ADVANCED API

Documentation

Abstract

In this module I learned advanced concept of API development in ASP.NET

Dhruvil Dobariya

dhruvildobariya21@gmail.com

INDEX

ACTION RESULT

1.1 INTRODUCTION

- We have four types of result which is possible to return in ASP.NET web API.
  - HttpResponseMessage
  - IHttpActionResult
  - Void
  - Any type of entity

1.2 VOID:

- Void means our action return nothing.

Example:

```
[HttpGet]
public void Ok()
{
    // code...
}
```

1.3 ENTITY:

- In this case our action return some entity in type of object which should be like, int, string, Product.

Example:

```
[HttpGet]
public List<Product> Ok()
{
    List<Product> products = new List<Product>();
    return products;
}
```

1.4 HTTPRESPONSEMESSAGE

- HttpResponseMessage represents HTTP Response Message as per MSDN definition.
- If the return type of the action method is one of the Web API's action results, then the API converts the return value to a HTTP Response Message.

- This action result gives us more flexibility to create our own custom message using it's properties.

Example:

```
[HttpGet]
public HttpResponseMessage Ok()
{
    return new HttpResponseMessage()
    {
        Content = new StringContent("This is content"),
        StatusCode = HttpStatusCode.OK, // 200
        RequestMessage = new HttpRequestMessage(HttpMethod.Get, "request
uri")
    };
}
```

1.5 IHttpActionResult

- The IHttpActionResult action result was introduced in Web API 2.
- It also returns the HttpResponseMessage. But the code we write to send the HTTP Response Message will be reduced with this interface.
- The preceding code returns the HTTP Response message as 200 Ok Status Code.
- As per IHttpActionResult's definition in the ASP.Net/Web API, it acts like a factory for HttpResponseMessage and comes with built-in responses, like Ok, BadRequest, NotFound, Unauthorized, Exception, Conflict and Redirect.

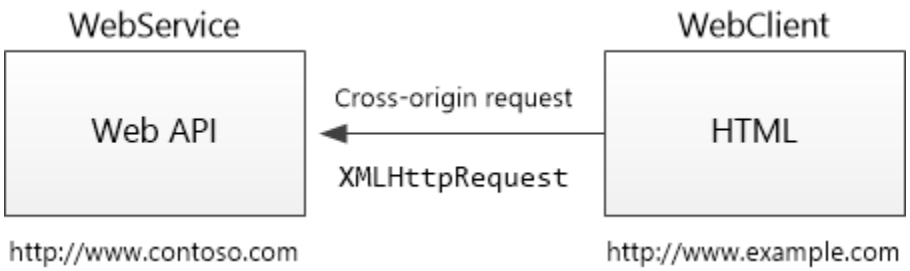
Example:

```
[HttpGet]
public IHttpActionResult Ok()
{
    return Ok("This is Content");
}
```

CORS

2.1 INTRODUCTION:

- When we use web API and frontend on different origins at that time CORS come in picture.
- CORS means cross origin resource sharing.
- For security reason browser restrict the cross origin requests.
- CORS allows us to access cross origin resource.



- Here is the example of same origin,
  - http://example.com/foo.html
  - http://example.com/bar.html
- Here is the example of cross origin,
  - http://example.net - Different domain
  - http://example.com:9000/foo.html - Different port
  - https://example.com/foo.html - Different scheme
  - http://www.example.com/foo.html - Different subdomain

2.2 ENABLE CORS

- We can enable CORS using “Microsoft.AspNet.WebApi.Cors” package.
- We can configure CORS using EnableCorsAttribute at action level, controller level and global level.
- We have four properties in for CORS.

Origins:

- Here, we need to set Origins which means from which domain the requests will accept.
- If we have more than one domain, then you can set as comma separated.
- Additionally, if you want any domain request to be accepted then use wild card as "*".

Request Headers:

- The Request header parameter specifies which Request headers are allowed.
- To allow any header set value to "*".

HTTP Methods:

- The methods property specifies which HTTP methods are allowed to access the resource.
- We can use comma-separated values when you have multiple HTTP methods like "get, put, post".
- When we want to allow all HTTP methods, then we should use the wildcard value "*".

Exposed Headers:

- By default, the browser does not expose all of the response headers to the application.
- Which mean browser only give access to client these headers which is exist in default set like, Cache-Control, Content-Language, Content-Type, Expires, Last-Modified, Pragma.
- Which means if we add any custom header in response then it don't accessed by client.
- So exposed header should help us to accessed our custom header by client which requested from cross origin.

2.2.1 Action Level:

- Here we use EnableCors attribute on action method.
- Action level configuration override configuration of controller level as well as global level configuration.

Example:

```
public HttpResponseMessage Get(int id, string name)
{
    HttpContext.Current.Response.Headers.Add("X-Id", id.ToString());
    HttpContext.Current.Response.Headers.Add("X-Name", name);
    return new HttpResponseMessage()
    {
        Content = new StringContent(JsonConvert.SerializeObject(new
{id,name})),
        StatusCode = HttpStatusCode.OK
    };
}
```

2.2.2 Controller Level:

- Here we use EnableCors attribute on controller, so it applied all the method of controller.

- Controller level configuration override global level configuration.

Example:

```
[EnableCors(origins: "www.something1.com, www.somthing2.com", headers: "X-
Header1, X-Header2", methods: "GET,PUT,POST,DELETE", exposedHeaders: "X-
Id,X-Name")]
public class HomeController : ApiController
{
    [HttpGet]
    public HttpResponseMessage Get(int id, string name)
    {
        HttpContext.Current.Response.Headers.Add("X-Id", id.ToString());
        HttpContext.Current.Response.Headers.Add("X-Name", name);
        return new HttpResponseMessage()
        {
            Content = new StringContent(JsonConvert.SerializeObject(new {
id, name })),
            StatusCode = HttpStatusCode.OK
        };
    }

    [HttpPost]
    public HttpResponseMessage Set(int id, string name)
    {
        HttpContext.Current.Response.Headers.Add("X-Id", id.ToString());
        HttpContext.Current.Response.Headers.Add("X-Name", name);
        return new HttpResponseMessage()
        {
            Content = new StringContent(JsonConvert.SerializeObject(new {
id, name })),
            StatusCode = HttpStatusCode.OK
        };
    }
}
```

2.2.3 Global Level:

- Here we configure COES in WebConfig.cs file.
- This configuration applied on whole application.

Example:

```
using System.Web.Http;
using System.Web.Http.Cors;
```

Advanced API

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

            // CORS configuration
            EnableCorsAttribute cors = new EnableCorsAttribute("*", "*",
"*", "X-Id, X-Name");
            config.EnableCors(cors);

            // Web API routes
            config.MapHttpAttributeRoutes();

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

FILTERS

3.1 INTRODUCTION: