Consume Web API By MVC In .NET Core (2), Client

George Updated date Dec 27, 2020 16.3k 3 7

Download Free .NET & JAVA Files API

This article will give you a way to consume Web API by a ASP.NET MVC Client in .NET Core with one line of code.

Introduction

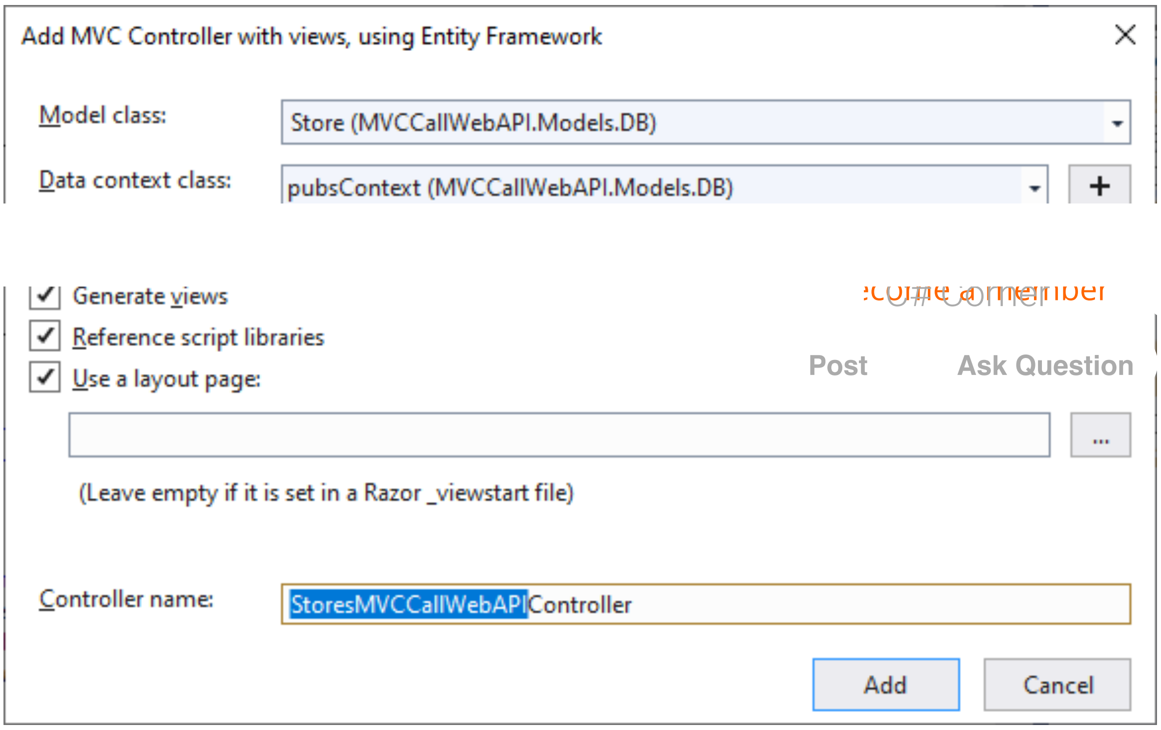
In the previous article (Part I of this article), we created a ASP.NET Core MVC app and associate a Web API service in it:

- MVC is a client/server app, with a web page as a client and SQL server as server, linked by Entity Framework;
- Web API is a Server side service, with a RESTful output for consumer, that is linked to database by entity framework.

For our test purposes, MVC and Web API are against two different database, MVC is against the database pubs, while Web API is against database DB_Demo_API.

In this article, we will make the MVC app as a client to consume Web API. For the purposes of convenient analysis and comparison, we will make another MVC module (controller/view) that's exactly the same as the previous MVC module but with a different name, *StoresMVCCallWebAPI* controller (see details from Part I, linked above, Part A, Step 1, 3, Add controller).

