Web API Interview Questions & Answers

Anoop Sharma

Author and Senior Software Engineer

DotNetTricks

Web API Interview Questions & Answers

Release History

- Initial Release 1.0.0 - 25th Mar 2019

About Dot Net Tricks

Dot Net Tricks is founded by Shailendra Chauhan (Microsoft MVP), in Jan 2010. Dot Net Tricks came into existence in the form of a blog post over various technologies including .NET, C#, SQL Server, ASP.NET, ASP.NET MVC, JavaScript, Angular, Node.js and Visual Studio etc.

The company which is currently registered by a name of Dot Net Tricks Innovation Pvt. Ltd. came into the shape in 2015. Dot Net Tricks website has an average footfall on the tune of 300k+ per month. The site has become a cornerstone when it comes to getting skilled-up on .NET technologies and we want to gain the same level of trust in other technologies. This is what we are striving for.

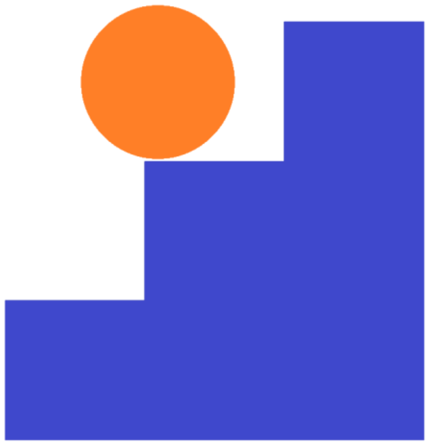
We have a very large number of trainees who have received training from our platforms and immediately got placement in some of the reputed firms testifying our claims of providing quality training. The website offers you a variety of free study material in the form of articles.

Dot Net Tricks Learning Platform

We have very robust technology platforms to answer the needs of all our trainees, no matter which program they have enrolled. We offer self-intuitive Learning Management System (LMS), which help our learners to learn code by doing and evaluates their learning.

Dot Net Tricks Pro

DotNetTricks Pro unlock the access of DotNetTricks premium features like unlimited access to all courses, source codes, assessments. Get help over email or phone. Upgrade your skills with curated learning paths tailored to today's developers and technology needs. Learn new skills and discover the world of possibilities with step-by-step guidance.



Start your journey today to learn coding. Because learning to code is the first step and foreword to advance your career. The detail about Dot Net Tricks Pro can be found here: https://www.dotnettricks.com/pro-membership

Dot Net Tricks Training Solutions

Instructor-led Training Programs

For a beginner who needs regular guidance, we have a fully packed Master Courses. They are almost equal to semester courses taught in engineering colleges when it comes to length, breadth of content delivery, the only difference instead of 5-6 months, they take approx. 16-weekend classes (2 months).

The detail about Master courses can be found here: https://www.dotnettricks.com/instructor-led-courses

Corporate Training (Online and Classroom)

Dot Net Tricks having a pool of mentors who help the corporate to enhance their employment skills as per changing technology landscape. Dot Net Tricks offers customized training programs for new hires and experienced employees through online and classroom mode. As a trusted and resourceful training partner, Dot Net Tricks helps the corporate to achieve success with its industry-leading instructional design and customer training initiatives.

Apart from these, we also provide on-demand boot camps and personalized project consultation.

The detail about Corporate Training can be found here: https://www.dotnettricks.com/corporate-training

Dedication

To my parents, Shri Vishnu Dutt Sharma (Father), Smt. Manju Sharma (Mother) and my elder brother, CA. Vineet Sharma for loving and supporting me in all aspect of life. Without their support, it would not be possible for me to write a Book. I especially want to say thanks to Shailendra Chauhan, CEO of DotNetTricks for providing me with the opportunity to write this book as well as the opportunity to join DotNetTricks as an Author.

-Anoop Kumar Sharma

Introduction

Writing a book has never been an easy task. It takes a great effort, patience and consistency with strong determination to complete it. Also, one should have a depth of knowledge over the subject is going to write.

So, what where my qualification to write this book? My qualification and inspiration come from my enthusiasm for and the experience with the technology and from my analytic and initiative nature. Being a trainer, analyst, consultant and blogger, I have thorough knowledge and understandings of .NET technologies. My inspiration and knowledge have also come from many years of my working experience and research over it.

So, the next question is who this book is for? This book covers useful Interview Questions and Answers on ASP.NET Web API. This book is appropriate for the novice as well as for senior-level professionals who want to strengthen their skills before appearing for an interview on ASP.NET Web API. This book is equally helpful to sharpen their programming skills and understanding ASP.NET Web API in a short time.

This book is not only the ASP.NET Web API interview book but it is more than that. This book helps you to get the depth knowledge of ASP.NET Web API with a simple and elegant way. This book is updated to the latest version of ASP.NET Web API 2.

I hope you will enjoy this book and find it useful. At the same time, I also encourage you to become a continuous reader of my blog https://www.dotnettricks.com/mentors/anoop-sharma and be part of the discussion. But most importantly practice a lot and enjoy the technology. That's what it's all about.

To get the latest information on ASP.NET Web API, I encourage you to follow the official Microsoft ASP.NET community website at www.asp.net. I also encourage you to read the article at www.dotnettricks.com that contains .NET, C#, ASP.NET MVC, EF, jQuery and many more tips, tricks and tutorials.

All the best for your interview and happy programming!

About the Author

Anoop Kumar Sharma – Senior Software Engineer and Blogger

Anoop Kumar Sharma is currently working as Senior Software Engineer in an MNC having vast experience on Microsoft .Net Technologies. He is also expertise in React, Node and MongoDB as well.

His several articles have been broadcast in a number of articles-of-the-day on the Official Microsoft ASP.NET Community Site. He has rewarded as C# Corner MVP and DZone MVB for his exceptional contributions to the tech communities.

He always loves to share his technical skills with the other person. Whenever he gets time, he writes Articles on his blog ittutorialswithexample.com which provides Articles on latest technology

How to Contact Us

Although the author of this book has tried to make this book as accurate as it possible but if there is something strikes you as odd, or you find an error in the book please drop a line via e-mail.

The e-mail addresses are listed as follows:

- mentor@dotnettricks.com
- info@dotnettricks.com

We are always happy to hear from our readers. Please provide your valuable feedback and comments!

You can follow us on YouTube, Facebook, Twitter, LinkedIn and Google Plus or subscribe to RSS feed.

Table of Contents

1
Introducing ASP.NET Web API

Q1. What is ASP.NET Web API?

Ans. ASP.NET Web API is a framework provided by the Microsoft with which we can easily build HTTP services that can reach a broad of clients, including browsers, mobile, IoT devices, etc. ASP.NET Web API provides an ideal platform for building RESTful applications on the .NET Framework.

Q2. What is the difference between ASP.NET Web API and WCF?

Ans. Web API is a Framework to build HTTP Services that can reach a board of clients, including browsers, mobile, IoT Devices, etc. and provided an ideal platform for building RESTful applications. It is limited to HTTP based services. ASP.NET framework ships out with the .NET framework and is Open Source. WCF i.e. Windows Communication Foundation is a framework used for building Service Oriented applications (SOA) and supports multiple transport protocol like HTTP, TCP, MSMQ, etc. It supports multiple protocols like HTTP, TCP, Named Pipes, MSMQ etc. WCF ships out with the .NET Framework. Both Web API and WCF can be self-hosted or can be hosted on the IIS Server.

Q3. When to prefer ASP.NET Web API over WCF?

Ans. It totally depends upon the requirement. Choose ASP.NET Web API is you want only HTTP based services only as Web API is a lightweight architecture and is good for the devices which have limited bandwidth. We can also create the REST services with the WCF, but that requires lots of configuration. In case, if you want a service that should support multiple transport protocol like HTTP, UDP, TCP, etc. then WCF will be a better option.

Q4. Which .NET Framework supports ASP.NET Web API?

Ans. First Version of ASP.NET Web API is introduced in .NET Framework 4. After that, all the later versions of the .NET Framework supports the ASP.NET Web API.

Q5. Can we consume ASP.NET Web API in applications created using other than .NET?

Ans. Yes, we can consume ASP.NET Web API in the applications created using another language than .NET but that application must have access/supports to the HTTP protocol.

Q6. What is the difference between ASP.NET MVC application and ASP.NET Web API application?

Ans. ASP.NET MVC is used to create a web application which returns both data as well as View whereas Web API is used to create HTTP based Services which only returns data not view. In an ASP.NET MVC application, requests are mapped to Action Methods whereas in the ASP.NET Web API request is mapped to Action based on the Action Verbs.

Q7. What are the RESTful Services?

Ans. REST stands for the Representational State Transfer. This term is coined by the Roy Fielding in 2000. RESTful is an Architectural style for creating loosely couple applications over the HTTP. In order to make API to be RESTful, it has to adhere the around 6 constraints that are mentioned below:

1. Client and Server Separation: Server and Clients are clearly isolated in the RESTful services.
2. Stateless: REST Architecture is based on the HTTP Protocol and the server response can be cached by the clients, but no client context would be stored on the server.
3. Uniform Interface: Allows a limited set of operation defined using the HTTP Verbs. For eg: GET, PUT, POST, Delete etc.
4. Cacheable: RESTful architecture allows the response to be cached or not. Caching improves performance and scalability.
5. Code-On-Demand
6. Layered System

Q8. What are the new features introduced in ASP.NET Web API 2.0?

Ans. The following features have been introduced in ASP.NET Web API 2.0:

1. Attribute Routing
2. CORS (Cross-Origin Resource Sharing)
3. OWIN (Open Web Interface for .NET) self-hosting
4. IHttpActionResult
5. Web API OData etc.

Q9. Can we return View from Web API?

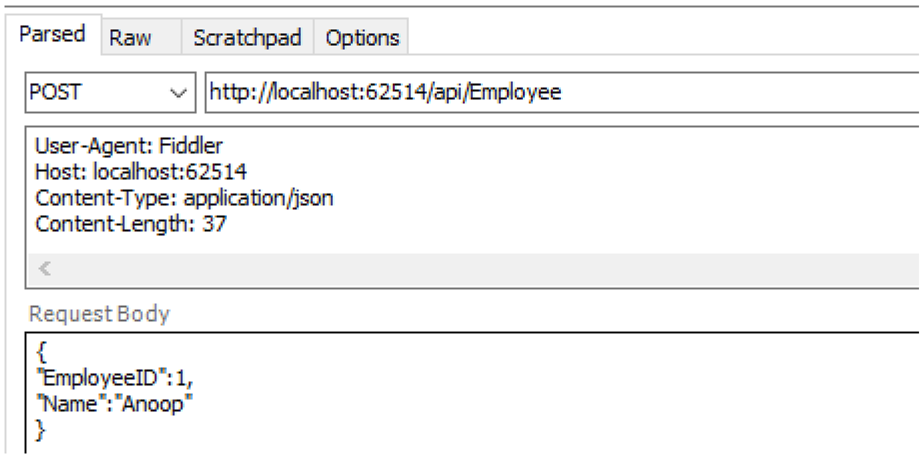
Ans. No, Web API does not return View but they return the data. APIController is meant for returning the data. So, if you need to return a view from the controller class, then make sure to use or inherit the Controller class.

Q10. Does ASP.NET Web API replace the WCF?

Ans. No, ASP.NET Web API didn't replace WCF Service as it is only used for creating RESTful Service i.e. non-SOAP based service.

Q11. What is the difference between Under-Posting and Over-Posting in ASP.NET Web API?

Ans. **Under-Posting:** Under-Posting happens when the client sends out some properties of the model, Just for an example, Suppose we have an employee class which have three properties i.e. EmployeeID, Name, Age. The client only sends out EmployeeID and Name then JSON Formatter assigns a default value for the Age i.e. Zero.

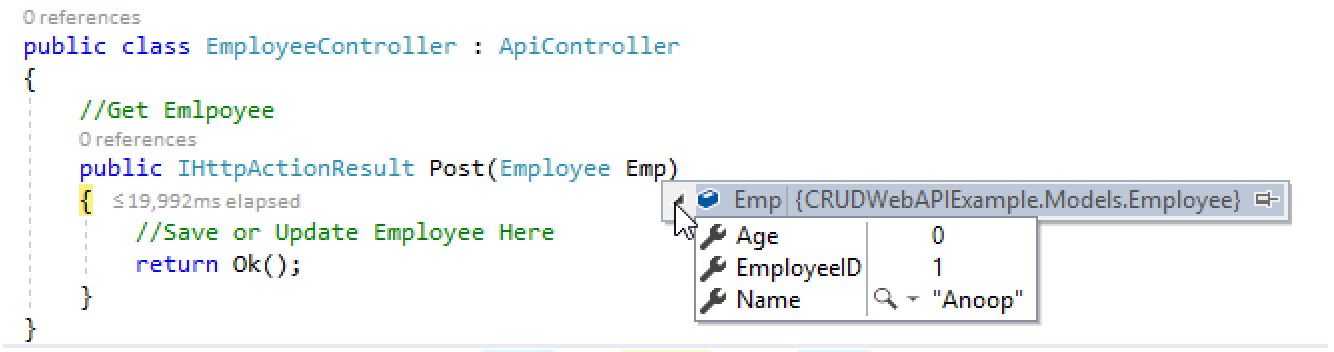


```
http://localhost:62514/api/Employee
```

```
User-Agent: Fiddler
Host: localhost:62514
Content-Type: application/json
Content-Length: 37
```

```
{
"EmployeeID":1,
"Name":"Anoop"
}
```

Image: Post few properties of the API Model using Fiddle



```
0 references
public class EmployeeController : ApiController
{
    //Get Emlpoyee
    0 references
    public IHttpActionResult Post(Employee Emp)
    { ≤19,992ms elapsed
        //Save or Update Employee Here
        return Ok();
    }
}
```

```
Emp {CRUDWebAPIExample.Models.Employee}
  Age          0
  EmployeeID   1
  Name         "Anoop"
```

Image: Receive all properties with the default value assigned to the Age property.

As we see that, Age is assigned a default value, but Client didn't set it as zero. To Force client to set a default value, make Age property as nullable or set the Required attribute with that property.

Over-Posting: A client can also send more data than your model expected. In this case, JSON formatter or XML Formatter simply ignores that value. When a client sends more data than expected in binding, then this process is known as Over-Posting.

2
ASP.NET Web API Fundamentals

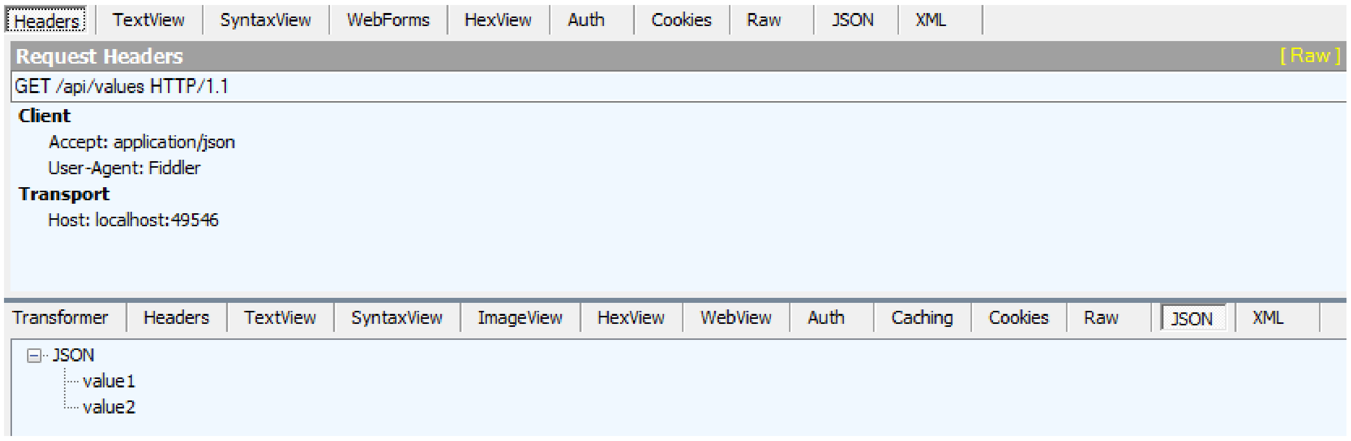
Q1. What is Request Verbs or HTTP Verbs?

Ans. In RESTful service, we can perform all types of CRUD (Create, Read, Update, Delete) Operation. In REST architecture, it is suggested to have a specific Request Verb or HTTP verb on the specific type of the call made to the server. Popular Request Verbs or HTTP Verbs are mentioned below:

1. HTTP Get: Used to get or retrieve the resource or information only.
2. HTTP Post: Used to create a new resource on the collection of resources.
3. HTTP Put: Used to update the existing Response
4. HTTP Delete: Used to Delete an existing resource.

Q2. What is the Request Header?

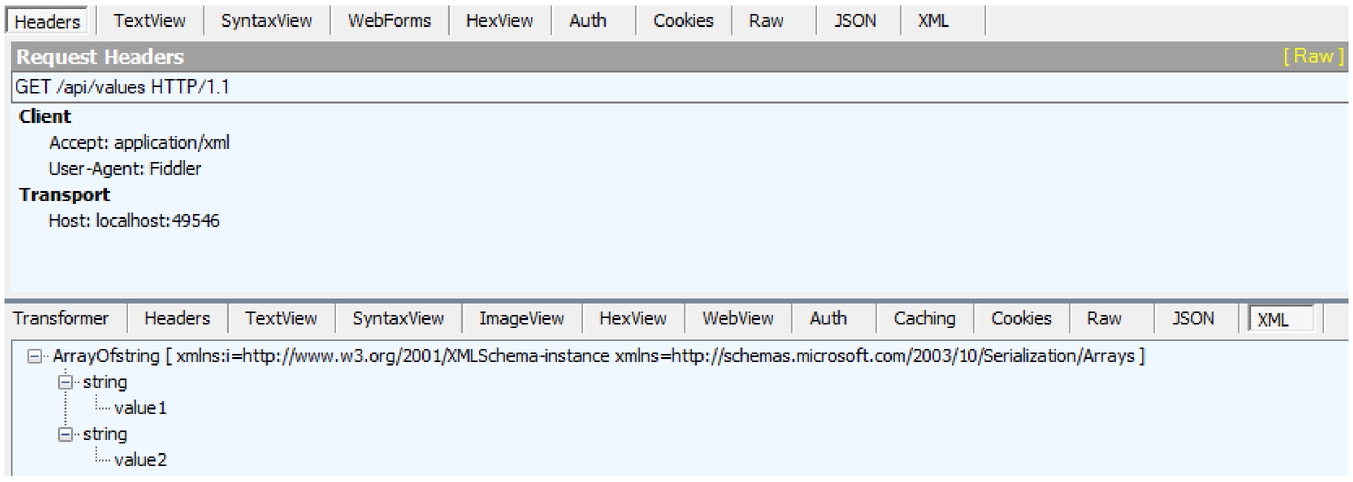
Ans. Request header is used to provide additional information related to the request. For example, if a client wants a response in JSON format, then the client needs to set the Accept request header to application/json or if want a response in XML format than Accept request header to application/xml.



```
GET /api/values HTTP/1.1
Client
    Accept: application/json
    User-Agent: Fiddler
Transport
    Host: localhost:49546
```

```
JSON
   value1
   value2
```

Image: Request header to get JSON Response in Fiddler



```
GET /api/values HTTP/1.1
Client
    Accept: application/xml
    User-Agent: Fiddler
Transport
    Host: localhost:49546
```

```
ArrayOfstring [ xmlns:i=http://www.w3.org/2001/XMLSchema-instance xmlns=http://schemas.microsoft.com/2003/10/Serialization/Arrays ]
    string
        value1
    string
        value2
```

Image: Request header to get XML Response in Fiddler

Q3. What is Request Body?

Ans. Request Body contains the data sent to the server on request. The request body is also known as Body Parameter.

Q4. What is Response Body?

Ans. Response Body is the data sent by the server as a response.

Q5. What are HTTP Status Codes?

Ans. HTTP Status Code Is 3-digit integer in which the first digit of the Status Code defines the class of response. Response Header of each API response contains the HTTP Status Code. HTTP Status Codes are grouped into five categories based upon the first number.

S. No.	HTTP Status Code	Description
1.	1XX	Informational
2.	2XX	Success
3.	3XX	Redirection
4.	4XX	Client-Side Error
5.	5XX	Server-Side Error

Table: HTTP Status Code with Description

Some of the commonly seen HTTP Status Codes are: 200 (Request is Ok), 201 (Created), 202 (Accepted), 204 (No Content), 301 (Moved Permanently), 400 (Bad Request), 401 (Unauthorized), 403 (Forbidden), 404 (Not Found), 500 (Internal Server Error), 502 (Bad Gateway), 503 (Service Unavailable) etc.

Q6. How to provide an alias name for Web API Action?

Ans. We can provide an alias name to Web API Action by using "ActionName" Attribute.

```
[ActionName("GetEmployee")]
public IHttpActionResult Get()
{
    return Ok();
}
```

DotNetTricks

Q7. What is Parameter Binding in ASP.NET Web API?

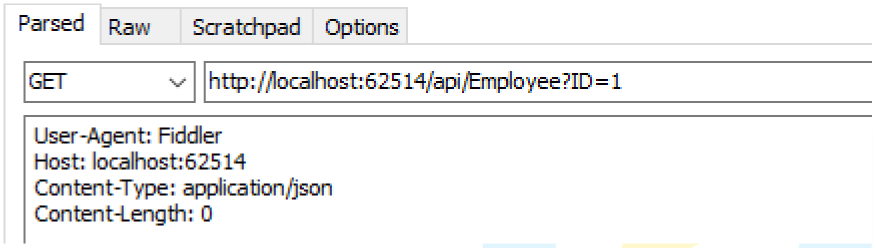
Ans. When Web API calls a method on a controller, it must set the values for the parameters, this particular process is known as Parameter Binding. By Default, Web API uses the below rules in order to bind the parameter:

1. **FromUri**: If the parameter is of "Simple" type, then Web API tries to get the value from the URI. Simple Type includes.Net Primitive type like int, double etc., DateTime, TimeSpan, GUID, string, any type which can be converted from the string type.
2. **FromBody**: If the parameter is of "Complex" type, then Web API will try to bind the values from the message body.

Q8. What is FromUri Attribute and How to use FromUri Attribute in ASP.NET Web API?

Ans. To force Web API to read a complex or Simple type from the URI, use [FormUri] attribute to the parameter in the Web API. The value should be in the form of key-value pairs just as a query string.

As in below example, ID=1 is passed in the URI.

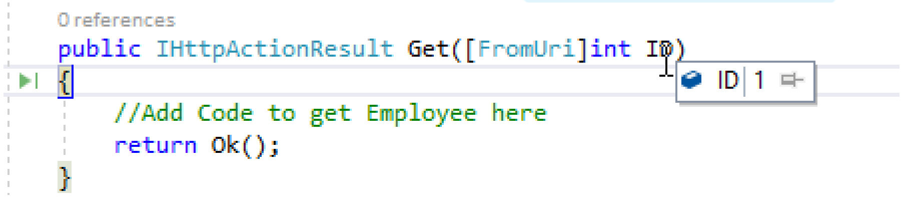


```
http://localhost:62514/api/Employee?ID=1

User-Agent: Fiddler
Host: localhost:62514
Content-Type: application/json
Content-Length: 0
```

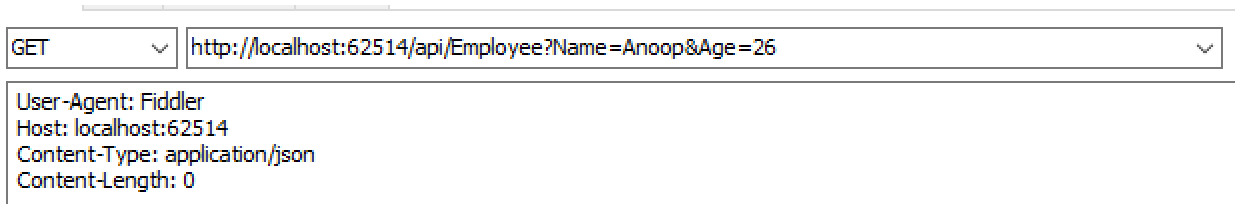
Image: Passed simple types in the URI in Fiddler

You will see the ID passed in URI is bind in the parameter of the Get Action of the Employee.



```
public IHttpActionResult Get([FromUri]int ID)
{
    //Add Code to get Employee here
    return Ok();
}
```

We can even pass the complex type from the URI as well but need to make sure that the property name must be written exactly as defined in the model.

