.NET Core: Interaction of Microservices via Web API

Shkurko Eugene

17 Jun 2020 CPOL

This article looks at the only thing that is missing from Christian Horsdal's, “Microservices in .NET Core: with examples in Nancy” - a tool for automating the interaction between microservices.

Here, we look at the use of the features of Nuget Shed.CoreKit.WebApi package and the auxiliary package Shed.CoreKit.WebApi.Abstractions in the development of the MicroCommerce application described in the aforementioned book of Christian Horsdal.

Download source code - 26.3 KB

Introduction

Almost everyone who has worked with microservices in the .NET Core probably knows the book of Christian Horsdal, “Microservices in .NET Core: with examples in Nancy” The ways of building an application based on microservices are well described here, monitoring, logging, and access control are discussed in detail. The only thing that is missing is a tool for automating the interaction between microservices.

In the usual approach, when developing a microservice, a web client for it is being developed in parallel. And every time the web interface of the microservice changes, additional efforts have to be expended for the corresponding changes in the web client. The idea of generating a pair of web-api / web-client using OpenNET is also quite laborious, I would like something more transparent for the developer.

So, with an alternative approach to the development of our application, I would like:

- The microservice structure is described by the .NET interface using attributes that describe the type of method, route and way of passing parameters, as is done in MVC.
- Microservice functionality is developed exclusively in the .NET class, implementing this interface. The publication of microservice endpoints should be automatic, not requiring complex settings.
- The web client for the microservice should be generated automatically based on the interface and provided through IoC container.
- There should be a mechanism for organizing redirects to the endpoints of microservices from the main application interacting with the user interface.

In accordance with these criteria, the **`Nuget Shed.CoreKit.WebApi`** package was developed. In addition to this, the auxiliary package **`Shed.CoreKit.WebApi.Abstractions`** was created, containing attributes and classes that can be used to develop common assembly projects where the functionality of the main package is not required.

Below, we look at the use of the features of these packages in the development of the MicroCommerce application described in the aforementioned book of Christian Horsdal.

Naming Agreement

Hereinafter, we use the following terminology:

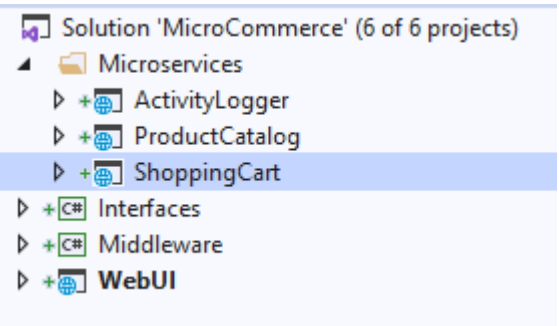
- **The microservice** is an ASP.NET Core application (project) that can be run console-based, by Internet Information Services (IIS) or in a Docker container.

- **The interface** is a .NET entity, a set of methods and properties without implementation.
- **The endpoint** is the path to the root of the microservice application or interface implementation. Examples: *http://localhost:5001, http://localhost:5000/products*
- **The route** is the path to the interface method from the endpoint. It can be defined by default in the same way as in MVC or set using the attribute.

MicroCommerce Application Structure

- `ProductCatalog` is a microservice that provides product information.
- `ShoppingCart` is a microservice that provides information about the user's purchases, as well as the ability to add/remove purchases. When the state of the user's basket changes, events are generated to notify other microservices.
- `ActivityLogger` is a microservice that collects information about events of other microservices. Provides an endpoint for receiving logs.
- `WebUI` is a user interface of the application that should be implemented as a Single Page Application.
- `Interfaces` - microservice interfaces and model classes
- `Middleware` - common functionality for all microservices

MicroCommerce Application Development



Create an empty .NET Core solution. Add the `WebUI` project as an empty ASP.NET Core WebApplication. Next, add `ProductCatalog`, `ShoppingCart`, `ActivityLog` microservice projects, as well as empty ASP.NET Core `WebApplication` projects. Finally, we add two class libraries - Interfaces and Middleware.

1. Interfaces Project. Microservice Interfaces and Model Classes

Install the `Shed.CoreKit.WebApi.Abstractions` nuget package.

Create the `IProductCatalog` interface and models for it:

```
//
// Interfaces/IProductCatalog.cs
//

using MicroCommerce.Models;
using Shed.CoreKit.WebApi;
using System;
using System.Collections.Generic;

namespace MicroCommerce
{
    public interface IProductCatalog
    {
        IEnumerable<Product> Get();

        [Route("get/{productId}")]
        public Product Get(Guid productId);
    }
}
```

```
//
// Interfaces/Models/Product.cs
//

using System;

namespace MicroCommerce.Models
```

```
{
    public class Product
    {
        public Guid Id { get; set; }

        public string Name { get; set; }

        public Product Clone()
        {
            return new Product
            {
                Id = Id,
                Name = Name
            };
        }
    }
}
```

Using the `Route` attribute is no different from that in ASP.NET Core MVC, but remember that this attribute must be from namespace `Shed.CoreKit.WebApi`, and no other. The same applies to the `HttpGet`, `HttpPut`, `HttpPost`, `HttpPatch`, `HttpDelete`, and `FromBody` attributes if used.

The rules for using attributes of the **Http [Methodname]** type are the same as in MVC, that is, if the prefix of the interface method name matches the name of the required Http method, then you do not need to additionally define it, otherwise we will use the corresponding attribute.

The `FromBody` attribute is applied to a method parameter if this parameter is to be retrieved from the request body. I note that like ASP.NET Core MVC, it *must always be specified, there are no default rules*. And in the method parameters, there can be only one parameter with this attribute.