The added Controller is like this:


Add MVC Controller with views, using Entity Framework

Model class: Store (MVCCallWebAPI.Models.DB)

Data context class: pubsContext (MVCCallWebAPI.Models.DB)

✓ Generate views

✓ Reference script libraries

✓ Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

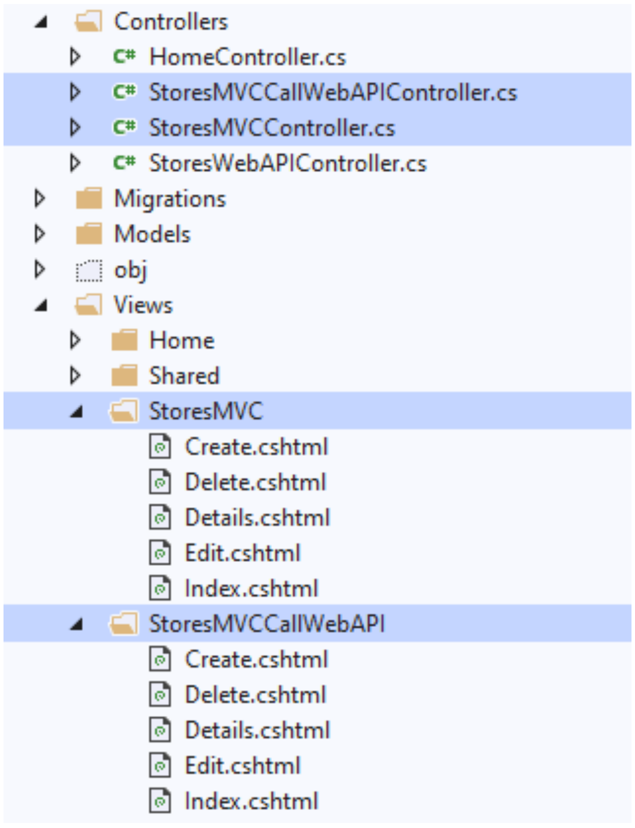
Controller name: StoresMVCCallWebAPIController

Add Cancel

Post Ask Question

Visual Studio Create

- A StroesMVCCallWebAPI controller (*Controllers/StoresMVCCallWebAPIController.cs*)
- Razor view files for Create, Delete, Details, Edit, and Index pages (*Views/StoresMVCCallWebAPI/*.cshtml*)


```
Controllers
    HomeController.cs
    StoresMVCCallWebAPIController.cs
    StoresMVCController.cs
    StoresWebAPIController.cs
Migrations
Models
obj
Views
    Home
    Shared
    StoresMVC
        Create.cshtml
        Delete.cshtml
        Details.cshtml
        Edit.cshtml
        Index.cshtml
    StoresMVCCallWebAPI
        Create.cshtml
        Delete.cshtml
        Details.cshtml
        Edit.cshtml
        Index.cshtml
```

The behavior of the new controller, *StroesMVCCallWebAPI*, is exactly the same as the old MVC controller, *StroesMVC.*

Run and Test the app

Modify the header of the file: *Views/Shared/_layout.cshtml* Views, shown below:

- At the app level, we modify as StoresMVCCallWebAPI controller, named as MVC Call Web API
- Move StoresMVC controller to the second level, with name as MVC app

```
<header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
        <div class="container">
            <a class="navbar-brand" asp-area="" asp-controller="StoresMVCCallWebAPI" asp-action="Index"><b>MVC Call Web API</b></a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
                    aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                <ul class="navbar-nav flex-grow-1">
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-controller="StoresMVC" asp-action="Index">MVC app</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-controller="Swagger" asp-action="Index">Web API</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
</header>
```

Then, we run the app,

Consume Web API By MVC Client In .NET Core Client

We can see the two controller endpoints: https://localhost:44350/StoresMVCCallWebAPI (above) and https://localhost:44350/StoresMVC (below) are exactly the same, because the are against the same database --- pubs,

Consume Web API By MVC Client In .NET Core Client

While the Web API (Swagger),

Consume Web API By MVC Client In .NET Core Client

has a different data set, that is due the fact that it is against the different database: DB_Demo_API,

Consume Web API By MVC Client In .NET Core Client

At the end of the article, the controller StoresMVCCallWebAPI will consume the Web API, then these two will share the same database, and get the same result.