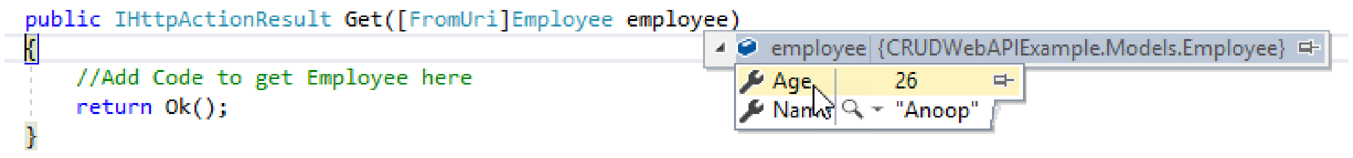


```
http://localhost:62514/api/Employee?Name=Anoop&Age=26

User-Agent: Fiddler
Host: localhost:62514
Content-Type: application/json
Content-Length: 0
```

Image: Passing a complex type in the URI from Fiddler

In code, you will receive the passed URI parameter as shown below:



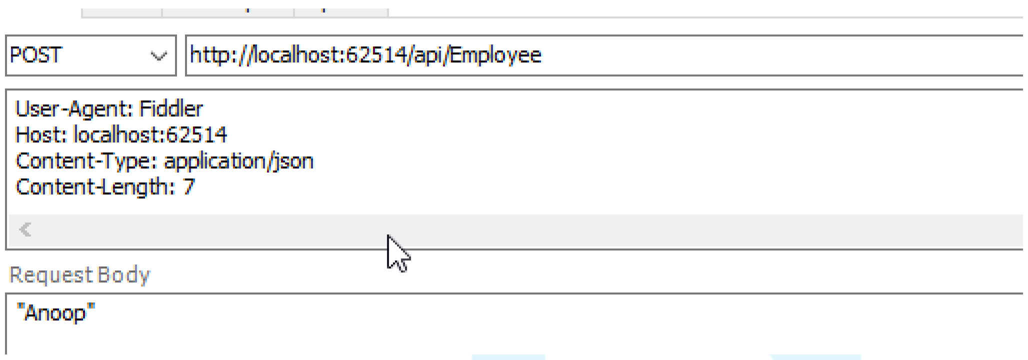
```
public IHttpActionResult Get([FromUri]Employee employee)
{
    //Add Code to get Employee here
    return Ok();
}
```

Image: Using FromUri for receiving a Complex type object parameter

Q9. What is FromBody Attribute and How to use this Attribute in ASP.NET Web API?

Ans. To force Web API to read a simple type from the request body, add FromBody Attribute to the parameter.

Example: Posting a simple type from the request body of the fiddler and binding it with the FromBody attribute in the ASP.NET Web API.

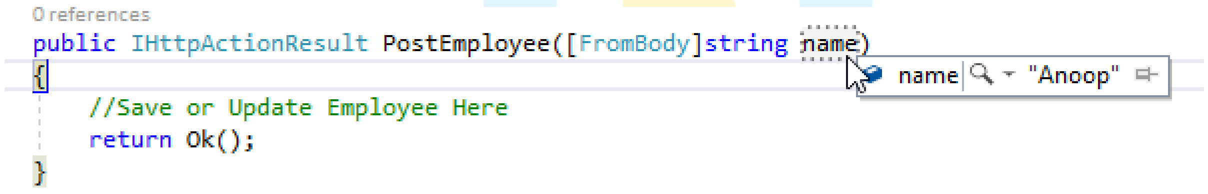


```
POST    http://localhost:62514/api/Employee

User-Agent: Fiddler
Host: localhost:62514
Content-Type: application/json
Content-Length: 7

Request Body
"Anoop"
```

Image: Posting a simple type from the request body of the fiddler

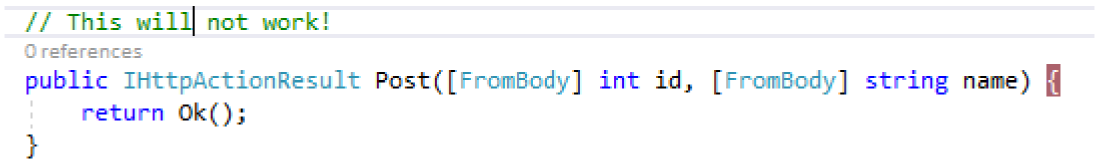


```
0 references
public IHttpActionResult PostEmployee([FromBody]string name)
{
    //Save or Update Employee Here
    return Ok();
}
```

Image: Receiving the value from FromBody in EmployeeController

Q10. Can we declare FromBody attribute on multiple parameters?

Ans. We can only declare only one FromBody attribute on the parameter to read data from the request body. The reason behind that is that the request body might be stored in the non-buffered stream that can be read only once.

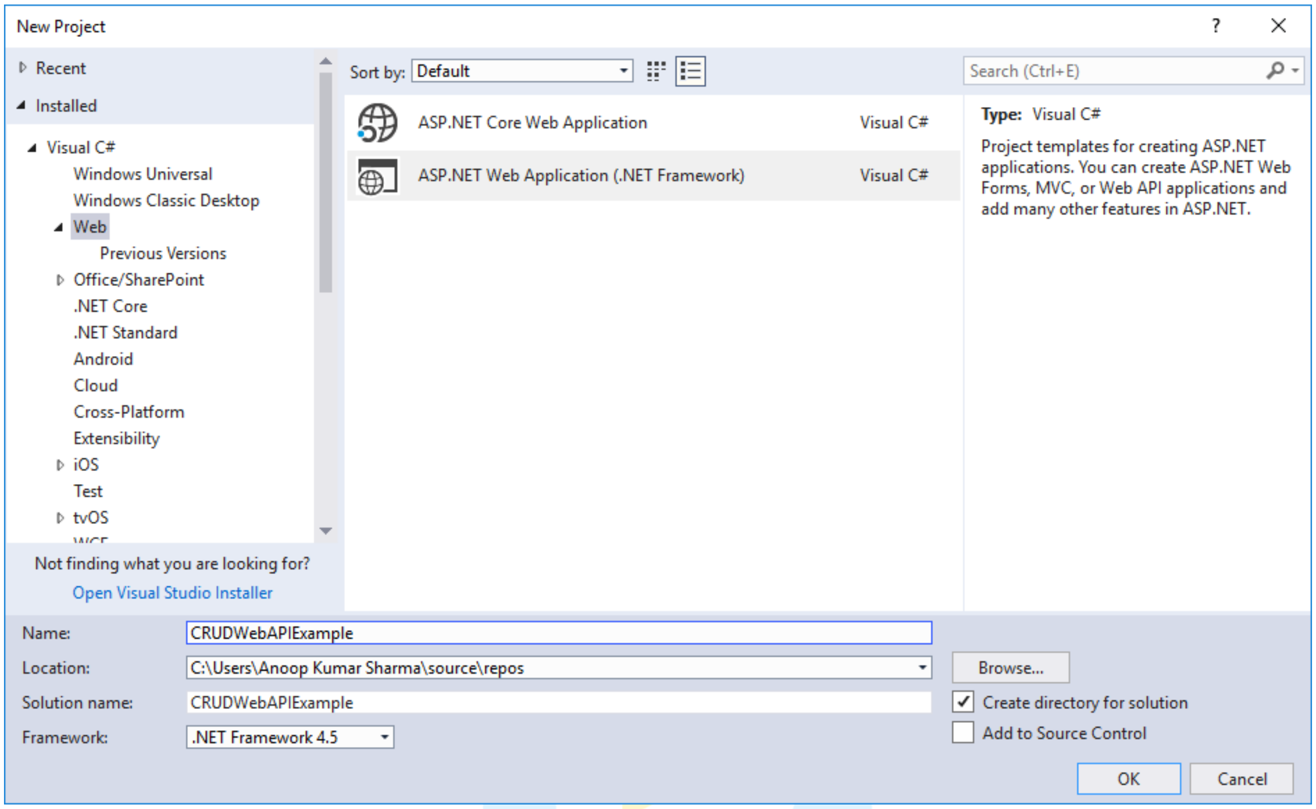


```
// This will not work!
0 references
public IHttpActionResult Post([FromBody] int id, [FromBody] string name) {
    return Ok();
}
```

Image: Incorrect way to declare FromBody attribute on multiple parameters.

Q11. How to get Data from the SQL Server Database using ASP.NET Web API?

Ans. Let's suppose we are creating a new project with Visual Studio. Select ASP.NET Web Application.



Sort by: Default

ASP.NET Core Web Application — Visual C#

ASP.NET Web Application (.NET Framework) — Visual C#

Type: Visual C#

Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Name: CRUDWebAPIExample

Location: C:\Users\Anoop Kumar Sharma\source\repos

Solution name: CRUDWebAPIExample

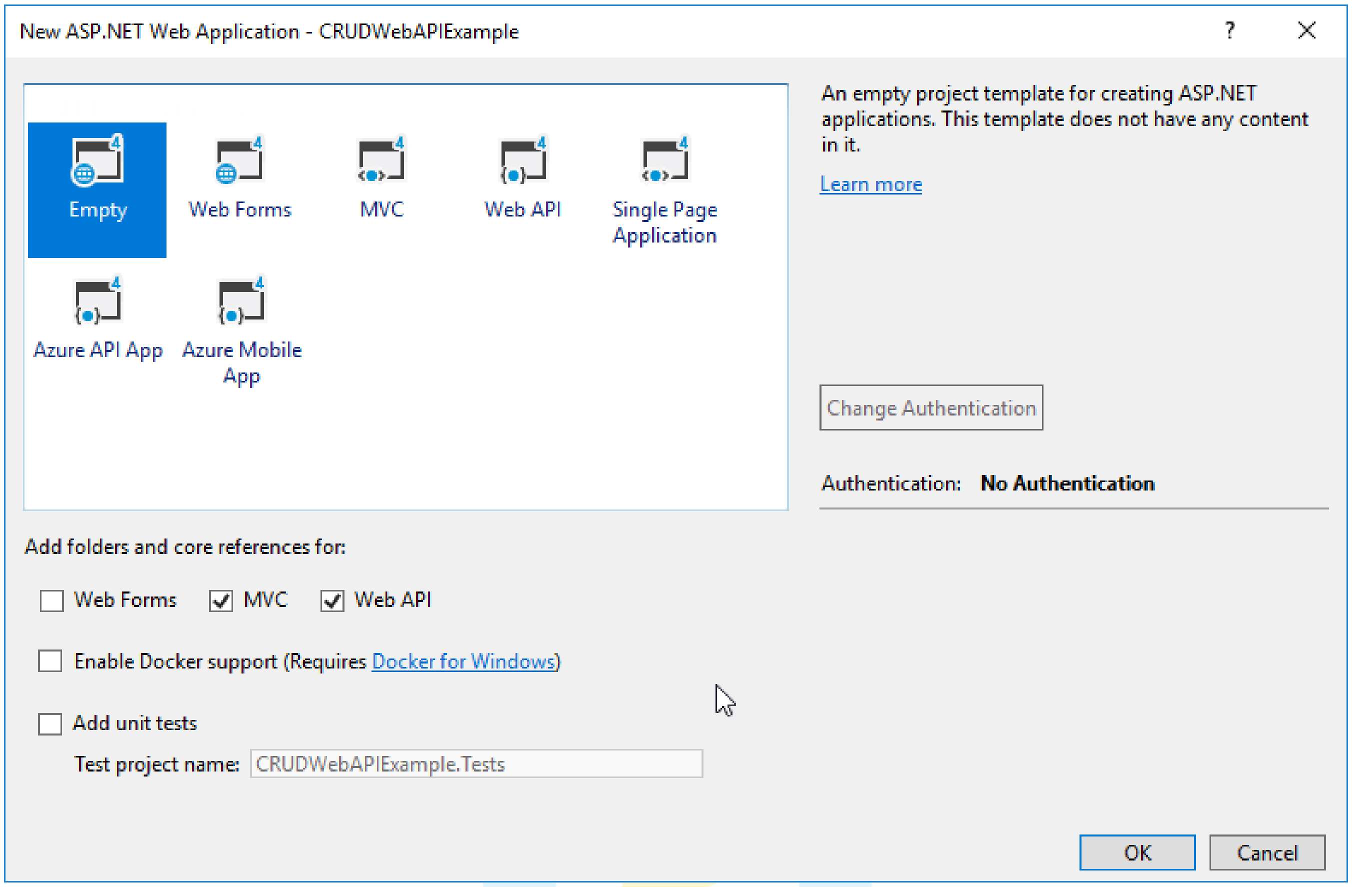
Framework: .NET Framework 4.5

[✓] Create directory for solution

[ ] Add to Source Control

Image: New Web Project dialog in Visual Studio

Select the template for the new project. We will select Empty project (selected MVC and Web API checkbox so that respective core classes added to the project) for the ASP.NET applications.


New ASP.NET Web Application - CRUDWebAPIExample

An empty project template for creating ASP.NET applications. This template does not have any content in it.

Authentication: **No Authentication**

Add folders and core references for:

☐ Web Forms ☑ MVC ☑ Web API

☐ Enable Docker support (Requires Docker for Windows)

☐ Add unit tests

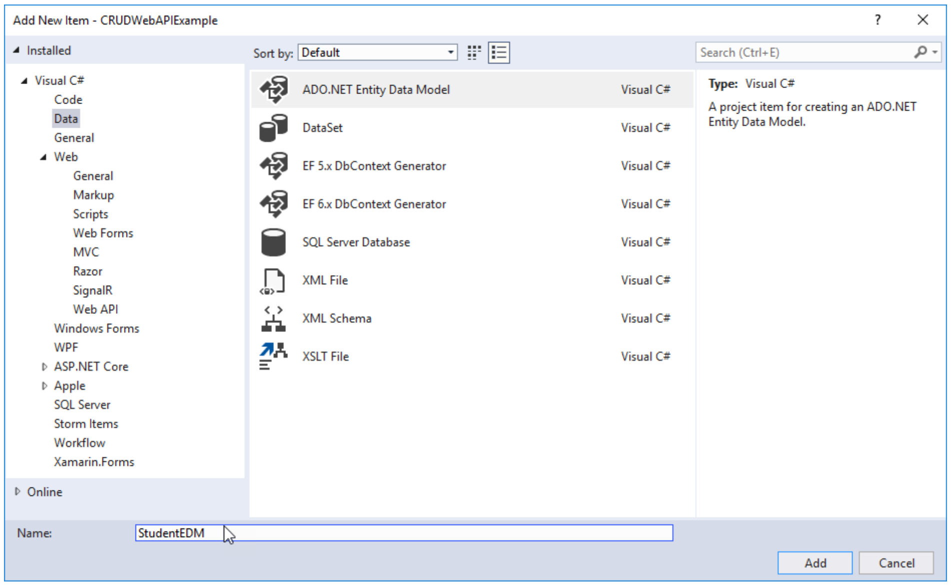
Test project name: CRUDWebAPIExample.Tests

Image: Create an Empty Web ASP.NET Web API Application

There are several ORM available by which we can connect and use the SQL Server Database with the ASP.NET web application. Here, we will use entity framework with database first approach for performing the CRUD operation in the project. Use the below script for the dummy database for performing the CRUD operations.

```
CREATE DATABASE WebAPIDB
GO
IF OBJECT_ID ('dbo.Students') IS NOT NULL
       DROP TABLE dbo.Students
GO
CREATE TABLE dbo.Students
       (
       ID     INT IDENTITY NOT NULL,
       Name   NVARCHAR (200) NULL,
       Age    INT NULL,
       Stream NVARCHAR (100) NULL,
       CONSTRAINT PK_Students PRIMARY KEY (ID)
       )
GO
INSERT INTO Students VALUES('Anoop Kumar Sharma',26,'Science')
INSERT INTO Students VALUES('Ajay',28,'Commerce')
INSERT INTO Students VALUES('Farhan',25,'Arts')
INSERT INTO Students VALUES('Mike',27,'Science')
```

Add ADO.NET Entity Data Model in the ASP.NET Web API project.


Add New Item - CRUDWebAPIExample

Installed

Sort by: Default

Search (Ctrl+E)

- Visual C#
  - Code
  - Data
  - General
  - Web
    - General
    - Markup
    - Scripts
    - Web Forms
    - MVC
    - Razor
    - SignalR
    - Web API
  - Windows Forms
  - WPF
  - ASP.NET Core
  - Apple
  - SQL Server
  - Storm Items
  - Workflow
  - Xamarin.Forms
- Online

ADO.NET Entity Data Model	Visual C#
DataSet	Visual C#
EF 5.x DbContext Generator	Visual C#
EF 6.x DbContext Generator	Visual C#
SQL Server Database	Visual C#
XML File	Visual C#
XML Schema	Visual C#
XSLT File	Visual C#

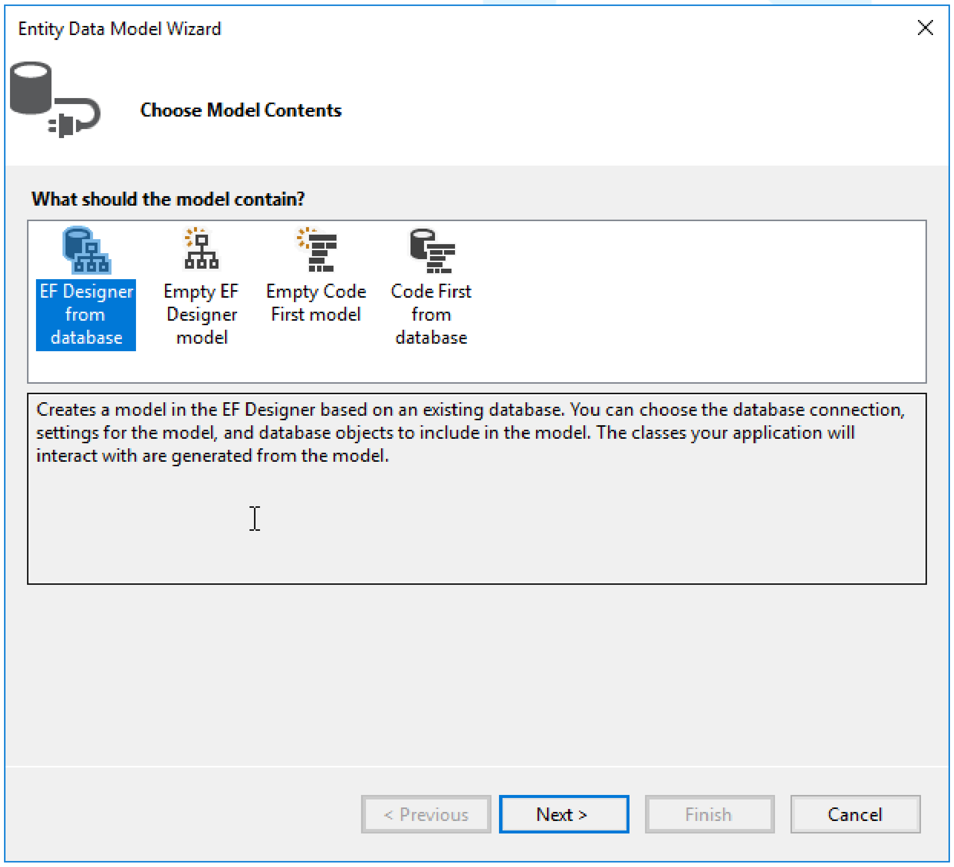
Type: Visual C#

A project item for creating an ADO.NET Entity Data Model.

Name: StudentEDM

Image: Add ADO.NET Entity Data Model in Project

And then select EF Designer from the database. Click on Next button


Entity Data Model Wizard

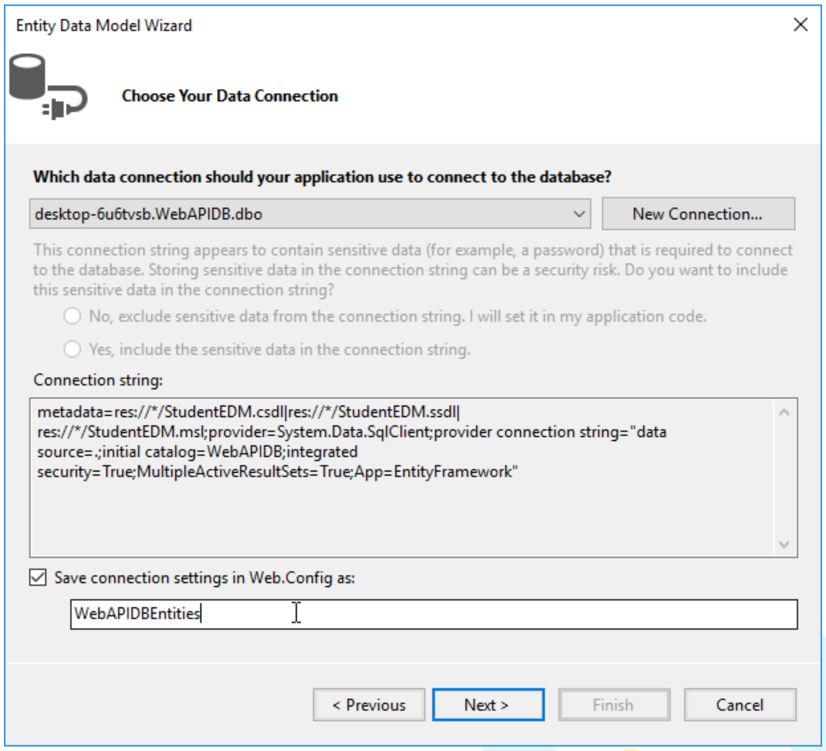
Choose Model Contents

What should the model contain?

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.

Image: Choose Model Contents in Entity Data Model Wizard

In the next step, you have to select the data connection in the Entity Data Model Wizard.



