C# Corner

ASK A QUESTION CONTRIBUTE

g Web API(s) In ASP.NET Core MVC Application

nsume Web APIs in ASP.NET Core MVC application using Factory Pattern and HttpClient

Ravi Raghav Jul 06 2018

4 10 76.1k

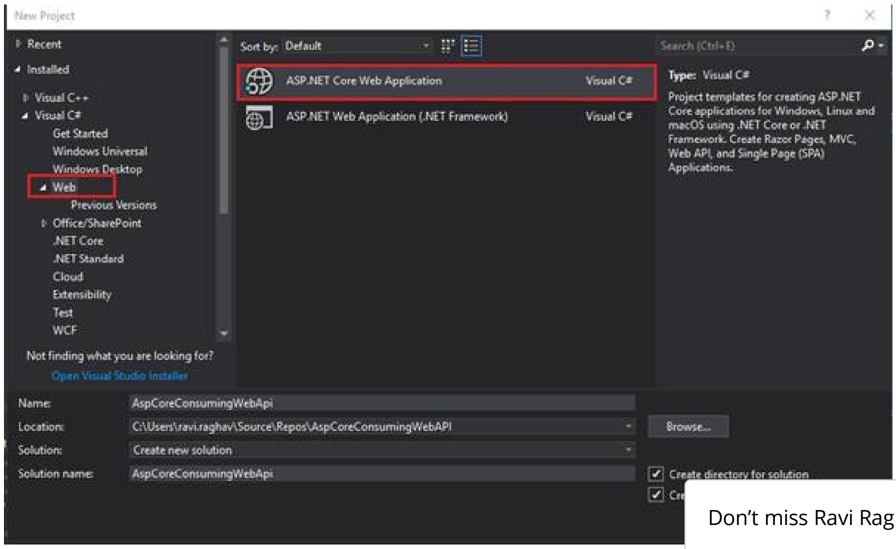
de Warrior?

nsume Web APIs in ASP.NET Core MVC application using Factory Pattern and HttpClient

rvice Oriented or Microservices), we need to make HttpClient calls to get and post the
›n, we don't connect to the database. We will have APIs that will connect to the database
o achieve that, we should know how to consume Web APIs.

So now, let's get started.

In our application, the APIs that we will use are created in my previous article. We will consume those APIs. Let's create our project. We will create a MVC application in ASP.NET Core 2.0.

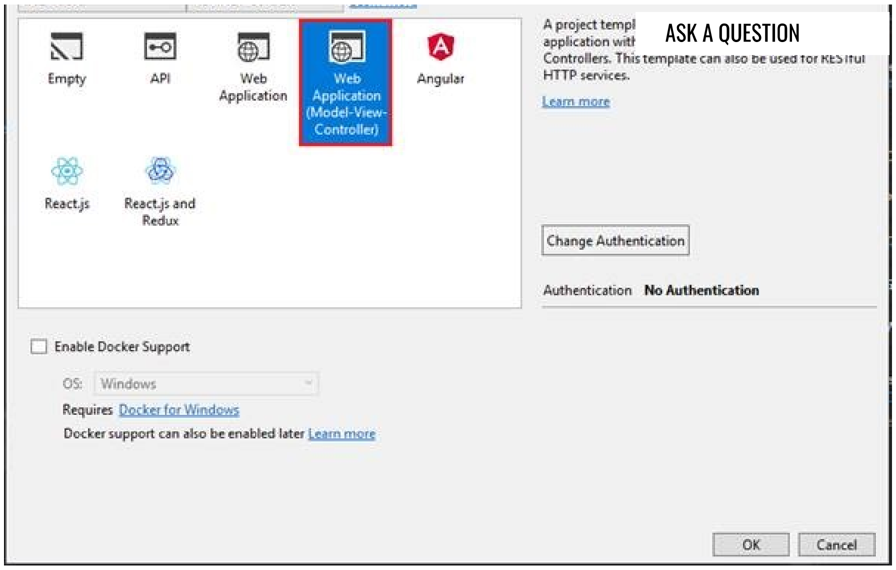


Type: Visual C#
Project templates for creating ASP.NET Core applications for Windows, Linux and macOS using .NET Core or .NET Framework. Create Razor Pages, MVC, Web API, and Single Page (SPA) Applications.

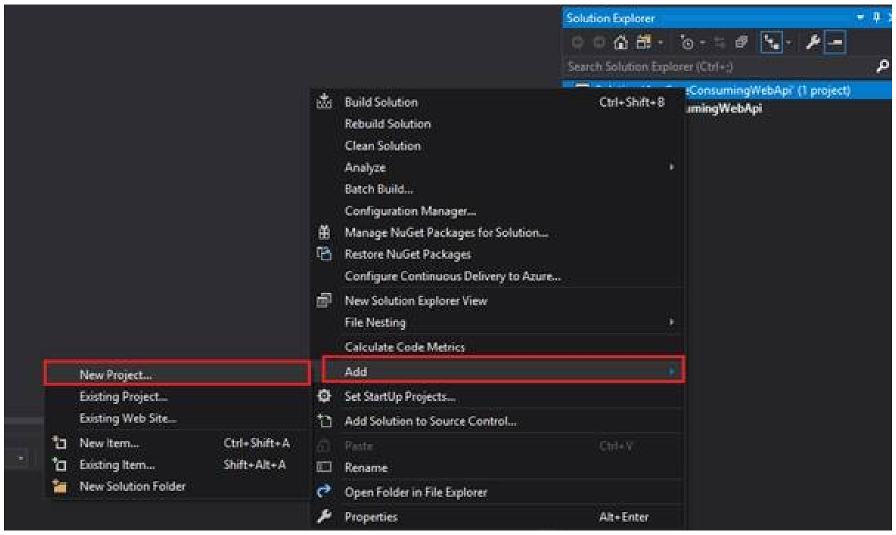
Name: AspCoreConsumingWebApi
Location: C:\Users\ravi.raghav\Source\Repos\AspCoreConsumingWebAPI
Solution: Create new solution
Solution name: AspCoreConsumingWebApi

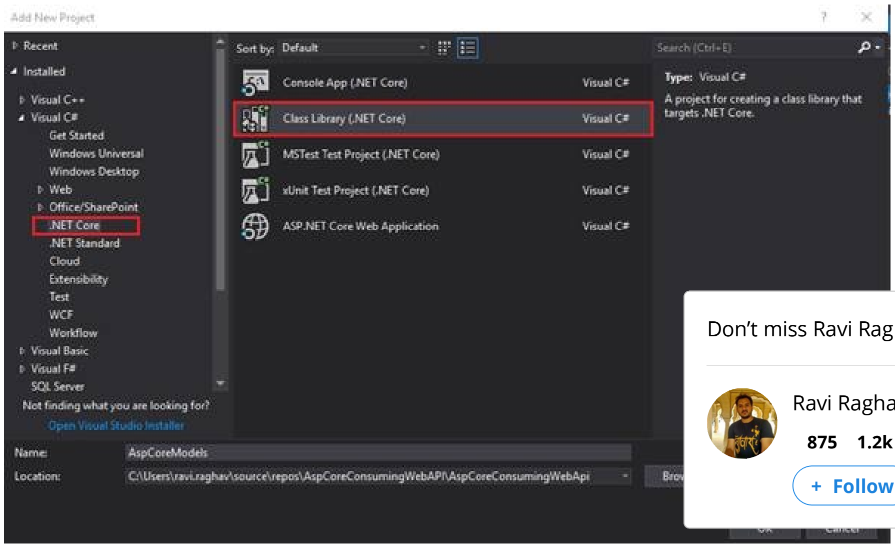
Don't miss Ravi Raghav's next article ×

Ravi Raghav
875 1.2k 654.6k
+ Follow

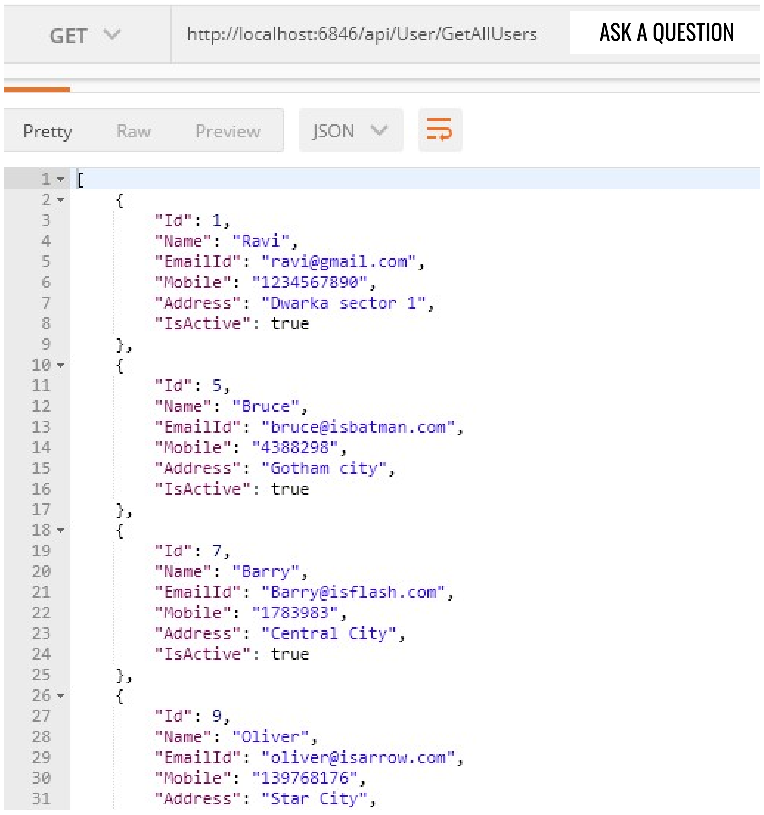

Empty
API
Web Application
Web Application (Model-View-Controller)
Angular
React.js
React.js and Redux
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.
Learn more
Change Authentication
Authentication No Authentication
Enable Docker Support
OS: Windows
Requires Docker for Windows
Docker support can also be enabled later Learn more
OK
Cancel

Once our project is created, we will create another project with the same solution. That project will contain models that will be used in APIs.




Add New Project
Recent
Installed
Visual C++
Visual C#
Get Started
Windows Universal
Windows Desktop
Web
Office/SharePoint
.NET Core
.NET Standard
Cloud
Extensibility
Test
WCF
Workflow
Visual Basic
Visual F#
SQL Server
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Console App (.NET Core) Visual C#
Class Library (.NET Core) Visual C#
MSTest Test Project (.NET Core) Visual C#
xUnit Test Project (.NET Core) Visual C#
ASP.NET Core Web Application Visual C#
Search (Ctrl+E)
Type: Visual C#
A project for creating a class library that targets .NET Core.
Name: AspCoreModels
Location: C:\Users\ravi.raghav\source\repos\AspCoreConsumingWebAPI\AspCoreConsumingWebApi
OK Cancel

