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

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

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

namespace WebApi.Models
{
   public class Friend
   {
     public int id { get; set;}
      public string firstname { get; set; }
      public string lastname { get; set; }
      public string location { get; set; }
      public DateTime dateOfHire { get; set; }

      public Friend(int id, string firstname, string lastname, string location, DateTime dateOfHire)
      {
         this.id = id;
         this.firstname = firstname;
         this.lastname = lastname;
         this.location = location;
         this.dateOfHire = dateOfHire;
      }
```

```
        public Friend()
        {
        }
    }
}
```

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.

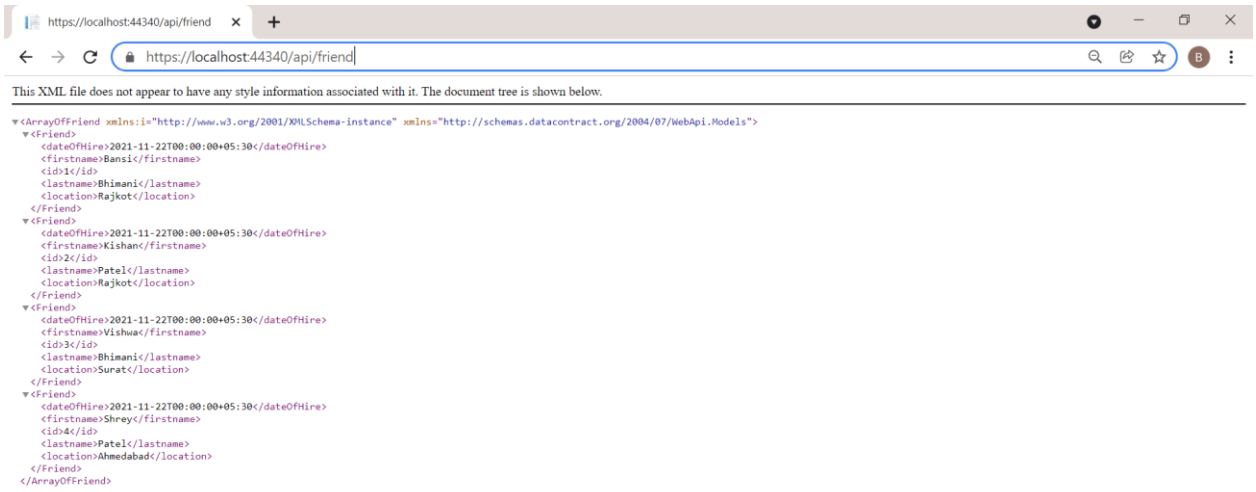
FriendController file:

```
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 FriendController : ApiController
  {
    List<Friend> friends = new List<Friend>
    {
      new Friend(1,"Bansi","Bhimani","Rajkot",DateTime.Today),
      new Friend(2,"Kishan","Patel","Rajkot",DateTime.Today),
      new Friend(3,"Vishwa","Bhimani","Surat",DateTime.Today),
      new Friend(4,"Shrey","Patel","Ahmedabad",DateTime.Today)
    };
    //GET :api/Friend
    [HttpGet]
    public List<Friend> Get()
    {
      return friends;
    }

    // GET: api/Friend/5
    [HttpGet]
    public Friend Get(int id)
    {
      Friend friend = friends.Find(f => f.id == id);
      return friend;
    }
    // POST: api/Friend
    [HttpPost]
    public List<Friend> Post([FromBody]Friend friend)
    {
      friends.Add(friend);
      return friends;
    }

    // PUT: api/Friend/5
    [HttpPut]
    public List<Friend> Put(int id, [FromBody]Friend friend)
    {
      Friend friendToUpdate = friends.Find(f => f.id == id);
      int index = friends.IndexOf(friendToUpdate);

      friends[index].firstname = friend.firstname;
      friends[index].lastname = friend.lastname;
      friends[index].location = friend.location;
      friends[index].dateOfHire = friend.dateOfHire;