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

desktop-6u6tvsb.WebAPIDB.dbo

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:

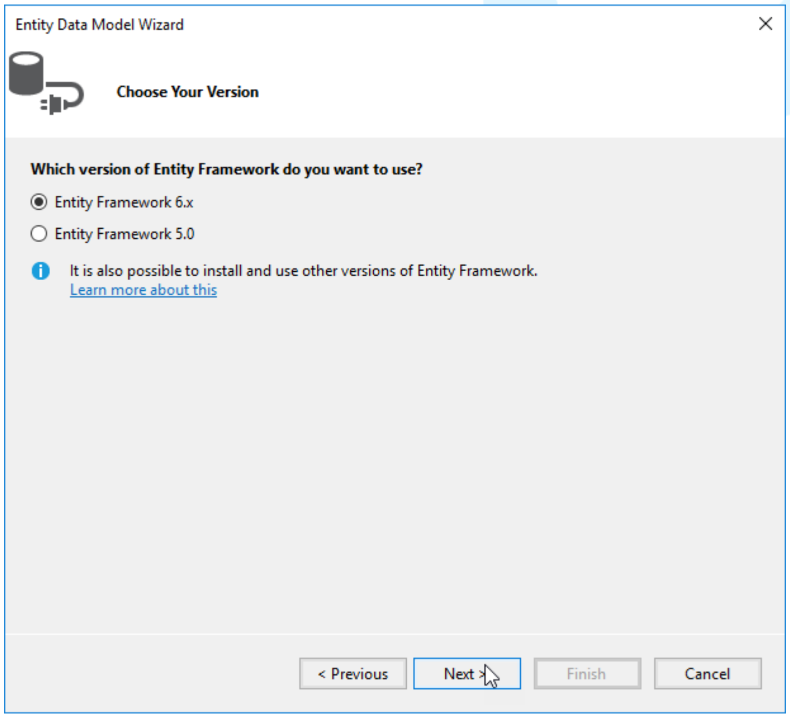
```
metadata=res://*/StudentEDM.csdl|res://*/StudentEDM.ssdl|
res://*/StudentEDM.msl;provider=System.Data.SqlClient;provider connection string="data
source=.;initial catalog=WebAPIDB;integrated
security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

☑ Save connection settings in Web.Config as:

WebAPIDBEntities

Image: Entity Data Model Connection Wizard

Select the version of the Entity Framework you want to use and click on next.



Which version of Entity Framework do you want to use?

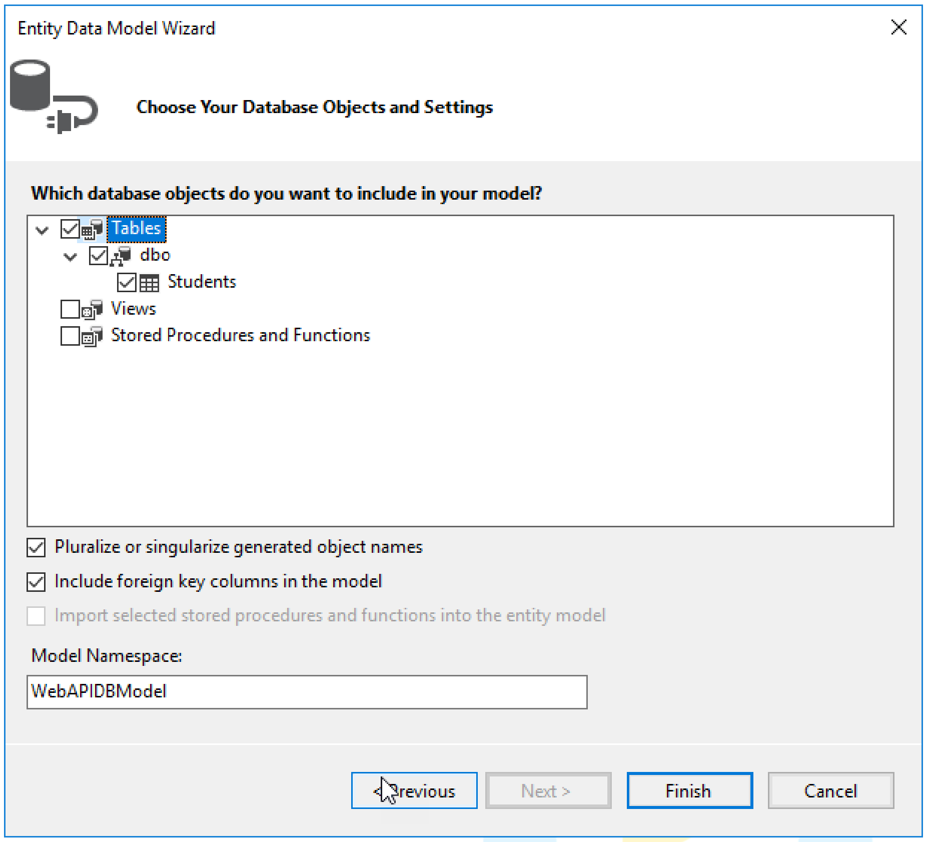
● Entity Framework 6.x

○ Entity Framework 5.0

It is also possible to install and use other versions of Entity Framework.
Learn more about this

Image: Choose the version of the Entity Framework

Select the database object you want to include in the model and click on finish button.


Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

- [✓] Tables
  - [✓] dbo
    - [✓] Students
- [ ] Views
- [ ] Stored Procedures and Functions

[✓] Pluralize or singularize generated object names

[✓] Include foreign key columns in the model

[ ] Import selected stored procedures and functions into the entity model

Model Namespace: WebAPIDBModel

Image: Choose Database objects and settings

After clicking on the finish, you will see the Database object model diagram just like below.

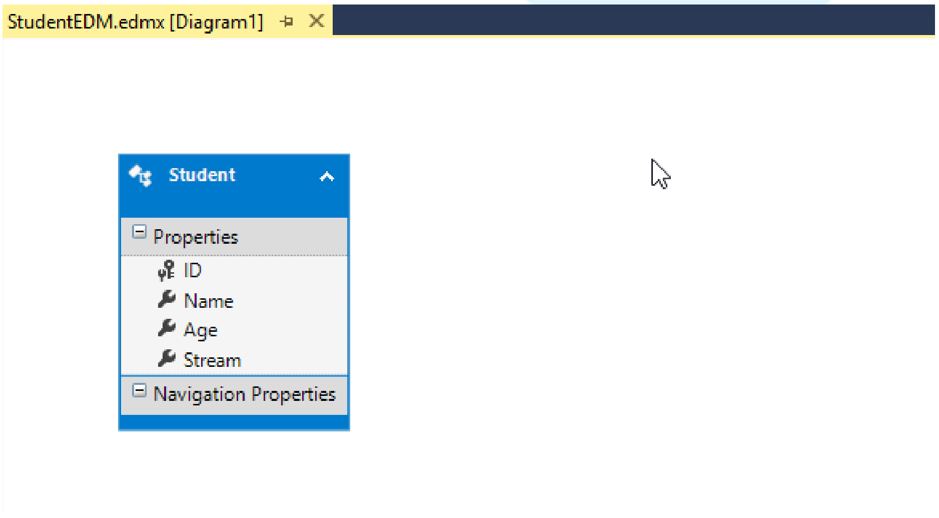
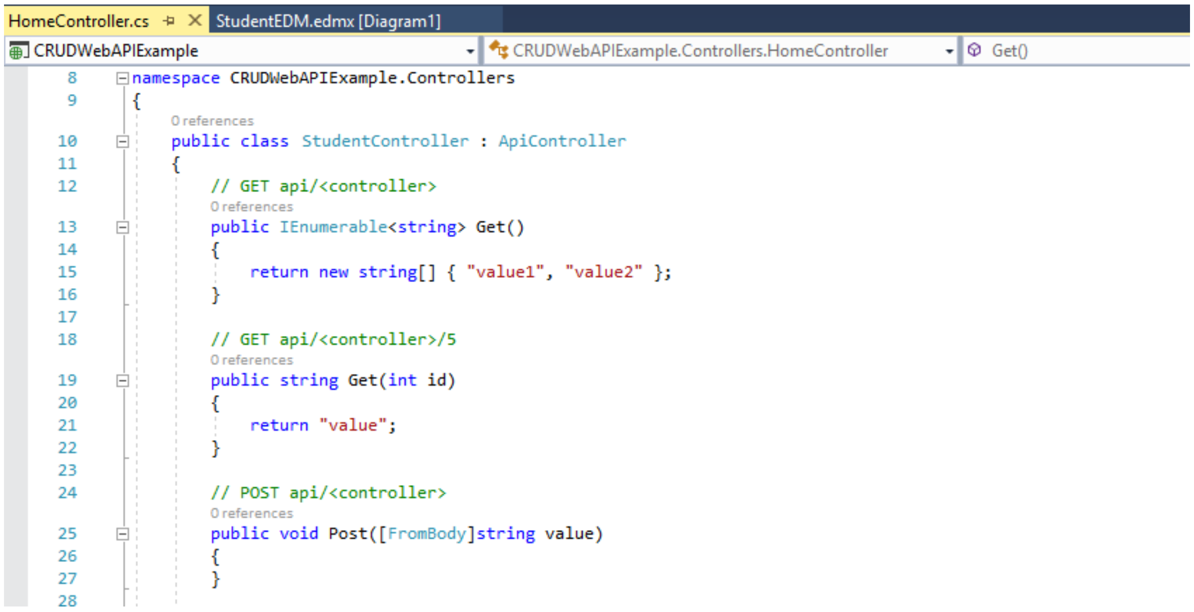


Image: Entity Data Model Diagram

Now, add a Web API Controller in the controllers.


HomeController.cs
StudentEDM.edmx [Diagram1]
CRUDWebAPIExample
CRUDWebAPIExample.Controllers.HomeController
Get()

```
namespace CRUDWebAPIExample.Controllers
{
    0 references
    public class StudentController : ApiController
    {
        // GET api/<controller>
        0 references
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET api/<controller>/5
        0 references
        public string Get(int id)
        {
            return "value";
        }

        // POST api/<controller>
        0 references
        public void Post([FromBody]string value)
        {
        }
```

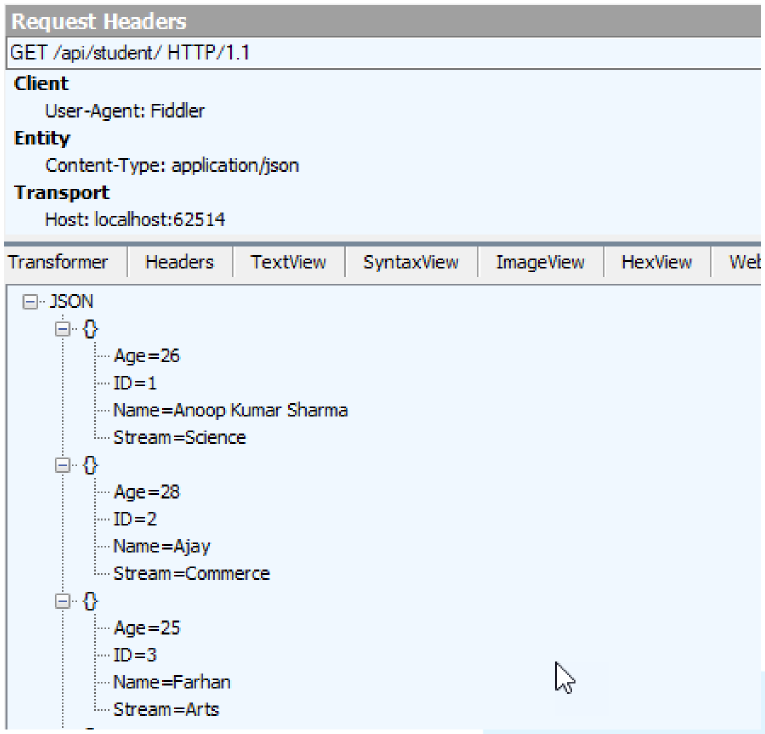
Image: Default Structure of Web API Controller

Now, let's change the Get method of the Web API as shown below.

```
// GET
        public IHttpActionResult Get()
        {
            using (WebAPIDBEntities objWebAPIDBEntities =new WebAPIDBEntities()) {
                //return All Student with HTTP Status Code OK
                return Ok(objWebAPIDBEntities.Students.ToList());
            }
        }

        // GET with ID
        public IHttpActionResult Get(int id)
        {
            using (WebAPIDBEntities objWebAPIDBEntities = new WebAPIDBEntities())
            {
                var student = objWebAPIDBEntities.Students.FirstOrDefault(x => x.ID == id);
                if (student==null) {
                    //return Error response with HTTP Status Code Notfound
                    return NotFound();
                }
                else
                {
                    //return student with an id request by client with HTTP Status Code OK
                    return Ok(student);
                }
            }
        }
```

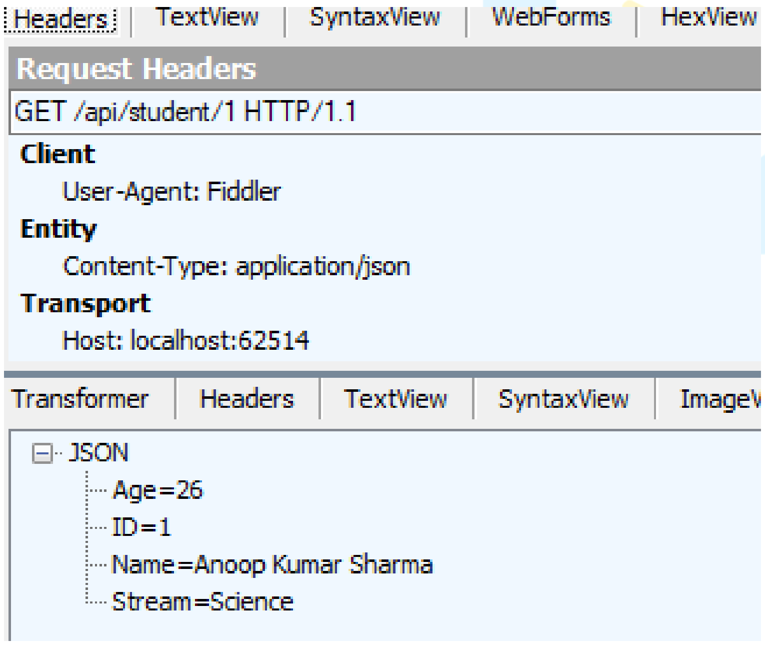
Let's hit the Get method to test both Get and Get with the ID parameter.



```
Request Headers
GET /api/student/ HTTP/1.1
Client
    User-Agent: Fiddler
Entity
    Content-Type: application/json
Transport
    Host: localhost:62514
```

```
JSON
  {}
    Age=26
    ID=1
    Name=Anoop Kumar Sharma
    Stream=Science
  {}
    Age=28
    ID=2
    Name=Ajay
    Stream=Commerce
  {}
    Age=25
    ID=3
    Name=Farhan
    Stream=Arts
```

Image: Get Method to get All Student Data in Fiddler



```
Request Headers
GET /api/student/1 HTTP/1.1
Client
    User-Agent: Fiddler
Entity
    Content-Type: application/json
Transport
    Host: localhost:62514
```

```
JSON
    Age=26
    ID=1
    Name=Anoop Kumar Sharma
    Stream=Science
```

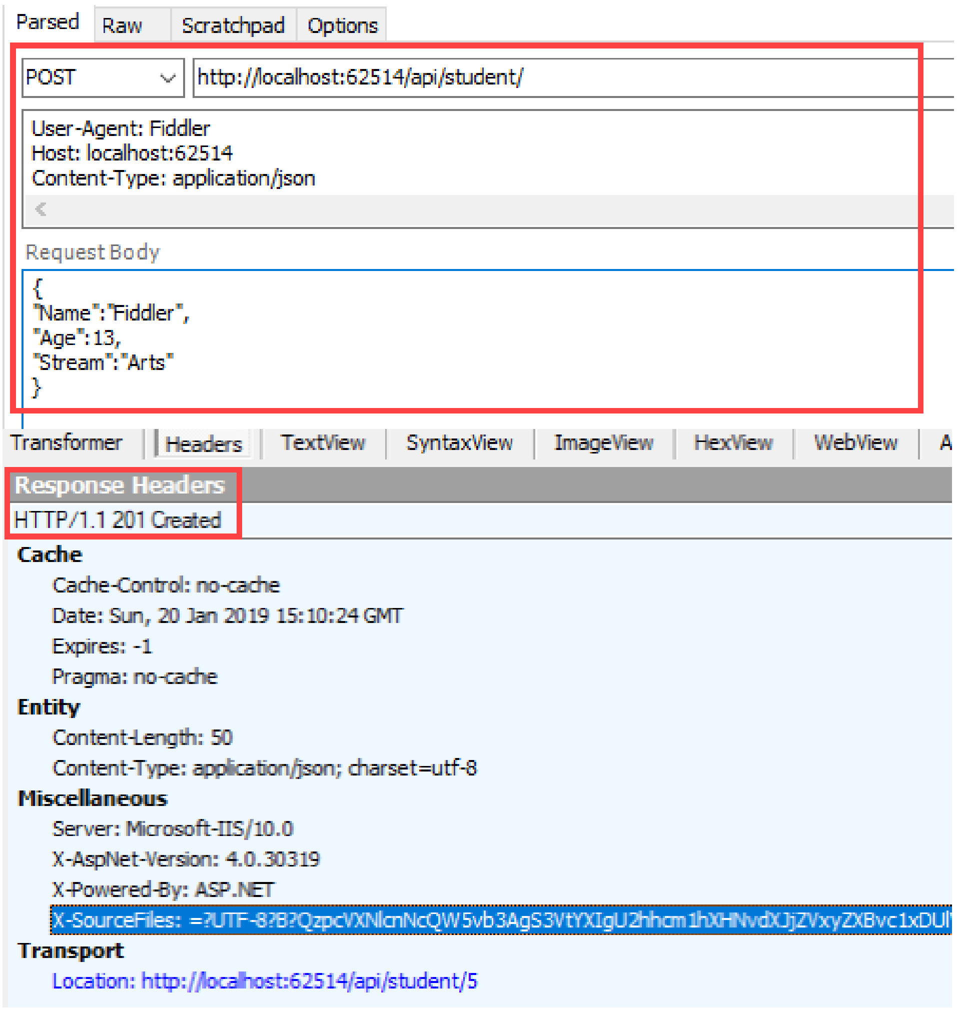
Image: Get only particular Student's Record in Fiddler

Q12. How to Post Data in ASP.NET Web API?

Ans. Let's include the POST method in the above example. Create a Post method with Post action name or other Custom Name with HTTP POST as an action verb. See the below code:

```
// POST
public IHttpActionResult Post([FromBody]Student student)
{
try
{
    using (WebAPIDBEntities objWebAPIDBEntities = new WebAPIDBEntities())
    {
        objWebAPIDBEntities.Students.Add(student);
        objWebAPIDBEntities.SaveChanges();
        //return HTTP Status Code Created after saving the student object
        return Created<Student>(Request.RequestUri+student.ID.ToString(),student);
    }
}
catch (Exception ex)
{
//return HTTP Status Code BadRequest if Data is not proper or an exception occur, with an
error message
    return BadRequest(ex.Message.ToString());
}
}
```

Let's test the above code with the fiddler.



```
POST    http://localhost:62514/api/student/

User-Agent: Fiddler
Host: localhost:62514
Content-Type: application/json

Request Body
{
"Name":"Fiddler",
"Age":13,
"Stream":"Arts"
}

Response Headers
HTTP/1.1 201 Created
Cache
    Cache-Control: no-cache
    Date: Sun, 20 Jan 2019 15:10:24 GMT
    Expires: -1
    Pragma: no-cache
Entity
    Content-Length: 50
    Content-Type: application/json; charset=utf-8
Miscellaneous
    Server: Microsoft-IIS/10.0
    X-AspNet-Version: 4.0.30319
    X-Powered-By: ASP.NET
    X-SourceFiles: =?UTF-8?B?QzpcVXNlcnNcQW5vb3AgS3VtYXIgU2hhcm1hXHNvdXJjZVxyZXBvc1xDU
Transport
    Location: http://localhost:62514/api/student/5
```

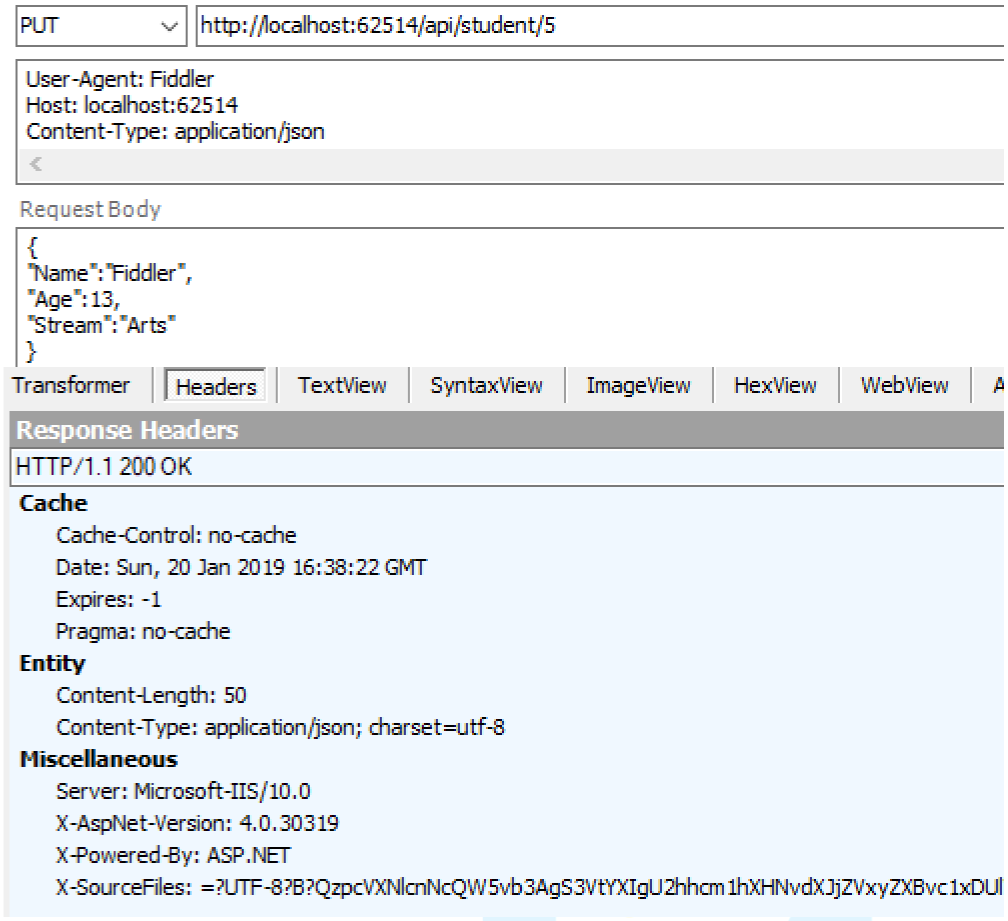
Image: Post Request Body in Web API in fiddler

Q13. How to Put or Update Data in ASP.NET Web API?

Ans. Let's add Put Action method in the Student API controller in order to demonstrate how can we update the data in ASP.NET Web API.

```
// PUT
public IHttpActionResult Put(int id, [FromBody]Student student)
{
    try
    {
        using (WebAPIDBEntities objWebAPIDBEntities = new WebAPIDBEntities())
        {
            var studentToUpdate = objWebAPIDBEntities.Students.FirstOrDefault(x => x.ID ==
id);
            if (studentToUpdate==null) {
            //return Error response with HTTP Status Code Notfound
            return NotFound();
            }
            else
            {
                studentToUpdate.Name = student.Name;
                studentToUpdate.Age = student.Age;
                objWebAPIDBEntities.SaveChanges();
                //return Http Status Code Ok after updating the student
                return Ok();
            }
        }
    }
    catch (Exception ex)
    {
    //return HTTP Status Code BadRequest if Data is not proper or an exception occur, with an
error message
    return BadRequest(ex.Message.ToString());
    }
}
```

Pass the ID from the FromURI and Student Object from FromBody in fiddle in order to test the above Put method code.



```
PUT    http://localhost:62514/api/student/5

User-Agent: Fiddler
Host: localhost:62514
Content-Type: application/json

Request Body
{
"Name":"Fiddler",
"Age":13,
"Stream":"Arts"
}
```

```
Response Headers
HTTP/1.1 200 OK
Cache
    Cache-Control: no-cache
    Date: Sun, 20 Jan 2019 16:38:22 GMT
    Expires: -1
    Pragma: no-cache
Entity
    Content-Length: 50
    Content-Type: application/json; charset=utf-8
Miscellaneous
    Server: Microsoft-IIS/10.0
    X-AspNet-Version: 4.0.30319
    X-Powered-By: ASP.NET
    X-SourceFiles: =?UTF-8?B?QzpcVXNlcnNcQW5vb3AgS3VtYXIgU2hhcm1hXHNvdXJjZVxyZXBvc1xDUl
```

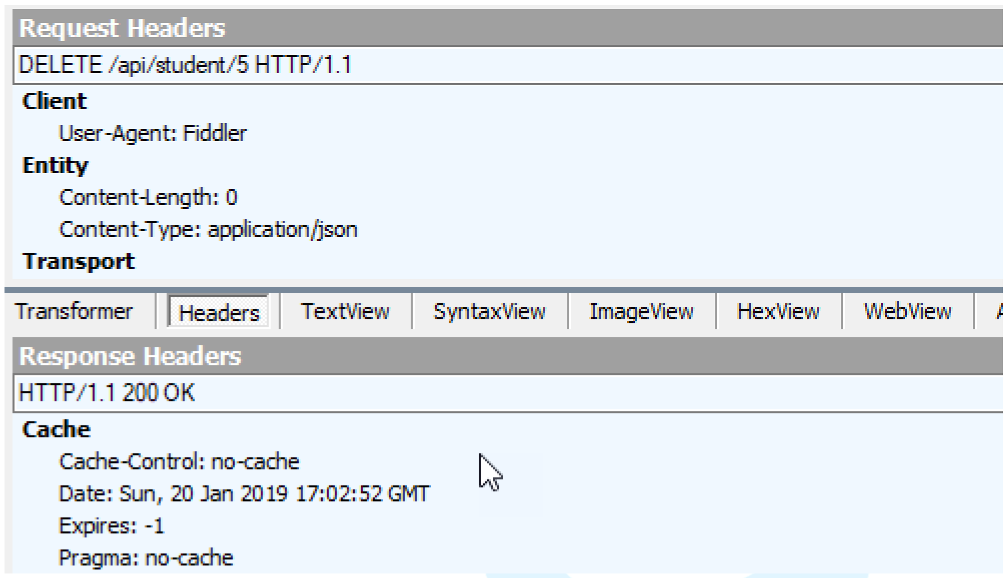
Image: Calling Put Action in Web API using fiddler

Q14. How to Delete Data in ASP.NET Web API?

Ans. Let's add Delete Action method in the Student API controller in order to demonstrate, how to delete data in ASP.NET Web API.

```
// DELETE
public IHttpActionResult Delete(int id)
{
    try
    {
        using (WebAPIDBEntities objWebAPIDBEntities = new WebAPIDBEntities())
        {
        var studentToDelete = objWebAPIDBEntities.Students.FirstOrDefault(x => x.ID == id);
        if (studentToDelete == null)
        {
            //return Error response with HTTP Status Code Notfound
            return NotFound();
        }
        else
        {
            objWebAPIDBEntities.Students.Remove(studentToDelete);
            objWebAPIDBEntities.SaveChanges();
            //return Http Status Code Ok after Deleting the student record
             return Ok();
          }
          }
     }
     catch (Exception ex)
     {
     //return HTTP Status Code BadRequest if Data is not proper or an exception occur, with an
error message
     return BadRequest(ex.Message.ToString());
     }
}
```

Let’s call the above Delete method using fiddler for testing.