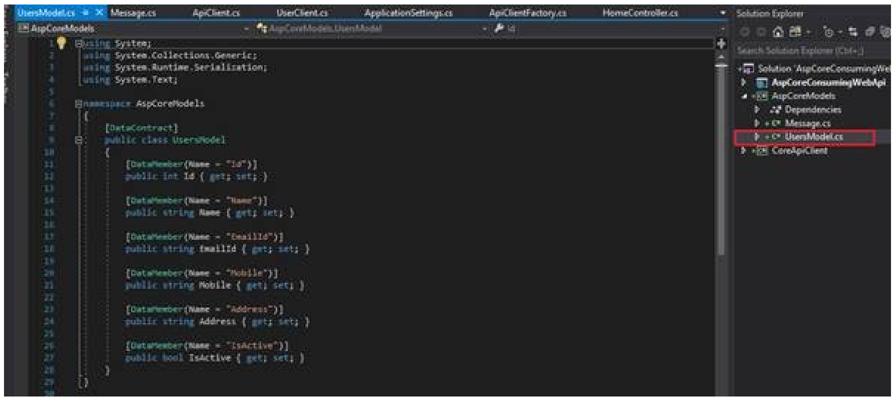
Now, to create models in this project, we will need the request and response of the Web API that we are about to consume



```
[
    {
        "Id": 1,
        "Name": "Ravi",
        "EmailId": "ravi@gmail.com",
        "Mobile": "1234567890",
        "Address": "Dwarka sector 1",
        "IsActive": true
    },
    {
        "Id": 5,
        "Name": "Bruce",
        "EmailId": "bruce@isbatman.com",
        "Mobile": "4388298",
        "Address": "Gotham city",
        "IsActive": true
    },
    {
        "Id": 7,
        "Name": "Barry",
        "EmailId": "Barry@isflash.com",
        "Mobile": "1783983",
        "Address": "Central City",
        "IsActive": true
    },
    {
        "Id": 9,
        "Name": "Oliver",
        "EmailId": "oliver@isarrow.com",
        "Mobile": "139768176",
        "Address": "Star City",
```

So, the GetAllUsers API returns the list of users with certain properties and since this method is get, so there is no request in the body.

Now, according to this, we will create our UsersModel.

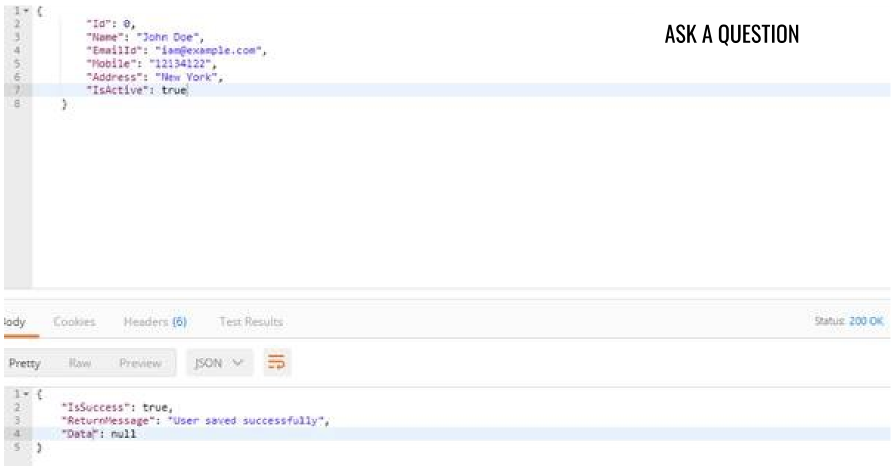


```
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.Text;

namespace AspCoreModels
{
    [DataContract]
    public class UsersModel
    {
        [DataMember(Name = "Id")]
        public int Id { get; set; }

        [DataMember(Name = "Name")]
        public string Name { get; set; }

        [DataMember(Name = "EmailId")]
        public string EmailId { get; set; }

        [DataMember(Name = "Mobile")]
        public string Mobile { get; set; }

        [DataMember(Name = "Address")]
        public string Address { get; set; }

        [DataMember(Name = "IsActive")]
        public bool IsActive { get; set; }
    }
}
```

I have decorated this class with DataContract attribute in order to make it serialized and decorated each property inside it with DataMember and given it's Name property the same as in the response JSON.

This Name property will be helpful in a way such that when you decide to rename your fields then your contract to Web API will not break.

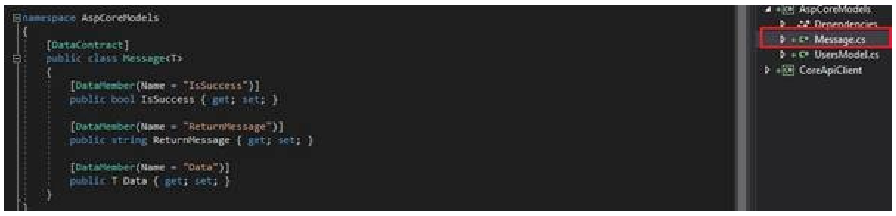
Now we see the response of SaveUsers,



```
{
    "Id": 0,
    "Name": "John Doe",
    "EmailId": "iam@example.com",
    "Mobile": "12134122",
    "Address": "New York",
    "IsActive": true
}
```