Create the `IShoppingCart` interface and models for it:

```
//
// Interfaces/IShoppingCart.cs
//

using MicroCommerce.Models;
using Shed.CoreKit.WebApi;
using System;
using System.Collections.Generic;

namespace MicroCommerce
{
    public interface IShoppingCart
    {
        Cart Get();

        [HttpPut, Route("addorder/{productId}/{qty}")]
        Cart AddOrder(Guid productId, int qty);

        Cart DeleteOrder(Guid orderId);

        [Route("getevents/{timestamp}")]
        IEnumerable<CartEvent> GetCartEvents(long timestamp);
    }
}
```

```
//
// Interfaces/IProductCatalog/Order.cs
//

using System;

namespace MicroCommerce.Models
{
    public class Order
    {
```

```
        public Guid Id { get; set; }

        public Product Product { get; set; }

        public int Quantity { get; set; }

        public Order Clone()
        {
            return new Order
            {
                Id = Id,
                Product = Product.Clone(),
                Quantity = Quantity

            };
        }
    }
}
```

```
//
// Interfaces/Models/Cart.cs
//

using System;

namespace MicroCommerce.Models
{
    public class Cart
    {
        public IEnumerable<Order> Orders { get; set; }
    }
}
```

```
//
// Interfaces/Models/CartEvent.cs
//

using System;

namespace MicroCommerce.Models
{
    public class CartEvent: EventBase
    {
        public CartEventTypeEnum Type { get; set; }
        public Order Order { get; set; }
    }
}
```

```
//
// Interfaces/Models/CartEventTypeEnum.cs
//

using System;

namespace MicroCommerce.Models
{
    public enum CartEventTypeEnum
    {
        OrderAdded,
        OrderChanged,
        OrderRemoved
    }
}
```

```
//
// Interfaces/Models/EventBase.cs
```

```
//

using System;

namespace MicroCommerce.Models
{
    public abstract class EventBase
    {
        private static long TimestampBase;

        static EventBase()
        {
            TimestampBase = new DateTime(2000, 1, 1).Ticks;
        }

        public long Timestamp { get; set; }

        public DateTime Time { get; set; }

        public EventBase()
        {
            Time = DateTime.Now;
            Timestamp = Time.Ticks - TimestampBase;
        }
    }
}
```

A few words about the basic type of events `EventBase`. When publishing events, we use the approach described in the book, i.e., any event contains a `Timestamp`, when polling the source of the event, the listener sends the last received timestamp. Unfortunately, the long type is incorrectly converted to the Number JavaScript type for large values, so we use some trick, we subtract the timestamp of the base date (`Timestamp = Time.Ticks - TimestampBase`). The specific value of the base date is absolutely unimportant.

Create the `IActivityLogger` interface and models for it:

```
//
// Interfaces/IActivityLogger.cs
//

using MicroCommerce.Models;
using System.Collections.Generic;

namespace MicroCommerce
{
    public interface IActivityLogger
    {
        IEnumerable<LogEvent> Get(long timestamp);
    }
}
```

```
//
// Interfaces/Models/LogEvent.cs
//

namespace MicroCommerce.Models
{
    public class LogEvent: EventBase
    {
        public string Description { get; set; }
    }
}
```

2. ProductCatalog Project

Open *Properties/launchSettings.json*, bind the project to port 5001.

```
//
// Properties/launchSettings.json
//

{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:60670",
      "sslPort": 0
    }
  },
  "profiles": {
    "MicroCommerce.ProductCatalog": {
      "commandName": "Project",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      },
      "applicationUrl": "http://localhost:5001"
    }
  }
}
```

Install `Shed.CoreKit.WebApi` nuget package to the project and add links to `Interfaces` and `Middleware` projects. `Middleware` project will be described in more detail below.

Create the `IProductCatalog` interface implementation:

```
//
// ProductCatalog/ProductCatalog.cs
//

using MicroCommerce.Models;
using System;
using System.Collections.Generic;
using System.Linq;

namespace MicroCommerce.ProductCatalog
{
    public class ProductCatalogImpl : IProductCatalog
    {
        private Product[] _products = new[]
        {
            new Product{ Id = new Guid("6BF3A1CE-1239-4528-8924-A56FF6527595"),
                         Name = "T-shirt" },
            new Product{ Id = new Guid("6BF3A1CE-1239-4528-8924-A56FF6527596"),
                         Name = "Hoodie" },
            new Product{ Id = new Guid("6BF3A1CE-1239-4528-8924-A56FF6527597"),
                         Name = "Trousers" }
        };

        public IEnumerable<Product> Get()
        {
            return _products;
        }

        public Product Get(Guid productId)
        {
            return _products.FirstOrDefault(p => p.Id == productId);
        }
    }
}
```

The `product` catalog is stored in a static field, to simplify the example. Of course, in a real application, you need to use some other storage, which can be provided as a dependency through Dependency Injection.

Now this implementation needs to be connected as an endpoint. If we used the traditional approach, we would have to use the MVC infrastructure, that is, create a controller, pass our implementation to it as a dependency, configure routing, etc. Using the `Shed.CoreKit.WebApi` Nuget package makes this much easier. It is enough to register our implementation in Dependency Injection, then declare it as the endpoint using the `UseWebApiEndpoint` extender method from the `Shed.CoreKit.WebApi` package. We do it in the Setup:

```
//
// ProductCatalog/Setup.cs
//

using MicroCommerce.Middleware;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Shed.CoreKit.WebApi;

namespace MicroCommerce.ProductCatalog
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddCorrelationToken();
            services.AddCors();
            // register the implementation as dependency
            services.AddTransient<IProductCatalog, ProductCatalogImpl>();
            services.AddLogging(builder => builder.AddConsole());
            services.AddRequestLogging();
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseCorrelationToken();
            app.UseRequestLogging();
            app.UseCors(builder =>
            {
                builder
                    .AllowAnyOrigin()
                    .AllowAnyMethod()
                    .AllowAnyHeader();
            });

            // bind the registered implementation to the endpoint
            app.UseWebApiEndpoint<IProductCatalog>();
        }
    }
}
```

This leads to the fact that methods appear in the microservice:
http://localhost: 5001/get
http://localhost: 5001/get/<productid>

The `UseWebApiEndpoint` method may accept an optional root parameter.
If we connect the endpoint this way: `app.UseWebApiEndpoint<IProductCatalog>("products")`, then the microservice endpoint will look like this:
http://localhost:5001/products/get
This can be useful if we need to connect several interfaces to the microservice.