```
Request Headers
DELETE /api/student/5 HTTP/1.1
Client
    User-Agent: Fiddler
Entity
    Content-Length: 0
    Content-Type: application/json
Transport
```

```
Response Headers
HTTP/1.1 200 OK
Cache
    Cache-Control: no-cache
    Date: Sun, 20 Jan 2019 17:02:52 GMT
    Expires: -1
    Pragma: no-cache
```

Image: Calling delete action of ASP.NET Web API using fiddler

3

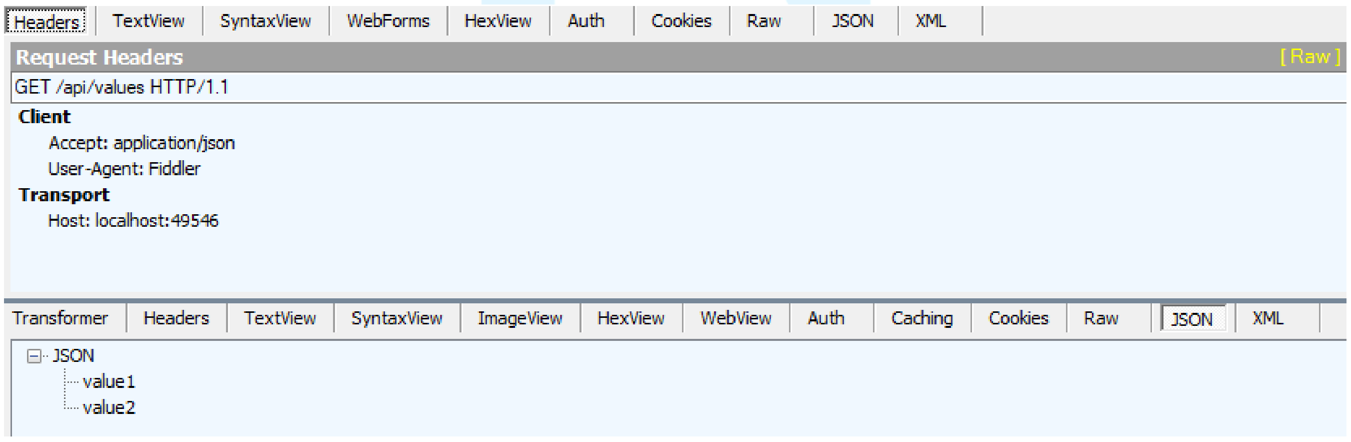
Content Negotiation

Q1. What is Content Negotiation in Web API?

Ans. Content Negotiation is the process of selecting the best representation for a given response when there are multiple representations available. Two main headers which are responsible for the Content Negotiation are:

1. Content-Type
2. Accept

The content-type header tells the server about the data, the server is going to receive from the client whereas another way to use Accept Header, which tells the format of data requested by the Client from a server. In the below example, we requested the data from the server in JSON format.



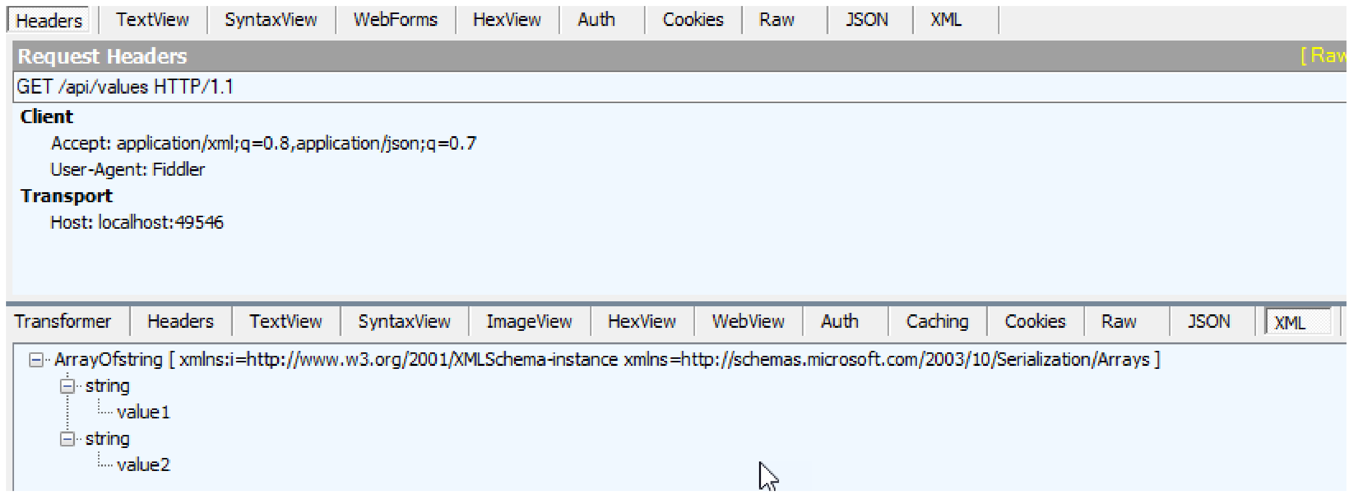
```
GET /api/values HTTP/1.1
Client
    Accept: application/json
    User-Agent: Fiddler
Transport
    Host: localhost:49546
```

```
JSON
    value1
    value2
```

Image: Request header to get JSON Response in Fiddler

Q2. What is the role of Quality Factor in Web API request?

Ans. If multiple Accept headers are passed in a Web API request, then the match which has a higher value of quality factor wins means content negotiator tries to provide the response in the format which has a high quality factor.



```
GET /api/values HTTP/1.1
Client
    Accept: application/xml;q=0.8,application/json;q=0.7
    User-Agent: Fiddler
Transport
    Host: localhost:49546
```

```
ArrayOfstring [ xmlns:i=http://www.w3.org/2001/XMLSchema-instance xmlns=http://schemas.microsoft.com/2003/10/Serialization/Arrays ]
    string
        value1
    string
        value2
```

Image: Pass multiple Accept Headers with a quality factor in Fiddler

In the above image, we passed multiple Accept headers with the different quality factor value. Application/xml has a higher q (i.e. Quality factor value), so response will be provided in the XML format.

Q3. If no Accept header will be specified, then the response will be in which format?

Ans. Web API provides JSON data as a response by default, in case if no Accept headers were provided.

Q4. What is Media-Type Formatter in ASP.NET Web API?

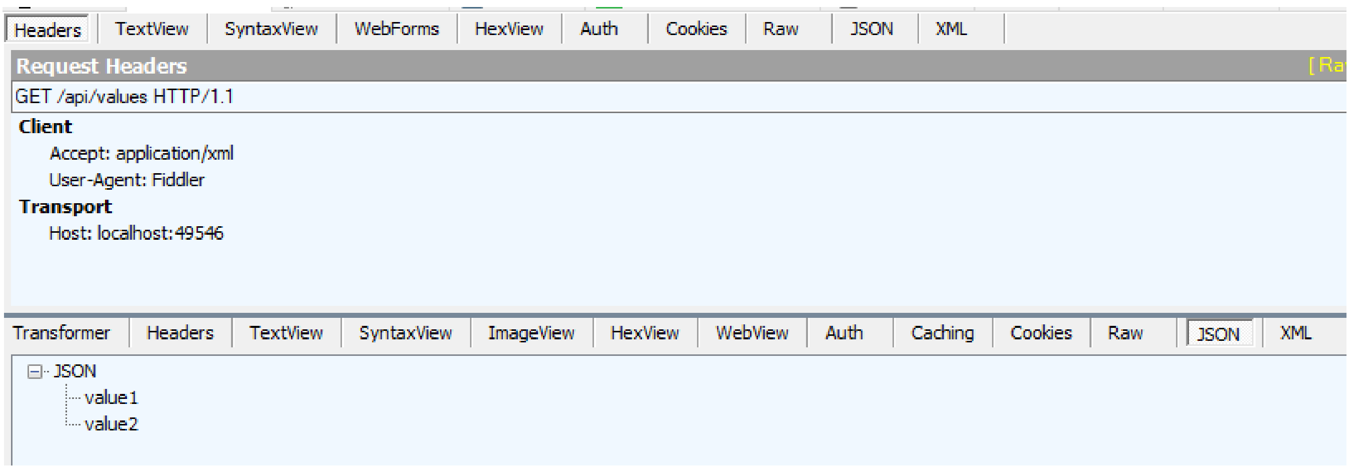
Ans. Media-Type formatter is an abstract class from which JsonMediaTypeFormatter (handle JSON format) and XmlMediaTypeFormatter (handle xml format) class derived from. Media-Type formatter are classes responsible for serializing the response data in the format that the client asked for.

Q5. How can we return data only in JSON or XML formatted data in Web API?

Ans. We can remove the formatters which we don't want by adding the below line in the WebApiConfig.cs file.

```
config.Formatters.Remove(config.Formatters.XmlFormatter);
```

In the above code, we have removed the XmlFormatter response from the code. So, if any person request for the Xml type Accept Header then he will only get the response in JSON format only.



```
Request Headers
GET /api/values HTTP/1.1
Client
    Accept: application/xml
    User-Agent: Fiddler
Transport
    Host: localhost:49546
```

```
JSON
    value1
    value2
```

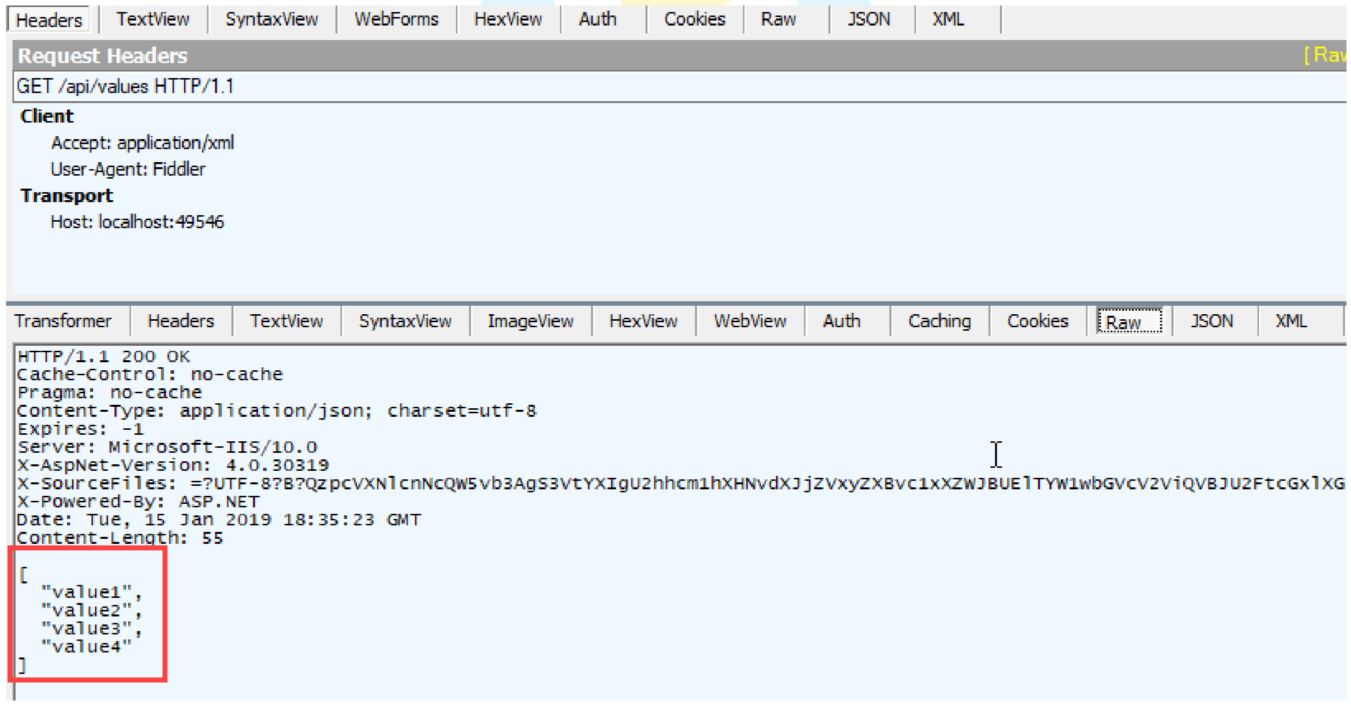
Image: JSON response received even on setting application/xml as Accept Header

Q6. How can we format or Indent the raw data in Web API?

Ans. We need to add the below line of code in WebApiConfig.cs file:

```
config.Formatters.JsonFormatter.SerializerSettings.Formatting =
Newtonsoft.Json.Formatting.Indented;
```

After calling the Get method of the API, you will see the raw response data is Indented accordingly.



```
Request Headers
GET /api/values HTTP/1.1
Client
    Accept: application/xml
    User-Agent: Fiddler
Transport
    Host: localhost:49546
```

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/10.0
X-AspNet-Version: 4.0.30319
X-SourceFiles: =?UTF-8?B?QzpcVXNlcnNcQW5vb3AgS3VtYXIgU2hhcm1hXHNvdXJjZVxyZXBvc1xXZWJBUElTYW1wbGVcV2ViQVBJU2FtcGxlXG
X-Powered-By: ASP.NET
Date: Tue, 15 Jan 2019 18:35:23 GMT
Content-Length: 55

[
  "value1",
  "value2",
  "value3",
  "value4"
]
```

Image: Indent raw JSON data in response in Fiddler.

4
Security

Q1. What is Authentication?

Ans. Authentication is knowing the identity of the user. It is about the validating the identity of the user which is accessing our system.

Q2. What is Authorization?

Ans. Authorization is deciding whether a user is allowed to perform an action or not. It is the second or the next step after the Authentication to achieve security.

Q3. What is the use of Authorize Attribute?

Ans. Web API provided a built-in authorization filter, i.e. Authorize Attribute. This filter checks whether the user is authenticated or not. If not, the user will see 401 Unauthorized HTTP Status Code.

Q4. What is Basic HTTP Authentication?

Ans. Basic HTTP Authentication is a mechanism, where the user is authenticated through the service in which the client pass username and password in the HTTP Authorization request headers. The credentials are formatted as the string “username:password”, based encoded.

Q5. How to create a custom Basic Authentication Web API filter?

Ans. Add a class BasicAuthenticationAttribute (you can create a class of any name) and inherit AuthorizationFilterAttribute class which provided details for the authorization filter. Override OnAuthorization method in the BasicAuthenticationAttribute class which will be called when a process request Authorization.

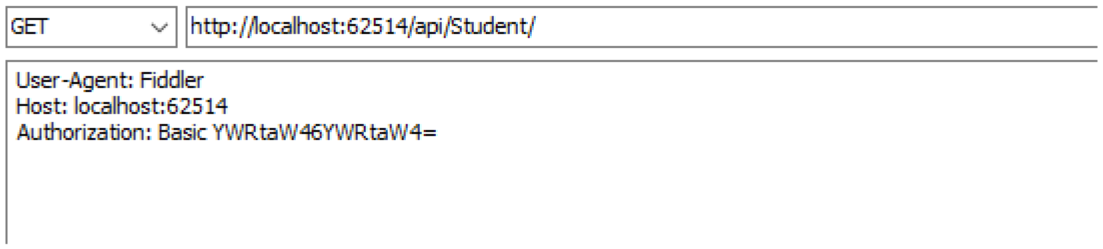
```
public class BasicAuthenticationAttribute: AuthorizationFilterAttribute
{
    public override void OnAuthorization(HttpActionContext actionContext)
    {
    //if no Authorization headers present in the Request Header, then return UnAuthorized 
Status Code
    if (actionContext.Request.Headers.Authorization==null) {
    actionContext.Response =actionContext.Request.CreateResponse(HttpStatusCode.Unauthorized);
    }
    else
    {
    //Get Authorization Token from the Request Headers
    string authenticationToken = actionContext.Request.Headers.Authorization.Parameter;
    //Decode Base64Encoded Authorization token to the string
    string decodedAuthToken
=Encoding.UTF8.GetString(Convert.FromBase64String(authenticationToken));
    if (decodedAuthToken.Split(':').Count()==2) {
    //Extracting UserName and Password from the decoded token
    string[] userNamePassword= decodedAuthToken.Split(':');
    string userName = userNamePassword[0];
    string password = userNamePassword[1];
    ValidateUser objValidateUser = new ValidateUser();
    if (objValidateUser.Login(userName,password)) {
    //If your application performs any custom authentication logic, you must set the principal
    Thread.CurrentPrincipal = new GenericPrincipal(new GenericIdentity(userName), null);
    if (HttpContext.Current!=null) {
    HttpContext.Current.User = Thread.CurrentPrincipal;
    }
    }
    else
    {
    actionContext.Response =actionContext.Request.CreateResponse(HttpStatusCode.Unauthorized);
    }
    }
    else
    {
    actionContext.Response =actionContext.Request.CreateResponse(HttpStatusCode.Unauthorized);
    }
    }
  }
}
```

In the above code, we have passed the username and password Login method of the ValidateUser class which is used to check or Validate the username and password. We can use the ValidateUser method of membership if the membership is implemented in the project.

Now, add the filter on the controller or the Action method which needs the Authentication before accessing the data. If user passes incorrect credentials, then it will return Unauthorized HTTP Status Code.

```
[BasicAuthenticationAttribute]
public class StudentController : ApiController
{
    // GET
    public IHttpActionResult Get()
    {
        using (WebAPIDBEntities objWebAPIDBEntities =new WebAPIDBEntities()) {
        //return All Student with HTTP Status Code OK
            return Ok(objWebAPIDBEntities.Students.ToList());
        }
    }
}
```

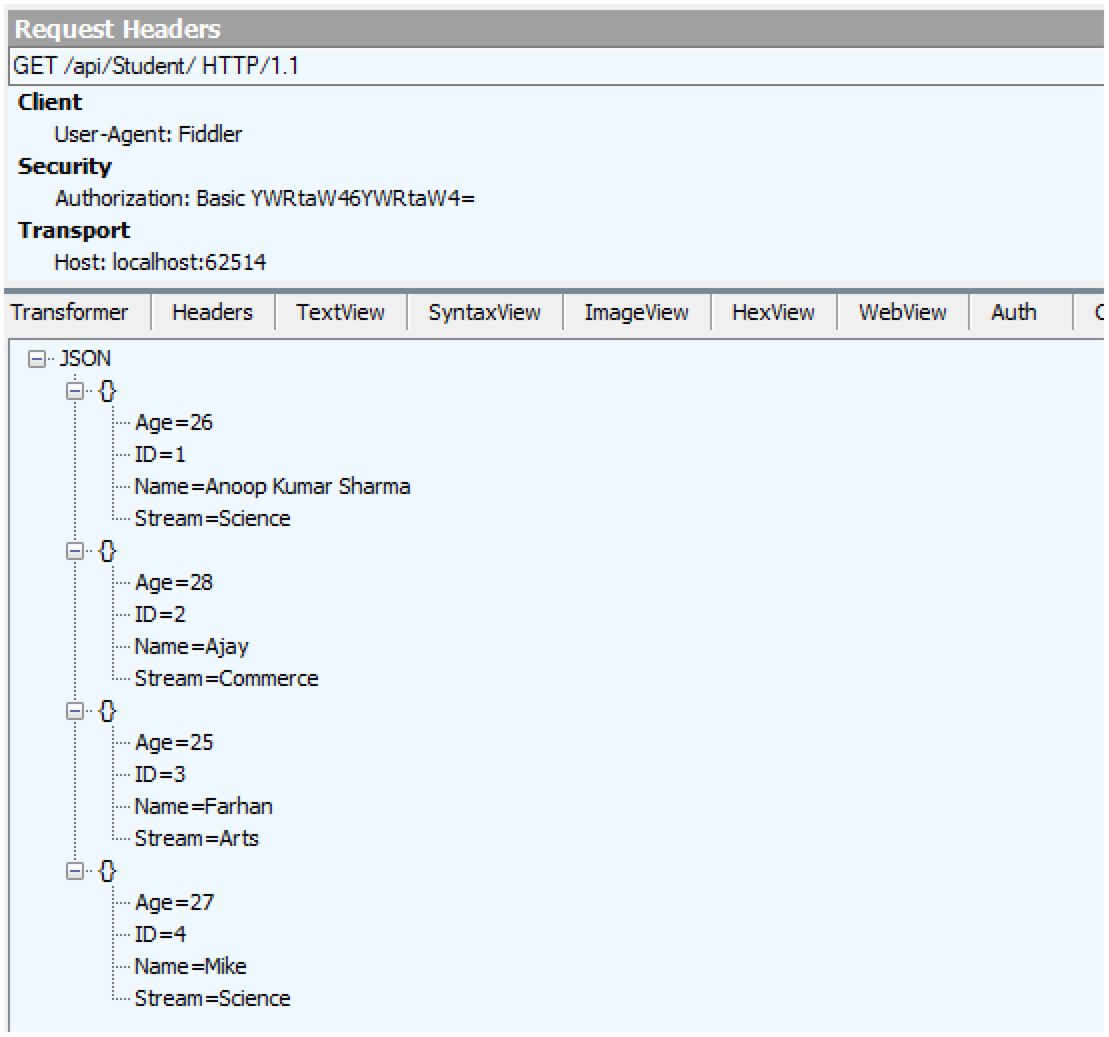
Let’s test the above code with the fiddler.



```
GET   http://localhost:62514/api/Student/

User-Agent: Fiddler
Host: localhost:62514
Authorization: Basic YWRtaW46YWRtaW4=
```

Image: Pass Authorization Header with Base64 encoded username and password

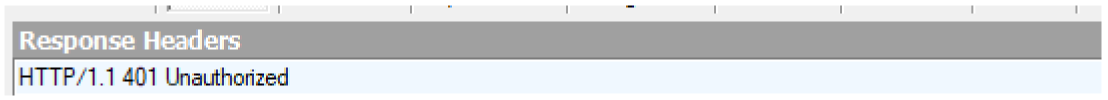


```
Request Headers
GET /api/Student/ HTTP/1.1
Client
    User-Agent: Fiddler
Security
    Authorization: Basic YWRtaW46YWRtaW4=
Transport
    Host: localhost:62514
```

```
JSON
  {}
    Age=26
    ID=1
    Name=Anoop Kumar Sharma
    Stream=Science
  {}
    Age=28
    ID=2
    Name=Ajay
    Stream=Commerce
  {}
    Age=25
    ID=3
    Name=Farhan
    Stream=Arts
  {}
    Age=27
    ID=4
    Name=Mike
    Stream=Science
```

Image: Response on passing correct credentials in fiddler

If the incorrect credential is passed then you will get the unauthorized status code in response.



```
Response Headers
HTTP/1.1 401 Unauthorized
```

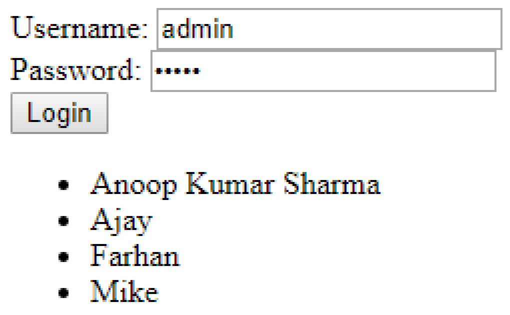
DotNetTricks

Q6. How to consume or call Web API with Basic HTTP Authentication using jQuery & Ajax if need to consume in the same Domain?

Ans. In the below code, we are getting the value of username and password fields on the button click event and send these details with the Authorization header to the client. If Username/Password doesn't correct, then the system shows an error to the user.