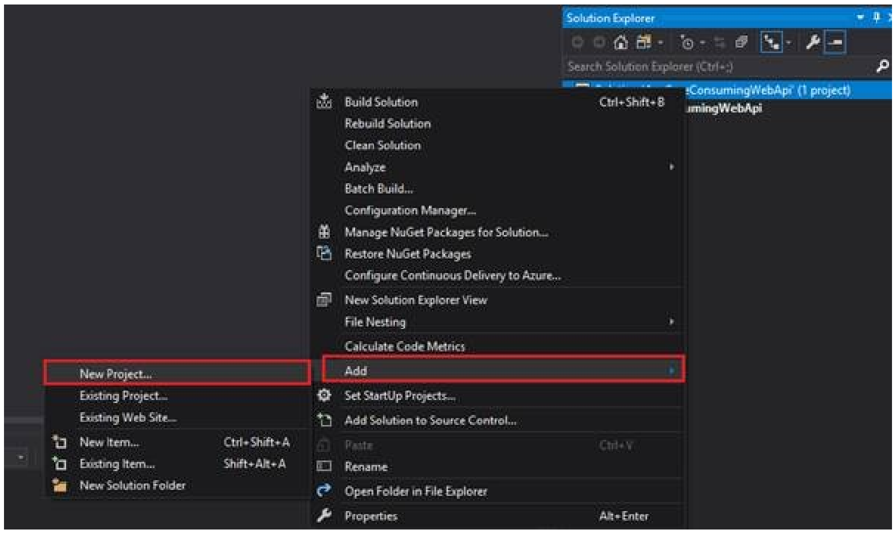
```
{
    "IsSuccess": true,
    "ReturnMessage": "User saved successfully",
    "Data": null
}
```

As we can see input is the user model and output is another model (Since we created API we know the structure of model so I will create another model class Message.cs , I will show you another way to get class from JSON)



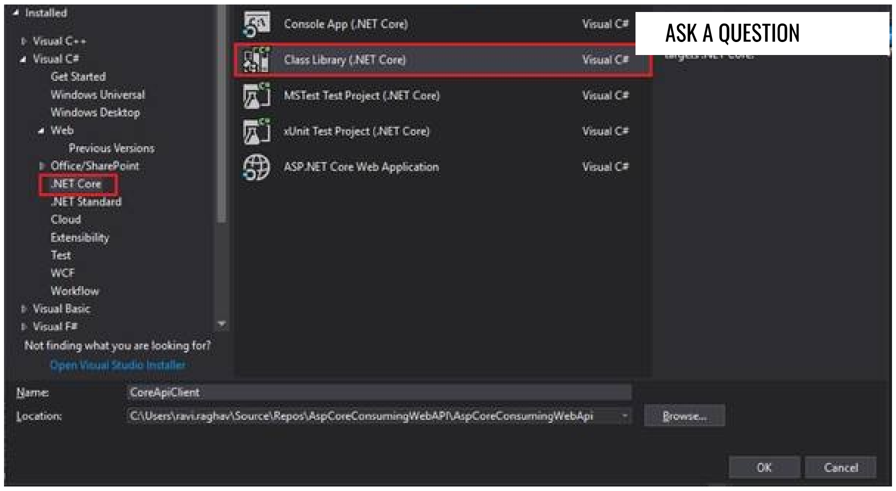
```
namespace AspCoreModels
{
    [DataContract]
    public class Message<T>
    {
        [DataMember(Name = "IsSuccess")]
        public bool IsSuccess { get; set; }

        [DataMember(Name = "ReturnMessage")]
        public string ReturnMessage { get; set; }

        [DataMember(Name = "Data")]
        public T Data { get; set; }
    }
}
```

Now we will create another project inside our solution CoreApiClient. This project will be responsible for communicating with our API.



C# Corner

C#Corner


Installed
Visual C++
Visual C#
Get Started
Windows Universal
Windows Desktop
Web
Previous Versions
Office/SharePoint
.NET Core
.NET Standard
Cloud
Extensibility
Test
WCF
Workflow
Visual Basic
Visual F#
Not finding what you are looking for?
Open Visual Studio Installer
Console App (.NET Core) Visual C#
Class Library (.NET Core) Visual C#
MSTest Test Project (.NET Core) Visual C#
xUnit Test Project (.NET Core) Visual C#
ASP.NET Core Web Application Visual C#
OK
Cancel
Browse...

Name: CoreApiClient
Location: C:\Users\ravi.raghav\Source\Repos\AspCoreConsumingWebAPI\AspCoreConsumingWebApi

ASK A QUESTION CONTRIBUTE

Now in this project we will create a partial class with some internal methods that will help us in consuming web API

Constructor of class

```
private readonly HttpClient _httpClient;
        private Uri BaseEndpoint { get; set; }

        public ApiClient(Uri baseEndpoint)
        {
            if (baseEndpoint == null)
            {
                throw new ArgumentNullException("baseEndpoint");
            }
            BaseEndpoint = baseEndpoint;
            _httpClient = new HttpClient();
        }
```

As you can see we have two private fields of type HttpClient and Uri. In constructor of this class we will pass the baseEndPoint of our API and then we will initialize our HttpClient object

Our next method is a common method to make get calls

```
private async Task<T> GetAsync<T>(Uri requestUrl)
        {

            var response = await _httpClient.GetAsync(requestUrl, HttpCompletionOption.ResponseHeade
            response.EnsureSuccessStatusCode();
            var data = await response.Content.ReadAsStringAsync();
            return JsonConvert.DeserializeObject<T>(data);
        }
```

As you can see this method will take requestUrl as input and then will make a http call to that URL and then we will have the response as string which then we will deserialize using JsonConvert which will be found in Newtonsoft.Json namespace.

