Như bạn có thể thấy trong đoạn mã dưới đây, phương thức ValidateUser lấy tên người dùng và mật khẩu làm tham số đầu vào và sau đó xác thực điều này. Nếu tên người dùng và mật khẩu hợp lệ thì nó sẽ trả về đối tượng UserMaster, nếu không nó sẽ trả về null. Sau đó chúng ta sẽ thảo luận khi nào và ở đâu chúng ta sẽ sử dụng phương pháp này.

UserMasterRepository.cs

```
namespace TokenAuthenticationInWebAPI.Models
{
    public class UserMasterRepository : IDisposable
    {
        // SECURITY_DBEntities it is your context class
        SECURITY_DBEntities context = new SECURITY_DBEntities();

        //This method is used to check and validate the user credentials
        public UserMaster ValidateUser(string username, string password)
        {
            return context.UserMasters.FirstOrDefault(user =>
                user.UserName.Equals(username, StringComparison.OrdinalIgnoreCase)
                && user.UserPassword == password);
        }

        public void Dispose()
        {
            context.Dispose();
        }
    }
}
```

Bước 6: Thêm một lớp để xác thực thông tin xác thực của người dùng yêu cầu mã thông báo.

Bây giờ chúng ta cần thêm một lớp có tên **MyAuthorizationServerProvider** vào ứng dụng của mình. Trong lớp đó, chúng ta cần viết logic để xác thực thông tin xác thực của người dùng và tạo mã thông báo truy cập.

Chúng ta cần kế thừa lớp **MyAuthorizationServerProvider** từ lớp **OAuthAuthorizationServerProvider** và sau đó cần ghi đè phương thức **ValidateClientAuthentication** và **GrantResourceOwnerCredentials**. Vì vậy, trước khi tiếp tục

và ghi đè hai phương pháp này, trước tiên chúng ta hãy hiểu chính xác những gì các phương pháp này sẽ thực hiện.

Phương thức xác thực **ValidateClientAuthentication**: Phương thức xác thực **ValidateClientAuthentication** được sử dụng để xác thực ứng dụng khách.

Phương thức **GrantResourceOwnerCredentials**: Phương thức **GrantResourceOwnerCredentials** được sử dụng để xác thực thông tin xác thực của khách hàng (tức là tên người dùng và mật khẩu). Nếu nó tìm thấy thông tin xác thực hợp lệ, thì chỉ nó tạo mã thông báo truy cập. Sau đó, khách hàng sử dụng mã thông báo truy cập này có thể truy cập các tài nguyên được ủy quyền từ Máy chủ Tài nguyên.

Như chúng ta đã thảo luận, mã thông báo truy cập đã ký chứa đủ thông tin để xác định người dùng. Bây giờ câu hỏi là làm thế nào. Hãy thảo luận chi tiết về điều này.

Đầu tiên, chúng ta cần tạo một thẻ hiện của lớp **ClaimsIdentity** và đối với phương thức khởi tạo của lớp **ClaimsIdentity**, chúng ta cần chuyển kiểu xác thực. Vì chúng ta sẽ sử dụng Xác thực dựa trên mã thông báo, vì vậy Loại xác thực là "mã thông báo mang".

Sau khi ta tạo cá thẻ **ClaimsIdentity**, sau đó cần thêm các xác nhận quyền sở hữu như Vai trò, Tên và Email, v.v. vào cá thẻ **ClaimsIdentity**. Đây là thông tin người dùng sẽ được bao gồm trong mã thông báo truy cập đã ký. Bạn có thể thêm bất kỳ số lượng xác nhận quyền sở hữu nào và khi bạn thêm nhiều xác nhận quyền sở hữu hơn, kích thước mã thông báo sẽ tăng lên.

MyAuthorizationServerProvider

Tạo một tệp lớp với tên **MyAuthorizationServerProvider.cs**, sau đó sao chép và dán đoạn mã sau vào đó.

MyAuthorizationServerProvider.cs

```
using Microsoft.Owin.Security.OAuth;
using System.Security.Claims;
using System.Threading.Tasks;

namespace TokenAuthenticationInWebAPI.Models
{
    public class MyAuthorizationServerProvider : OAuthAuthorizationServerProvider
    {
        public override async Task
ValidateClientAuthentication(OAuthValidateClientAuthenticationContext context)
        {
            context.Validated();
        }

        public override async Task
```

```

GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)
{
    using (UserMasterRepository _repo = new UserMasterRepository())
    {
        var user = _repo.ValidateUser(context.UserName, context.Password);
        if (user == null)
        {
            context.SetError("invalid_grant", "Provided username and password is incorrect");
            return;
        }

        var identity = new ClaimsIdentity(context.Options.AuthenticationType);
        identity.AddClaim(new Claim(ClaimTypes.Role, user.UserRoles));
        identity.AddClaim(new Claim(ClaimTypes.Name, user.UserName));
        identity.AddClaim(new Claim("Email", user.UserEmailID));

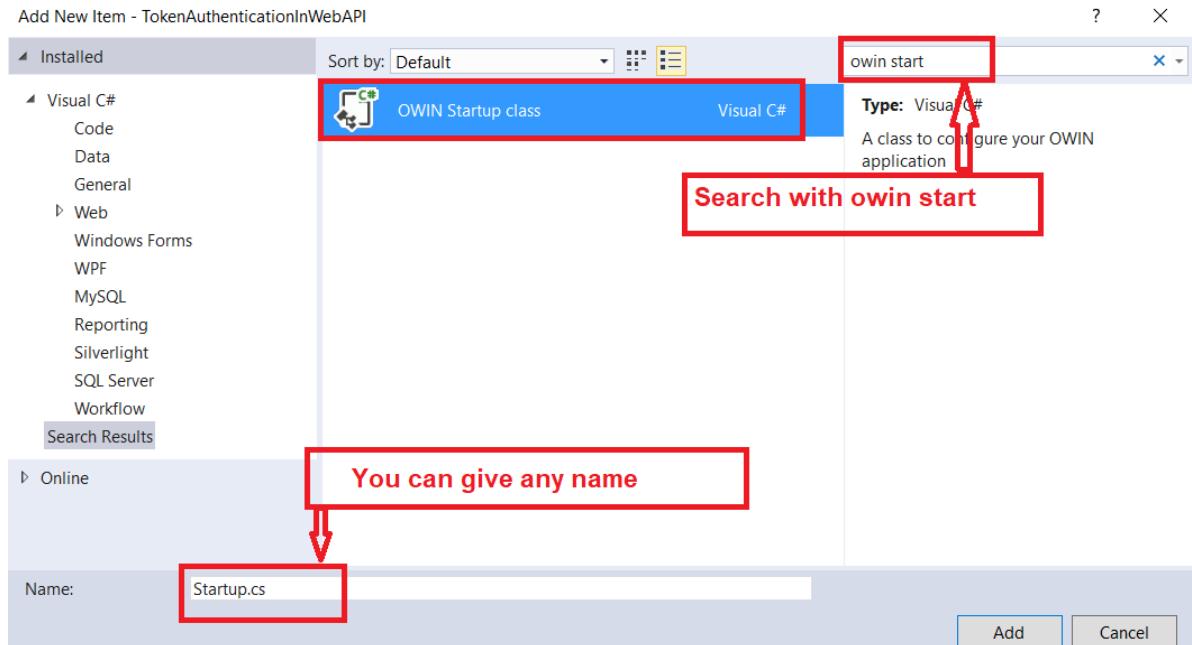
        context.Validated(identity);
    }
}
}
}
}

```

Bước 7: Thêm lớp OWINStartup.

Bây giờ chúng ta cần thêm lớp OWINStartup nơi chúng ta sẽ cấu hình Máy chủ ủy quyền OAuth. Đây sẽ là máy chủ ủy quyền của chúng tôi.

