Module-6

27. Building WEB API.

Understanding HTTP Verbs and Implement GET, POST, PUT, DELETE:

The HTTP verbs comprise a major portion of our “uniform interface” constraint and provide us the action counterpart to the noun-based resource. The primary or most commonly used HTTP verbs are POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations, respectively. There are a number of other verbs, too, but are utilized less frequently. Of those less-frequent methods, OPTIONS and HEAD are used more often than others.

Action method can be named as HTTP verbs like Get, Post, Put, Patch or Delete. However, we can append any suffix with HTTP verbs for more readability. For example, Get method can be GetAllStudents() or any other name which starts with Get.

Analogy with crud operations:

Create – Post

Read - Get

Update - Put

Delete – Delete

Get:

This method is used to retrieve a representation of a resource. A GET request is considered safe because as HTTP specifies, these requests are used only to read data and not change it.

The disadvantage of GET requests is that they can only supply data in the form of parameters encoded in the URI or as cookies in the cookie request header.

For example: Displaying your registered account details on any website has no effect on the account. If you are using Internet Explorer you can refresh a page and the page will show information that resulted from a GET, without displaying any kind of warning. We have also other HTTP components like PROXIES that automatically retry GET requests if they encounter a temporary network connection problem.

Tasks that you can do with GET requests are:

- You can cache the requests
- It can remain in the browser history

- You can bookmark the requests
- You can apply length restrictions with GET requests
- It is used only to retrieve data

POST

The HTTP POST request is used to create a new record in the data source. For some resources, it may be used to alter the internal state. For others, its behavior may be that of a remote procedure call. If you try to refresh a page while using a POST request, then you will get an error like page can't be refreshed. Why? Because the request cannot be repeated without explicit approval by the user.

Some important points about POST requests:

- Data will be re-submitted
- It cannot be bookmarked
- It cannot be cached
- Parameters are not saved in browser history
- No restrictions. Binary data is also allowed
- Data is not displayed in the URL

PUT

The HTTP PUT method is used to update an existing record in the data source.

Simply PUT updates data in the repository, replacing any existing data with the supplied data.

For example: Suppose we wanted to change the image that we have something already on the server, So, we just upload a new one using the PUT verb.

Some good points of PUT requests are:

- It completes as possible
- It's as seamless as possible
- Easy to use via HTML
- It should integrate well with servers that already support PUT

DELETE

The HTTP DELETE request is used to delete an existing record in the data source.

A DELETE request is as simple as its name implies; it just deletes, it is used to delete a resource identified by a URI and on successful deletion, it returns HTTP status 200 (OK) along with a response body. The important and unique thing about a DELETE request is that the client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully.

Some HTTP infrastructures do not support the DELETE method. In such cases, the request **should** be submitted as a POST with an additional X-HTTP-Method-Override header set to DELETE.

Example DELETE request: DELETE /data/myDataApp/myorder/-/takenOrdernumber('00777')

First let us have a look at the code in Friend.cs.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web;