      return friends;
    }

    // DELETE: api/Friend/5
    [HttpDelete]
    public List<Friend> Delete(int id)
    {
      Friend friend = friends.Find(f => f.id == id);
      friends.Remove(friend);
      return friends;
    }
  }
}
```

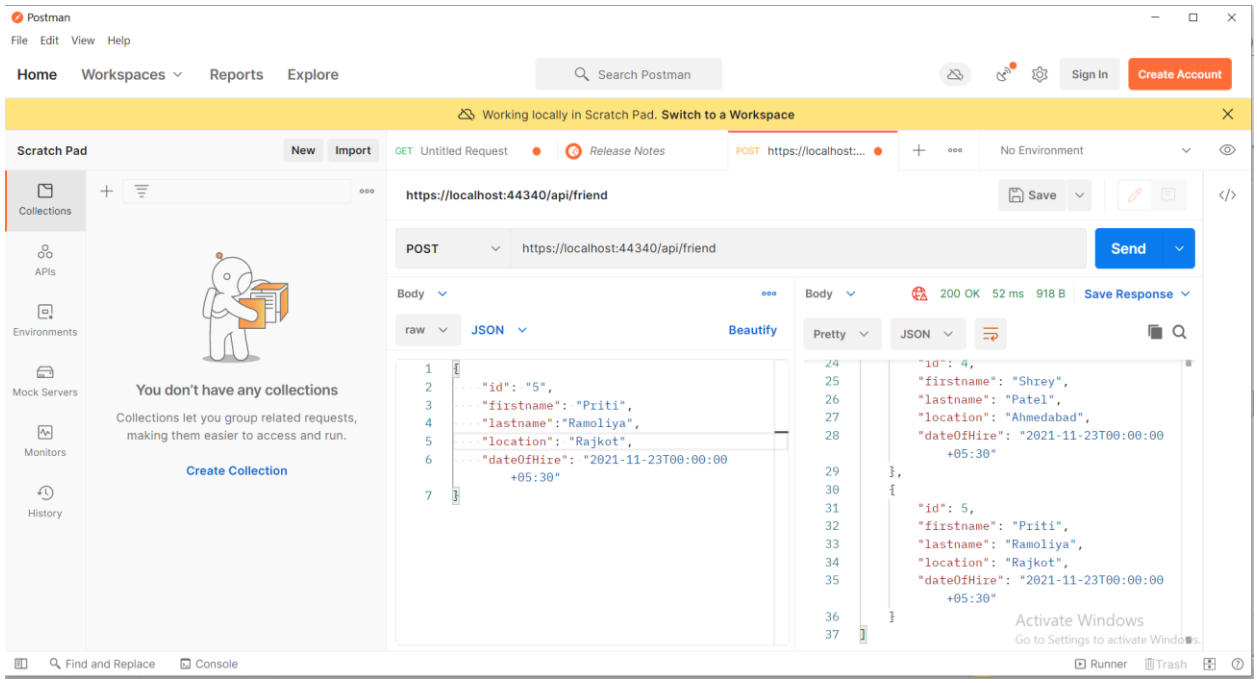
As per the Web API naming convention, action method that starts with a work "Get" will handle HTTP GET request. We can either name it only Get or with any suffix.

Demonstration of get request:



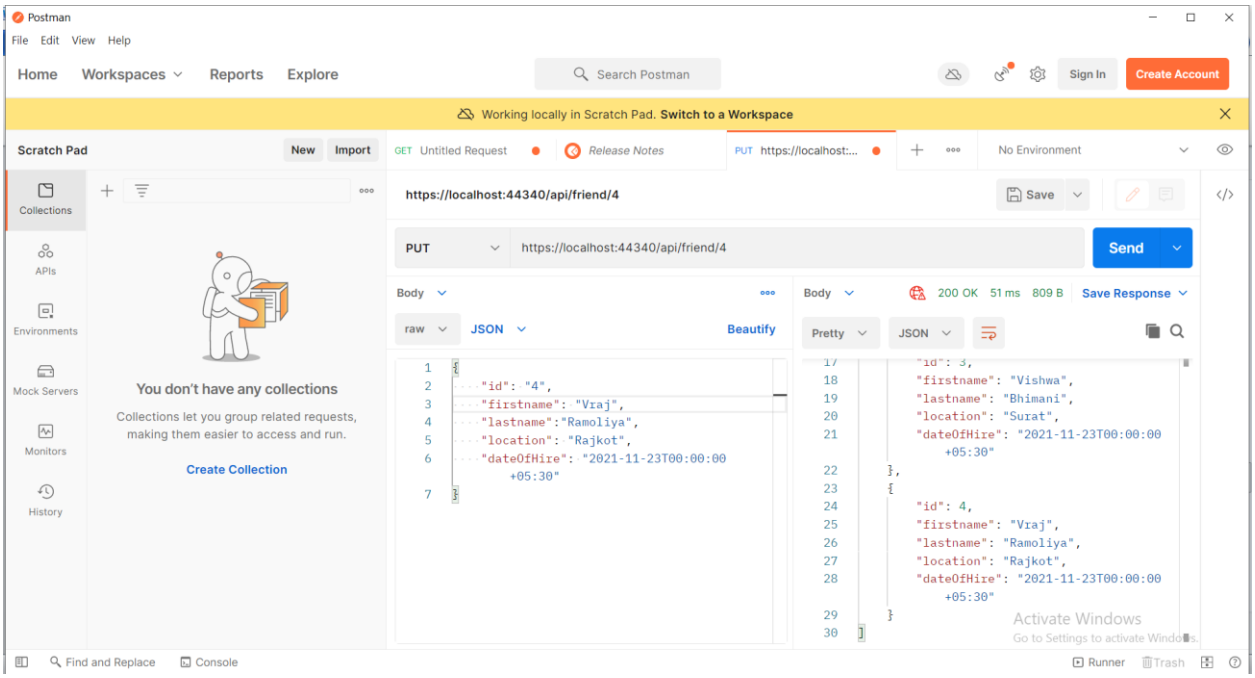
The HTTP POST request is used to create a new record in the data source. So let's create an action method in our FriendController to insert new friend record.

Demonstration of post:

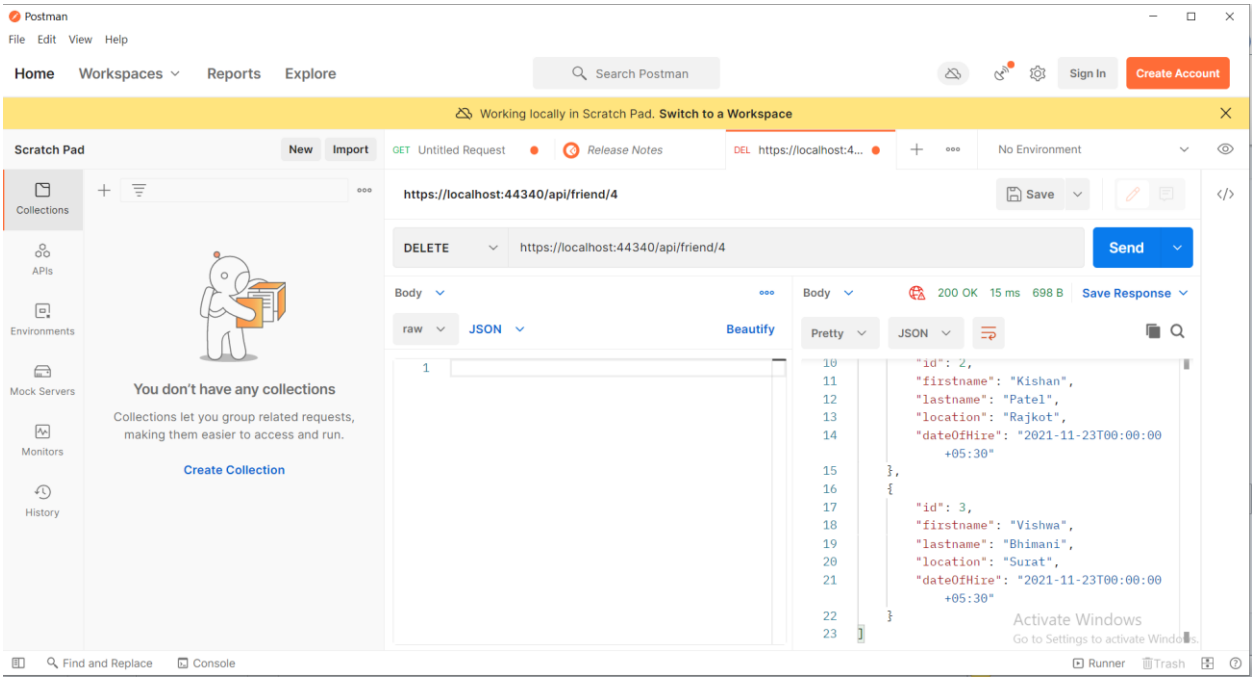


The HTTP PUT method is used to update an existing record in the data source.

Its demonstration is as shown :



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



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.