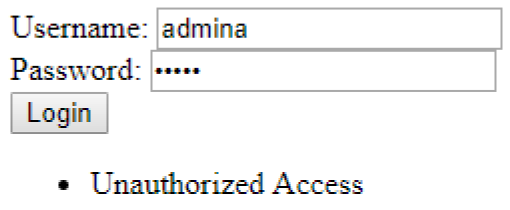
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title></title>
    <script src="../Scripts/jquery-3.3.1.js"></script>
    <script>
        $(document).ready(function(){
            $('#btnLogin').click(function () {
                //Get username and password from the textbox
                var username = $('#txtUserName').val()
                var password = $('#txtPassword').val()
                //Send Base64 encoded username:password in request
                //On Success, show the Data to the User else show an error to the user
                $.ajax({
                    type: 'GET',
                    url: 'http://localhost:62514/api/Student',
                    dataType: 'json',
                    headers: {
                        'Authorization': 'Basic ' + btoa(username + ":" + password)
                    },
                    success: function (data) {
                        $('#ulOutPut').empty();
                        $.each(data, function (index, val) {
                            $('#ulOutPut').append('<li>'+val.Name+'</li>');
                        });
                    },
                    complete: function (jqXHR) {
                        if (jqXHR.status == '401') {
                            $('#ulOutPut').empty();
                            $('#ulOutPut').append('<li>Unauthorized Access</li>');
                        }
                    }
                });
            });
        });
    </script>
</head>
<body>
    <div>
        Username: <input type="text" id="txtUserName"/><br/>
        Password: <input type="password" id="txtPassword"/><br />
        <button id="btnLogin">Login</button>
    </div>
    <ul id="ulOutPut">

    </ul>
</body>
</html>
```



- Anoop Kumar Sharma
- Ajay
- Farhan
- Mike

Image: On passing correct credentials, you will get the data as shown in the above image as a response



- Unauthorized Access

Image: On passing incorrect credentials, you will get an unauthorized error on the view.

Q7. What are the Advantages and Disadvantages of Basic Authentication?

Ans. Advantages of Basic Authentication are:

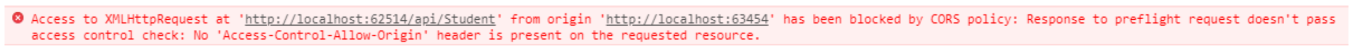
1. Supported by All major browsers.
2. Simple Protocol
3. Internet Standard

The disadvantage of Basic Authentication are:

1. User Password is sent in each request as plain text.
2. No way to logout except by ending browser session.
3. Vulnerable to CSRF i.e. cross-site request forgery.

Q8. What is the Same-Origin Policy?

Ans. Browser security prevents a web page from making an AJAX request to another domain. This restriction is known as Same-Origin Policy. It prevents a malicious site from reading the data from another website.



```
Access to XMLHttpRequest at 'http://localhost:62514/api/Student' from origin 'http://localhost:63454' has been blocked by CORS policy: Response to preflight request doesn't pass
access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.
```

Image: Error message on request has been blocked by the CORS Policy in Console of Developer Tools in Web Browser

In case, Client wants to hit the Web Service or API from the other domain, then CORS i.e. Cross-Origin Resource Sharing play an important role. Using CORS, a server can explicitly allow some cross-origin request while rejecting others.

Q9. How to enable CORS in Web API project?

Ans. First, add the reference for the CORS from the NuGet Package manager. Right click on References, and click on Manage NuGet Package Manager and search for the CORS as shown in below image.

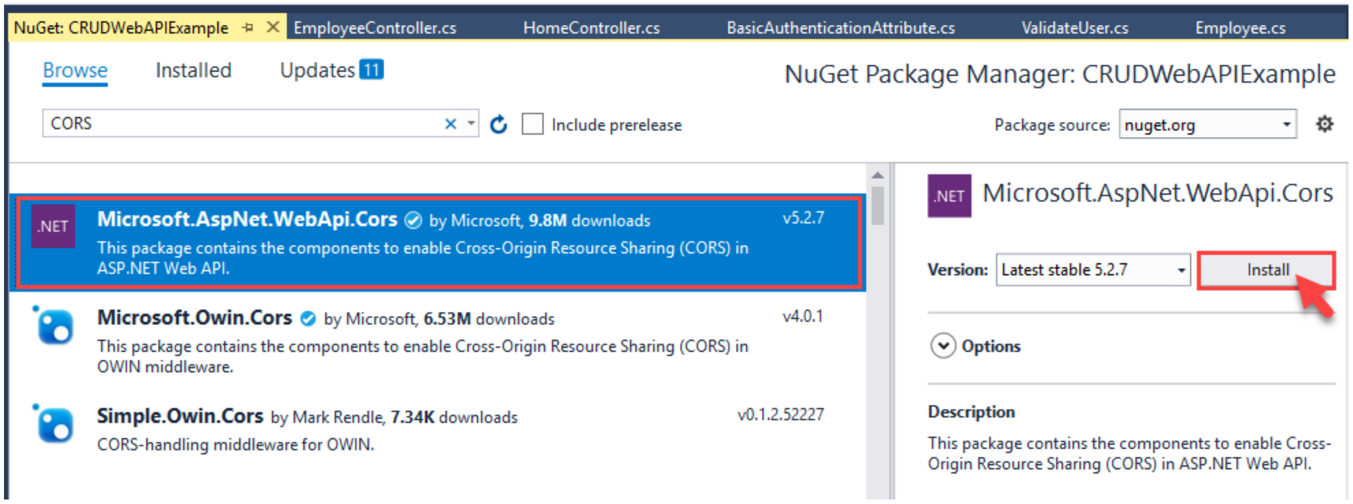

NuGet: CRUDWebAPIExample
EmployeeController.cs
HomeController.cs
BasicAuthenticationAttribute.cs
ValidateUser.cs
Employee.cs
Browse
Installed
Updates 11
NuGet Package Manager: CRUDWebAPIExample
CORS
Include prerelease
Package source: nuget.org
Microsoft.AspNet.WebApi.Cors by Microsoft, 9.8M downloads v5.2.7
This package contains the components to enable Cross-Origin Resource Sharing (CORS) in ASP.NET Web API.
Microsoft.Owin.Cors by Microsoft, 6.53M downloads v4.0.1
This package contains the components to enable Cross-Origin Resource Sharing (CORS) in OWIN middleware.
Simple.Owin.Cors by Mark Rendle, 7.34K downloads v0.1.2.52227
CORS-handling middleware for OWIN.
Microsoft.AspNet.WebApi.Cors
Version: Latest stable 5.2.7
Install
Options
Description
This package contains the components to enable Cross-Origin Resource Sharing (CORS) in ASP.NET Web API.

Image: Install Microsoft.AspNet.WebApi.Cors dll from the NuGet Package Manager

Or we can also install the package from the NuGet package manager console by using the below command:

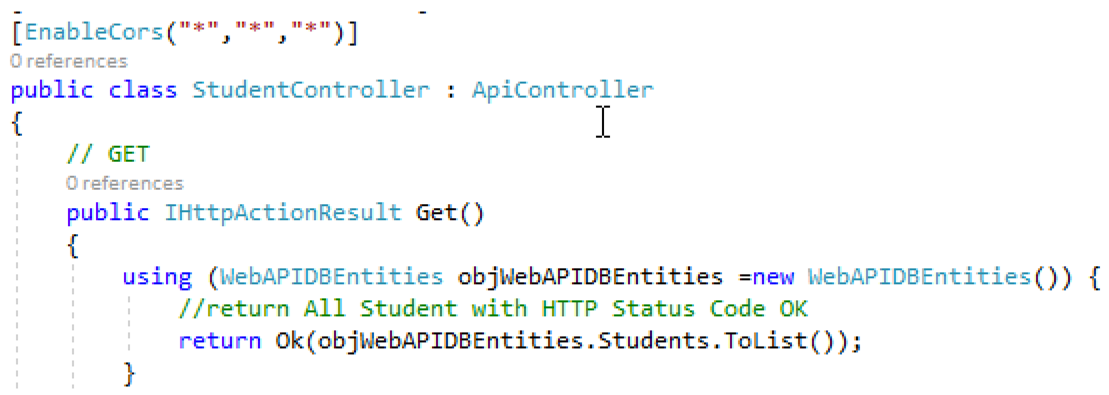
```
Install-Package Microsoft.AspNet.WebApi.Cors
```

After that add the below lines in the WebApiConfig.cs file to enable CORS.

```
EnableCorsAttribute cors = new EnableCorsAttribute("*", "*", "*");
config.EnableCors(cors);
```

EnableCorsAttribute Class accepts origin, headers, methods, etc. in the constructor of the class. Wildcard value "*" in origin means that any origin, header, methods are allowed. In the request header, we can check the Origin of the request and in the response header, we can check the Access-Control Header. We can only allow CORS for the particular domain, method, as well as header.

We can also enable CORS on the controller or Action method level. Make sure that config.EnableCors() method is still present in the WebApiConfig.cs file. Decorate the controller with the EnableCors as shown below:



```
[EnableCors("*","*","*")]
0 references
public class StudentController : ApiController
{
    // GET
    0 references
    public IHttpActionResult Get()
    {
        using (WebAPIDBEntities objWebAPIDBEntities =new WebAPIDBEntities()) {
            //return All Student with HTTP Status Code OK
            return Ok(objWebAPIDBEntities.Students.ToList());
        }
```

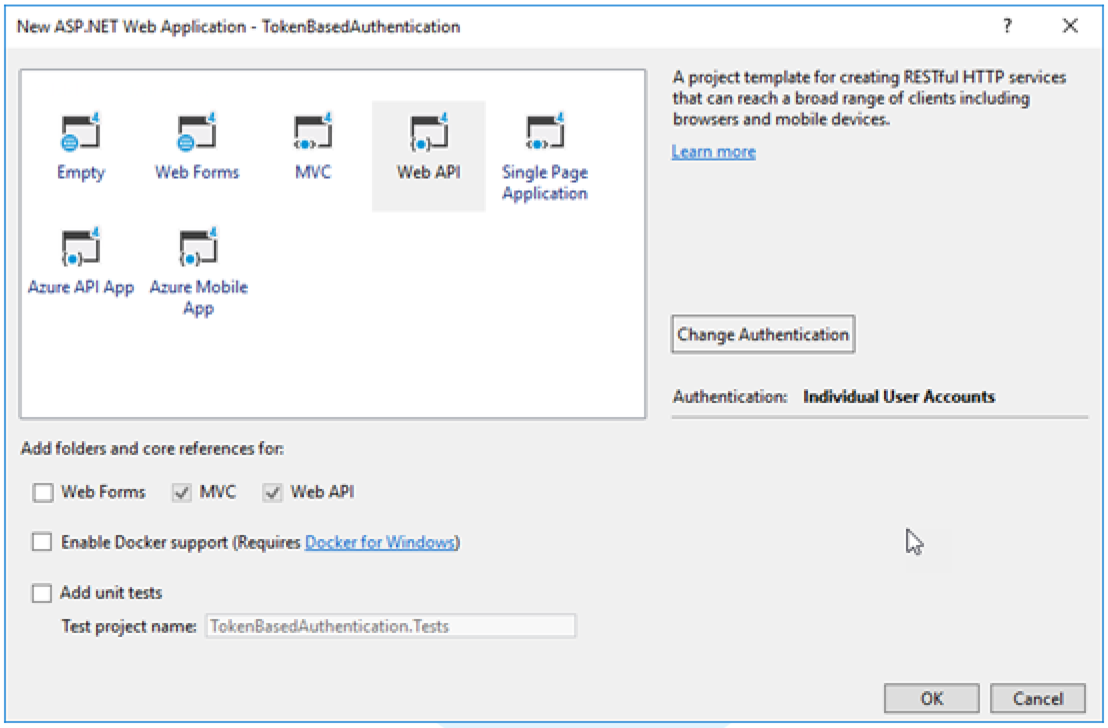
Q10. How can we Disable CORS for an Action only in Web API project?

Ans. Use [DisableCors] attribute to the action in order to disable CORS for an action method.

Q11. How to implement Token-based Authentication in Web API project?

Ans. In Token Based Authentication, Username and Password of a user is validated on the server and a token is generated as a response which is stored on the client side and used in Authorization header in the request which needs to be Authenticated. Let's see how to implement the Token-based authentication in detail:

1. Create a New Web API project and Select the Authentication as Individual User Accounts as we will use the Account controller to register and login the user in the application.



New ASP.NET Web Application - TokenBasedAuthentication

A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Authentication: **Individual User Accounts**

Add folders and core references for:

- [ ] Web Forms
- [✓] MVC
- [✓] Web API
- [ ] Enable Docker support (Requires Docker for Windows)
- [ ] Add unit tests

Test project name: TokenBasedAuthentication.Tests

2. Once the application is created, open web.config file and change the connection string as per your requirement. Open Startup.Auth.cs, you will see ConfigureAuth methods which accept the IAppBuilder interface to manage the middleware for the application. OAuthAuthorizationServerOptions class provides the information needed to control Authorization server middleware behaviour.

```
public void ConfigureAuth(IAppBuilder app)
{
    // Configure the db context and user manager to use a single instance per request
    app.CreatePerOwinContext(ApplicationDbContext.Create);
    app.CreatePerOwinContext<ApplicationUserManager>(ApplicationUserManager.Create);

    // Enable the application to use a cookie to store information for the signed in
    user
    // and to use a cookie to temporarily store information about a user logging in
    with a third party login provider
    app.UseCookieAuthentication(new CookieAuthenticationOptions());
    app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);

    // Configure the application for OAuth based flow
    PublicClientId = "self";
    OAuthOptions = new OAuthAuthorizationServerOptions
    {
        TokenEndpointPath = new PathString("/Token"),
        Provider = new ApplicationOAuthProvider(PublicClientId),
        AuthorizeEndpointPath = new PathString("/api/Account/ExternalLogin"),
        AccessTokenExpireTimeSpan = TimeSpan.FromDays(14),
        // In production mode set AllowInsecureHttp = false
        AllowInsecureHttp = true
    };

    // Enable the application to use bearer tokens to authenticate users
    app.UseOAuthBearerTokens(OAuthOptions);
}
```

The request path client applications communicate with directly as part of the OAuth protocol. It accepts pathstring which must start with a slash like “/Token”. **AccessTokenExpireTimeSpan** used to define the validity of the token. AllowInsecureHttp is a boolean property used to allow authorize and token requests to arrive on HTTP URI addresses.

3. Account controller has a Register method to register the user. Below is the HTML and jQuery code

```
<h2>Register</h2>
    <table>
    <tbody>
    <tr>
        <td>Email: </td>
        <td><input type="text" id="txtEmail" /></td>
    </tr>
    <tr>
        <td>Password: </td>
        <td><input type="password" id="txtPassword" /></td>
    </tr>
    <tr>
        <td>Confirm Password: </td>
        <td><input type="password" id="txtConfirmPassword" /></td>
    </tr>
    <tr>
        <td></td>
        <td><input type="button" id="btnRegister" value="Register" /></td>
    </tr>
    <tr>
        <td colspan="2">
        <div id="divRegMessage"></div></td>
    </tr>
    </tbody>
</table>
```

Register UI Preview will be something like below:

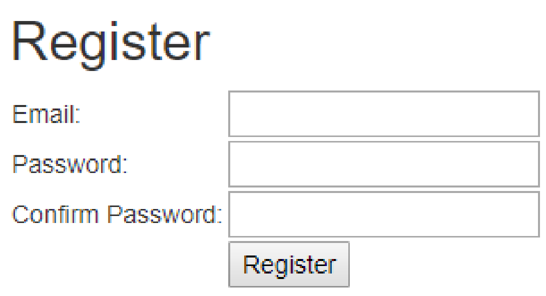


Image: Sample User Interface for User Registration

On clicking on register button, we will save the user details in the database with the help of API and Ajax call.

```
$(document).ready(function () {

//Method to Register
$('#btnRegister').click(function () {
    $.ajax({
        url: 'http://localhost:61515/api/Account/Register',
        method: 'Post',
        data: {
            Email: $('#txtEmail').val(),
            Password: $('#txtPassword').val(),
            ConfirmPassword: $('#txtConfirmPassword').val()
        },
        success: function () {
            $('#divRegMessage').text("User Registered Successfully!");
            $('#txtEmail').val() = '';
            $('#txtPassword').val() = '';
            $('#txtConfirmPassword').val() = '';
        },
        error: function (error) {
            $('#divRegMessage').text(error.responseText);
        }
    });
});
});
```

4. Let's create a User interface so that user can login from that interface.

```
<h2>Login</h2>
    <table>
    <tr>
        <td>Email: </td>
        <td><input type="text" id="txtLoginEmail" /></td>
    </tr>
    <tr>
        <td>Password: </td>
        <td><input type="text" id="txtLoginPassword" /></td>
    </tr>
    <tr>
        <td></td>
        <td><input type="button" id="btnLogin" value="Login" /></td>
    </tr>
    <tr>
        <td colspan="2">
            <div id="divLoginMessage"></div>
        </td>
    </tr>
</table>
```

On login user interface, we will hit the token method with the username, password, grant_type in an Ajax call.

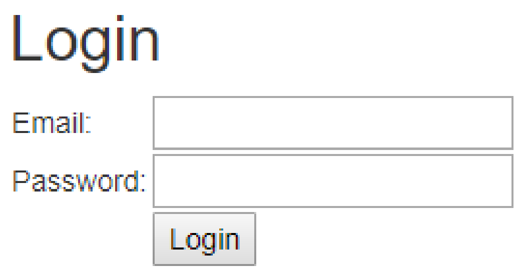


Image: Login view for Token-Based Authentication

If successfully authenticated, we will store the accessToken in the sessionStorage else show the error message to the user. We can also redirect user to the page on which the client needs some data.

```
$(document).ready(function () {

$('#btnLogin').click(function () {
    $.ajax({
        url: 'http://localhost:61515/token',
        method: 'Post',
        data: {
            username: $('#txtLoginEmail').val(),
            password: $('#txtLoginPassword').val(),
            grant_type: 'password'
        },
        success: function (response) {
            sessionStorage.setItem("accessToken", response.access_token);
        },
        error: function (error) {
            $('#divLoginMessage').text(error.responseText);
        }
    });
});
});
```

5. In order test Authorize functionality, we have created ValuesController which is decorated with [Authorize] attribute. If the user directly tries to call that method, the user will get an error message i.e. “Authorization has been denied for this request” with 401 HTTP Status Code Unauthorized.

```
[Authorize]
public class ValuesController : ApiController
{
    // GET api/values
    List<Student> lstStudent = new List<Student>() {
        new Student{ID=1,Name="Anoop Sharma",state="Delhi",Country="India" },
        new Student{ID=2,Name="Mark",state="Pubjab",Country="India" }
    };
    public IHttpActionResult Get()
    {
        return Ok(lstStudent);
    }
}
```

And a user interface with the load button which will check the sessionStorage(“accessToken”) stored in browser after the successful Authentication.

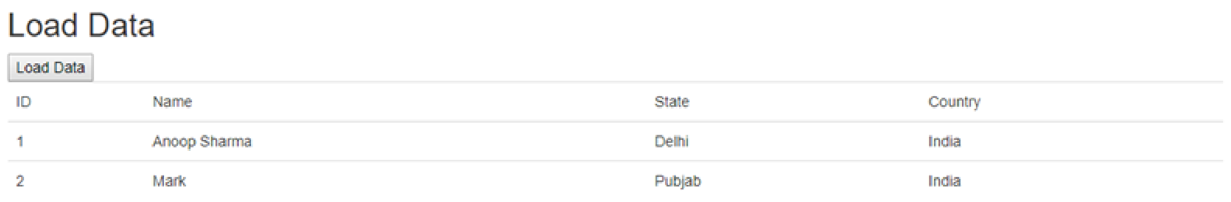
```
<h2>Load Data</h2><input type="button" id="btnLoadData" value="Load Data" />
<div id="divData">
</div>
```

If this sessionStorage is not present then show a failure message to the user. If present, then sent that Token to the Authorization header.

```
$(document).ready(function () {

    $('#btnLoadData').click(function () {
    $('#divData').text('');
    if (sessionStorage.getItem("accessToken") == null) {
        alert('Token Expired');
    return false;
    }
    $.ajax({
        url: 'http://localhost:61515/api/values',
        method: 'Get',
        headers: {
            'Authorization': 'Bearer ' + sessionStorage.getItem("accessToken")
        },
        success: function (result) {
            var data = "<table class='table'><tr><td>ID</td><td>Name</td>
<td>State</td> <td>Country</td></tr>";
            $.each(result, function (index, value) {
                data += "<tr><td>" + value.ID + "</td><td>" + value.Name
                         + "</td><td>" + value.state + "</td><td>"
                         + value.Country + "</td></tr>";
            });
            data += "</table>";
            $('#divData').html(data);
        },
        error: function (error) {
            $('#divData').text(error.responseText);
        }
        });
    });

});
```



Load Data

Load Data

ID	Name	State	Country
1	Anoop Sharma	Delhi	India
2	Mark	Pubjab	India

Image: Load Data if the user is Authenticated using Token based authentication

5
Routing

Q1. How Web API Routes HTTP request to the Controller ASP.NET MVC?

Ans. In ASP.NET Web API, HTTP request maps to the controller. In order to determine which action is to invoke, the Web API framework uses a routing table.

Q2. How to define a route in ASP.NET MVC?

Ans. Web API supports two types of routing i.e.

1. Convention-based routing
2. Attribute Routing

When we create a Web API project, it will add WebApiConfig.cs class in App_Start folder with the default route values as shown below:

```
public static void Register(HttpConfiguration config)
{
        // Web API configuration and services

        // Web API routes
        config.MapHttpAttributeRoutes();

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

Each entry in the routing table contains a route template. The Default route template for the Web API project is "api/{controller}/{id}. As we can see in the template that "api" is a string literal path segment (provided by default in the route in order to prevent collision with the ASP.NET MVC Routing. If don't like this default literal path then you can change the default route template) and {controller" and {id} are placeholder variables. When Web API receives an HTTP request, it will try to match the URI with one of the route template defined in the routing table. In case, if not matched with any of the route templates then the client receives 404 Not found error.

Q3. How to include Action Name in the routing template?

Ans. In Web API, the default route template uses the HTTP Verbs to select the action. We can also use the Action Name in route so that URI with the action name can call the respective action of the controller. For that, we have to change the default route template in the WebConfig.cs file to as shown below:

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

In the above route template, the {action} parameter names the action method of the controller. We can also override the name of the Action Name attribute on Action name as shown below:

```
public class StudentController : ApiController
{
    List<Students> lstStudent = new List<Students>()
    {
        new Students{ ID=1,Name="Anoop Sharma",Age=26,City="New Delhi" },
        new Students{ ID=2,Name="Mike",Age=28,City="New Delhi" },
        new Students{ ID=3,Name="Lance",Age=24,City="Turkey" }
    };

    [HttpGet]
    [ActionName("GetStudent")]
    public IHttpActionResult Get()
    {
        return Ok(lstStudent);
    }

    [HttpGet]
    [ActionName("GetStudentByID")]
    public IHttpActionResult Get(int ID)
    {
        return Ok(lstStudent.FirstOrDefault(x=>x.ID==ID));
    }

    [HttpGet]
    [ActionName("GetStudentByName")]
    public IHttpActionResult Get(string Name)
    {
        return Ok(lstStudent.FirstOrDefault(x => x.Name== Name));
    }
}
```

Q4. How to prevent an Action Method to be invoked in Web API?

Ans. Use [NonAction] attribute, in order to prevent an action method to be invoked in Web API.

Q5. What is Attribute based routing and How can we enable it in a Web API project?

Ans. Web API 2 supports a new type of routing known as Attribute based routing, which uses attributes to define the routes. Attribute routing gives more control over the URI's in the Web API. Attribute based routing is enabled by default in the Web API project. You can also enable it by adding the below lines in the Register method of the WebApiConfig.cs. To enable the routing, call MapHttpAttributeRoutes during configuration. This extension method is defined in the System.Web.Http.HttpConfigurationExtensions class.

```
// Web API routes
config.MapHttpAttributeRoutes();
```

Add the Route Attribute in the Action name with the route template as a parameter.

```
public class StudentController : ApiController
{
    List<Student> lstStudent = new List<Student>()
    {
        new Student{ ID=1,Name="Anoop Sharma",Age=26,City="New Delhi" },
        new Student{ ID=2,Name="Mike",Age=28,City="New Delhi" },
        new Student{ ID=3,Name="Lance",Age=24,City="Turkey" }
    };
    List<Teacher> lstTeacher = new List<Teacher>() {
        new Teacher{ID=1,Name="Mark",lstStudentID= new List<int>{2,3}},
        new Teacher{ID=2,Name="Spencer",lstStudentID= new List<int>{1,2,3}}
    };

    [Route("api/Student/")]
    public IHttpActionResult Get()
    {
        return Ok(lstStudent);
    }
    [Route("api/Student/{ID}")]
    public IHttpActionResult GetStudentByID(int ID)
    {
        return Ok(lstStudent.FirstOrDefault(x => x.ID == ID));
    }
    [Route("api/StudentsUnderTeacher/{ID}")]
    public IHttpActionResult GetStudentsUnderTeacher(int ID)
    {
        return Ok(lstTeacher.Where(x => x.ID== ID));
    }
}
```

Q6. What is the use of RoutePrefix attribute?

