* 1. **Web api**
     + http protocol
  + controller – action – routing
  + (GET/POST/PUT/PATCH/DELETE)

Eg:

[HttpPost]

public void SaveNewValue([FromBody]string value)

{

}

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.

* + 1. Difference between Web API and MVC controller

| Web API Controller | MVC Controller |
| --- | --- |
| Derives from System.Web.Http.ApiController class | Derives from System.Web.Mvc.Controller class. |
| Method name must start with Http verbs otherwise apply http verbs attribute. | Must apply appropriate Http verbs attribute. |
| Specialized in returning data. | Specialized in rendering view. |
| Return data automatically formatted based on Accept-Type header attribute. Default to json or xml. | Returns ActionResult or any derived type. |
| Requires .NET 4.0 or above | Requires .NET 3.5 or above |

* + 1. ASP.NET Web API vs WCF

| Web API | WCF |
| --- | --- |
| Open source and ships with .NET framework. | Ships with .NET framework |
| Supports only HTTP protocol. | Supports HTTP, TCP, UDP and custom transport protocol. |
| Maps http verbs to methods | Uses attributes based programming model. |
| Uses routing and controller concept similar to ASP.NET MVC. | Uses Service, Operation and Data contracts. |
| Does not support Reliable Messaging and transaction. | Supports Reliable Messaging and Transactions. |
| Web API can be configured using HttpConfiguration class but not in web.config. | Uses web.config and attributes to configure a service. |
| Ideal for building RESTful services. | Supports RESTful services but with limitations. |

1. **OData**
2. The Open Data Protocol (OData) is a data access protocol for the web. OData provides a uniform way to query and manipulate data sets through CRUD operations (create, read, update, and delete).
3. OData defines parameters that can be used to modify an OData query. The client sends these parameters in the query string of the request URI. For example, to sort the results, a client uses the $orderby parameter:
4. http://localhost/Products?$orderby=Name
5. The OData specification calls these parameters query options. You can enable OData query options for any Web API controller in your project — the controller does not need to be an OData endpoint. This gives you a convenient way to add features such as filtering and sorting to any Web API application.

* Web API 5.2
* OData v4
* Visual Studio 2017 (download Visual Studio 2017 [here](https://visualstudio.microsoft.com/downloads/))
* Entity Framework 6
* .NET 4.7.2

## Configure the OData endpoint

Open the file App\_Start/WebApiConfig.cs. Add the following **using** statements:

C#Copy

using ProductService.Models;

using Microsoft.AspNet.OData.Builder;

using Microsoft.AspNet.OData.Extensions;

Then add the following code to the **Register** method:

C#Copy

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// New code:

ODataModelBuilder builder = new ODataConventionModelBuilder();

builder.EntitySet<Product>("Products");

config.MapODataServiceRoute(

routeName: "ODataRoute",

routePrefix: null,

model: builder.GetEdmModel());

}

}

**Inherits odatacontroller:**

public class ProductsController : ODataController

public async Task<IHttpActionResult> Patch([FromODataUri] int key, Delta<Product> product)

{

OData supports two different semantics for updating an entity, PATCH and PUT.

* PATCH performs a partial update. The client specifies just the properties to update.
* PUT replaces the entire entity.
  1. **Query Options**

The following are the OData query options that ASP.NET WebAPI supports:

* $orderby: Sorts the fetched record in particular order like ascending or descending.

[*?$orderby=ProductName desc*](http://localhost:50875/v1/Products/Product/allproducts?$orderby=ProductName%20desc)

*$orderby=ProductName asc*

* $select: Selects the columns or properties in the result set. Specifies which all attributes or properties to include in the fetched result.
* $skip: Used to skip the number of records or results. For example, I want to skip first 100 records from the database while fetching complete table data, then I can make use of $skip.

[*$top=5&$skip=3*](http://localhost:50875/v1/Products/Product/allproducts?$top=5&$skip=3)

[*$orderby=ProductName asc &$skip=6*](http://localhost:50875/v1/Products/Product/allproducts?$orderby=ProductName%20asc%20&$skip=6)

* $top: Fetches only top n records. For e.g. I want to fetch top 10 records from the database, then my particular service should be OData enabled to support $top query option. -  "?**$top=2"**

*$orderby=ProductId asc&$top=5*

* $expand: Expands the related domain entities of the fetched entities.
* $filter: Filters the result set based on certain conditions, it is like where clause of LINQ. For e.g. I want to fetch the records of 50 students who have scored more than 90% marks, and then I can make use of this query option.

**$filter=ProductName eq 'computer'**

[*$filter=ProductId lt 3*](http://localhost:50875/v1/Products/Product/allproducts?$filter=ProductId%20lt%203)

[*$filter=ProductId ge 3 and ProductId le 5*](http://localhost:50875/v1/Products/Product/allproducts?$filter=ProductId%20ge%203%20and%20ProductId%20le%205)

[*$filter=substringof('IPhone',ProductName)*](http://localhost:50875/v1/Products/Product/allproducts?$filter=substringof(%27IPhone%27,ProductName))

* $inlinecount: This query option is mostly used for pagination at client side. It tells the count of total entities fetched from the server to the client.

[Queryable]

[GET("allproducts")]

[GET("all")]

public HttpResponseMessage Get()

{

var products = \_productServices.GetAllProducts().AsQueryable();

var productEntities = products as List<ProductEntity> ?? products.ToList();

if (productEntities.Any())

return Request.CreateResponse(HttpStatusCode.OK, productEntities.AsQueryable());

throw new ApiDataException(1000, "Products not found", HttpStatusCode.NotFound);

}

* 1. Standard filter operators

The Web API supports the standard OData filter operators listed in the following table.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| **Comparison Operators** | | |
| **Eq** | Equal | $filter=revenue eq 100000 |
| **Ne** | Not Equal | $filter=revenue ne 100000 |
| **Gt** | Greater than | $filter=revenue gt 100000 |
| **Ge** | Greater than or equal | $filter=revenue ge 100000 |
| **Lt** | Less than | $filter=revenue lt 100000 |
| **Le** | Less than or equal | $filter=revenue le 100000 |
| **Logical Operators** | | |
| **And** | Logical and | $filter=revenue lt 100000 and revenue gt 2000 |
| **Or** | Logical or | $filter=contains(name,'(sample)') or contains(name,'test') |
| **Not** | Logical negation | $filter=not contains(name,'sample') |
| **Grouping Operators** | | |
| **( )** | Precedence grouping | (contains(name,'sample') or contains(name,'test')) and revenue gt 5000 |

* 1. Standard query functions

The web API supports these standard OData string query functions.

| **Function** | **Example** |
| --- | --- |
| **contains** | $filter=contains(name,'(sample)') |
| **endswith** | $filter=endswith(name,'Inc.') |
| **startswith** | $filter=startswith(name,'a') |

Paging:

[Queryable(PageSize = 10)]

* 1. Query Constraints:

[Queryable(AllowedQueryOptions =AllowedQueryOptions.Filter | AllowedQueryOptions.OrderBy)]

[GET("allproducts")]

[GET("all")]

public HttpResponseMessage Get()