This is all you need to do. You can start the microservice and test its methods.

The rest of the code in `Setup` configures and enables additional features.
A pair of `services.AddCors()` / `app.UseCors(...)` allows the use of cross-domain requests in the project. This is necessary when redirecting requests from the UI.
A pair of `services.AddCorrelationToken()` / `app.UseCorrelationToken()` enables the use of correlation tokens when logging requests, as described in the book of Christian Horsdal. We will discuss this later.
Finally, a pair of `services.AddRequestLogging()` / `app.UseRequestLogging()` enables request logging from a Middleware project. We will return to this later too.

3. ShoppingCart Project

Bind the project to port 5002 in the same way as `ProductCatalog`.

Add to the project the `Shed.CoreKit.WebApi` nuget package and links to `Interfaces` and `Middleware` projects.

Create the `IShoppingCart` interface implementation.

```
//
// ShoppingCart/ShoppingCart.cs
//

using MicroCommerce.Models;
using System;
using System.Collections.Generic;
using System.Linq;

namespace MicroCommerce.ShoppingCart
{
    public class ShoppingCartImpl : IShoppingCart
    {
        private static List<Order> _orders = new List<Order>();
        private static List<CartEvent> _events = new List<CartEvent>();
        private IProductCatalog _catalog;

        public ShoppingCartImpl(IProductCatalog catalog)
        {
            _catalog = catalog;
        }

        public Cart AddOrder(Guid productId, int qty)
        {
            var order = _orders.FirstOrDefault(i => i.Product.Id == productId);
            if(order != null)
            {
                order.Quantity += qty;
                CreateEvent(CartEventTypeEnum.OrderChanged, order);
            }
            else
            {
                var product = _catalog.Get(productId);
                if (product != null)
                {
                    order = new Order
                    {
                        Id = Guid.NewGuid(),
                        Product = product,
                        Quantity = qty
                    };

                    _orders.Add(order);
                    CreateEvent(CartEventTypeEnum.OrderAdded, order);
                }
            }

            return Get();
        }
```

```
        public Cart DeleteOrder(Guid orderId)
        {
            var order = _orders.FirstOrDefault(i => i.Id == orderId);
            if(order != null)
            {
                _orders.Remove(order);
                CreateEvent(CartEventTypeEnum.OrderRemoved, order);
            }

            return Get();
        }

        public Cart Get()
        {
            return new Cart
            {
                Orders = _orders
            };
        }

        public IEnumerable<CartEvent> GetCartEvents(long timestamp)
        {
            return _events.Where(e => e.Timestamp > timestamp);
        }

        private void CreateEvent(CartEventTypeEnum type, Order order)
        {
            _events.Add(new CartEvent
            {
                Timestamp = DateTime.Now.Ticks,
                Time = DateTime.Now,
                Order = order.Clone(),
                Type = type
            });
        }
    }
}
```

Here, as in `ProductCatalog`, we use `static` fields as storage. But this microservice still uses calls to `ProductCatalog` to get information about the product, so we pass the link to `IProductCatalog` to the constructor as a dependency.

Now this dependency needs to be defined in DI, and for this, we use the `AddWebApiEndpoints` extender method from the `Shed.CoreKit.WebApi` package. This method registers the `WebApi` client generator as factory method for the `IProductCatalog` interface in the IoC container.
When generating a `WebApi` client, the factory uses the dependency `System.Net.Http.HttpClient`. If the application requires some special settings for `HttpClient` (credentials, special headers / tokens, etc.), this should be done when registering `HttpClient` in the IoC container.

```
//
// ShoppingCart/Settings.cs
//

using MicroCommerce.Middleware;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Shed.CoreKit.WebApi;
using System.Net.Http;

namespace MicroCommerce.ShoppingCart
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
```

```
            services.AddCorrelationToken();
            services.AddCors();
            services.AddTransient<IShoppingCart, ShoppingCartImpl>();
            services.AddTransient<HttpClient>();
            // register a dependency binded to the endpoint
            services.AddWebApiEndpoints
            (new WebApiEndpoint<IProductCatalog>(new System.Uri("http://localhost:5001")));
            services.AddLogging(builder => builder.AddConsole());
            services.AddRequestLogging();
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseCorrelationToken();
            app.UseRequestLogging("getevents");
            app.UseCors(builder =>
            {
                builder
                    .AllowAnyOrigin()
                    .AllowAnyMethod()
                    .AllowAnyHeader();
            });

            app.UseWebApiEndpoint<IShoppingCart>();
        }
    }
}
```

The `AddWebApiEndpoints` method can take an arbitrary number of parameters, so it is possible to configure all dependencies with a single call to this method.
Otherwise, all settings are similar to `ProductCatalog`.

4. ActivityLogger Project

Bind the project to port 5003 in the same way as `ProductCatalog`.

Install the `Shed.CoreKit.WebApi` nuget package to the project and add links to `Interfaces` and `Middleware` projects.