Next will be a property for date types

```
private static JsonSerializerSettings MicrosoftDateFormatSettings
        {
            get
            {
                return new JsonSerializerSettings
                {
                    DateFormatHandling = DateFormatHandling.Micros
                };
            }
        }
```

Don't miss Ravi Raghav's next article ×

Ravi Raghav
875 1.2k 654.6k
+ Follow

Our next method will be to create content for post request from our models

```
private HttpContent CreateHttpContent<T>(T content)
        {
            var json = JsonConvert.SerializeObject(content, MicrosoftDateFormatSettings);
            return new StringContent(json, Encoding.UTF8, "application/json");
        }
```

This is a very simple method which will serialize our model object before sending it to the request

Our next method(s) will be common methods to make post calls

```
private async Task<Message<T>> PostAsync<T>(Uri requestUrl, T content)
{

    var response = await _httpClient.PostAsync(requestUrl.ToString(), CreateHttpContent<T>
(content));
    response.EnsureSuccessStatusCode();
    var data = await response.Content.ReadAsStringAsync();
    return JsonConvert.DeserializeObject<Message<T>>(data);
}


private async Task<Message<T1>> PostAsync<T1, T2>(Uri requestUrl, T2 content)
{

    var response = await _httpClient.PostAsync(requestUrl.ToString(), CreateHttpContent<T2>
(content));
    response.EnsureSuccessStatusCode();
    var data = await response.Content.ReadAsStringAsync();
    return JsonConvert.DeserializeObject<Message<T1>>(data);
}
```

These methods will help us make post calls to the server. There are two types of methods both return Message<T> (here message is our model class), first one takes input model and returns the same type of model inside Message and the other one takes different input and returns different Output model in Message<T>. I have added message here because this is the pattern of my web APIs.

You can simply remove Message and make its return Type T just like in GetAsync method.

There is one more method in this class.

```
private Uri CreateRequestUri(string relativePath, string queryString = "")
        {
            var endpoint = new Uri(BaseEndpoint, relativePath);
            var uriBuilder = new UriBuilder(endpoint);
            uriBuilder.Query = queryString;
            return uriBuilder.Uri;
        }
```

This method will take a string url and query string and will return a Uri type which will be passed in get and post common methods.

Here is the entire class

```
public partial class ApiClient
    {

        private readonly HttpClient _httpClient;
        private Uri BaseEndpoint { get; set; }

        public ApiClient(Uri baseEndpoint)
        {
                throw new ArgumentNullException("baseEndpoint");
            }
            BaseEndpoint = baseEndpoint;
            _httpClient = new HttpClient();
        }

        /// <summary>
        /// Common method for making GET calls
        /// </summary>
        private async Task<T> GetAsync<T>(Uri requestUrl)
        {
            addHeaders();
            var response = await _httpClient.GetAsync(requestUrl, HttpCompletionOption.ResponseHeade
            response.EnsureSuccessStatusCode();
            var data = await response.Content.ReadAsStringAsync();
            return JsonConvert.DeserializeObject<T>(data);
        }

        /// <summary>
        /// Common method for making POST calls
        /// </summary>
        private async Task<Message<T>> PostAsync<T>(Uri requestUrl, T content)
        {
            addHeaders();
            var response = await _httpClient.PostAsync(requestUrl.ToString(), CreateHttpContent<T>
(content));
            response.EnsureSuccessStatusCode();
            var data = await response.Content.ReadAsStringAsync();
            return JsonConvert.DeserializeObject<Message<T>>(data);
        }
        private async Task<Message<T1>> PostAsync<T1, T2>(Uri requestUrl, T2 content)
        {
            addHeaders();
            var response = await _httpClient.PostAsync(requestUrl.ToString(), CreateHttpContent<T2>
(content));
            response.EnsureSuccessStatusCode();
            var data = await response.Content.ReadAsStringAsync();
            return JsonConvert.DeserializeObject<Message<T1>>(data);
        }

        private Uri CreateRequestUri(string relativePath, string queryString = "")
        {
            var endpoint = new Uri(BaseEndpoint, relativePath);
            var uriBuilder = new UriBuilder(endpoint);
            uriBuilder.Query = queryString;
            return uriBuilder.Uri;
        }

        private HttpContent CreateHttpContent<T>(T content)
        {
            var json = JsonConvert.SerializeObject(content, MicrosoftDateFormatSettings);
            return new StringContent(json, Encoding.UTF8, "application/json");
        }

        private static JsonSerializerSettings MicrosoftDateFormatSettings
        {
            get
            {
                return new JsonSerializerSettings
                {
                    DateFormatHandling = DateFormatHandling.Micros
                };
            }
        }

        private void addHeaders()
        {
```

Don't miss Ravi Raghav's next article ×

Ravi Raghav

875 1.2k 654.6k

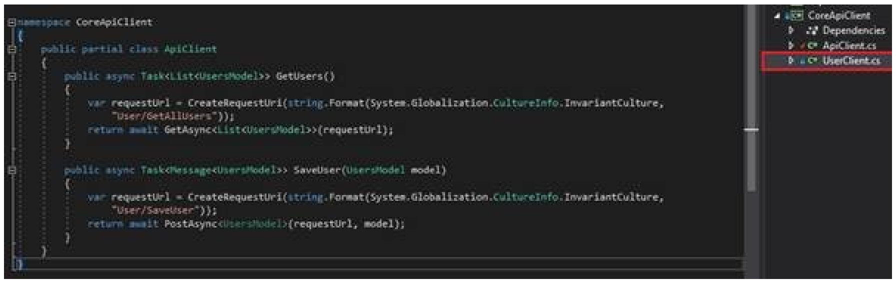
+ Follow

```
        }
    }
```

Here I have added another method addHeaders and called it in get and post common methods. This is just to demonstrate how you can send your custom headers in requests.