How to Consume RESTful APIs

We can see the most comprehensive list of ways to consume RESTful APIs in your C# projects from this article 《A Few Great Ways to Consume RESTful API in C#, we borrowed here,

"There are several ways to consume a RESTful API in C#,

- HttpWebRequest/Response Class
- WebClient Class
- HttpClient Class
- RestSharp NuGet Package
- ServiceStack Http Utils
- Flurl
- DalSoft.RestClient

Every one of these has pros and cons."

In this article, we will choose to use HttpClient from Microsoft for our project. In practice or production, you may choose different ones.

One Line Code Implementation

The database context is used in each of the CRUD methods in the both MVC Controller and Web API ApiController. They have the same methods, same signatures, and implementations. For each action, we will use one line code to redirect the direct database pubs, to the Web API that is against database DB_Demo_API.

POST

We start from Create, because this is simplest. Get rid of all other actions, we have the controller class with Create method:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using MVCCallWebAPI.Models.DB;

namespace MVCCallWebAPI.Controllers
{
    public class StoresMVCCallWebAPIController : Controller
    {
        private readonly pubsContext _context;

        public StoresMVCCallWebAPIController(pubsContext context)
        {
            _context = context;
        }

        // POST: StoresMVCCallWebAPI/Create
        // To protect from overposting attacks, enable the specific p
        // For more details, see http://go.microsoft.com/fwlink/?
LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("Stor_Id,Stor_N
        {
            if (ModelState.IsValid)
            {
                _context.Add(store);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(store);
        }
    }
}
```

The Create method with an input Object Store, the following two-line code saves the object into the database (pubs) through entity framework.

```
_context.Add(store);
await _context.SaveChangesAsync();
```

Replace this two line code with class HttpClient, and the method PostAsJsonAsync to call Web API,

```
HttpClient client = new HttpClient();
string url = "https://localhost:44350/api/storesAPI/";
await client.PostAsJsonAsync<Store>(url, store);
```

We got the Create method like this,

```
public async Task<IActionResult> Create([Bind("Stor_Id,Stor_Name,Stor
{
    if (ModelState.IsValid)
    {
        //_context.Add(store);
        //await _context.SaveChangesAsync();

        // Consume API
        HttpClient client = new HttpClient();
        string url = "https://localhost:44350/api/storesWebAPI/";

        await client.PostAsJsonAsync<Store>(url, store);

        return RedirectToAction(nameof(Index));
    }
    return View(store);
}
```

We can move the two line shared code (Create HttpClient class and define url address) into class level, then we only use one line code to complete the job to consume Web API for the Create method.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Json;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Newtonsoft.Json;
using WebMVCCore5.Models.DB;

namespace WebMVCCore5.Controllers
{
    public class StoresMVCCallAPIController : Controller
    {
        private readonly pubsContext _context;

       HttpClient client = new HttpClient();
       string url = "https://localhost:44330/api/storesWebAPI/";

        public StoresMVCCallAPIController(pubsContext context)
        {
            _context = context;
        }

        // POST: StoresMVCCallAPI/Create
        // To protect from overposting attacks, enable the specific p
        // For more details, see http://go.microsoft.com/fwlink/?
LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("Stor_Id,Stor_N
        {
            if (ModelState.IsValid)
            {
                //_context.Add(store);
                //await _context.SaveChangesAsync();

                // Consume API
                await client.PostAsJsonAsync<Store>(url, store);

                return RedirectToAction(nameof(Index));
            }
            return View(store);
        }
```

DELETE

We use DeleteAsync method to do the job,

```
// POST: StoresMVCCallAPI/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(string id)
{
    // Original Code:
    //var store = await _context.Stores.FindAsync(id);
    //_context.Stores.Remove(store);
    //await _context.SaveChangesAsync();

    // Consume API
    await client.DeleteAsync(url + id);

    return RedirectToAction(nameof(Index));
}
```

PUT (Edit)

We need to bring two parameters, one is id, another is the object, and we use PutAsJsonAsync method

```
try
{
    // Original code:
    //_context.Update(store);
    //await _context.SaveChangesAsync();

    // Consume API
    await client.PutAsJsonAsync<Store>(url + id, store);
}
```