Để làm như vậy, hãy chuyển đến Solution Explorer > Right kích chọn Project Name từ Solution Explorer > Add > New Item > Select OWIN Startup class > Nhập tên lớp là Startup.cs > và sau đó nhấp vào Add button như thể hiện trong hình ảnh dưới đây.



Khi bạn đã tạo lớp Khởi động Owin, hãy sao chép và dán đoạn mã dưới đây vào đó.

Startup.cs

```

using System;
using Microsoft.Owin;
using Owin;
using TokenAuthenticationInWebAPI.Models;
using Microsoft.Owin.Security.OAuth;
using System.Web.Http;

[assembly: OwinStartup(typeof(TokenAuthenticationInWebAPI.App_Start.Startup))]

namespace TokenAuthenticationInWebAPI.App_Start
{
    // In this class we will Configure the OAuth Authorization Server.
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            // Enable CORS (cross origin resource sharing) for making request using browser from
            // different domains
            app.UseCors(Microsoft.Owin.Cors.CorsOptions.AllowAll);
        }
    }
}

```

```

OAuthAuthorizationServerOptions options = new OAuthAuthorizationServerOptions
{
    AllowInsecureHttp = true,
    //The Path For generating the Toekn
    TokenEndpointPath = new PathString("/token"),
    //Setting the Token Expired Time (24 hours)
    AccessTokenExpireTimeSpan = TimeSpan.FromDays(1),
    //MyAuthorizationServerProvider class will validate the user credentials
    Provider = new MyAuthorizationServerProvider()
};

//Token Generations
app.UseOAuthAuthorizationServer(options);
app.UseOAuthBearerAuthentication(new OAuthBearerAuthenticationOptions());
HttpConfiguration config = new HttpConfiguration();
WebApiConfig.Register(config);
}
}
}

```

Hiểu mã lớp Khởi động Owin:

Ở đây, ta đã tạo một phiên bản mới của lớp OAuthAuthorizationServerOptions và sau đó đặt các tùy chọn của nó như sau:

1. Ở đây, ta đặt đường dẫn để tạo mã thông báo là “http://localhost:portnumber/token”. Sau đó, chúng ta sẽ xem cách đưa ra Yêu cầu HTTP request Post để tạo mã thông báo truy cập.
2. Ta đã chỉ định thời gian hết hạn cho mã thông báo truy cập là 24 giờ. Vì vậy, nếu người dùng cố gắng sử dụng cùng một mã thông báo truy cập sau 24 giờ kể từ thời điểm phát hành, thì yêu cầu này sẽ bị từ chối và mã trạng thái HTTP 401 sẽ được trả lại.

3. Ta cũng chỉ định việc triển khai cách xác thực thông tin xác thực của khách hàng cho những người dùng yêu cầu mã thông báo truy cập trong lớp tùy chỉnh có tên MyAuthorizationServerProvider.

Cuối cùng, ta đã chuyển các tùy chọn cho phương thức mở rộng UseOAuthAuthorizationServer sẽ thêm phần mềm trung gian xác thực vào đường ống.

Bước 8: Thêm Bộ điều khiển Web API.

Bây giờ chúng ta cần tạo tài nguyên Web API. Để làm như vậy, hãy thêm Bộ điều khiển Web API trống, nơi ta sẽ thêm một số phương pháp phương thức để ta có thể kiểm tra Xác thực dựa trên mã thông báo có hoạt động tốt hay không.

Đi tới **Solution Explorer > Rồi kích chọn Controllers folder > Add > Controller > Chọn WEB API 2 Controller – Empty** > kích chọn **Add .** > **Enter** tên người điều khiển là TestController.cs > finally nhấp vào nút Thêm sẽ tạo TestController.

Khi bạn đã tạo TestController, hãy sao chép và dán đoạn mã sau.