Create the `IActivityLogger` interface implementation.

```
//
// ActivityLogger/ActivityLogger.cs
//

using MicroCommerce;
using MicroCommerce.Models;
using System.Collections.Generic;
using System.Linq;

namespace MicroCommerce.ActivityLogger
{
    public class ActivityLoggerImpl : IActivityLogger
    {
        private IShoppingCart _shoppingCart;

        private static long timestamp;
        private static List<LogEvent> _log = new List<LogEvent>();

        public ActivityLoggerImpl(IShoppingCart shoppingCart)
        {
            _shoppingCart = shoppingCart;
```

```
        }

        public IEnumerable<LogEvent> Get(long timestamp)
        {
            return _log.Where(i => i.Timestamp > timestamp);
        }

        public void ReceiveEvents()
        {
            var cartEvents = _shoppingCart.GetCartEvents(timestamp);

            if(cartEvents.Count() > 0)
            {
                timestamp = cartEvents.Max(c => c.Timestamp);
                _log.AddRange(cartEvents.Select(e => new LogEvent
                {
                    Description = $"{GetEventDesc(e.Type)}: '{e.Order.Product.Name}
                                  ({e.Order.Quantity})'"
                }));
            }
        }

        private string GetEventDesc(CartEventTypeEnum type)
        {
            switch (type)
            {
                case CartEventTypeEnum.OrderAdded: return "order added";
                case CartEventTypeEnum.OrderChanged: return "order changed";
                case CartEventTypeEnum.OrderRemoved: return "order removed";
                default: return "unknown operation";
            }
        }
    }
}
```

It also uses a dependency on another microservice (`IShoppingCart`). But one of the tasks of this service is to listen to the events of other services, so we add an additional `ReceiveEvents()` method, which we will call from the scheduler. We will add it to the project additionally.

```
//
// ActivityLogger/Scheduler.cs
//

using Microsoft.Extensions.Hosting;
using System;
using System.Threading;
using System.Threading.Tasks;

namespace MicroCommerce.ActivityLogger
{
    public class Scheduler : BackgroundService
    {
        private IServiceProvider ServiceProvider;

        public Scheduler(IServiceProvider serviceProvider)
        {
            ServiceProvider = serviceProvider;
        }

        protected override Task ExecuteAsync(CancellationToken stoppingToken)
        {
            Timer timer = new Timer(new TimerCallback(PollEvents), stoppingToken, 2000, 2000);
            return Task.CompletedTask;
        }

        private void PollEvents(object state)
        {
            try
```

```
            {
                var logger = ServiceProvider.GetService
                            (typeof(MicroCommerce.IActivityLogger)) as ActivityLoggerImpl;
                logger.ReceiveEvents();
            }
            catch
            {

            }
        }
    }
}
```

Project settings are similar to the previous paragraph.
Additionally, we only need to add the previously developed scheduler.

```
//
// ActivityLogger/Setup.cs
//

using System.Net.Http;
using MicroCommerce;
using MicroCommerce.Middleware;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Shed.CoreKit.WebApi;

namespace MicroCommerce.ActivityLogger
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddCorrelationToken();
            services.AddCors();
            services.AddTransient<IActivityLogger, ActivityLoggerImpl>();
            services.AddTransient<HttpClient>();
            services.AddWebApiEndpoints(new WebApiEndpoint<IShoppingCart>
                     (new System.Uri("http://localhost:5002")));
            // register the scheduler
            services.AddHostedService<Scheduler>();
            services.AddLogging(builder => builder.AddConsole());
            services.AddRequestLogging();
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseCorrelationToken();
            app.UseRequestLogging("get");
            app.UseCors(builder =>
            {
                builder
                    .AllowAnyOrigin()
                    .AllowAnyMethod()
                    .AllowAnyHeader();
            });

            app.UseWebApiEndpoint<IActivityLogger>();
        }
```

```
    }
}
```

5. WebUI Project. The User Interface

Bind the project to port 5000 in the same way as `ProductCatalog`.

Install the `Shed.CoreKit.WebApi` nuget package to the project. Links to `Interfaces` and `Middleware` projects are needed only if we going to use calls to microservices inside this project.

In fact, this is a usual ASP.NET project and we can use MVC in it, i.e., to interact with the UI, we can create controllers that use our microservice interfaces as dependencies. But it's more interesting and practical to leave only the user interface behind this project, and redirect all calls from the UI directly to microservices. For this, the `UseWebApiRedirect` extender method from the `Shed.CoreKit.WebApi` package is used.

```
//
// WebUI/Setup.cs
//

using MicroCommerce.Interfaces;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Shed.CoreKit.WebApi;
using System.Net.Http;

namespace MicroCommerce.Web
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.Use(async (context, next) =>
            {
                //  when root calls, the start page will be returned
                if(string.IsNullOrEmpty(context.Request.Path.Value.Trim('/')))
                {
                    context.Request.Path = "/index.html";
                }

                await next();
            });
            app.UseStaticFiles();
            // add redirects to microservices
            app.UseWebApiRedirect("api/products", new WebApiEndpoint<IProductCatalog>
                                  (new System.Uri("http://localhost:5001")));
            app.UseWebApiRedirect("api/orders", new WebApiEndpoint<IShoppingCart>
                                  (new System.Uri("http://localhost:5002")));
            app.UseWebApiRedirect("api/logs", new WebApiEndpoint<IActivityLogger>
                                  (new System.Uri("http://localhost:5003")));
        }
    }
}
```

Everything is very simple. Now, if, for example, a request to '*http://localhost:5000/api/products/get*' comes from the UI, it will be automatically redirected to '*http://localhost:5001/get*'. Of course, for this, microservices must allow cross-domain requests, but we

have allowed this earlier (see CORS in the implementation of microservices).