namespace WebApi.Models
{
   public class Friend
   {
      /// <summary>
      /// It is friend id
      /// </summary>
      [Required(ErrorMessage = "Friend id is required")]
      [Range(1, 100)]
      public int id { get; set; }
      /// <summary>
      /// It is firstname of friend
      /// </summary>
      public string firstname { get; set; }
      /// <summary>
      /// It is lastname of friend
      /// </summary>
      public string lastname { get; set; }
      /// <summary>
      /// It is location of friend
      /// </summary>
      public string location { get; set; }
      /// <summary>
      /// It is hiredate of friend
      /// </summary>
      [Required(ErrorMessage = "Hire date is required")]
      public DateTime hiredate { get; set; }
   }
}
```

Then look at the code in WebApiConfig.cs.

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Web.Http;

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

         // Web API routes
         config.MapHttpAttributeRoutes();

         // friend route
         config.Routes.MapHttpRoute(
            name: "Friend",
            routeTemplate: "api/friend/{id}",
            defaults: new { controller = "Friend", id = RouteParameter.Optional },
            constraints: new { id = "/d+" }
         ) ;

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

Then look at the code in FriendController which has all the methods.

FriendController file:

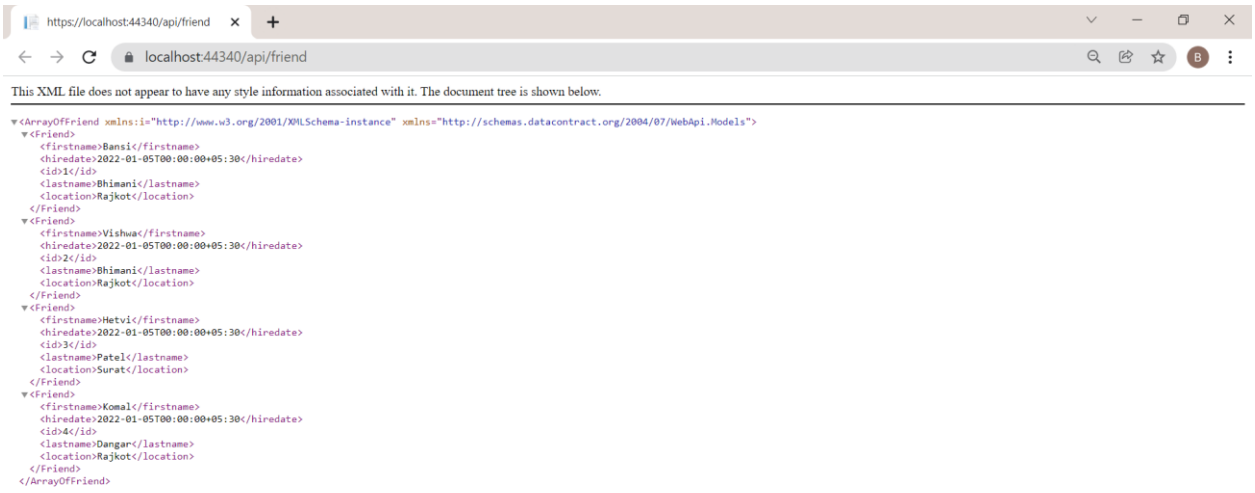
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using WebApi.Models;

namespace WebApi.Controllers
{
   public class FriendList
   {
   List<Friend> friends = new List<Friend>();
   public FriendList()
   {
      friends.Add(new Friend { id = 1, firstname="Bansi", lastname="Bhimani", location="Rajkot", hiredate=DateTime.Today
});
      friends.Add(new Friend() { id = 2, firstname = "Vishwa", lastname = "Bhimani", location = "Rajkot", hiredate =
DateTime.Today });
      friends.Add(new Friend() { id = 3, firstname = "Hetvi", lastname = "Patel", location = "Surat", hiredate =
DateTime.Today });
```

```
        friends.Add(new Friend() { id = 4, firstname = "Komal", lastname = "Dangar", location = "Rajkot", hiredate =
DateTime.Today });
        }

    public List<Friend> GetCompleteList()
    {
        return friends;
    }

    public Friend GetListByID(int id)
    {
        return friends.Where(temp => temp.id == id).FirstOrDefault();

    }

    public void AddItem(Friend newFriend)
    {
        friends.Add(newFriend);
    }

    public void RemoveItem(int id)
    {
        friends.Remove(friends.Where(x => x.id == id).FirstOrDefault());
    }

    public void EditItem(int id, Friend EditFriend)
    {
        friends[id] = EditFriend;
    }
}
    public class FriendController : ApiController
    {
        FriendList objFriendList = new FriendList();
        // Read all friend
        public List<Friend> Get()
        {
            List<Friend> friends= objFriendList.GetCompleteList();
            return friends;
        }

        // Read friend by id
        public Friend Get(int id)
        {
            return objFriendList.GetListByID(id);
        }

        // add friend
        public void Post(Friend newFriend)
        {
            objFriendList.AddItem(newFriend);
        }

        // edit friend by id
        public void Put(int id, Friend editFriend)
        {
            objFriendList.EditItem(id, editFriend);
        }

        // Delete friend
```

```
        public void Delete(int id)
        {
            objFriendList.RemoveItem(id);
        }
    }

}
```

As per the Web API naming convention, action method that starts with a work "Get" will handle HTTP GET request. We can either name it only Get or with any suffix.

Demonstration of get request:



Understanding JSON Structure:

JSON (JavaScript Object Notation) is most widely used data format for data interchange on the web. This data interchange can happen between two computer applications at different geographical locations or running within the same machine.

The good thing is that JSON is a human-readable as well as a machine-readable format. So while applications/libraries can parse the JSON documents – humans can also look at the data and derive the meaning from it.

Web API handles HTTP request with JSON or XML data and parses it to a Student object based on Content-Type header value and the same way it converts insertedStudent object into JSON or XML based on Accept header value.