TestController.cs

```
using System.Linq;
using System.Security.Claims;
using System.Web.Http;

namespace TokenAuthenticationInWebAPI.Controllers
{
    public class TestController : ApiController
    {
        //This resource is For all types of role
        [Authorize(Roles = "SuperAdmin, Admin, User")]
        [HttpGet]
        [Route("api/test/resource1")]
        public IHttpActionResult GetResource1()
        {
            var identity = (ClaimsIdentity)User.Identity;
            return Ok("Hello: " + identity.Name);
        }

        //This resource is only For Admin and SuperAdmin role
        [Authorize(Roles = "SuperAdmin, Admin")]
        [HttpGet]
```

```

[Route("api/test/resource2")]
public IHttpActionResult GetResource2()
{
    var identity = (ClaimsIdentity)User.Identity;
    var Email = identity.Claims
        .FirstOrDefault(c => c.Type == "Email").Value;

    var UserName = identity.Name;

    return Ok("Hello " + UserName + ", Your Email ID is :" + Email);
}

//This resource is only For SuperAdmin role
[Authorize(Roles = "SuperAdmin")]
[HttpGet]
[Route("api/test/resource3")]
public IHttpActionResult GetResource3()
{
    var identity = (ClaimsIdentity)User.Identity;
    var roles = identity.Claims
        .Where(c => c.Type == ClaimTypes.Role)
        .Select(c => c.Value);

    return Ok("Hello " + identity.Name + "Your Role(s) are: " + string.Join(",",
roles.ToList()));
}
}

```

Ở đây, trong bộ điều khiển ở trên, ta đã tạo ba tài nguyên như sau,

1./api/test/resource1 - Tài nguyên này có thể được truy cập bởi cả ba loại vai trò như Quản trị viên, SuperAdmin và Người dùng

2./api/test/resource2 - Người dùng có vai trò Admin và SuperAdmin có thể truy cập tài nguyên này

3./api/test/resource3 - Chỉ những người dùng có vai trò SuperAdmin mới có thể truy cập tài nguyên này

Để kiểm tra điều này, ta sẽ sử dụng một công cụ khách có tên là Postman. Trước tiên, bạn cần chạy ứng dụng Web API của mình. Nếu bạn chưa quen với Postman thì vui lòng đọc phần sau, nơi ta đã thảo luận về cách sử dụng Postman để kiểm tra các dịch vụ phần còn lại của Web API.

Bước 9: Kiểm tra xác thực mã thông báo

Test1: Không có Mã thông báo truy cập, hãy thử đưa ra yêu cầu theo dõi URI

Bạn cần thay đổi số cổng. Bạn phải cung cấp số cổng nơi ứng dụng Web API của bạn đang chạy.

The screenshot shows the Postman interface with a GET request to `http://localhost:65061/api/test/resource1`. The response status is `401 Unauthorized`. The JSON body contains the message: `"Message": "Authorization has been denied for this request."`

Đúng như dự đoán, bạn đã nhận được 401 câu trả lời trái phép

Kiểm tra 2: Có gắng tạo mã thông báo Access với thông tin đăng nhập không hợp lệ

Vì ta không có bất kỳ người dùng nào sử dụng thử nghiệm tên, vì vậy hãy thử tạo Mã truy cập cho người dùng thử nghiệm. Chọn loại phương thức là POST (1), nhập URL là `http://localhost:PortNumber/token` (2) và sau đó nhấp vào tab body (3), sau đó chọn `x-www-form-urlencoded` (4) và sau đó nhập 3 tham số (5)

1. tên người dùng (giá trị: thử nghiệm)
2. mật khẩu (giá trị: thử nghiệm)
3. Cấp_tỉnh (giá trị: mật khẩu)

Và sau đó nhấp vào nút Gửi (6).