Ans. RoutePrefix attribute is used to set a common route prefix for an entire controller. It is used when the route of a controller starts with the same prefix.

```
[RoutePrefix("api/Student")]
public class StudentController : ApiController
{
    List<Student> lstStudent = new List<Student>()
    {
        new Student{ ID=1,Name="Anoop Sharma",Age=26,City="New Delhi" },
        new Student{ ID=2,Name="Mike",Age=28,City="New Delhi" },
        new Student{ ID=3,Name="Lance",Age=24,City="Turkey" }
    };
    List<Teacher> lstTeacher = new List<Teacher>() {
        new Teacher{ID=1,Name="Mark",lstStudentID= new List<int>{2,3}},
        new Teacher{ID=2,Name="Spencer",lstStudentID= new List<int>{1,2,3}}
    };
    [Route("")]
    public IHttpActionResult Get()
    {
        return Ok(lstStudent);
    }
    [Route("{ID}")]
    public IHttpActionResult GetStudentByID(int ID)
    {
        return Ok(lstStudent.FirstOrDefault(x => x.ID == ID));
    }
    [Route("~/api/StudentsUnderTeacher/{ID}")]
    public IHttpActionResult GetStudentsUnderTeacher(int ID)
    {
        return Ok(lstTeacher.Where(x => x.ID== ID));
    }
}
```

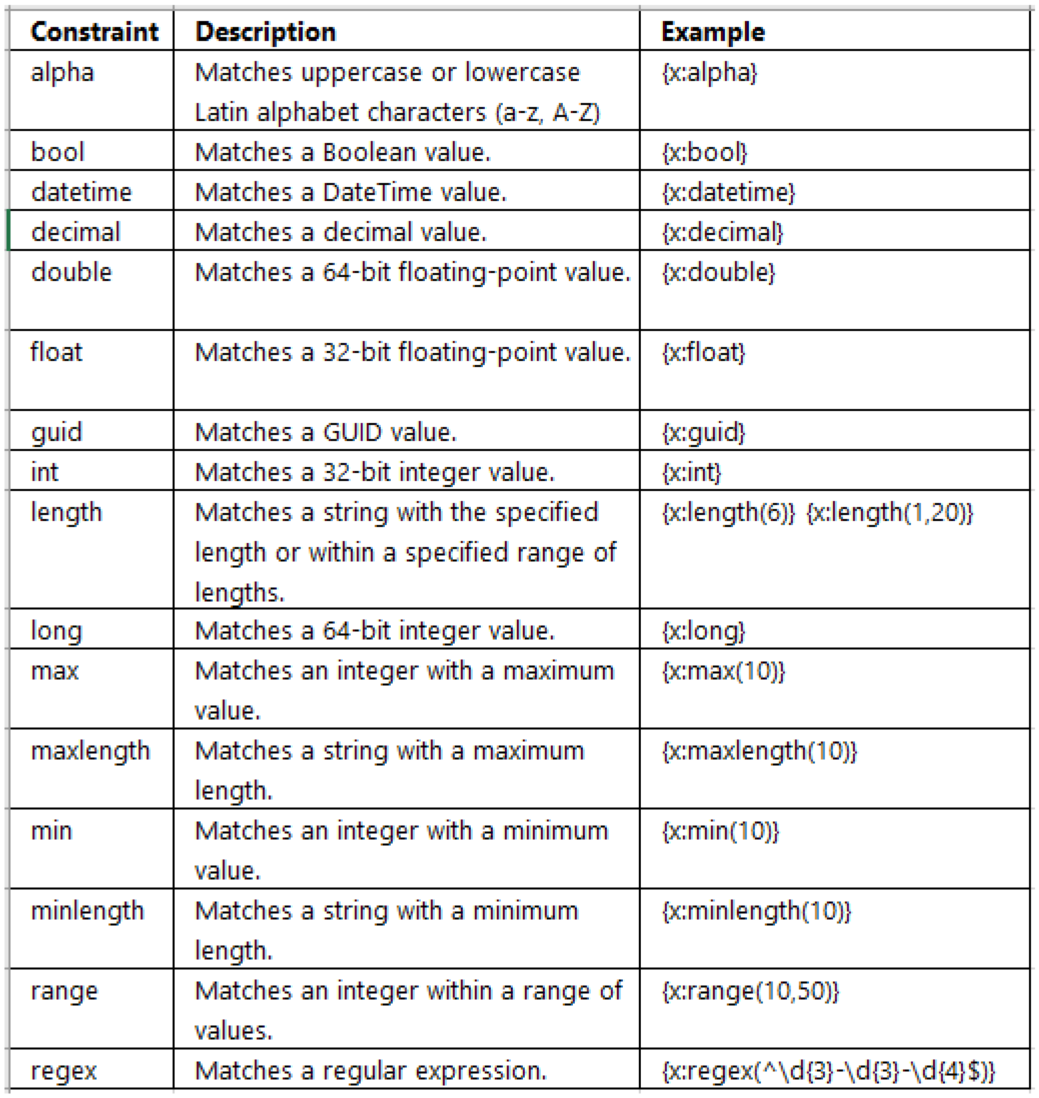
Use a tilde symbol (as we used on `[Route("~/api/StudentsUnderTeacher/{ID}")]`) on the method attribute in order to override the route prefix.

Q7. What are the Route Constraints?

Ans. Route constraints let you restrict how the parameters in the route template are matched. The syntax for the route constraints is “{parameter:constraint}”.

```
public class StudentController : ApiController
{
    List<Student> lstStudent = new List<Student>()
    {
        new Student{ ID=1,Name="Anoop Sharma",Age=26,City="New Delhi" },
        new Student{ ID=2,Name="Mike",Age=28,City="New Delhi" },
        new Student{ ID=3,Name="Lance",Age=24,City="Turkey" }
    };
    //Alpha: Matches uppercase or lowercase Latin alphabet characters (a-z, A-Z)
    [Route("api/Get/{name:alpha}")]
    public IHttpActionResult GetStudentByName(string name)
    {
        return Ok(lstStudent.Where(x=>x.Name==name));
    }
    //int: Matches a 32-bit integer value and with a minimum value 1.
    [Route("api/Get/{ID:int:min(1)}")]
    public IHttpActionResult GetStudentByID(int ID)
    {
        return Ok(lstStudent.FirstOrDefault(x => x.ID == ID));
    }
}
```

In the above example, we have set the id parameter to be the type of int 32 with min value to be 1 and name of type string. Web API supports different type of route constraints. Check the below list of the Route Constraints that are supported by Web API:



Constraint	Description	Example
alpha	Matches uppercase or lowercase Latin alphabet characters (a-z, A-Z)	{x:alpha}
bool	Matches a Boolean value.	{x:bool}
datetime	Matches a DateTime value.	{x:datetime}
decimal	Matches a decimal value.	{x:decimal}
double	Matches a 64-bit floating-point value.	{x:double}
float	Matches a 32-bit floating-point value.	{x:float}
guid	Matches a GUID value.	{x:guid}
int	Matches a 32-bit integer value.	{x:int}
length	Matches a string with the specified length or within a specified range of lengths.	{x:length(6)} {x:length(1,20)}
long	Matches a 64-bit integer value.	{x:long}
max	Matches an integer with a maximum value.	{x:max(10)}
maxlength	Matches a string with a maximum length.	{x:maxlength(10)}
min	Matches an integer with a minimum value.	{x:min(10)}
minlength	Matches a string with a minimum length.	{x:minlength(10)}
range	Matches an integer within a range of values.	{x:range(10,50)}
regex	Matches a regular expression.	{x:regex(^\d{3}-\d{3}-\d{4}$)}

Image: List of Route Constraint supported by the Web API

Q8. Can we use both Convention based routing and attribute routing in the same Web API application?

Ans. Yes, we can use convention-based routing and attribute routing in the same Web API project without any issue.

6
Web API Versioning

Q1. Why we need Web API Versioning?

Ans. As Business Requirement and Business grow with the time, Versioning of the Web API is required. Versioning of Web API ensures that no existing client will be impacted because of those business changes.

Q2. In How many ways we can do Web API Versioning?

Ans. We can do Web API Versioning in the following ways:

1. URI
2. Query String Parameter
3. Custom Header Parameter
4. Accept Header Parameter

Q3. How to achieve Web API Versioning with the URI?

Ans. Suppose we have two Controller, i.e. EmployeeV1Controller which returns an EmployeeV1 object with ID, Name, Age, City and State properties and EmployeeV2Controller which returns an EmployeeV2 object with ID, FirstName, LastName, DOB, City, State, Country etc. properties.

EmployeeV1 Controller:

```
public class EmployeeV1Controller : ApiController
{
    List<EmployeeV1> lstEmployeeV1 = new List<EmployeeV1>() {
        new EmployeeV1(){ ID=1,Name="Anoop Sharma",Age=26,City="New Delhi",State="Delhi" },
        new EmployeeV1(){ ID=2,Name="Mike",Age=28,City="New Delhi",State="Delhi" },
        new EmployeeV1(){ ID=3,Name="Utkarsh",Age=22,City="Amritsar",State="Punjab" }
    };
    // GET: EmployeeV1
    public IHttpActionResult Get()
    {
        return Ok(lstEmployeeV1);
    }

    public IHttpActionResult Get(int Id)
    {
        return Ok(lstEmployeeV1.FirstOrDefault(x=>x.ID==Id));
    }
}
```

EmployeeV2 Controller:

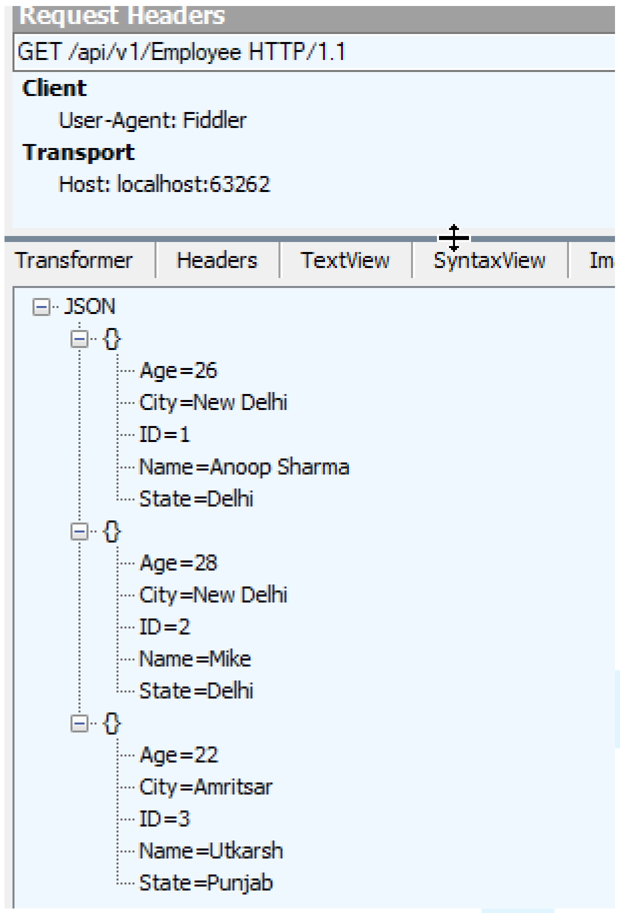
```
public class EmployeeV2Controller : ApiController
{
    List<EmployeeV2> lstEmployeeV2 = new List<EmployeeV2>() {
    new EmployeeV2(){
        ID =1,FirstName="Anoop",LastName="Sharma",Age=26,
        DOB =new DateTime(1991,12,27), City="New Delhi",State="Delhi",Country="India" },
    new EmployeeV2(){
        ID =2,FirstName="Mike",LastName="Ogg",Age=28,
        DOB =new DateTime(1990,10,16),City="New Delhi",State="Delhi",Country="India" },
    new EmployeeV2(){
        ID =3,FirstName="Utkarsh",LastName="Singh",Age=22,
        DOB =new DateTime(1998,06,08),City="Amritsar",State="Punjab",Country="India" }
    };
    // GET: EmployeeV1
    public IHttpActionResult Get()
    {
        return Ok(lstEmployeeV2);
    }

    public IHttpActionResult Get(int Id)
    {
        return Ok(lstEmployeeV2.FirstOrDefault(x => x.ID == Id));
    }
}
```

Now we need to configure the route template in WebApiConfig.cs file as shown below:

```
config.Routes.MapHttpRoute(
    name: "V1Route",
    routeTemplate: "api/V1/Employee/{id}",
    defaults: new { controller="EmployeeV1", id = RouteParameter.Optional }
);
config.Routes.MapHttpRoute(
    name: "V2Route",
    routeTemplate: "api/V2/Employee/{id}",
    defaults: new { controller="EmployeeV2", id = RouteParameter.Optional }
);
```

Now let's test the API with the Fiddler.



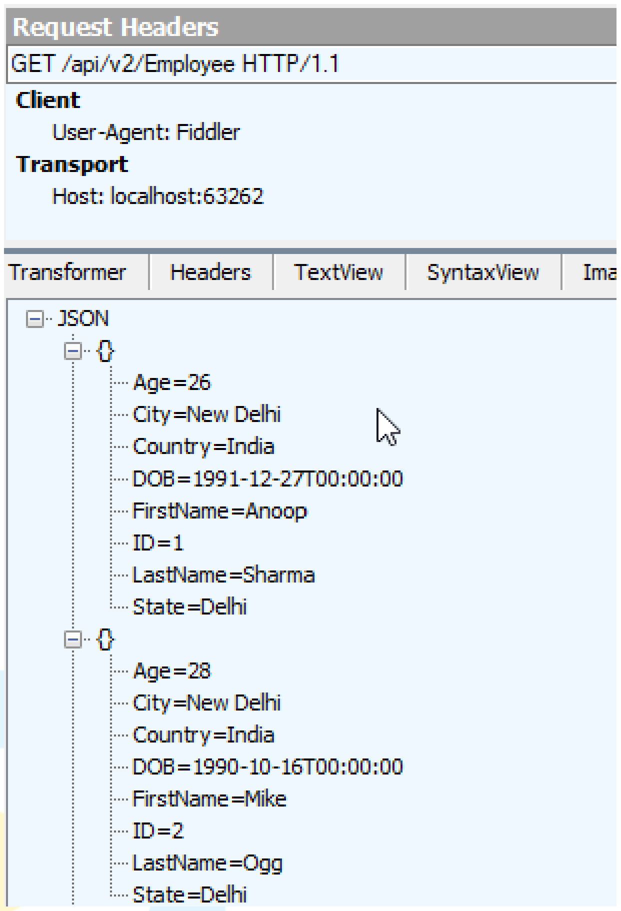
Request Headers

GET /api/v1/Employee HTTP/1.1

Client
User-Agent: Fiddler

Transport
Host: localhost:63262

```
JSON
  {}
    Age=26
    City=New Delhi
    ID=1
    Name=Anoop Sharma
    State=Delhi
  {}
    Age=28
    City=New Delhi
    ID=2
    Name=Mike
    State=Delhi
  {}
    Age=22
    City=Amritsar
    ID=3
    Name=Utkarsh
    State=Punjab
```



Request Headers

GET /api/v2/Employee HTTP/1.1

Client
User-Agent: Fiddler

Transport
Host: localhost:63262

```
JSON
  {}
    Age=26
    City=New Delhi
    Country=India
    DOB=1991-12-27T00:00:00
    FirstName=Anoop
    ID=1
    LastName=Sharma
    State=Delhi
  {}
    Age=28
    City=New Delhi
    Country=India
    DOB=1990-10-16T00:00:00
    FirstName=Mike
    ID=2
    LastName=Ogg
    State=Delhi
```

Image: Testing V1 and V2 Web API using Fiddler.

We can also do the same with the Attribute based Routing as shown in the below code.

```
[Route("api/V1/Employee")]
public IHttpActionResult Get()
{
    return Ok(lstEmployeeV1);
}
[Route("api/V1/Employee/{Id}")]
public IHttpActionResult Get(int Id)
{
    return Ok(lstEmployeeV1.FirstOrDefault(x=>x.ID==Id));
}
```

Q4. How to achieve Web API Versioning with the Query string parameter?

Ans. We can do the Web API Versioning by using the query string parameter of the URL. Using that query string parameter, we can check which version of the data is requested by the client. When a request comes, SelectController() method of the DefaultHttpControllerSelector class selects the controller information from the URI. We will create a custom class CustomSelectorController class and override the SelectController method in order to achieve Web API Versioning with Query String Parameter.

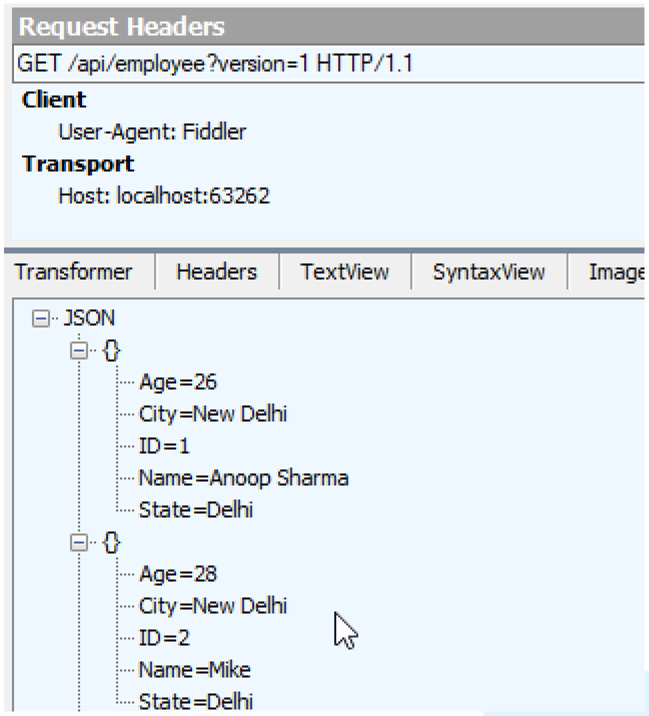
```
public class CustomSelectorController:DefaultHttpControllerSelector
{
    HttpConfiguration _config;
    public CustomSelectorController(HttpConfiguration config) : base(config)
    {
        _config = config;
    }

    public override HttpControllerDescriptor SelectController(HttpRequestMessage request)
    {
        //returns all possible Web API Controllers
        var controllers = GetControllerMapping();
        //return the information about the route
        var routeData = request.GetRouteData();
        //get the controller name passed
        var controllerName = routeData.Values["controller"].ToString();
        string apiVersion = "1";
        //get querystring from the URI
        var versionQueryString = HttpUtility.ParseQueryString(request.RequestUri.Query);
        if (versionQueryString["version"] != null)
        {
           apiVersion = Convert.ToString(versionQueryString["version"]);
        }
        if (apiVersion == "1")
        {
            controllerName = controllerName + "V1";
        }
        else
        {
            controllerName = controllerName + "V2";
        }
        //
        HttpControllerDescriptor controllerDescriptor;
        //check the value in controllers dictionary. TryGetValue is an efficient way to check
the value existence
        if (controllers.TryGetValue(controllerName, out controllerDescriptor))
        {
            return controllerDescriptor;
        }
        return null;
    }
}
```

Replace the IHttpControllerSelector with CustomSelectorController in WebApiConfig.cs.

```
config.Services.Replace(typeof(IHttpControllerSelector), new
CustomSelectorController(config));
```

Now test the API with Fiddler.


Request Headers

GET /api/employee?version=1 HTTP/1.1

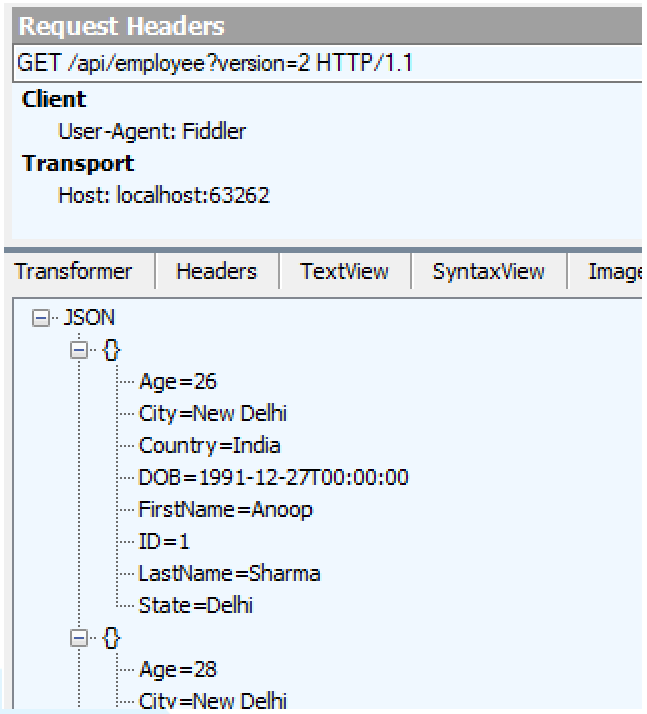
Client

User-Agent: Fiddler

Transport

Host: localhost:63262

```
JSON
  {}
    Age=26
    City=New Delhi
    ID=1
    Name=Anoop Sharma
    State=Delhi
  {}
    Age=28
    City=New Delhi
    ID=2
    Name=Mike
    State=Delhi
```


Request Headers

GET /api/employee?version=2 HTTP/1.1

Client

User-Agent: Fiddler

Transport

Host: localhost:63262

```
JSON
  {}
    Age=26
    City=New Delhi
    Country=India
    DOB=1991-12-27T00:00:00
    FirstName=Anoop
    ID=1
    LastName=Sharma
    State=Delhi
  {}
    Age=28
    City=New Delhi
```

Image: Testing Web API Versioning with Query String Parameter in Fiddler

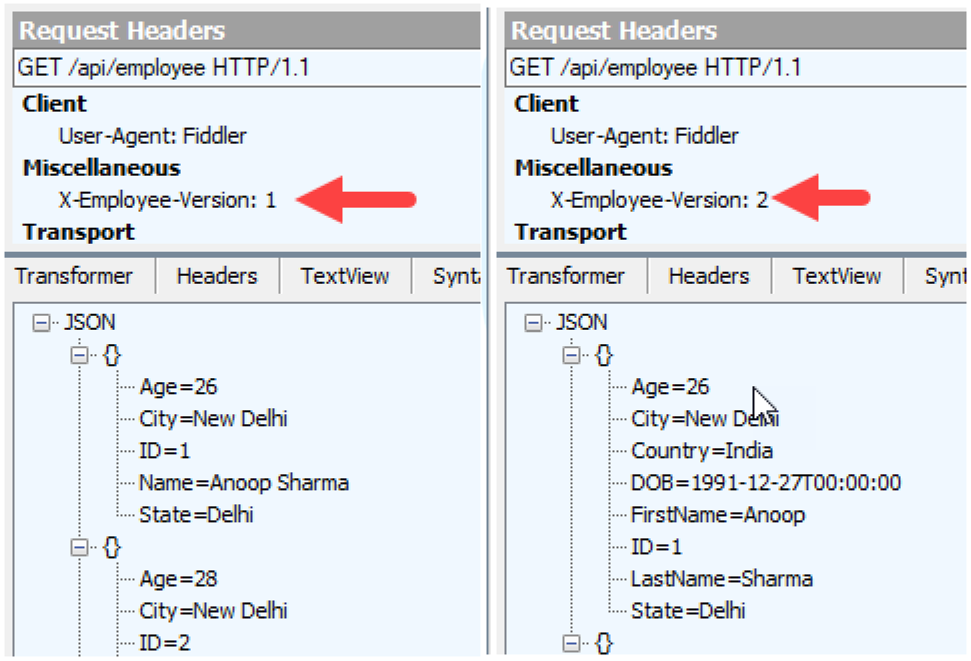
Q4. How to achieve Web API Versioning with the Custom Header Parameter?