As you have noticed this is a partial class. You may wonder why we have create this as a partial class.

It's because now we will create a UserClient class in this project which will be partial of this so that it can access its methods. Here is the UserClient class with two methods to call list of users and save user.



```
namespace CoreApiClient
{
    public partial class ApiClient
    {
        public async Task<List<UsersModel>> GetUsers()
        {
            var requestUrl = CreateRequestUri(string.Format(System.Globalization.CultureInfo.InvariantCulture,
                "User/GetAllUsers"));
            return await GetAsync<List<UsersModel>>(requestUrl);
        }

        public async Task<Message<UsersModel>> SaveUser(UsersModel model)
        {
            var requestUrl = CreateRequestUri(string.Format(System.Globalization.CultureInfo.InvariantCulture,
                "User/SaveUser"));
            return await PostAsync<UsersModel>(requestUrl, model);
        }
    }
}
```

Here is the entire UserClient class

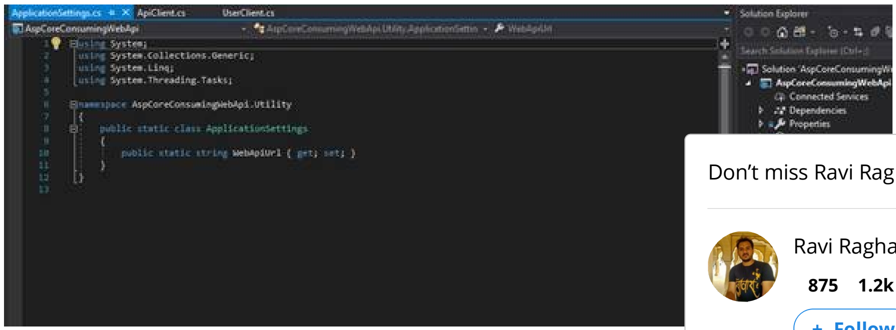
```
public partial class ApiClient
    {
        public async Task<List<UsersModel>> GetUsers()
        {
            var requestUrl = CreateRequestUri(string.Format(System.Globalization.CultureInfo.Invaria
                "User/GetAllUsers"));
            return await GetAsync<List<UsersModel>>(requestUrl);
        }

        public async Task<Message<UsersModel>> SaveUser(UsersModel model)
        {
            var requestUrl = CreateRequestUri(string.Format(System.Globalization.CultureInfo.Invaria
                "User/SaveUser"));
            return await PostAsync<UsersModel>(requestUrl, model);
        }
    }
```

Now our work in model and apiClient project is done. We just have to call this api client through our web.

For this we will use lazy loading and factory pattern (Singleton class)

Now in our web project we will add a folder called utility and in the we will add a class ApplicationSettings.cs

Now in our home controller we will add this below code.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace AspCoreConsumingWebApi.Utility
{
    public static class ApplicationSettings
    {
        public static string WebApiUrl { get; set; }
    }
}
```

This class will hold our base web API URL.

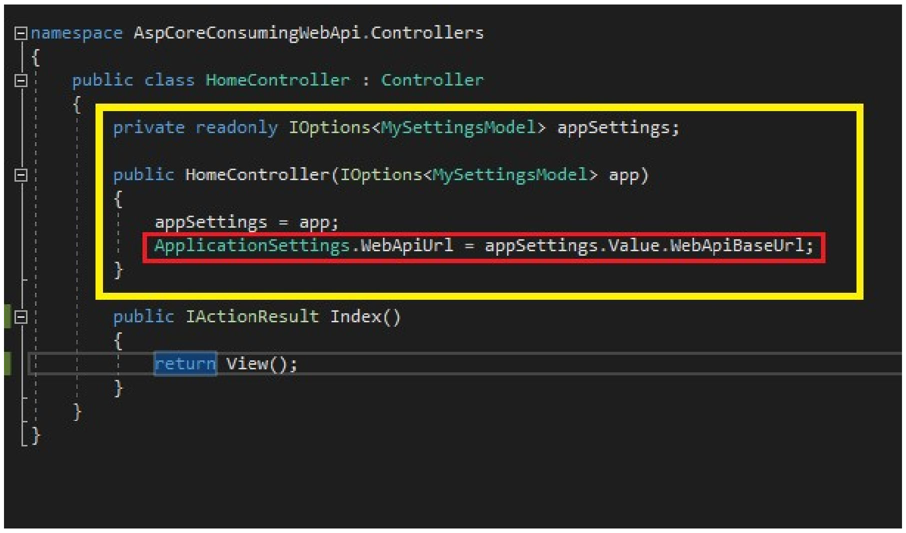
Now we will add our basi API Url in appsettings.json

```
      "WebApiBaseUrl": "http://localhost:6846/api/"
   }
```

After modifying out appsettings file, we will read it in our controller and set it into ApplicationSettings.cs

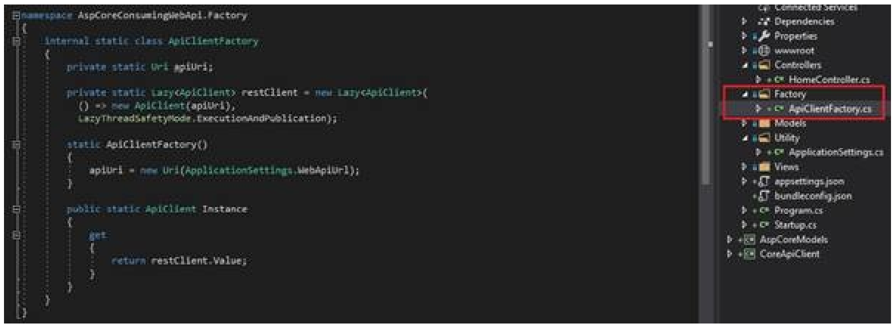
The part in yellow is how to read values from appsettings.json.Clik Here to know more.

The part in red is how to get value and set it in ApplicationSettings.