GET/id

There are three places to use the GET method, but actually, they are the same. We use

GetStringAsync. Here, due to getting data, we need to Deserialize the JSON into class, we use JsonConvert.DeserializeObject in Newtonsoft.Json namespace.

```
// GET: StoresMVCCallAPI/Delete/5
public async Task<IActionResult> Delete(string id)
{
    if (id == null)
    {
        return NotFound();
    }

    // Original code:
    //var store = await _context.Stores
    //    .FirstOrDefaultAsync(m => m.Stor_Id == id);

    // Consume API
    var store = JsonConvert.DeserializeObject<Store>
(await client.GetStringAsync(url + id));

    if (store == null)
    {
        return NotFound();
    }

    return View(store);
}
```

GET

The same as Get/ID, but we use List<Store>

```
// GET: StoresMVCCallAPI
public async Task<IActionResult> Index()
{
    // Original code:
    //return View(await _context.Stores.ToListAsync());

    // Consume API
    return View(JsonConvert.DeserializeObject<List<Store>>
(await client.GetStringAsync(url)).ToList());
}
```

Finally, we got the full code where the changed one line of code for each method noted as // Consume API, and the // Original code is comment out:

```
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Json;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using MVCCallWebAPI.Models.DB;
using Newtonsoft.Json;

namespace MVCCallWebAPI.Controllers
{
    public class StoresMVCCallWebAPIController : Controller
    {
        private readonly pubsContext _context;

       HttpClient client = new HttpClient();
       string url = "https://localhost:44350/api/storesWebAPI/";

        public StoresMVCCallWebAPIController(pubsContext context)
        {
            _context = context;
        }


        // GET: StoresMVCCallAPI
        public async Task<IActionResult> Index()
        {
            // Original code:
            //return View(await _context.Stores.ToListAsync());

            // Consume API
            return View(JsonConvert.DeserializeObject<List<Store>>
(await client.GetStringAsync(url)).ToList());
        }

        // GET: StoresMVCCallWebAPI/Details/5
        public async Task<IActionResult> Details(string id)
        {
            if (id == null)
            {
                return NotFound();
            }

            // Original code:
            //var store = await _context.Stores
            //    .FirstOrDefaultAsync(m => m.Stor_Id == id);

            // Consume API
            var store = JsonConvert.DeserializeObject<Store>
(await client.GetStringAsync(url + id));

            if (store == null)
            {
                return NotFound();
            }

            return View(store);
        }

        // GET: StoresMVCCallWebAPI/Create
        public IActionResult Create()
        {
            return View();
        }

        // POST: StoresMVCCallWebAPI/Create
        // To protect from overposting attacks, enable the specific p
        // For more details, see http://go.microsoft.com/fwlink/?
LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("Stor_Id,Stor_N
        {
            if (ModelState.IsValid)
            {
                // Original code:
                //_context.Add(store);
                //await _context.SaveChangesAsync();

                // Consume API
                await client.PostAsJsonAsync<Store>(url, store);

                return RedirectToAction(nameof(Index));
            }
            return View(store);
        }

        // GET: StoresMVCCallWebAPI/Edit/5
        public async Task<IActionResult> Edit(string id)
        {
            if (id == null)
            {
                return NotFound();
            }

            // Original code:
            //var store = await _context.Stores.FindAsync(id);

            // Consume API
            var store = JsonConvert.DeserializeObject<Store>
(await client.GetStringAsync(url + id));

            if (store == null)
            {
                return NotFound();
            }
            return View(store);
        }

        // POST: StoresMVCCallWebAPI/Edit/5
        // To protect from overposting attacks, enable the specific p
        // For more details, see http://go.microsoft.com/fwlink/?
LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(string id, [Bind("Stor_
        {
            if (id != store.Stor_Id)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    // Original code:
                    //_context.Update(store);
                    //await _context.SaveChangesAsync();

                    // Consume API
                    await client.PutAsJsonAsync<Store>
(url + id, store);
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!StoreExists(store.Stor_Id))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            return View(store);
        }

        // GET: StoresMVCCallWebAPI/Delete/5
        public async Task<IActionResult> Delete(string id)
        {
            if (id == null)
            {
                return NotFound();
            }

            // Original code:
            //var store = await _context.Stores
            //    .FirstOrDefaultAsync(m => m.Stor_Id == id);

            // Consume API
            var store = JsonConvert.DeserializeObject<Store>
(await client.GetStringAsync(url + id));

            if (store == null)
            {
                return NotFound();
            }

            return View(store);
        }

        // POST: StoresMVCCallWebAPI/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(string id)
        {
            // Original Code:
            //var store = await _context.Stores.FindAsync(id);
            //_context.Stores.Remove(store);
            //await _context.SaveChangesAsync();

            // Consume API
            await client.DeleteAsync(url + id);

            return RedirectToAction(nameof(Index));
        }

        private bool StoreExists(string id)
        {
            return _context.Stores.Any(e => e.Stor_Id == id);
        }
    }
}
```