Ans. Custom headers are used to provide some additional information related to the request or to implement some custom server-side logic. In this method, client send a custom header with the version specified. Web API CustomSelectorController will check the custom header. If present, then uses its value in order to serve the respective controller.

```
public override HttpControllerDescriptor SelectController(HttpRequestMessage request)
{
    //returns all possible Web API Controllers
    var controllers = GetControllerMapping();
    //return the information about the route
    var routeData = request.GetRouteData();
    //get the controller name passed
    var controllerName = routeData.Values["controller"].ToString();
    string apiVersion = "1";
    //Custom Header Name to be check
    string customHeaderForVersion = "X-Employee-Version";
    if (request.Headers.Contains(customHeaderForVersion))
    {
        //If same custom header sent multiple times by client then multiple value received
which are separated by the comma
        apiVersion = request.Headers.GetValues(customHeaderForVersion).FirstOrDefault();
        if (!string.IsNullOrEmpty(apiVersion) && apiVersion.Contains(','))
        {
            apiVersion = apiVersion.Split(',')[0];
        }
    }

    If (apiVersion == "1")
    {
        controllerName = controllerName + "V1";
    }
    else
    {
        controllerName = controllerName + "V2";
      }
      //
      HttpControllerDescriptor controllerDescriptor;
      //check the value in controllers dictionary. TryGetValue is an efficient way to check 
the value existence
      if (controllers.TryGetValue(controllerName, out controllerDescriptor))
      {
          return controllerDescriptor;
      }
      return null;
}
```

Replace the IHttpControllerSelector with CustomSelectorController in WebApiConfig.cs as done in previous example. Now test the code with the fiddler.



```
Request Headers
GET /api/employee HTTP/1.1
Client
    User-Agent: Fiddler
Miscellaneous
    X-Employee-Version: 1
Transport

JSON
  {}
    Age=26
    City=New Delhi
    ID=1
    Name=Anoop Sharma
    State=Delhi
  {}
    Age=28
    City=New Delhi
    ID=2
```

```
Request Headers
GET /api/employee HTTP/1.1
Client
    User-Agent: Fiddler
Miscellaneous
    X-Employee-Version: 2
Transport

JSON
  {}
    Age=26
    City=New Delhi
    Country=India
    DOB=1991-12-27T00:00:00
    FirstName=Anoop
    ID=1
    LastName=Sharma
    State=Delhi
  {}
```

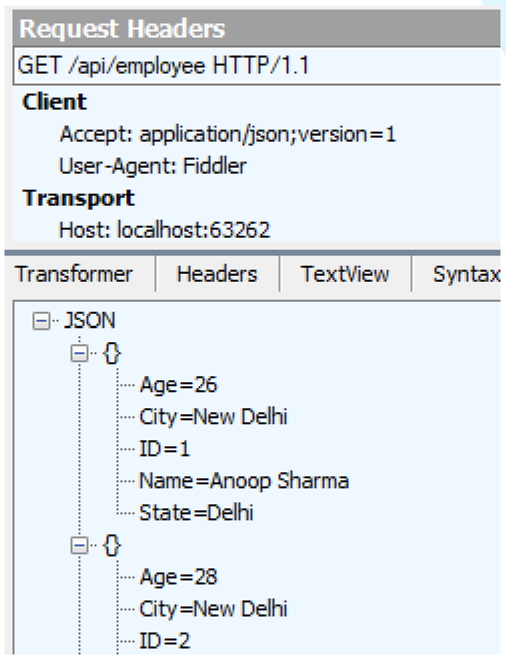
Image: Testing Web API using Fiddler for Custom Header API Versioning

Q5. How to achieve Web API Versioning with the Accept Header?

Ans. Accept header is used to request the file format of the data the client wants to receive. In a request, the client will send a version to be required in the Accept header.

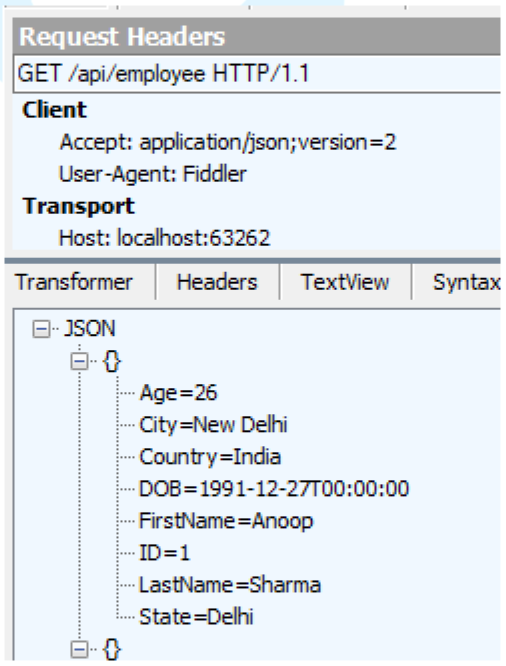
```
public override HttpControllerDescriptor SelectController(HttpRequestMessage request)
{
    //returns all possible Web API Controllers
    var controllers = GetControllerMapping();
    //return the information about the route
    var routeData = request.GetRouteData();
    //get the controller name passed
    var controllerName = routeData.Values["controller"].ToString();
    string apiVersion = "1";
    //Get the Accept Header which contains the version
    var acceptHeader = request.Headers.Accept
        .Where(b => b.Parameters.Count(t => t.Name.ToLower() == "version")>0);
    if (acceptHeader.Any())
    {
        //Get the first value of the version from the Accept Headers
        apiVersion = acceptHeader.First().Parameters.First(x => x.Name.ToLower() ==
"version").Value;
    }
    if (apiVersion == "1")
    {
        controllerName = controllerName + "V1";
    }
    else
    {
        controllerName = controllerName + "V2";
    }
    //
    HttpControllerDescriptor controllerDescriptor;
    //check the value in controllers dictionary. TryGetValue is an efficient way to check the
value existence
    if (controllers.TryGetValue(controllerName, out controllerDescriptor))
    {
    return controllerDescriptor;
    }
    return null;
}
```

Now test the above code using Fiddler.



```
Request Headers
GET /api/employee HTTP/1.1
Client
    Accept: application/json;version=1
    User-Agent: Fiddler
Transport
    Host: localhost:63262
```

```
JSON
  {}
    Age=26
    City=New Delhi
    ID=1
    Name=Anoop Sharma
    State=Delhi
  {}
    Age=28
    City=New Delhi
    ID=2
```



```
Request Headers
GET /api/employee HTTP/1.1
Client
    Accept: application/json;version=2
    User-Agent: Fiddler
Transport
    Host: localhost:63262
```

```
JSON
  {}
    Age=26
    City=New Delhi
    Country=India
    DOB=1991-12-27T00:00:00
    FirstName=Anoop
    ID=1
    LastName=Sharma
    State=Delhi
  {}
```

Image: Sending the version parameter in the Accept Header for the Web API Versioning

7

Exception Handling and Dependency Injection

Q1. What is Exception handling?

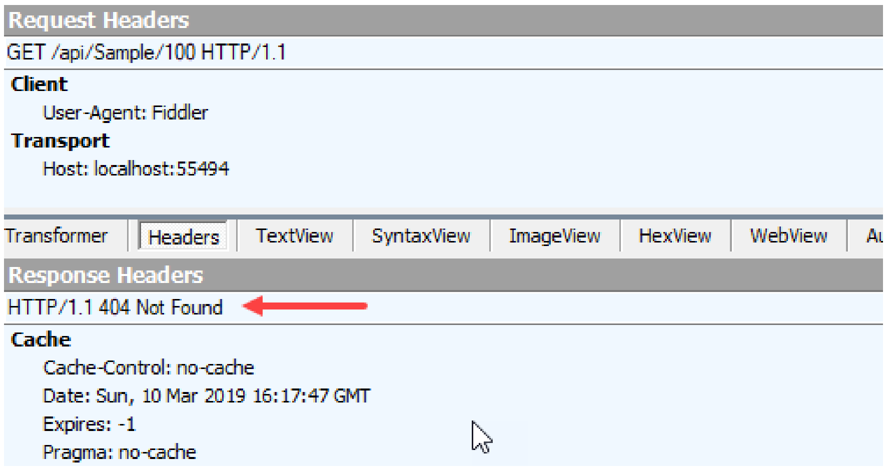
Ans. Exception handling is a technique to handle runtime error in the application code. In multiple ways we can handle the error in ASP.NET Web API, some of them are listed below:

1. HttpResponseException
2. HttpError
3. Exception Filters etc.

Q2. How to handle the exception with the help of HttpResponseException Class?

Ans. HttpResponseException class is derived from the Exception class and have two overloaded versions in which one accepts HttpStatusCode and another accepts HttpResponseMessage. We can directly specify HttpStatusCode in the exception constructor, just like we have done in the below example.

```
[HttpGet]
public IHttpActionResult GetData(int ID)
{
    StudentBAL objStudentBAL = new StudentBAL();
    var student = objStudentBAL.getStudentDetailsByID(ID);
    if (student == null)
    {
        throw new HttpResponseException(HttpStatusCode.NotFound);
        //We can also return NotFound Result Code directly
        //return NotFound();
    }
    return Ok(student);
}
```



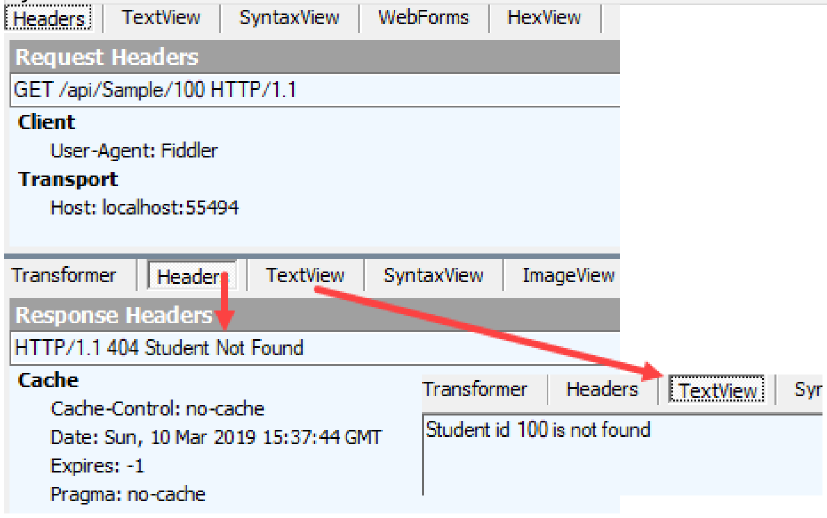
Request Headers
GET /api/Sample/100 HTTP/1.1
Client
User-Agent: Fiddler
Transport
Host: localhost:55494

Response Headers
HTTP/1.1 404 Not Found
Cache
Cache-Control: no-cache
Date: Sun, 10 Mar 2019 16:17:47 GMT
Expires: -1
Pragma: no-cache

Image: Returning 404 Not Found HttpStatusCode using HttpResponseException Class

We can have more control over the response return by the HttpResponseException, as we can include HttpResponseMessage in HttpResponseException.

```
[HttpGet]
public IHttpActionResult GetData(int ID)
{
    StudentBAL objStudentBAL = new StudentBAL();
    var student = objStudentBAL.getStudentDetailsByID(ID);
    if (student == null)
    {
        //Initializing the HttpResponseMessage class with specific Status Code
        var message = new HttpResponseMessage(HttpStatusCode.NotFound)
        {
            Content = new StringContent(string.Format("Student id {0} is not found", ID)),
            ReasonPhrase = "Student Not Found"
        };
        throw new HttpResponseException(message);
    }
    return Ok(student);
}
```



Request Headers
GET /api/Sample/100 HTTP/1.1
Client
User-Agent: Fiddler
Transport
Host: localhost:55494

Response Headers
HTTP/1.1 404 Student Not Found
Cache
Cache-Control: no-cache
Date: Sun, 10 Mar 2019 15:37:44 GMT
Expires: -1
Pragma: no-cache

Student id 100 is not found

Image: Including HttpResponseMessage in HttpResponseException

DotNetTricks

Q3. How can we create custom Exception Filter in ASP.NET Web API?

Ans. An Exception Filter is executed on getting an unhandled exception except not for the HttpResponseException as it is designed for returning specific Http response. We can create a class as an exception filter by inheriting the ExceptionFilterAttribute Class and overriding the OnException method of the class. We can use exception filter for logging the exception in the database as well. We can also check the type of exception and return HttpResponseMessage according to the type of Exception. A basic example of the Exception Filter is provided below:

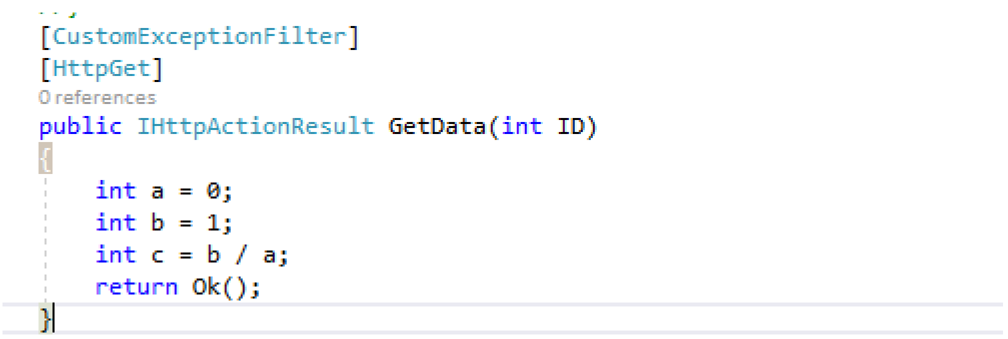
```
public class CustomExceptionFilter : ExceptionFilterAttribute
{
    public override void OnException(HttpActionExecutedContext actionExecutedContext)
    {
        //We can use Exception Message and StackTrace value for Exception Logging in the
database
        string exMessage = actionExecutedContext.Exception.Message;
        string exStackTrace = actionExecutedContext.Exception.StackTrace;
        actionExecutedContext.Response = new
HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError)
        {
            Content=new StringContent("An unhandled Error is thrown by the API"),
            ReasonPhrase="Internal Server Error"
        };

        base.OnException(actionExecutedContext);
    }
}
```

Q4. How can we register custom Exception Filter in ASP.NET Web API?

Ans. We can register Custom Exception Filter in multiple ways:

1. On Action Method
2. On Controller
3. Globally

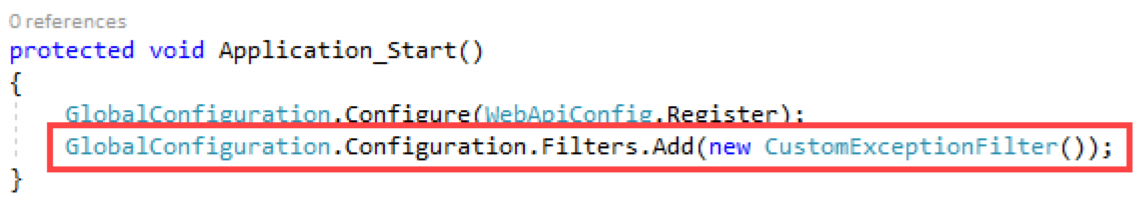


```
[CustomExceptionFilter]
[HttpGet]
0 references
public IHttpActionResult GetData(int ID)
{
    int a = 0;
    int b = 1;
    int c = b / a;
    return Ok();
}
```

Image: Registering CustomExceptionFilter on the Action Name in Web API

Q5. How to register an Exception Filter globally?

Ans. In order to apply the Exception filter globally to all Web API controllers, we need to add the instance of the exception filter to the GlobalConfiguration.Configuration.Filters collection in the Global.asax.cs file.



```
0 references
protected void Application_Start()
{
    GlobalConfiguration.Configure(WebApiConfig.Register);
    GlobalConfiguration.Configuration.Filters.Add(new CustomExceptionFilter());
}
```

Image: Registering an Exception filter globally in Global.asax.cs file

Q6. What is Dependency Injection?

Ans. Dependency injection is a software design pattern to create loosely coupled code. Dependency can be any object that requires another object. The main purpose of dependency injection is to remove the dependencies, reduce the complexity in the software, improves the unit testing, increases the reusability and makes it more maintainable.

Q7. What are the different types of techniques to implement Dependency Injection in Web API?

Ans. There are three types of techniques to implement Dependency injection in ASP.NET Web API:

1. Constructor Injection: Most popular type of dependency injection technique. Dependency is injected from the class constructor.
2. Setter Injection: Setter injection is also known as property injection.
3. Method Injection

Q8. What are Dependency Injection Containers or DI Containers?

Ans. Dependency Injection Containers or DI Containers are the frameworks to create and inject the dependencies automatically. The main role of DI Containers is to create an object and inject them if required. Some of the popular DI Containers are:

1. Ninject
2. Autofac
3. Unity
4. Castle Windsor etc.

Q9. How to implement Dependency Injection using Ninject DI Container?

Ans. Let's create a new Empty Web API Project and create a dependency first in the Web API project. Once we create a dependency in the project, we will try to remove it with the help of Ninject DI Container.

Add a model named as Student with the below structure:

```
public class Student
{
    public int ID { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public string Stream{ get; set; }
}
```

Add another class StudentBAL which is responsible for interacting with the DataAccess Layer or Database.

```
public class StudentBAL
{
    //Sample Data, in real project data will be fetched from the DAL or other services
    List<Student> lstStudents = new List<Student>() {
        new Student(){ID=1,Name="Anoop Sharma",Age=26,Stream="Science" },
        new Student(){ID=2,Name="Ajay",Age=28,Stream="Commerce" },
        new Student(){ID=3,Name="Farhan",Age=25,Stream="Arts" },
        new Student(){ID=4,Name="Mike",Age=27,Stream="Science" }
    };

    public List<Student> getAllStudentDetails()
    {
        return lstStudents;
    }

    public Student getStudentDetailsByID(int ID)
    {
        return lstStudents.FirstOrDefault(x => x.ID == ID);
    }
}
```

Add Student Web API Controller and call the StudentBAL from it. As we can clearly see that StudentController class need StudentBAL, In order to perform its function and hence creates a dependency here.

```
public class StudentController : ApiController
{
    StudentBAL obj = new StudentBAL();
    public IHttpActionResult Get()
    {
        return Ok(obj.getAllStudentDetails());
    }
    [HttpGet]
    public IHttpActionResult Get(int ID)
    {
        var student = obj.getStudentDetailsByID(ID);
        if (student==null) {
            return NotFound();
        }
        else
        {
            return Ok(student);
        }
    }
}
```

In order to inject the dependency, we need to create an interface named IStudent.

```
public interface IStudent
{
    List<Student> getAllStudentDetails();
    Student getStudentDetailsByID(int ID);
}
```

Now implement the method in StudentBAL class as shown below.

```
public class StudentBAL: IStudent
{
    //Sample Data, in real project data will be fetched from the DAL or other services
    List<Student> lstStudents = new List<Student>() {
        new Student(){ID=1,Name="Anoop Sharma",Age=26,Stream="Science" },
        new Student(){ID=2,Name="Ajay",Age=28,Stream="Commerce" },
        new Student(){ID=3,Name="Farhan",Age=25,Stream="Arts" },
        new Student(){ID=4,Name="Mike",Age=27,Stream="Science" }
    };
    public List<Student> getAllStudentDetails()
    {
        return lstStudents;
    }
    public Student getStudentDetailsByID(int ID)
    {
        return lstStudents.FirstOrDefault(x => x.ID == ID);
    }
}
```

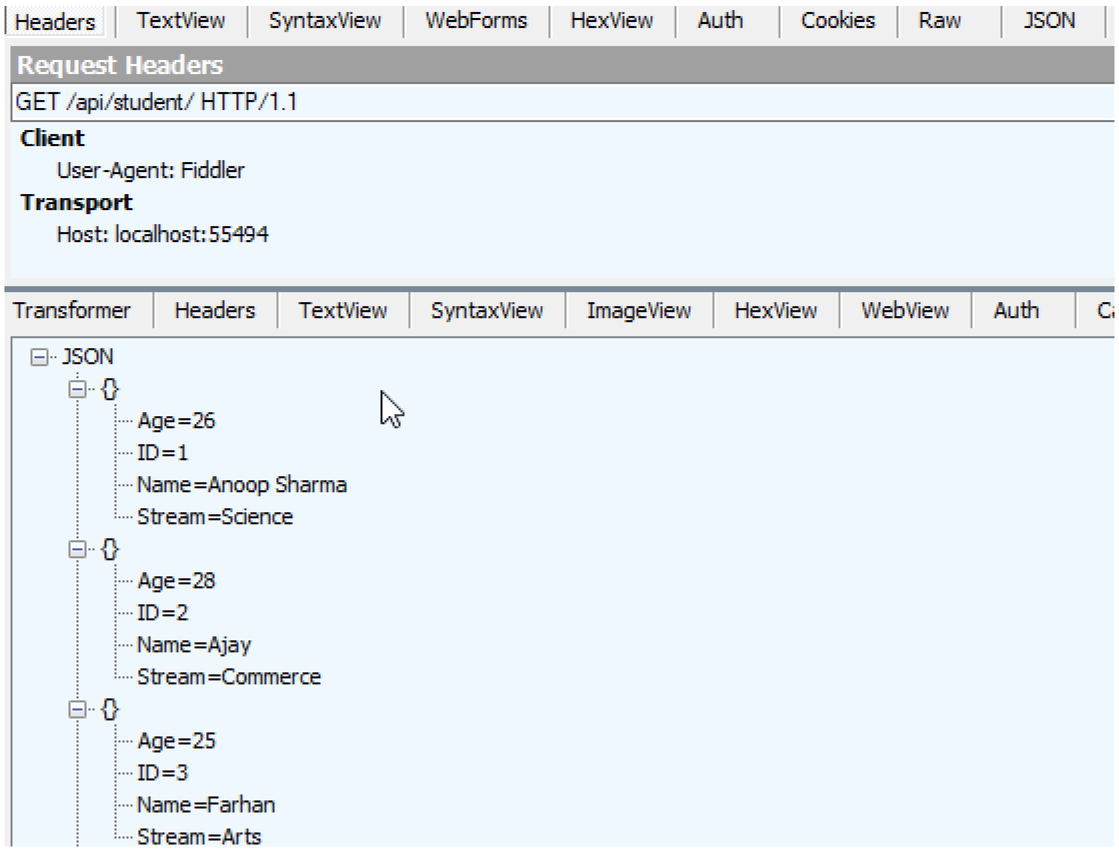
Now, add a parameterized constructor in StudentController and pass the IStudent interface as the parameter.

```
public class StudentController : ApiController
{
    IStudent _IStudent;
    public StudentController(IStudent student)
    {
        _IStudent = student;
    }
    public IHttpActionResult Get()
    {
        return Ok(_IStudent.getAllStudentDetails());
    }
    [HttpGet]
    public IHttpActionResult Get(int ID)
    {
        var student = _IStudent.getStudentDetailsByID(ID);
        if (student==null) {
            return NotFound();
        }
        else
        {
            return Ok(student);
        }
    }
}
```

Add the Ninject library in the project. Ninject is the lightning fast, ultra-lightweight Dependency injector for the .Net Applications. Make sure to install Ninject.Web.WebApi and Ninject.Web.WebApi.WebHost packages from the NuGet package manager. Once all packages are downloaded, Ninject.Web.Common.cs file will be added in the App_Start method. Add or Bind the interface or modules in the RegisterServices method.

```
private static void RegisterServices(IKernel kernel)
{
    kernel.Bind<IStudent>().To<StudentBAL>();
}
```

Build and run the project. Now open the Postman or Rest Client in order to test the Web API.



Request Headers

```
GET /api/student/ HTTP/1.1
Client
    User-Agent: Fiddler
Transport
    Host: localhost:55494
```

```
JSON
  {}
    Age=26
    ID=1
    Name=Anoop Sharma
    Stream=Science
  {}
    Age=28
    ID=2
    Name=Ajay
    Stream=Commerce
  {}
    Age=25
    ID=3
    Name=Farhan
    Stream=Arts
```

Image: Testing the Web API after adding Ninject Dependency Container.

8
Deployment

Q1. How we can Deploy the ASP.NET Web API?

Ans. We can deploy Web API in multiple ways, but some of the popular ways are:

1. Deployment on the IIS Server
2. Self-Hosting

Q2. What are the steps required in order to Deploy the ASP.NET Web API on the IIS Server?

Ans. We can deploy Web API in various ways. Some people deploy directly on the IIS server, some prefer to publish it first and then move the files via FTP. Let's see How can we deploy ASP.NET Web API on the local IIS Server.