Now all that remains is to develop a user interface, and Single Page Application is best suited for this. You can use Angular or React, but we just create a small page using the ready-made bootstrap theme and the knockoutjs framework.

```
<!DOCTYPE html>  <!-- WebUI/wwwroot/index.html -->
<html>
<head>
    <meta charset="utf-8" />
    <title></title>
    <link rel="stylesheet"

     href="https://cdnjs.cloudflare.com/ajax/libs/bootswatch/4.5.0/materia/bootstrap.min.css"
/>"
    <style type="text/css">
        body {
            background-color: #0094ff;
        }

        .panel {
            background-color: #FFFFFF;
            margin-top:20px;
            padding:10px;
            border-radius: 4px;
        }

        .table .desc {
            vertical-align: middle;
            font-weight:bold;
        }

        .table .actions {
            text-align:right;
            white-space:nowrap;
            width:40px;
        }
    </style>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"

            integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="

            crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.5.1/knockout-latest.min.js">
    </script>
    <script src="../index.js"></script>
</head>
<body>
    <div class="container">
        <div class="row">
            <div class="col-12">
                <div class="panel panel-heading">
                    <div class="panel-heading">
                        <h1>MicroCommerce</h1>
                    </div>
                </div>
            </div>
            <div class="col-xs-12 col-md-6">
                <div class="panel panel-default">
                    <h2>All products</h2>
                    <table class="table table-bordered" data-bind="foreach:products">
                        <tr>
                            <td data-bind="text:name"></td>
                            <td class="actions">
                                <a class="btn btn-primary"

                                    data-bind="click:function(){$parent.addorder(id, 1);}">ADD</a>
                            </td>
                        </tr>
                    </table>
```

```
                </div>
            </div>
            <div class="col-xs-12 col-md-6">
                <div class="panel panel-default" data-bind="visible:shoppingCart()">
                    <h2>Shopping cart</h2>
                    <table class="table table-bordered"

                     data-bind="foreach:shoppingCart().orders">
                        <tr>
                            <td data-bind="text:product.name"></td>
                            <td class="actions" data-bind="text:quantity"></td>
                            <td class="actions">
                                <a class="btn btn-primary"

                                 data-bind="click:function(){$parent.delorder(id);}">DELETE</a>
                            </td>
                        </tr>
                    </table>
                </div>
            </div>
            <div class="col-12">
                <div class="panel panel-default">
                    <h2>Operations history</h2>
                    <!-- ko foreach:logs -->
                    <div class="log-item">
                        <span data-bind="text:time"></span>
                        <span data-bind="text:description"></span>
                    </div>
                    <!-- /ko -->
                </div>
            </div>
        </div>
    </div>

    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js">
    </script>
    <script>
        var model = new IndexModel();
        ko.applyBindings(model);
    </script>
</body>
</html>
```

```
//
// WebUI/wwwroot/index.js
//

function request(url, method, data) {
    return $.ajax({
        cache: false,
        dataType: 'json',
        url: url,
        data: data ? JSON.stringify(data) : null,
        method: method,
        contentType: 'application/json'
    });
}

function IndexModel() {
    this.products = ko.observableArray([]);
    this.shoppingCart = ko.observableArray(null);
    this.logs = ko.observableArray([]);
    var _this = this;

    this.getproducts = function () {
        request('/api/products/get', 'GET')
            .done(function (products) {
                _this.products(products);
```

```
                console.log("get products: ", products);
            }).fail(function (err) {
                console.log("get products error: ", err);
            });
    };

    this.getcart = function () {
        request('/api/orders/get', 'GET')
            .done(function (cart) {
                _this.shoppingCart(cart);
                console.log("get cart: ", cart);
            }).fail(function (err) {
                console.log("get cart error: ", err);
            });
    };

    this.addorder = function (id, qty) {
        request(`/api/orders/addorder/${id}/${qty}`, 'PUT')
            .done(function (cart) {
                _this.shoppingCart(cart);
                console.log("add order: ", cart);
            }).fail(function (err) {
                console.log("add order error: ", err);
            });
    };

    this.delorder = function (id) {
        request(`/api/orders/deleteorder?orderId=${id}`, 'DELETE')
            .done(function (cart) {
                _this.shoppingCart(cart);
                console.log("del order: ", cart);
            }).fail(function (err) {
                console.log("del order error: ", err);
            });
    };

    this.timestamp = Number(0);
    this.updateLogsInProgress = false;
    this.updatelogs = function () {
        if (_this.updateLogsInProgress)
            return;

        _this.updateLogsInProgress = true;
        request(`/api/logs/get?timestamp=${_this.timestamp}`, 'GET')
            .done(function (logs) {
                if (!logs.length) {
                    return;
                }

                ko.utils.arrayForEach(logs, function (item) {
                    _this.logs.push(item);
                    _this.timestamp = Math.max(_this.timestamp, Number(item.timestamp));
                });
                console.log("update logs: ", logs, _this.timestamp);
            }).fail(function (err) {
                console.log("update logs error: ", err);
            }).always(function () { _this.updateLogsInProgress = false; });
    };

    this.getproducts();
    this.getcart();
    this.updatelogs();
    setInterval(() => _this.updatelogs(), 1000);
}
```

I will not explain in detail the implementation of the UI, as this is beyond the scope of the article's topic, I'll just say that the JavaScript model defines properties and collections for binding from the HTML markup, as well as functions that respond to button clicks for accessing `WebApi` endpoints that are redirected to the corresponding microservices. What the user interface looks like and how it works, we will discuss later in the section "Testing the application".

7. A Few Words About the General Functionality

In this article, we did not touch on some other aspects of application development, such as journaling, health monitoring, authentication, and authorization. This is all considered in detail in the book of Christian Horsdahl and is quite applicable within the frame of the above approach. However, these aspects are too specific for each application and it makes no sense to put them in a Nuget package, it is better to just create a separate assembly within the application. We have created such an assembly - this is **Middleware**. For example, just add here the functionality for query logging, which we already linked when developing microservices (see paragraphs 2-4).