The screenshot shows the Postman interface with a POST request to `http://localhost:65061/token`. The request type is POST (1), URL is `http://localhost:65061/token` (2), Body tab is selected (3), Content type is `x-www-form-urlencoded` (4). The body parameters are: `username: test`, `password: test`, `grant_type: password`. The response status is `400 Bad Request`. The JSON body shows the error: `"error": "invalid_grant", "error_description": "Provided username and password is incorrect"`.

Khi bạn nhấp vào nút gửi, bạn sẽ nhận được trạng thái là 400 Yêu cầu Xấu như mong đợi và nó cũng cho biết rằng trong mô tả lỗi rằng tên người dùng và mật khẩu được cung cấp không chính xác. Hãy tạo mã thông báo truy cập với thông tin xác thực hợp lệ cho người dùng Anurag có mật khẩu ta là 123456 như thể hiện trong hình ảnh bên dưới.

The screenshot shows the Postman interface with a red box highlighting the 'Request section' where a POST request is made to `http://localhost:65061/token`. The 'Body' tab is selected, showing form-data with fields `username`, `password`, and `grant_type` all set to their respective values. The 'Response Section' below shows a successful response with status `200 OK` and time `12008 ms`. The response body contains a JSON object with an access token, token type (bearer), and expiration information.

```

{
  "access_token": "wYGFs9epP8BsU_PnDmv1NN7cewyA0j9EpE-yctRo_dMc9F7hRDuSAE3Kwa0UkDsYm_6b-1_QzWAThs_P2oTpjmi0_581wf3ATfvF8fpG5ceC8qV1Hd2nSrMcQNo81HGtnfocNz3Fk8bxF9Aqv-2hbtI67ksNh7-KbKc12qlhA2azw167zPGo0luk5EDfOynd-0kls98dfqvQt0wbVC0p9DneJGB_Vxq1Shn21VVYkp76k_zms4QZTxXn9axTjS1bsFS08v3Tb4eKpI58nWTMkmYeemLgUIC_1RXB9j8MhcdgfDZR56uYFVSYzb",
  "token_type": "bearer",
  "expires_in": 86399
}

```

Như bạn có thể thấy khi bạn nhấp vào nút gửi, bạn sẽ nhận được mã trạng thái 200 Ok cùng với đó bạn sẽ nhận được mã thông báo truy cập chứa đủ thông tin để xác định người dùng Anurag. Bạn cũng có thể thấy trong phần Phản hồi rằng loại mã thông báo là Mang và thời gian mã thông báo hết hạn tính bằng giây.

Test3: Truy cập tài nguyên bị hạn chế bằng mã thông báo truy cập.

`/api/test/resource2`

Đầu tiên, hãy sao chép mã thông báo truy cập mà chúng ta vừa tạo trong ví dụ trước mà chúng ta sẽ sử dụng mã thông báo như được hiển thị bên dưới.

Ủy quyền: Bearer Access_Token (giá trị)

The screenshot shows the Postman interface with a red box highlighting the 'Request section' where a GET request is made to `http://localhost:65061/api/test/resource2`. The 'Headers' tab is selected, showing an `Authorization` header with the value `Bearer wYGFs9epP8BsU_PnDmv1NN7cewyA0j9EpE-yctRo_dMc9F...`. The 'Response Section' below shows a successful response with status `200 OK` and time `316 ms`. The response body contains the message `"Hello Anurag, Your Email ID is :Anurag@g.com"`.

Các bạn có thể thấy rằng, khi bấm vào nút Gửi, bạn sẽ nhận được 200 Ok như mong đợi vì tài nguyên/api/test/resource2 đã được truy cập bởi Roles Admin và SuperAdmin và ở đây người dùng Anurag có Role Admin nên chúng ta nhận được phản ứng trên.

Nhưng người dùng ở trên không thể truy cập tài nguyên/api/test/resource3 vì tài nguyên3 chỉ có thể được truy cập bởi người dùng có vai trò là SuperAdmin. Hãy chứng minh điều này.

