**1) What is Web API?**

**Web API is the technology by which you can expose data over HTTP. It is a framework which helps us to build/develop HTTP services. So there will be a client server communication using HTTP protocol.**

**Using ASP.NET Web API, we can create non-SOAP based services like plain XML or JSON strings, etc. with many other advantages including:**

* **Create resource-oriented services using the full features of HTTP**
* **Exposing services to a variety of clients easily like browsers or mobile devices, etc.**

**2) Explain Web API Routing?**

Routing is the mechanism of pattern matching as we have in MVC. These routes will get registered in Route Tables. Below is the sample route in Web API –

routes.MapHttpRoute(

name: “MyFirstWebAPIRoute”,

routeTemplate: “api/{controller}/{id}, //as you can see “api” is at the beginning.

defaults: new { id = RouteParameter.Optional}

);

**3) Give an example of MVC Routing?**

routes.MapRoute(

name: “MyRoute”, //route name

url: “{controller}/{action}/{id}”, //route pattern

defaults: new

{

controller = “a4academicsController”,

action = “a4academicsAction”,

id = UrlParameter.Optional

});

**4) What is Representational state transfer or REST?**

REST is architectural style, which has **defined guidelines for creating services which are scalable**.

REST used with HTTP protocol using its verbs GET, POST, PUT and DELETE.

**5) What are the advantages of using REST in Web API?**

REST always *used to make less data transfers between client and server* which makes REST an ideal for using it in mobile apps. Web API supports HTTP protocol thereby it reintroduces the old way of HTTP verbs for communication.

**REST defines 6 architectural constraints** which make any web service – a true RESTful API.

**1-Uniform interface**

**A resource in the *system should have only one logical URI*** *and that should provide a way to fetch related or additional data.*

**2-Client–server**

This essentially means that *client application and server application MUST be able to evolve separately* without any dependency on each other. A client should know only resource URIs and that’s all.

**3-Stateless**

Make all client-server interaction stateless. Server will not store anything about latest HTTP request client made. It will treat each and every request as new. No session, no history.

**4-Cacheable**

Caching brings performance improvement for client side, and better scope for scalability for a server because the load has reduced.

**5-Layered system**

REST allows you to use a layered system architecture where you deploy the APIs on server A, and store data on server B and authenticate requests in Server C,

**6-Code on demand (optional)**

**6) What New Features are introduced in ASP.NET Web API 2.0?**

New features added in Web API 2.0 version in Italic

* *OWIN (Open Web Interface for .NET) Self Hosting*
* *Attribute Routing*
* *External Authentication*
* *Web API OData*
* *IHttpActionResult*
* *CORS (Cross-Origin Resource Sharing)*

**Advantages of ASP.NET Web API**

* It works the HTTP way using standard HTTP verbs like GET, POST, PUT, DELETE, etc. for all CRUD operations
* Complete support for routing
* Response generated in JSON or XML format using MediaTypeFormatter
* It has the ability to be hosted in IIS as well as self-host outside of IIS
* Supports Model binding and Validation
* Support for OData

**7) List out the steps to be made for Web API to work in Web Forms?**

1. Create a Web API Controller
2. Add a routing table to Application\_Start method of Global.asax
3. Make a jQuery AJAX Call to Web API method and get data

**8) Explain how to give alias name for action methods in Web API?**

Using attribute “ActionName” we can give alias name for Web API actions. Eg:

[HttpPost]

*[ActionName(“AliasTestAction”)]*

public void UpdateTestCustomer(Customer c)

{

TestCustomerRepository.AddCustomer(c);

}

**9) What is the difference between MVC Routing and Web API Routing?**

In *Web API pattern we can find “api/”* at the beginning which makes it distinct from MVC routing.

**10) How can we pass multiple complex types in Web API?**

Below are the methods to pass the complex types in Web API –

1. Using ArrayList
2. NewtonsoftJArray

**11) Write a code snippet for passing arraylist in Web API?**

Below is the code snippet for passing arraylist –

ArrayList paramList = new ArrayList();

Category c = new Category { CategoryId = 1, CategoryName = “SmartPhones”};

Product p = new Product { ProductId = 1, Name = “Iphone”, Price = 500, CategoryID = 1 };

paramList.Add(c);

paramList.Add(p);

**12) How we can handle errors in Web API?**

Below are the list of classes which can be used for error handling –

* HttpResponseException
* Exception Filters
* Registering Exception Filters
* HttpError

**13) Explain how we can handle error from “HttpResponseException”?**

This returns the HTTP status code what you specify in the constructor. Eg :

public TestClass MyTestAction(int id)

{

TestClass c = repository.Get(id);

if (c == null)

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

return c;

}

**14) How to register Web API exception filters?**

Below are the options to register Web API exception filters –

* From Action
* From Controller
* Global registration

**15) Write a code snippet to register exception filters from action?**

Below is the code snippet for registering exception filters from action –

*[NotImplExceptionFilter]*

public TestCustomer GetMyTestCustomer(int custid)

{

//Your code goes here

}

**16) Write a code snippet to register exception filters from controller?**

Below is the code snippet for registering exception filters from controller –

*[NotImplExceptionFilter]*

public class TestCustomerController : Controller

{

//Your code goes here

}

**17) Write a code snippet to register exception filters globally?**

Below is the code snippet for registering exception filters globally –

GlobalConfiguration.Configuration.Filters.Add( new MyTestCustomerStore.NotImplExceptionFilterAttribute());

**18) What is “Under-Posting” and “Over-Posting” in Web API?**

“Under-Posting” – If the client sends less data than expected in binding then it’s called under-posting “Over-Posting” – If the client sends more data than expected in binding then it’s called over-posting.

**19) Give an example of creating custom action filter in Web API?**

Below is the sample code for creating custom action filter –

public class MyCustomModelAttribute :ActionFilterAttribute

{

public override void OnActionExecuting(HttpActionContext actionContext)

{

if (actionContext.ModelState.IsValid == false)

{

//Code goes here

}

}

}

In case validation fails here it returns HTTP response which contains validation errors.

**20) How to apply custom action filter in WebAPI.config?**

Add a new action filter in “Register” method as shown –

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

config.Filters.Add(new MyCustomModelAttribute());

// …

}

}

**21) How to set the custom action filter in action methods in Web API?**

Below is the sample code of action with custom action filter –

public class MyCustomerTestController :ApiController

{

[MyCustomModelAttribute]

public HttpResponseMessagePost(MyTestCustomer customer)

{

// …

}

}

**22) Why to use “FromUri” in Web API?**

In Web API to **read complex types from URL** we will use “FromUri” attribute to the parameter in action method. Eg:

[Route("TestFromUri")]