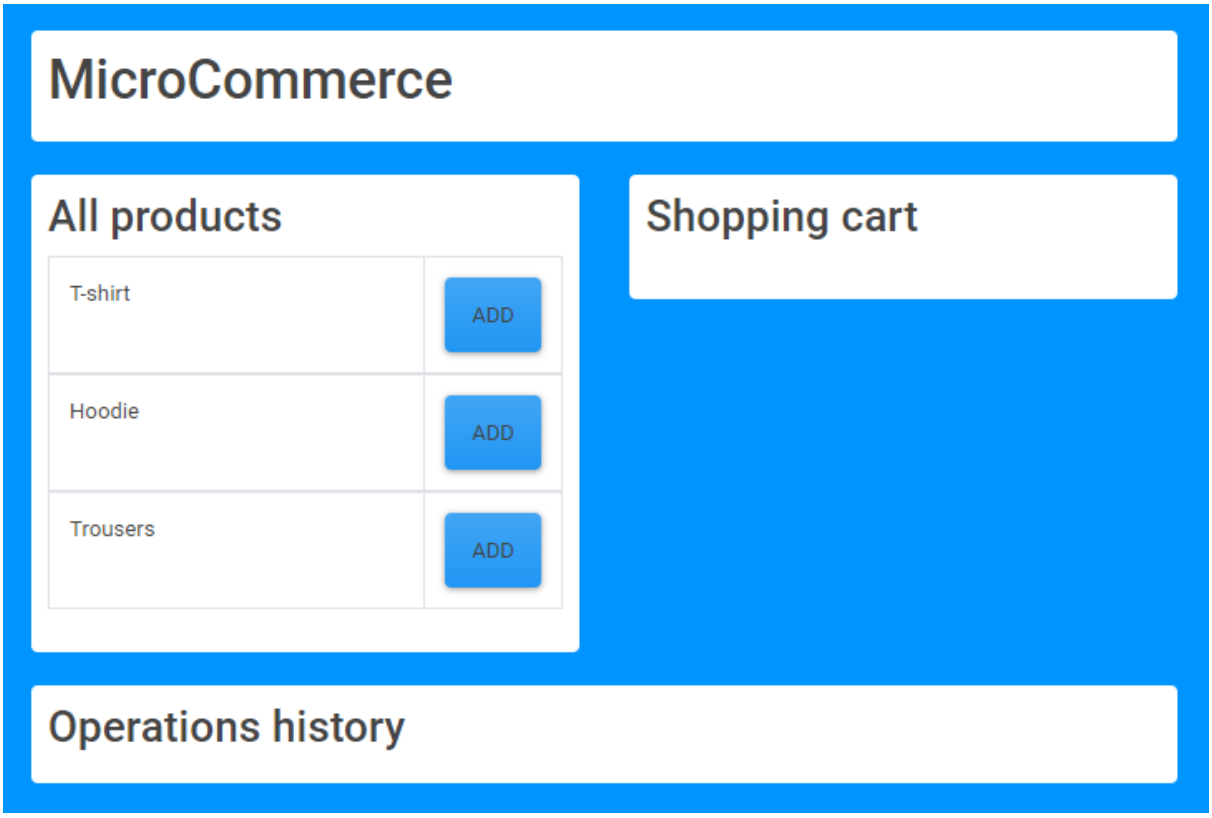
```
//
// Middleware/RequestLoggingExt.cs
//

using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;

namespace MicroCommerce.Middleware
{
    public static class RequestLoggingExt
    {
        private static RequestLoggingOptions Options = new RequestLoggingOptions();

        public static IApplicationBuilder UseRequestLogging
               (this IApplicationBuilder builder, params string[] exclude)
        {
            Options.Exclude = exclude;

            return builder.UseMiddleware<RequestLoggingMiddleware>();
        }

        public static IServiceCollection AddRequestLogging(this IServiceCollection services)
        {
            return services.AddSingleton(Options);
        }
    }

    internal class RequestLoggingMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger _logger;
        private RequestLoggingOptions _options;

        public RequestLoggingMiddleware(RequestDelegate next,
               ILoggerFactory loggerFactory, RequestLoggingOptions options)
        {
            _next = next;
            _options = options;
            _logger = loggerFactory.CreateLogger("LoggingMiddleware");
        }

        public async Task InvokeAsync(HttpContext context)
        {
            if(_options.Exclude.Any
            (i => context.Request.Path.Value.Trim().ToLower().Contains(i)))
            {
                await _next.Invoke(context);
                return;
            }

            var request = context.Request;
            _logger.LogInformation($"Incoming request: {request.Method},
            {request.Path}, [{HeadersToString(request.Headers)}]");
            await _next.Invoke(context);
            var response = context.Response;
            _logger.LogInformation($"Outgoing response: {response.StatusCode},
            [{HeadersToString(response.Headers)}]");
        }
```

```
        private string HeadersToString(IHeaderDictionary headers)
        {
            var list = new List<string>();
            foreach(var key in headers.Keys)
            {
                list.Add($"'{key}':[{string.Join(';', headers[key])}]");
            }

            return string.Join(", ", list);
        }
    }

    internal class RequestLoggingOptions
    {
        public string[] Exclude = new string[] { };
    }
}
```

A pair of `AddRequestLogging()` / `UseRequestLogging(...)` methods allows us to enable query logging in the microservice. The `UseRequestLogging` method can also take an arbitrary number of exception paths. We used this in `ShoppingCart` and in `ActivityLogger` to exclude event polls from logging and to avoid log overflows. But again, journaling, like any other common functionality, is an exclusive responsibility of developers and is implemented within the frame of a specific project.

Testing of the Application

We launch the solution, we see on the left a list of products to add to the basket, an empty basket on the right and the history of operations below, also empty so far.