Run and Test the app

The MVC module,

Consume Web API By MVC Client In .NET Core Client

The MVC module calls Web API: the data is different from the MVC module above, but the same as the Web API module.

Consume Web API By MVC Client In .NET Core Client

The Web API module,

Consume Web API By MVC Client In .NET Core Client

Conclusion

In this article, we used MVC, and Web API templates to build out three apps, one is MVC module, one is Web API, then we used HttpClient in MVC module to consume Web API with only one line of code written manually.

.NET Core ASP.NET MVC Code First Database First Entity Framework HTTPClient Web API

Next Recommended Reading

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

OUR BOOKS

George TOP 500

I have been an IT professional for more than 20 years since I got my first Microsoft certificate MCP in 1999. I used to be a scientist in atmospheric science field in 90s, handling big data from satellites and everywhere... Read more

214 1m 2

7 3

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

Great Explanation for Consuming Web api in Mvc core. However I have a question the method PostAsJsonAsync won't work for .NET Core 3 + version. Is there any alternative for it?

Muhammad Hanif Dec 24, 2020

1981 125 0 0 2 Reply

Hi, Muhammad, thank you for your reading and comment. We choose class HttpClient to do the job, however, it does not support the method bring a object as parameter, where PostAsJsonAsync and PutAsJsonAsync actually belong to HttpClient extension class, say, HttpClientJsonExtensions. You need to add this name space: using System.Net.Http.Json;

George Dec 24, 2020

214 11.1k 1m 0

Sorry, my article is for advanced developer, some details might missing. After you add the line: await client.PostAsJsonAsync<Store>(url, store); the compiling will not pass. You should right click the line and add the missing name space, otherwise, you need manually add it by yourself: using System.Net.Http.Json; --- but in the final code, everything is included. Hope this helps.

George Dec 24, 2020

214 11.1k 1m 0

FEATURED ARTICLES

Azure Duration Functions - How To Use And Implement It

Easily Use Flurl Testable HttpClient

Legacy Classes And Legacy Interface Of Collections API

It's Not About How You Inject Your Services, It's About How You Test Them

SPFx Form Customizer Extension To Customize SharePoint New/Edit/Display Form Of List/Libraries

TRENDING UP

01 Azure Durable Functions - An Overview

02 How To Upload Files Into Azure Blog Storage Using Azure Functions In C#

03 Change Data Capture - Another Way To Implement The Incremental Load

04 Azure Duration Functions - How To Use And Implement It

05 Rockin' The Code World with dotNetDave ft. Khalid Abuhakmeh Ep. 52

06 Creating Search Feature In Blazor Server Grid

07 Rockin' The Code World with dotNetDave ft. Steve Jones Ep. 53

08 Growth Mindset Show Ep. 11 - 2022

09 How To Handle Nullable Reference In .NET 6

10 Distributed Transaction in C# Microservices using SAGA Pattern

Learn Machine Learning With Python

CHALLENGE YOURSELF

Azure Developer Skill Challenge

GET CERTIFIED

Python Developer