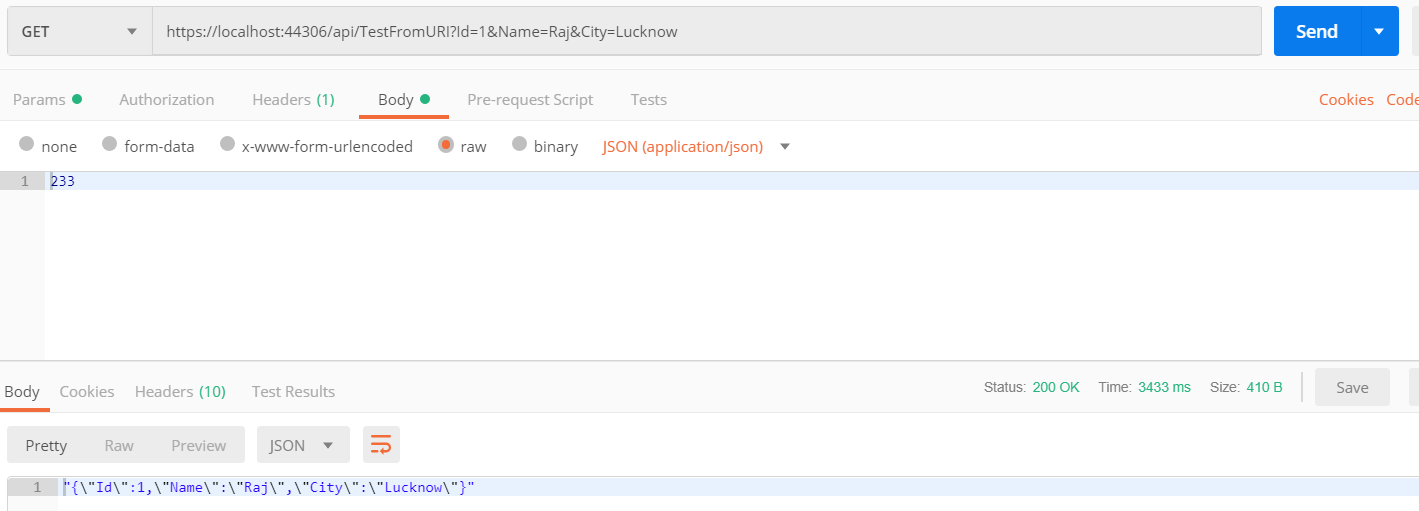
public string TestFromUri([FromBody] int CustId, [FromUri] Employee employee)

{

return JsonConvert.SerializeObject(employee);

}

Request would be: https://localhost:44306/api/TestFromURI?Id=1&Name=Raj&City=Lucknow



**23) Why to use “FromBody” in Web API?**

This attribute is used to force Web API **to read the simple type from message body**. “FromBody” attribute is along with parameter. Eg:

[Route("TestFromBody")]

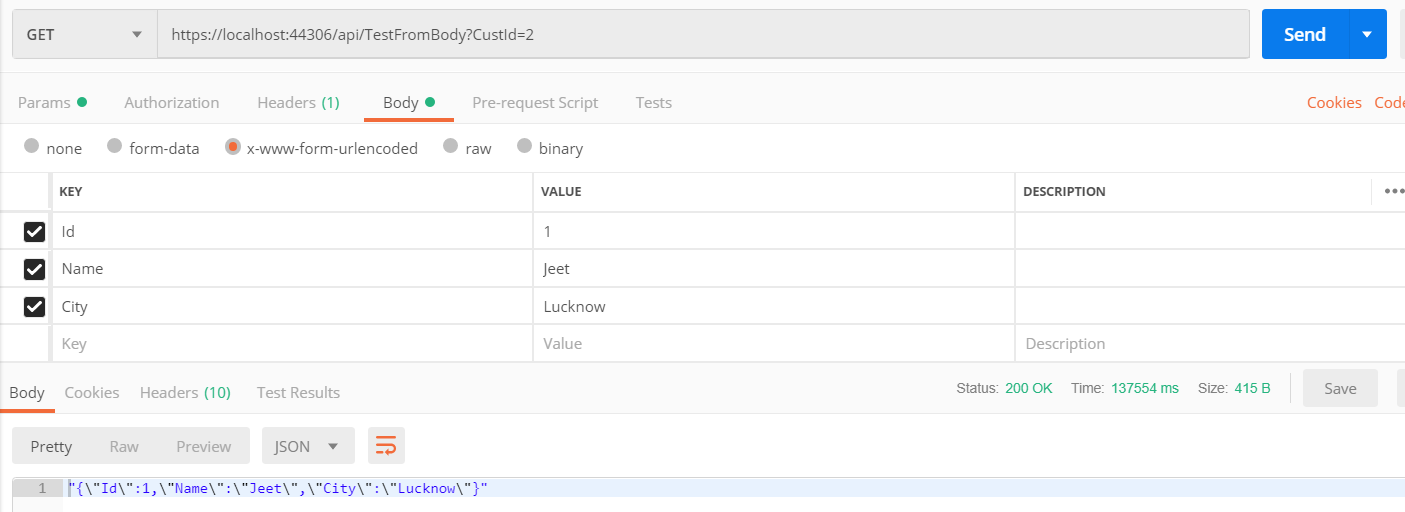
public string TestFromBody([FromUri] int CustId, [FromBody] Employee employee)