The screenshot shows the Postman interface with a red box highlighting the 'Request section' where a GET request is made to `http://localhost:65061/api/test/resource3`. The 'Headers' tab is selected, showing an `Authorization` header with the value `Bearer wYGFs9epP8BsU_PnDmv1NN7cewyA0j9EpE-yctRo_dMc9F...`. The 'Response Section' below shows a failed response with status `401 Unauthorized` and time `97 ms`. The response body contains the message `"Message": "Authorization has been denied for this request."`.

Như bạn có thể thấy, phản hồi 401 là trái phép. Nhưng bạn tạo mã thông báo cho người dùng có Vai trò là SuperAdmin, thì bạn có thể truy cập tài nguyên này.

Hãy xem lớp MyAuthorizationServiceProvider

```
public class MyAuthorizationServiceProvider : OAuthAuthorizationServiceProvider
{
    public override async Task ValidateClientAuthentication(OAuthValidateClientAuthenticationContext context)
    {
        context.Validated();
    }

    public override async Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)...
}
```

Phương pháp đầu tiên, tức là phương thức ValidateClientAuthentication chịu trách nhiệm xác thực Khách hàng, trong ví dụ này, ta giả định rằng ta chỉ có một ứng dụng khách nên ta sẽ luôn trả về rằng nó đã được xác thực thành công. Nhưng trong thời gian thực, bạn có thể có nhiều khách hàng và bạn cần xác thực các khách hàng.

7.4.4 Advantages of using Token Based Authentication

Khả năng mở rộng của Máy chủ:

Mã thông báo được khách hàng gửi đến máy chủ là độc lập có nghĩa là nó chứa đủ dữ liệu để xác định người dùng cần thiết để xác thực. Do đó, bạn có thể dễ dàng thêm nhiều máy chủ hơn vào trang web của mình, không phụ thuộc vào các cửa hàng phiên chia sẻ.

Khớp nối lỏng lẻo:

Ứng dụng khách không bị ràng buộc hoặc kết hợp với bất kỳ cơ chế xác thực cụ thể nào. Mã thông báo được tạo, xác thực và thực hiện xác thực chỉ do máy chủ thực hiện.

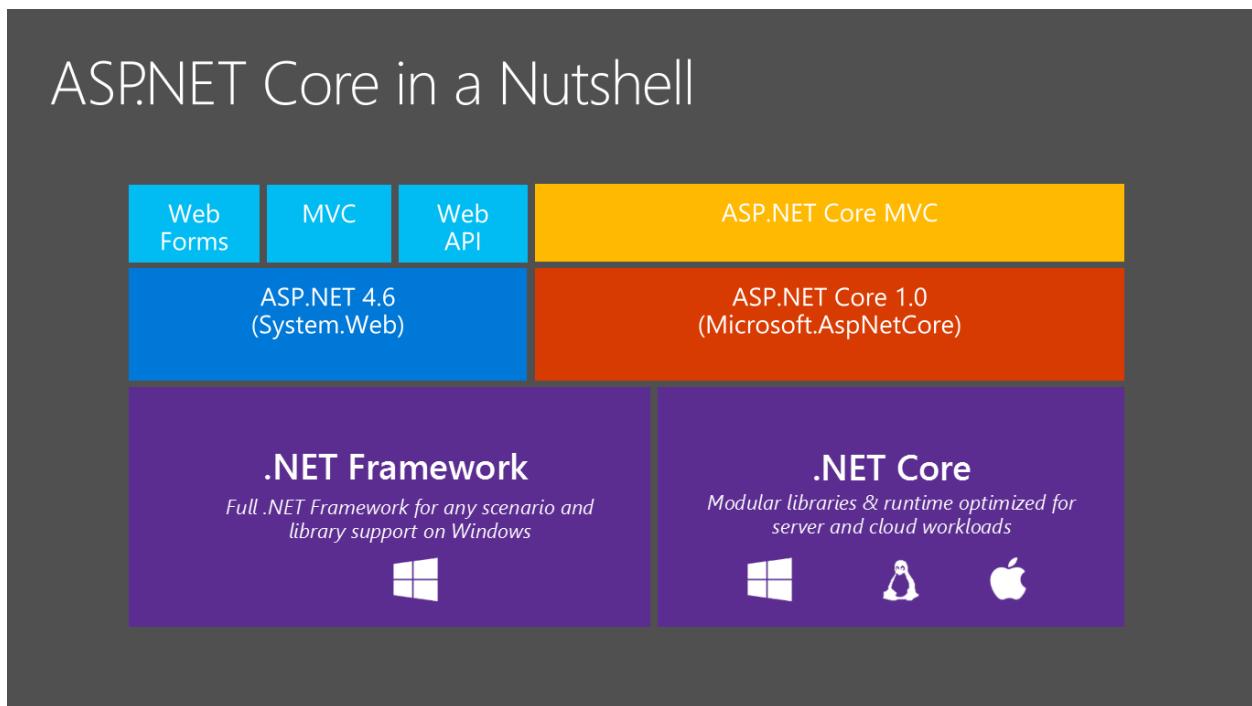
Thân thiện với thiết bị di động:

Cookie và trình duyệt giống nhau, nhưng xử lý cookie trên các nền tảng gốc như Android, iOS, Windows Phone không phải là một nhiệm vụ dễ dàng. Phương pháp dựa trên mã thông báo đơn giản hóa điều này rất nhiều.

CHƯƠNG 8 ASP.NET CORE WEB API

8.1 Giới thiệu

ASP.NET Core là một framework mã nguồn mở, hiệu suất cao và đa nền tảng dùng để xây dựng các ứng dụng hiện đại có kết nối với [Internet](#) và dựa trên mô hình đám mây. ASP.NET Core được phát triển bởi tập đoàn Microsoft và cộng đồng lập trình viên. ASP.NET Core cũng là một framework kiểu module có khả năng thực thi trên framework .NET, Windows và .NET Core đa nền tảng.



Hình 28. Vị trí .NET Core trong mô hình phát triển .NET

ASP.NET Core còn được xem là sự kết hợp giữa ASP.NET [MVC](#) và ASP.NET Web API tạo thành một mô hình lập trình đơn. Mặc dù được xây dựng mới, ASP.NET Core vẫn có tính tương thích cao với ASP.NET [MVC](#). Hơn nữa, các ứng dụng ASP.NET Core hỗ trợ kiểu phiên bản “side by side”, tức là cùng chạy trên một máy tính với việc lựa chọn nhiều phiên bản ASP.NET Core khác nhau. Điều này là không thể với các phiên bản ASP.NET trước kia. Phiên bản ASP.NET Core mới nhất tính đến thời điểm viết bài này là ASP.NET 2.1.

Với ASP.NET Core, bạn có thể:

Xây dựng các ứng dụng web, các dịch vụ, ứng dụng IoT và các phần backend mobile.

Sử dụng các công cụ phát triển ưa thích trên Windows, macOS và Linux.

Triển khai trên đám mây hoặc tại chỗ.

Chạy trên .NET Core hoặc .NET Framework.

8.2 Tại sao dùng ASP.NET Core?

Như bạn đã biết, có hàng triệu lập trình viên đã và đang sử dụng ASP.NET 4.x để xây dựng các ứng dụng Web. ASP.NET Core là một phiên bản thiết kế lại của ASP.NET 4.x, với nhiều thay đổi kiến

trúc giúp framework nhẹ hơn và có tính module nhiều hơn. Do đó, các lập trình viên có thể tiếp tục xây dựng ứng dụng bằng ASP.NET Core với nền tảng hiệu suất và tính tương thích tốt hơn.