```
namespace AspCoreConsumingWebApi.Controllers
{
    public class HomeController : Controller
    {
        private readonly IOptions<MySettingsModel> appSettings;

        public HomeController(IOptions<MySettingsModel> app)
        {
            appSettings = app;
            ApplicationSettings.WebApiUrl = appSettings.Value.WebApiBaseUrl;
        }

        public IActionResult Index()
        {
            return View();
        }
    }
}
```

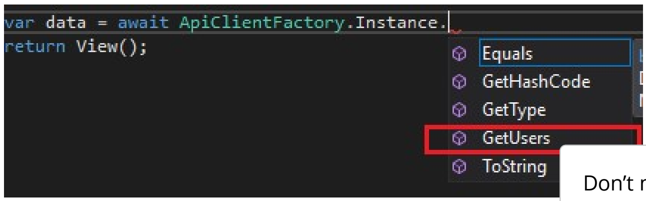
Now we will create a Factory folder and inside it we will create a factory class.



```
namespace AspCoreConsumingWebApi.Factory
{
    internal static class ApiClientFactory
    {
        private static Uri apiUri;

        private static Lazy<ApiClient> restClient = new Lazy<ApiClient>(
          () => new ApiClient(apiUri),
          LazyThreadSafetyMode.ExecutionAndPublication);

        static ApiClientFactory()
        {
            apiUri = new Uri(ApplicationSettings.WebApiUrl);
        }

        public static ApiClient Instance
        {
            get
            {
                return restClient.Value;
            }
        }
    }
}
```

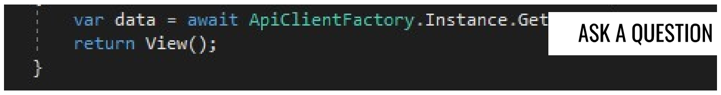
As you can see it is a singleton class and it used lazy loading to initialize our ApiClient. In the static constructor of our class we will get the value from ApplicationSettings file and set it in the apiUri field in the class.

Now how to use this class?



```
var data = await ApiClientFactory.Instance.
return View();
```

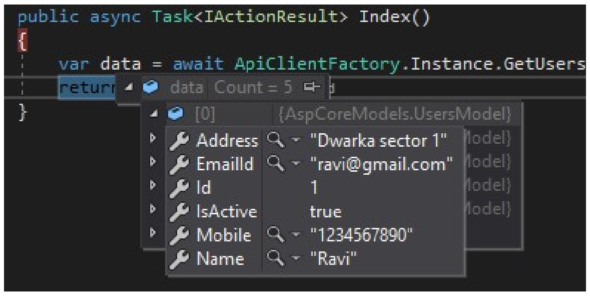
This is the method to use this class. Since our methods in api client were async Task s
method GetUsers also in the suggestion (Note :- we will also see all our methods in th



```
    var data = await ApiClientFactory.Instance.Get
    return View();
}
```

This is how we will call it.

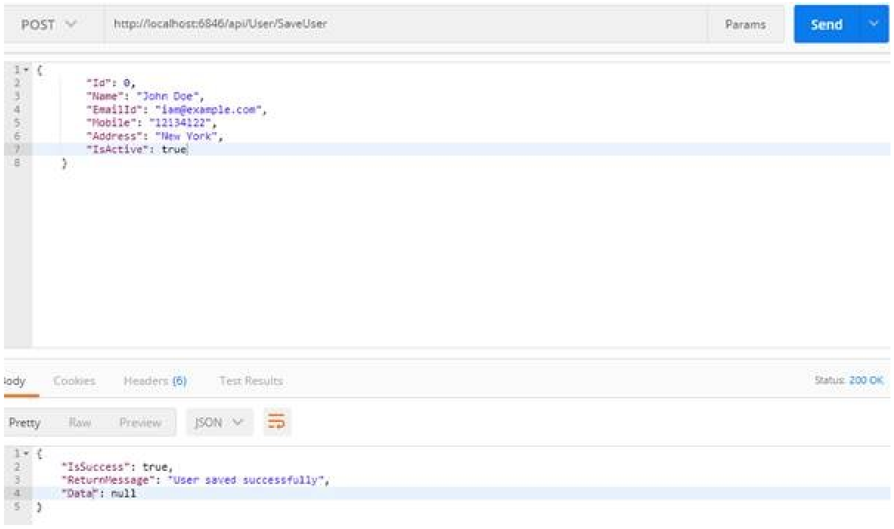
Now our API is called so let's run it and see the result.