{

return JsonConvert.SerializeObject(employee);

}

Request would be: https://localhost:44306/api/TestFromBody?CustId=2



**24) Basic difference between Web Services and Web APIs**

**Web Service:**

1) It is a SOAP based service and returns data as XML.

2) It supports the HTTP, UDP, and custom transport (TCP) protocol.

3) It can be hosted on IIS and Self-hosting.

4) It is not open source, but can be used by any client that understands XML.

5) It requires a SOAP protocol to receive and send data over the network, so it is not a light-weight architecture.

6) WCF supports message queues, message security, duplex communication, transaction whereas Web API doesn’t support.

**Web API:**

1) A Web API is a HTTP based service and returns JSON or XML data by default.

2) It only supports the HTTP protocol.

3) It can be hosted on IIS and Self-hosting.

4) It is open source and it can be used by any client that understands JSON or XML.

5) It is light-weight architecture and good for devices which have limited bandwidth, like mobile devices.

6) It offers support for content negotiation whereas WCF doesn’t support.

**25) How to get specific output from the same WebAPI**

First we need to add formatter for both xml and json in **WebApiConfig.config** file

// Adding formatter for Json

config.Formatters.JsonFormatter.MediaTypeMappings.Add(

new QueryStringMapping("type", "json", new MediaTypeHeaderValue("application/json")));

// Adding formatter for XML

config.Formatters.XmlFormatter.MediaTypeMappings.Add(

new QueryStringMapping("type", "xml", new MediaTypeHeaderValue("application/xml")));

**For JSON response make the request as**

<https://localhost:44306/api/employee?type=json>

**For XML response make the request as**

https://localhost:44306/api/employee?type=xml

**26) How to implementing Caching in Web API**

1- First we need to override the **“OnActionExecuted”** method as below

public class CacheFilter : ActionFilterAttribute

{

public int TimeDuration { get; set; }

public override void OnActionExecuted(HttpActionExecutedContext actionExecutedContext)

{

actionExecutedContext.Response.Headers.CacheControl = new CacheControlHeaderValue

{

MaxAge = TimeSpan.FromSeconds(TimeDuration),

MustRevalidate = true,

Public = true

};

}

}

2- Then we can add that filter class to any api method which we want to cache

[CacheFilter(TimeDuration = 100)]

public async Task<IHttpActionResult> GetData()

{

Dictionary<object, object> obj = new Dictionary<object, object>();

obj.Add("1", "Punjab");

obj.Add("2", "Assam");

return Ok(obj);

}

**OR**

Simply we can use the **OutputCache attribute**

[System.Web.Mvc.**OutputCache**(Duration = 100)]

**27) How to allow a method to be invoked only when specific parameter is passed.**

Here is the e.g. this method is invoked only when there is integer in query string.

[Route("GetNumber/{num:regex([0-9])}")]

public int GetNumber(int num)

{

return num;