ASP.NET mang lại các lợi ích như sau:

- Dùng để xây dựng giao diện Web (Web UI) cũng như các API Web.
- Tích hợp các framework phía client hiện đại và các quy trình làm việc phát triển.
- Hệ thống cấu hình sẵn có trên đám mây.
- Tích hợp sẵn nhúng phụ thuộc.
- Đường ống (pipeline) yêu cầu HTTP mang tính module, hiệu suất cao và nhẹ ký.
- Có khả năng lưu trữ (host) ở IIS, Nginx, Apache, Docker, hoặc tự host ở các tiến trình riêng.
- Tạo mới phiên bản app side-by-side với .NET Core.
- Tạo công cụ đơn giản hóa phát triển web hiện đại.
- Khả năng xây dựng, chạy trên Windows, macOS, và Linux.
- Mã nguồn mở và tập trung vào cộng đồng phát triển mã nguồn.

ASP.NET Core được cung cấp dưới dạng các gói NuGet. Bạn có thể sử dụng các gói này để tối ưu hóa ứng dụng khi chỉ nhúng những thành phần cần thiết. Trên thực tế, các ứng dụng ASP.NET Core 2.x cũng chỉ yêu cầu một gói NuGet đơn lẻ.

8.3 Xây dựng các API và giao diện (UI) Web dùng ASP.NET Core MVC

- ASP.NET Core [MVC](#) cung cấp các tính năng để xây dựng API Web và ứng dụng web như sau:
- Mô hình [MVC](#) (Model-View-Controller) giúp tạo ra các API web và ứng dụng web có khả năng thực nghiệm.
- Razor Pages (phần mới ở ASP.NET Core 2.0) là một mô hình lập trình dựa theo cơ chế trang web giúp xây dựng giao diện web dễ và hiệu quả hơn.
- Razor cung cấp một cú pháp hiệu quả cho các trang Razor và view [MVC](#).
- Tag Helper cho phép mã nguồn phía server tham gia vào việc tạo và phát sinh phần tử HTML ở các tập tin Razor.
- Tích hợp hỗ trợ cho các định dạng đa dữ liệu và việc dàn xếp nội dung cho phép các API web tiếp cận nhiều khách hàng hơn, bao gồm trên các trình duyệt và thiết bị mobile khác nhau.
- Ràng buộc mô hình tự động ánh xạ dữ liệu từ các yêu cầu HTTP đến các tham số phương thức hành động.
- Xác thực mô hình tự động thực thi xác thực dữ liệu phía client và server.

8.4 Phát triển phía client

- ASP.NET Core tích hợp liền mạch với các thư viện và framework phía client, bao gồm including Angular, React, và Bootstrap.
- ASP.NET Core nhắm đến .NET Framework
- ASP.NET Core có thể nhắm đến .NET Core hoặc .NET Framework. Các ứng dụng ASP.NET Core nhắm đến .NET Framework không phải là ứng dụng đa nền tảng khi chỉ chạy trên Windows. Hiện chưa có kế hoạch loại bỏ sự hỗ trợ .NET Framework ở ASP.NET Core. Nhìn chung, ASP.NET Core được tạo thành các thư viện .NET Standard (bản .NET chuẩn). Các ứng dụng viết bằng .NET Standard 2.0 cho thể chạy bất kỳ đâu nếu .NET Standard 2.0 được hỗ trợ.
- Có một số lợi ích khi nhắm đến .NET Core so với .NET Framework đó là:
 - Đa nền tảng, chạy trên macOS, Linux và Windows.
 - Nâng cao hiệu suất
 - Tạo phiên bản side-by-side
 - Các API mới
 - Mã nguồn mở

Kết luận

ASP.NET Core là một mô hình lập trình mới sử dụng cho các ứng dụng chạy đa nền tảng (macOS, Linux và Windows) và có khả năng triển khai trên đám mây. ASP.NET Core cũng hỗ trợ ASP.NET [MVC](#), ASP.NET Web API, backend mobile và cũng là mã nguồn mở. Bạn cần phân tích kỹ yêu cầu của ứng dụng để quyết định xem có nên dùng ASP.NET Core hay không. Mời bạn tiếp tục theo dõi phần tiếp theo.