```
public async Task<IActionResult> Index()
{
    var data = await ApiClientFactory.Instance.GetUsers
    return
         data  Count = 5
           [0]   {AspCoreModels.UsersModel}
             Address   "Dwarka sector 1"
             EmailId   "ravi@gmail.com"
             Id        1
             IsActive  true
             Mobile    "1234567890"
             Name      "Ravi"
}
```

As you can see we get list of 5 users and this is the expanded view of first model.

So we have done a get request successfully. Now we will do the post request.



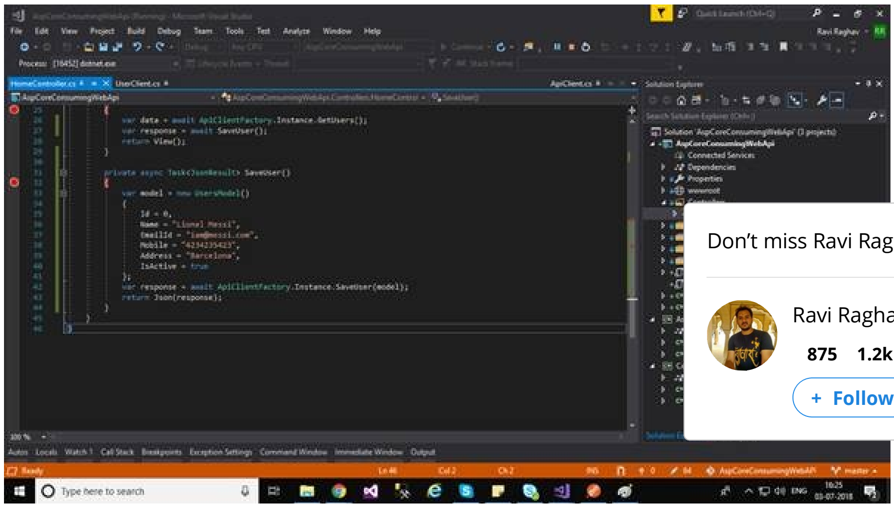
```
POST http://localhost:6846/api/User/SaveUser
```

```
{
        "Id": 0,
        "Name": "John Doe",
        "EmailId": "iam@example.com",
        "Mobile": "12134122",
        "Address": "New York",
        "IsActive": true
    }
```

Status: 200 OK

```
{
    "IsSuccess": true,
    "ReturnMessage": "User saved successfully",
    "Data": null
}
```

According to our save request here is our controller (with some hard coded values, you can get them from your view).

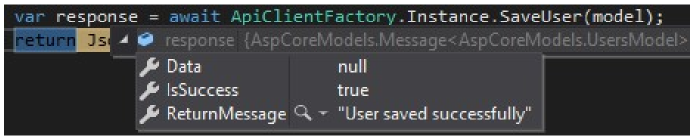


```
        {
            var data = await ApiClientFactory.Instance.GetUsers();
            var response = await SaveUser();
            return View();
        }

        private async Task<JsonResult> SaveUser()
        {
            var model = new UsersModel()
            {
                Id = 0,
                Name = "Lionel Messi",
                EmailId = "iam@messi.com",
                Mobile = "4234235423",
                Address = "Barcelona",
                IsActive = true
            };
            var response = await ApiClientFactory.Instance.SaveUser(model);
            return Json(response);
        }
    }
}
```

C# Corner

ASK A QUESTION CONTRIBUTE

Now let's run it.



```
var response = await ApiClientFactory.Instance.SaveUser(model);
return Js  response {AspCoreModels.Message<AspCoreModels.UsersModel>
             Data           null
             IsSuccess      true
             ReturnMessage  "User saved successfully"
```

Congratulations! You have successfully consumed web APIs in your ASP.NET Core 2.0 MVC application.

If you wish to see/download the code please Click Here!

Summary

In this article we have seen how to consume web APIs in ASP.NET core MVC application using factory pattern and lazy loading. We have used partial classes and HttpClient to make web requests.

Hope you all liked it.

Happy coding!

ASP.NET Core Consuming Web APIs MVC MVC Application

Ravi Raghav TOP 1000

Been developing in several technologies ASP.NET, ASP.NET CORE ,MVC SharePoint, few bits of Android

875 654.6k

4 10

Type your comment here and press Enter Key (Minimum 10 characters)

Hey ravi, i want to know why did you precisely used singleton. I mean is it better to use Dependency injection rather than using singleton. I am making a application in which multiple users can call many apis, so it is better to use singleton or DI

Parth Trehan Aug 03, 2018

1779 5 0 0 7 Reply 7

Hope you got your answer.

Ravi Raghav Aug 03, 2018

875 1.2k 654.6k 0

Yeah, I mean I am calling the apis a lot of time, so does that mean I should use Singleton?

Parth Trehan Aug 04, 2018

1779 5 0

I am creating a .net client to call lot of APIs(from the controller like in your pr architecture you used(singletoo)? Or I should just make a class and static func

Parth Trehan

1779 5 0

Don't miss Ravi Raghav's next article ×

Ravi Raghav

875 1.2k 654.6k

+ Follow

Nice Article, Thank you for sharing.........

Viknaraj Manogararajah Jul 14, 2018

107 16.6k 176.3k 1 1

C# Corner

ASK A QUESTION CONTRIBUTE

TRENDING UP

01 Transform An Existing MVC App To A Real-Time App Using SignalR

02 Implement Gmail And Facebook Based Authentication In ASP.NET Core 2.2

03 ASP.NET Core - MVC Request Life Cycle

04 Implement Microsoft And Twitter Based Authentication In ASP.NET Core 2.2

05 The Future of .NET

06 Exception Handling In MVC With Filters And Application Insights

07 .NET 5 Is The Future Of .NET - What Every .NET Developer Must Know

Don't miss Ravi Raghav's next article ×

Ravi Raghav

875 1.2k 654.6k

+ Follow

C# Corner

09 When To Use Static Classes In C#

C# Corner

ASK A QUESTION CONTRIBUTE

10 What Is the Most Popular Operating System?

View All

LEARN MORE

About Us Contact Us Privacy Policy Terms Media Kit Sitemap Report a Bu

Don't miss Ravi Raghav's next article ×

Ravi Raghav

875 1.2k 654.6k

+ Follow