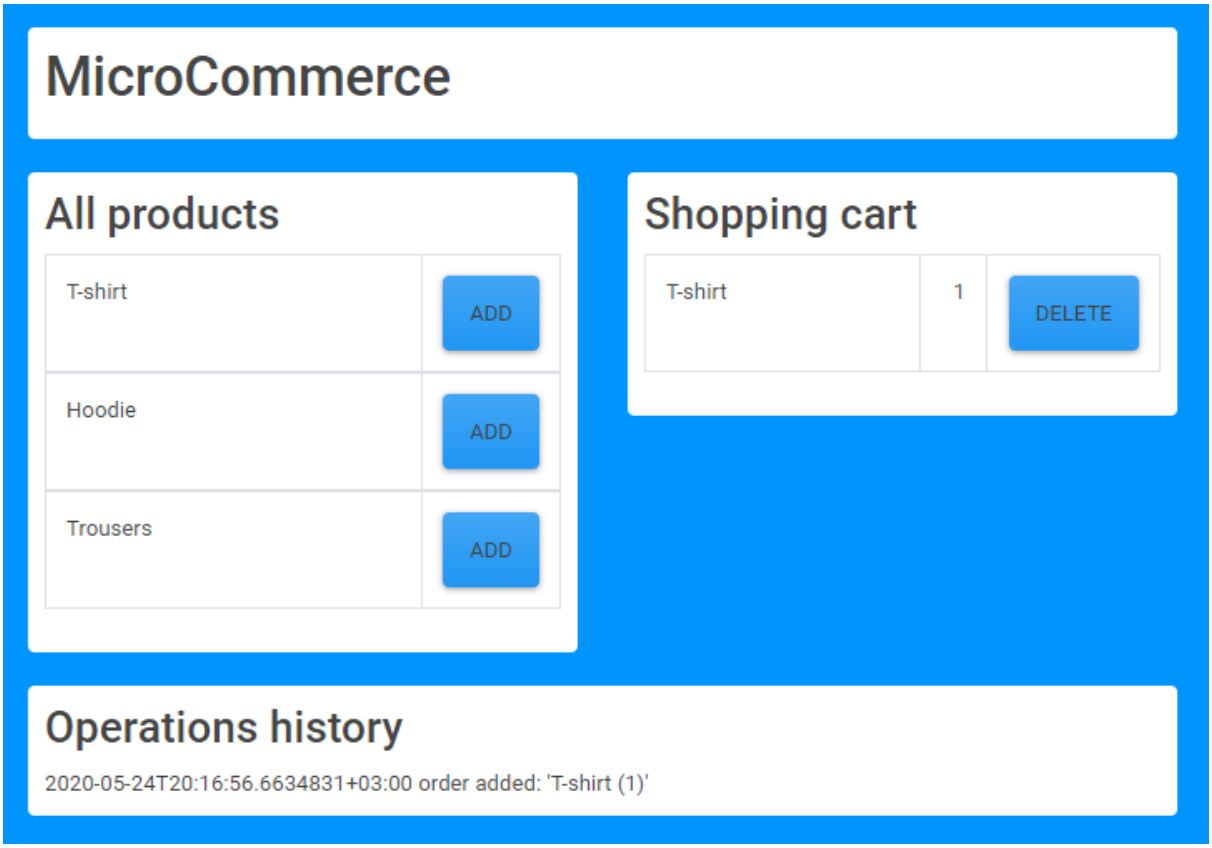
In the consoles of microservices, we see that at startup, the UI has already requested and received some data. For example, to get a list of products, a request was sent *http://localhost:5000/api/products/get*, which was redirected to *http://localhost:5001/get*.

```
info: LoggingMiddleware[0]
      Incoming request: OPTIONS, /get, ['Connection':[keep-alive], 'Accept':[*/*], 'Ac
cept-Encoding':[gzip, deflate, br], 'Accept-Language':[ru-RU,ru;q=0.9,en-US;q=0.8,en;q
=0.7], 'Host':[localhost:5001], 'Referer':[http://localhost:5000/], 'User-Agent':[Mozi
lla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83
.0.4103.97 Safari/537.36], 'Origin':[http://localhost:5000], 'Access-Control-Request-M
ethod':[GET], 'Access-Control-Request-Headers':[content-type,x-requested-with], 'Sec-F
etch-Mode':[cors], 'Sec-Fetch-Site':[same-site], 'Sec-Fetch-Dest':[empty]]
info: LoggingMiddleware[0]
      Outgoing response: 204, ['Access-Control-Allow-Headers':[content-type,x-requeste
d-with], 'Access-Control-Allow-Methods':[GET], 'Access-Control-Allow-Origin':[*], 'Cor
relation-Token':[3ede0e79-56eb-44de-83d9-b57348dae139]]
info: LoggingMiddleware[0]
      Incoming request: GET, /get, ['Connection':[keep-alive], 'Content-Type':[applica
tion/json], 'Accept':[application/json, text/javascript, */*; q=0.01], 'Accept-Encodin
g':[gzip, deflate, br], 'Accept-Language':[ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7], 'Host
':[localhost:5001], 'Referer':[http://localhost:5000/], 'User-Agent':[Mozilla/5.0 (Win
dows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.97 S
afari/537.36], 'Origin':[http://localhost:5000], 'X-Requested-With':[XMLHttpRequest],
'Sec-Fetch-Site':[same-site], 'Sec-Fetch-Mode':[cors], 'Sec-Fetch-Dest':[empty]]
info: LoggingMiddleware[0]
      Outgoing response: 200, ['Date':[Sun, 07 Jun 2020 07:40:01 GMT], 'Transfer-Encod
ing':[chunked], 'Content-Type':[application/json], 'Server':[Kestrel], 'Access-Control
-Allow-Origin':[*], 'Correlation-Token':[af92cfea-5442-48f1-93a2-aa03b9e1b131]]
```