1. Right click on the Web API project and click on publish.

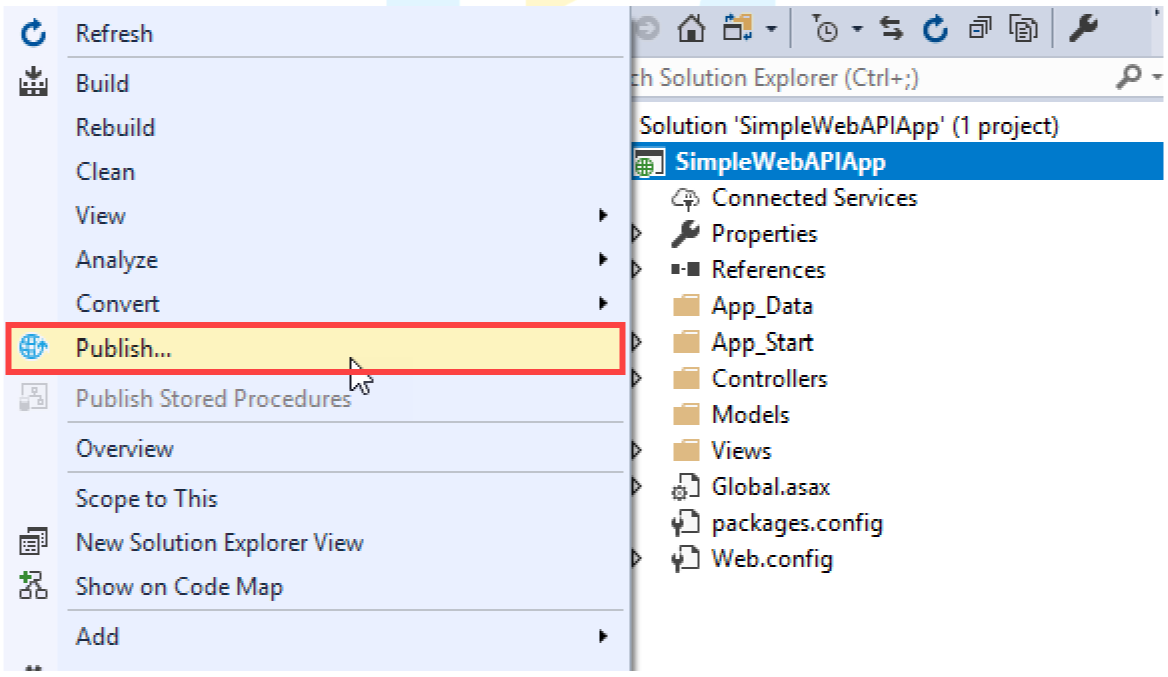
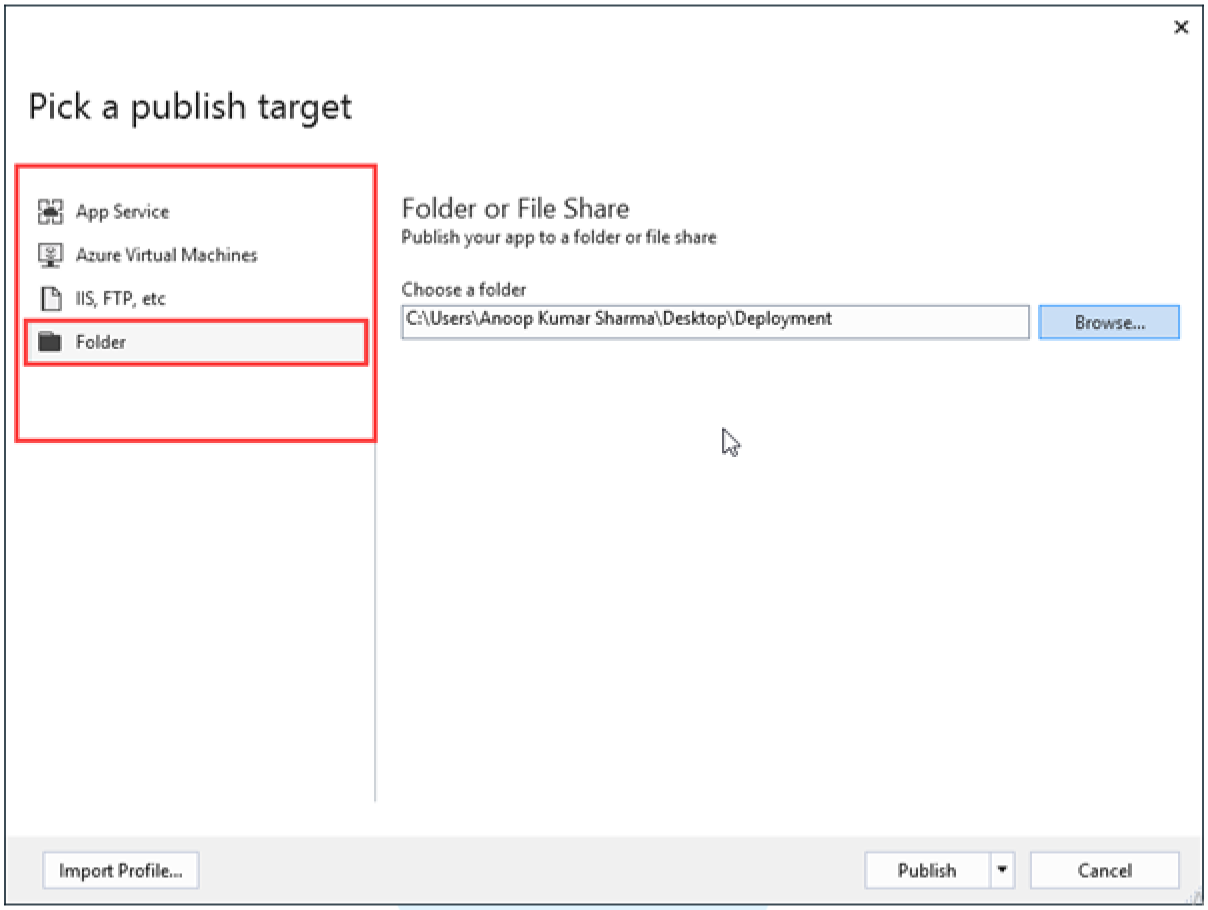


Image: Right Click on the Project to publish the Web API project

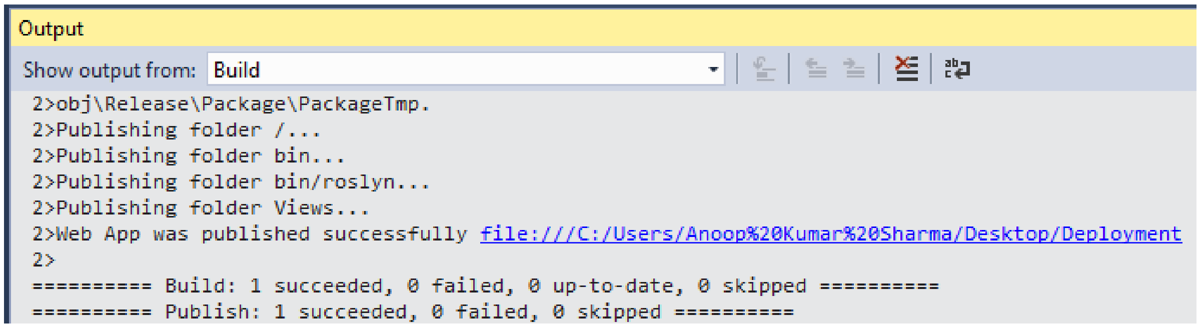
2. Select the Publish Target. In our case, we already created a Directory where we want to publish the Code.



Folder or File Share
Publish your app to a folder or file share
Choose a folder
C:\Users\Anoop Kumar Sharma\Desktop\Deployment

Image: Pick a publish target

3. Check the Success message if project published successfully. If not you will get an error in the output window.



Show output from: Build

```
2>obj\Release\Package\PackageTmp.
2>Publishing folder /...
2>Publishing folder bin...
2>Publishing folder bin/roslyn...
2>Publishing folder Views...
2>Web App was published successfully file:///C:/Users/Anoop%20Kumar%20Sharma/Desktop/Deployment
2>
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

Image: Output window showing the Publish succeeded

4. Open the IIS server by running the "inetmgr" command or from the start menu option.

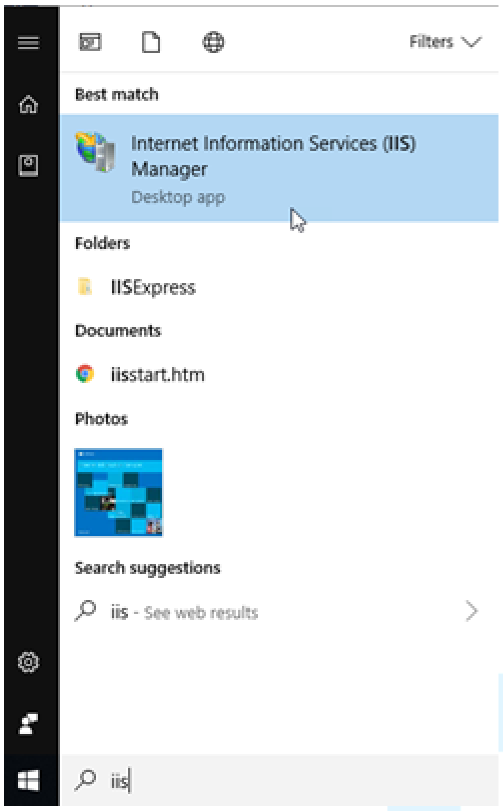


Image: Searching IIS Manager in the Start Menu

5. Add a new website/web application in the IIS Server.

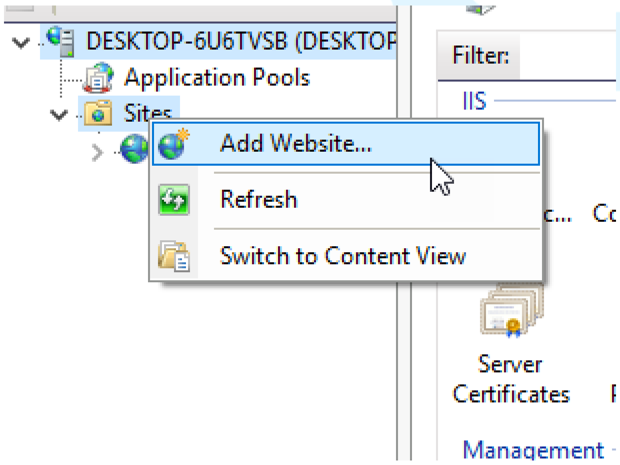


Image: Add Website in the IIS Server

6. Add the Site name, physical path (where you will put the published code), port, etc. as per your need on the IIS Server.

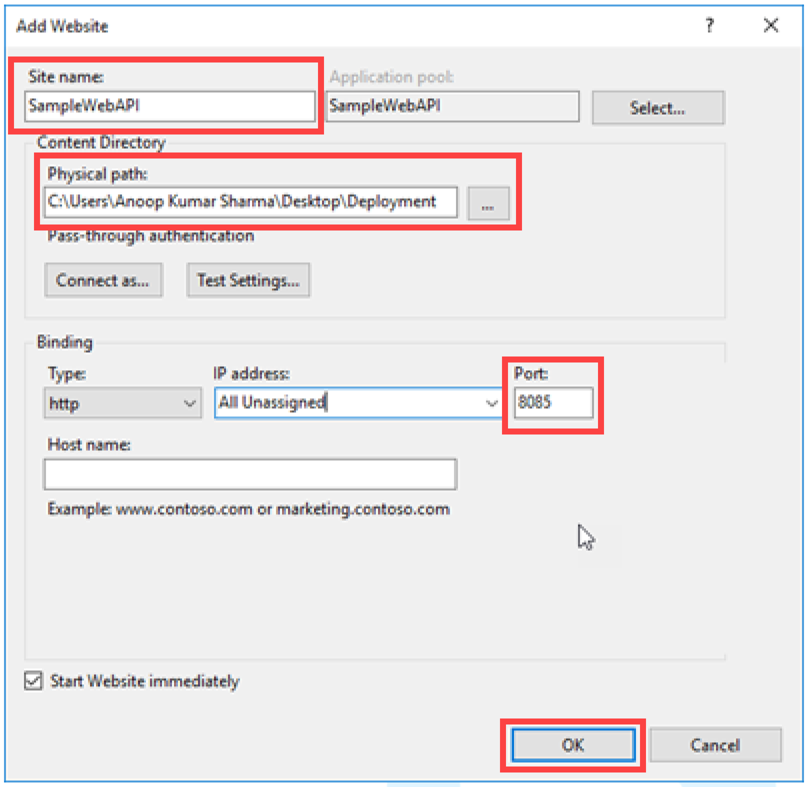
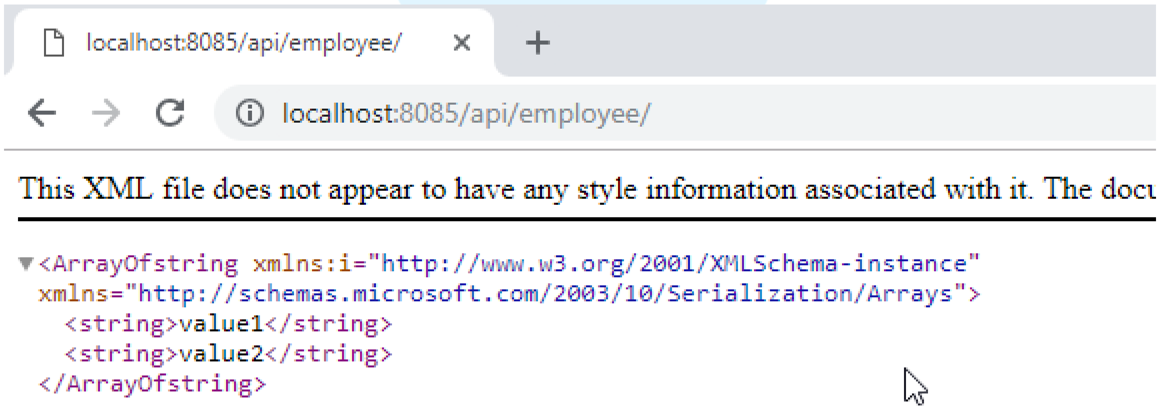


Image: Add physical path and the binding for the Web API during IIS Deployment.

7. Now open the browser and hit the API with the port mentioned during deployment of the application. If everything is fine you will get the response from the IIS server.



This XML file does not appear to have any style information associated with it. The docu

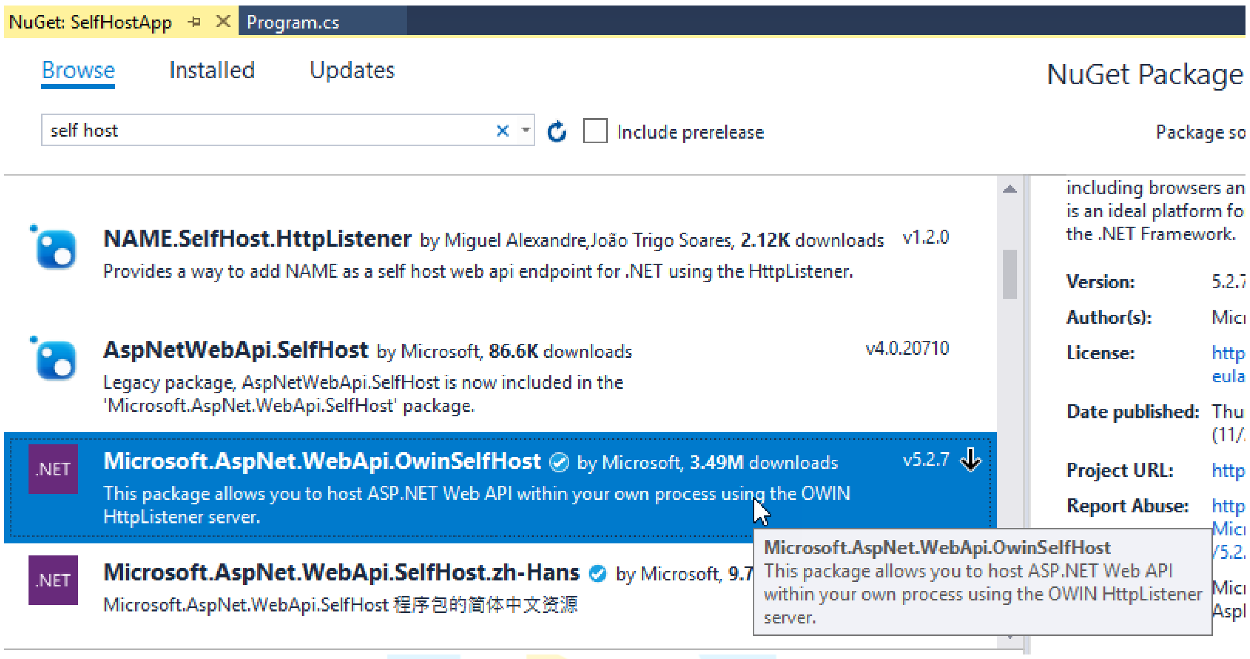
```
▼<ArrayOfstring xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
   <string>value1</string>
   <string>value2</string>
 </ArrayOfstring>
```

Image: Web API Deployed Successfully. Testing the API from the Browser

Q3. What are the steps required to Self-Host an ASP.NET Web API?

Ans. Sometimes deploying an application feels to be quite heavy on the IIS server in case you are deploying Web API. With OWIN, that stands for Open Web Interface for the .Net, we can self-host a Web API. Let's see How to Self-Host a Web API step by step:

1. Create an Empty Console Application Project and install Microsoft.AspNet.WebApi.OwinSelfHost package from NuGet package manager which allows us to host ASP.NET Web API with your own process using the OWIN HttpListener server.



NAME.SelfHost.HttpListener by Miguel Alexandre,João Trigo Soares, 2.12K downloads v1.2.0
Provides a way to add NAME as a self host web api endpoint for .NET using the HttpListener.

AspNetWebApi.SelfHost by Microsoft, 86.6K downloads v4.0.20710
Legacy package, AspNetWebApi.SelfHost is now included in the 'Microsoft.AspNet.WebApi.SelfHost' package.

Microsoft.AspNet.WebApi.OwinSelfHost by Microsoft, 3.49M downloads v5.2.7
This package allows you to host ASP.NET Web API within your own process using the OWIN HttpListener server.

Microsoft.AspNet.WebApi.SelfHost.zh-Hans by Microsoft, 9.7
Microsoft.AspNet.WebApi.SelfHost 程序包的简体中文资源

Microsoft.AspNet.WebApi.OwinSelfHost
This package allows you to host ASP.NET Web API within your own process using the OWIN HttpListener server.

including browsers an
is an ideal platform fo
the .NET Framework.

Version: 5.2.7
Author(s): Mici
License: http eula
Date published: Thu (11/
Project URL: http
Report Abuse: http Mici /5.2.
Mici
Aspl

Image: Install Microsoft.AspNet.WebApi.OwinSelfHost package from NuGet package manager

2. Add an OWIN Startup class in the project.

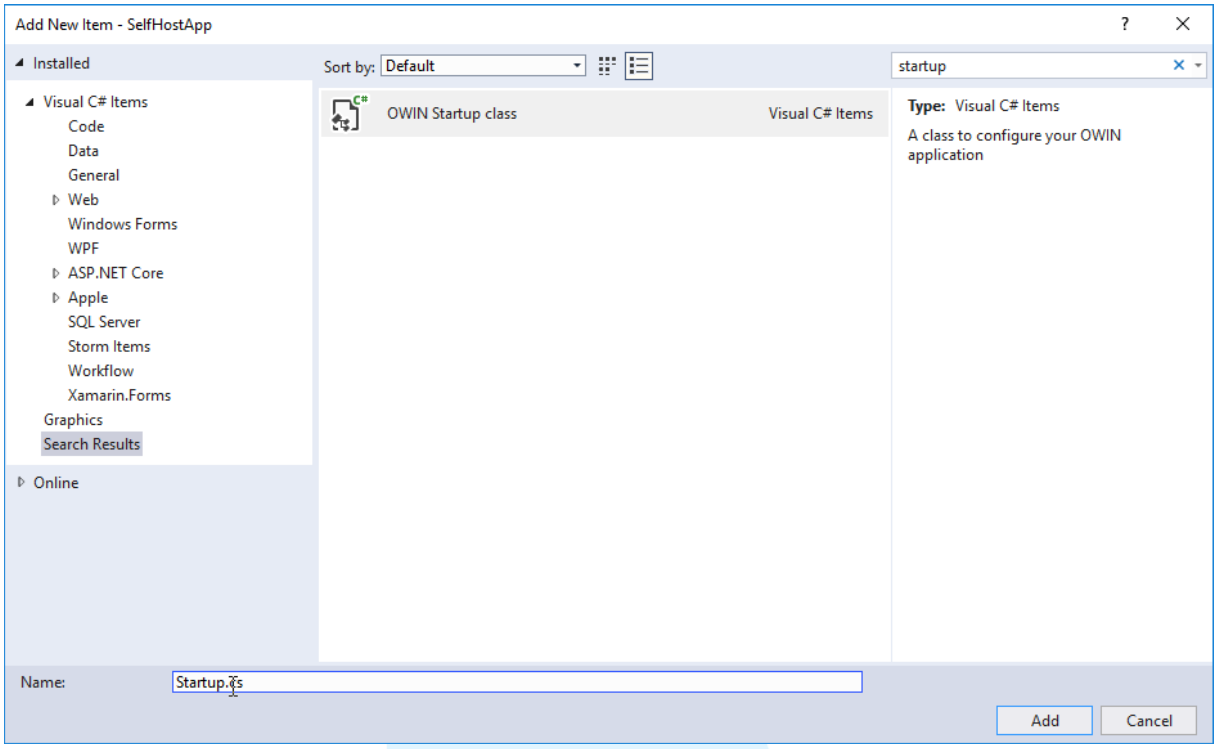


Image: Add OWIN Startup class in the project

Add the default route template just like we have in WebApiConfig.cs file.

```
public class Startup
{
    public void Configuration(IAppBuilder app)
    {
    // Configure Web API for self-host.
    var config = new HttpConfiguration();

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

    app.UseWebApi(config);
    }
}
```

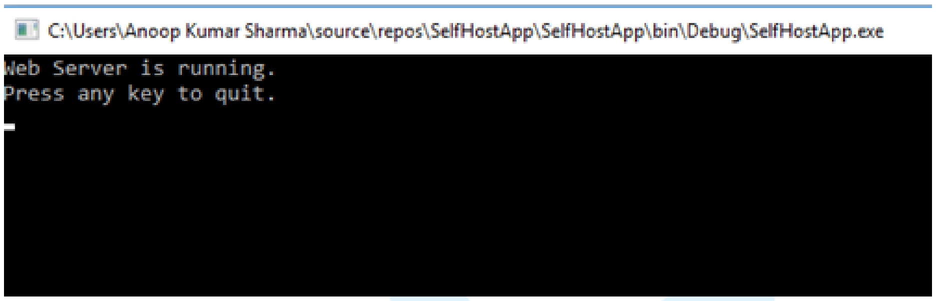
3. Add a Controller Folder and add a HomeController Class and Inherit the ApiController class in it.

```
public class HomeController:ApiController
{
    public IEnumerable<string> Get()
     {
         return new string[] { "Value1", "Value2" };
     }
}
```

4. Start the web server using WebApp.Start and pass the startup class name as an argument with the domain and port details.

```
class Program
{
    static void Main(string[] args)
    {
        using (WebApp.Start<Startup>("http://localhost:8090"))
        {
        Console.WriteLine("Web Server is running.");
        Console.WriteLine("Press any key to quit.");
        Console.ReadLine();
        }
    }
}
```

5. Now run the application. You will see Web Server is running message in the command prompt.

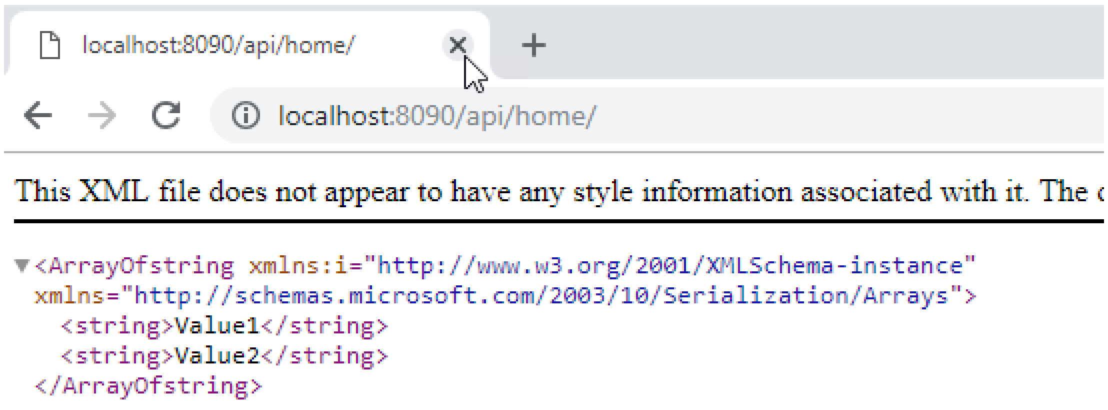


C:\Users\Anoop Kumar Sharma\source\repos\SelfHostApp\SelfHostApp\bin\Debug\SelfHostApp.exe

```
Web Server is running.
Press any key to quit.
```

Image: Starting up the Web Server

6. Now open a browser or Rest Client and test the Web API.



This XML file does not appear to have any style information associated with it. The c

```
▼<ArrayOfstring xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
   <string>Value1</string>
   <string>Value2</string>
 </ArrayOfstring>
```

Image: Testing Self-Hosted Web API in the browser

References

This book has been written by referring to the following sites:

1. https://docs.microsoft.com/en-us/aspnet/web-api - Microsoft Docs - ASP.NET Web API
2. https://stackoverflow.com/questions/tagged/asp.net-web-api - Stack Overflow - ASP.NET Web API
3. https://www.dotnettricks.com/learn/webapi - Dot Net Tricks - ASP.NET Web API