}

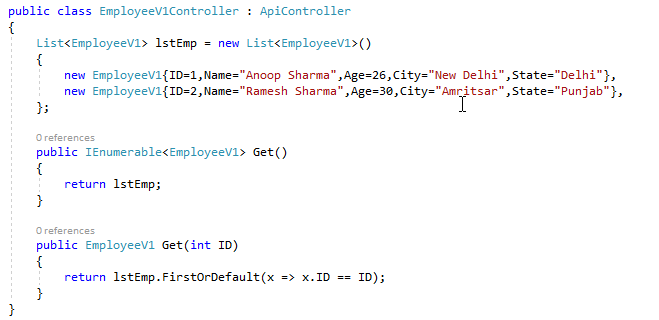
**28) Web API versioning**

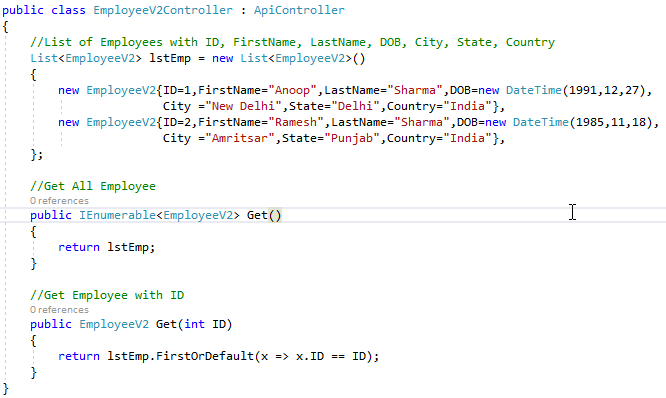
Web API Versioning can be done by using the following methods:

1. URI
2. QueryString parameter
3. Custom Header parameter
4. Accept Header parameter

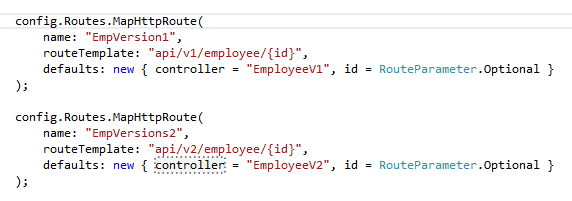
**Versioning using URI**: Let’s say, we have an existing running API in which one URI returns some response. All Clients are consuming the same API and one client wants some changes by requesting to add new properties. With Versioning, we can achieve the same without breaking the existing API flow. In this case, Web API Versioning using URI is one of the best ways to achieve the same.

For Demonstration, we have two controller EmployeeV1 and EmployeeV2. Both will return different data as EmployeeV1 return employees details with ID, Name, Age, City, State property and EmployeeV2 returns employees with ID, FirstName, LastName (In V1, we have name property), DOB (In V1 we have Age property), City, State, Country etc. property.