Similarly, the UI received the status of an order basket from `ShoppingCart`.

```
info: LoggingMiddleware[0]
      Incoming request: OPTIONS, /get, ['Connection':[keep-alive], 'Accept':[*/*], 'Ac
cept-Encoding':[gzip, deflate, br], 'Accept-Language':[ru-RU,ru;q=0.9,en-US;q=0.8,en;q
=0.7], 'Host':[localhost:5002], 'Referer':[http://localhost:5000/], 'User-Agent':[Mozi
lla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83
.0.4103.97 Safari/537.36], 'Origin':[http://localhost:5000], 'Access-Control-Request-M
ethod':[GET], 'Access-Control-Request-Headers':[content-type,x-requested-with], 'Sec-F
etch-Mode':[cors], 'Sec-Fetch-Site':[same-site], 'Sec-Fetch-Dest':[empty]]
info: LoggingMiddleware[0]
      Outgoing response: 204, ['Access-Control-Allow-Headers':[content-type,x-requeste
d-with], 'Access-Control-Allow-Methods':[GET], 'Access-Control-Allow-Origin':[*], 'Cor
relation-Token':[31af0137-9e17-47ae-97fd-b94798e1c9ec]]
info: LoggingMiddleware[0]
      Incoming request: GET, /get, ['Connection':[keep-alive], 'Content-Type':[applica
tion/json], 'Accept':[application/json, text/javascript, */*; q=0.01], 'Accept-Encodin
g':[gzip, deflate, br], 'Accept-Language':[ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7], 'Host
':[localhost:5002], 'Referer':[http://localhost:5000/], 'User-Agent':[Mozilla/5.0 (Win
dows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.97 S
afari/537.36], 'Origin':[http://localhost:5000], 'X-Requested-With':[XMLHttpRequest],
'Sec-Fetch-Site':[same-site], 'Sec-Fetch-Mode':[cors], 'Sec-Fetch-Dest':[empty]]
```

When click the ADD button, the product is added to the basket. If this product has already been added, the quantity increases.



The microservice ShoppingCart is sent a request *http://localhost:5002/addorder/<productid>*.

```
info: LoggingMiddleware[0]
      Incoming request: PUT, /addorder/6bf3a1ce-1239-4528-8924-a56ff6527595/1, ['Conne
ction':[keep-alive], 'Content-Type':[application/json], 'Accept':[application/json, te
xt/javascript, */*; q=0.01], 'Accept-Encoding':[gzip, deflate, br], 'Accept-Language':
[ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7], 'Host':[localhost:5002], 'Referer':[http://loca
lhost:5000/], 'User-Agent':[Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.
36 (KHTML, like Gecko) Chrome/83.0.4103.97 Safari/537.36], 'Origin':[http://localhost:
5000], 'Content-Length':[0], 'X-Requested-With':[XMLHttpRequest], 'Sec-Fetch-Site':[sa
me-site], 'Sec-Fetch-Mode':[cors], 'Sec-Fetch-Dest':[empty]]
info: LoggingMiddleware[0]
      Outgoing response: 200, ['Date':[Sun, 07 Jun 2020 07:54:45 GMT], 'Transfer-Encod
ing':[chunked], 'Content-Type':[application/json], 'Server':[Kestrel], 'Access-Control
-Allow-Origin':[*], 'Correlation-Token':[d80a97bd-3efa-4487-b36c-94ac5a826992]]
```

But since `ShoppingCart` does not store the list of products, it receives information about the ordered product from `ProductCatalog`.

```
info: LoggingMiddleware[0]
      Incoming request: GET, /get/6bf3a1ce-1239-4528-8924-a56ff6527595, ['Content-Type
':[application/json; charset=utf-8], 'Accept':[application/json], 'Host':[localhost:50
01], 'Request-Id':[|89035ea9-435de72e6bc9986d.1.], 'Content-Length':[0], 'Correlation-
Token':[d80a97bd-3efa-4487-b36c-94ac5a826992]]
info: LoggingMiddleware[0]
      Outgoing response: 200, ['Date':[Sun, 07 Jun 2020 07:54:45 GMT], 'Transfer-Encod
ing':[chunked], 'Content-Type':[application/json], 'Server':[Kestrel], 'Correlation-To
ken':[d80a97bd-3efa-4487-b36c-94ac5a826992]]
```

Please note that before sending a request to `ProductCatalog`, a correlation token was assigned. This allows us to track the chain of related queries in case of a failure.

Upon completion of the operation, `ShoppingCart` publishes an event that tracks and logs the `ActivityLogger`. In turn, the UI periodically polls this microservice and displays the received data in the operation history panel. Of course, the entries in the history appear with some delay, because it is a parallel mechanism that does not depend on the operation of adding a product.

Conclusion

Nuget package `Shed.CoreKit.WebApi` allows us to:

- fully focus on developing the business logic of the application, without making additional efforts on the issues of microservices interaction;
- describe the structure of the microservice with the .NET interface and use it both in the development of the microservice itself and for generating the Web client (the Web client for the microservice is generated by the factory method after registering the interface in the IoC container and is provided as a dependency);
- Register microservice interfaces as dependencies in the IoC container;
- organize redirection of requests from the Web UI to microservices without additional efforts in the development of UI.

Drawbacks

The idea of interacting via Web api is quite attractive, but there are some problems:

- we must take care of providing microservice configurations on which our microservice depends;
- if our microservice is overloaded, we would like to launch several additional instances of this microservice to reduce the load.

To handle this, we need another architecture, namely star-shaped architecture, that is, interaction through a common bus.

How to organize interaction through a common bus like MQ service, we will consider in the next article

History

- 14th June, 2020: Initial version

License

About the Author

Shkurko Eugene

Software Developer (Senior) AMTOSS

Ukraine

No Biography provided

Comments and Discussions

1 message has been posted for this article Visit **https://www.codeproject.com/Articles/5271067/NET-Core-Interaction-of-Microservices-via-Web-API** to post and view comments on this article, or click **here** to get a print view with messages.