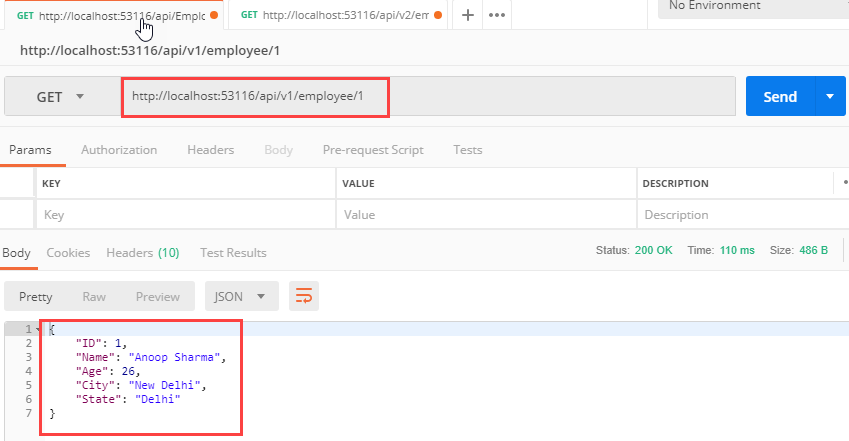




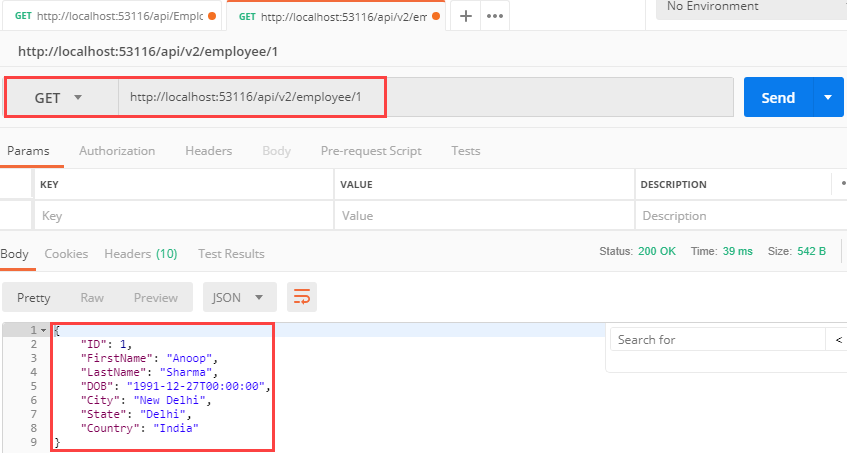
Now open WebApiConfig.cs file and configure the route as mentioned in below image.



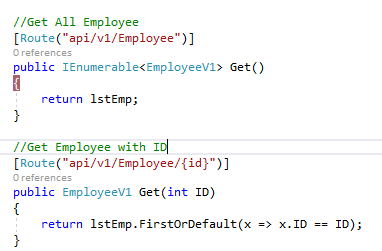
In the above image, we have configured the route so that if Web API receives an HTTP request, it tries to match with the one of the routes in the routing table and call the mapped API controller. Let’s try to hit the API with the Postman.



In the above image, we hit the Web API with the configured V1 route. As we already specified in the route, “/api/v1/employee” will call the EmployeeV1Controller. Now, let's call the V2 API with the Postman. In the below image, we can clearly see that we are able to call EmployeeV2Controller without any issue.



We can also use attribute-based routing for URI based API Versioning.



**29) How to check for specific headers before processing the Web Api request**

[HttpGet]

[Route("TestHeaders")]

//[HeaderFilter] // Use this when you want to do this using Filters

public string TestHeaders([FromBody] object jsonData)

{

var re = Request;

var headers = re.Headers;

string token = string.Empty;

if (headers.Contains("Custom"))

{

token = ((IList<string>)headers.GetValues("Custom"))[0];

}

return token;

}

public class HeaderFilter : ActionFilterAttribute

{

public override void OnActionExecuting(HttpActionContext actionContext)

{

var re = actionContext.Request;

var headers = re.Headers;

string token = string.Empty;

if (headers.Contains("Custom"))

{

token = ((IList<string>)headers.GetValues("Custom"))[0];

}

else

{

actionContext.Response = actionContext.Request.CreateErrorResponse(

HttpStatusCode.BadRequest, "Not a proper request");

}

}

}

**In Dot Net Core use the [FromHeader] attribute.**

[HttpGet]

public string TestHeaders([FromHeader]string Custom)

{

string token = string.Empty;

if (!string.IsNullOrEmpty(Custom))

{

token = Custom;

}

return token;

}

|  |  |
| --- | --- |
| **Method** | **Usage** |
| GET | To retrieve the information from the server. Parameters will be appended in the query string. |
| POST | To create a new resource. |
| PUT | To update an existing resource. |
| HEAD | Identical to GET except that server do not return the message body. |
| OPTIONS | It represents a request for information about the communication options supported by the web server. |
| DELETE | To delete an existing resource. |
| PATCH | To full or partial update the resource. |

**Ways to overload action methods in web api**

1. Overload action methods using a different verb (HttpGet, HttpPost)
2. Overload action methods using the [ActionName] attribute
3. Overload action methods by using attribute routing
4. Overload action methods using the [NonAction] attribute

TokenBased Authenticaton Method override

<https://blogs.msdn.microsoft.com/mvpawardprogram/2017/05/02/adding-webapi-oauth-auth/>

<https://stackoverflow.com/questions/26357054/return-more-info-to-the-client-using-oauth-bearer-tokens-generation-and-owin-in>