

Monolith vs



Microservices



2-3. Microservice Architecture and Docker

Asp.NET & searching for microservices

Agenda

- Asp.net Core
- Microservices Scenario
- Searching for microservices

ASP.NET Core Overview

- history and features of .NET, the runtime, and ASP.NET Core, the web server.

A Brief History of .NET

- .NET Framework in 2000
- .NET Core has become .NET: a unified platform for building great modern apps.
- runs equally well on x64 and ARM processors.
- You can install it on a machine just as easily as embed it in a Docker container.
- One can bundle .NET into a single executable or leverage the .NET runtime installed on the machine.

.NET (2)

- the next version of .NET Core would not be called .NET Core 4.0 but rather would be called .NET 5.
- This one unified base class library and runtime can now power cloud properties, desktop apps, phone apps, IoT devices, machine learning applications, and console games: “One .NET.”
- .NET 6.0 carries on this tradition
- “Current release” versions are supported for 18 months. .NET Core 2.2 is designated as a “Current release.” It was released in late 2018, so it’s support has already ended.
- “Long-term support” versions are supported for 3 years from their release date. .NET Core 3.1 is designated as an LTS release. It came out in 2019, so it will be supported until 2022.
- .NET 6 is designated as an LTS release. It shipped in November 2021, so it will be supported until 2024.
- <https://dotnet.microsoft.com/platform/support/policy/dotnet-core>.

Stacks

- On the web server stack, we have ASP.NET Core MVC, Web API, and SignalR.
- Added to this, we now have gRPC and Blazor, a web-component framework that runs in the browser on WebAssembly.
- On the desktop stack, we have WinForms, including a designer new in .NET Core 5 and WPF.
- Added to this, we have XAML-based apps built-in Xamarin for use on mobile devices, and MAUI is a new cross-platform UI toolkit previewing during the .NET 6 timeframe.
- Unfortunately, WCF and WebForms have not been ported
- There's some great work that allows an old WebForms app to run on Blazor.
- If you're in need of WCF, CoreWCF was donated to the .NET Foundation and may meet the need.

Installing .NET 6.0 and ASP.NET Core

- <https://dotnet.microsoft.com/download>
- let's build an ASP.NET Core project. There's a lot more going on in this template.

```
mkdir helloweb
```

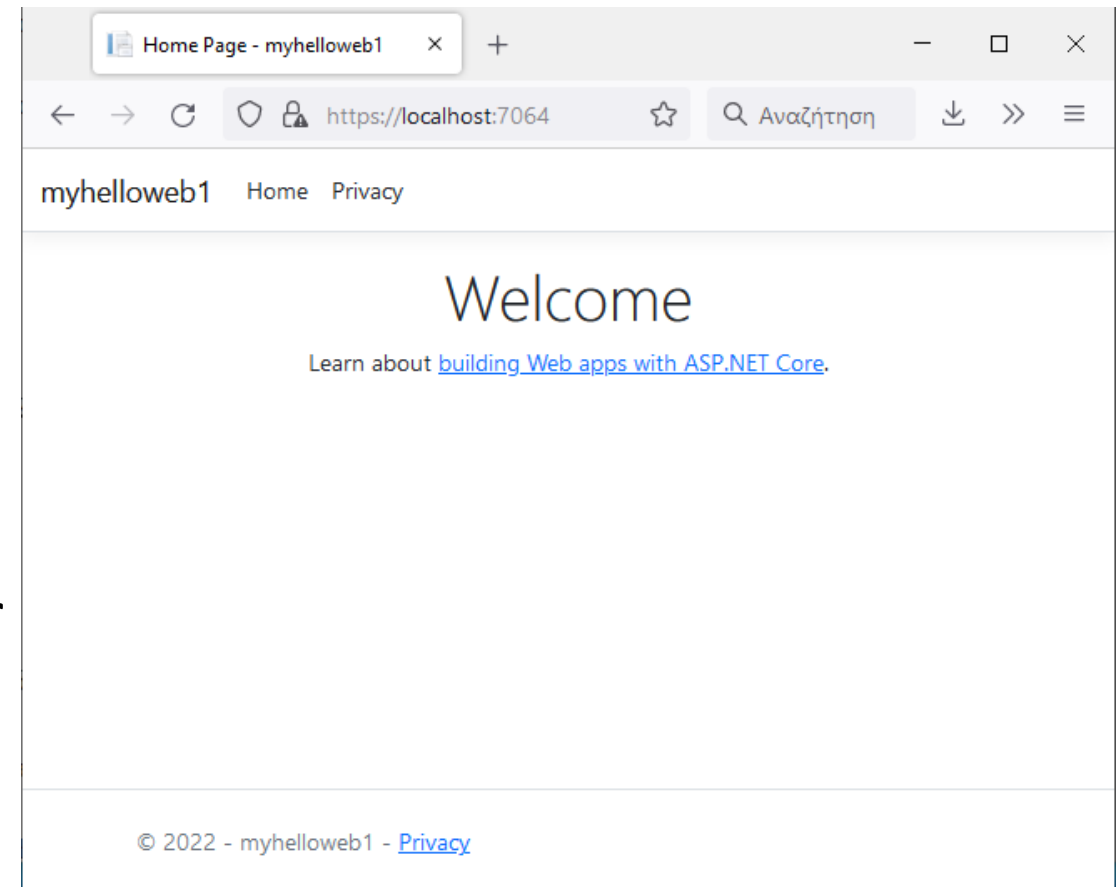
```
cd helloweb
```

```
dotnet new mvc --name myhelloweb
```

- Like the console app, this scaffolds out a new ASP.NET Core website with controllers,
- views, and JavaScript and CSS files. You can run the website in the same way:
- `dotnet run`

MVC .NET ports

- As the website starts up, note the web address the site runs on. The HTTP port
- will be a random unused port between 5000 and 5300, and HTTPS will be randomly
- selected between 7000 and 7300. For example, the console output could list https://
- localhost:7000/. Copy the URL from your console, open a browser, and paste the
- URL. You'll see your sample website running.



Visual Studio

- <https://visualstudio.microsoft.com/vs>
- Community Edition

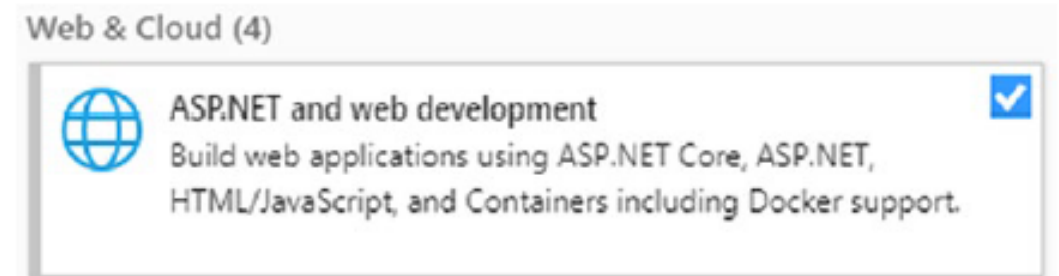


Figure 2-2. ASP.NET install component

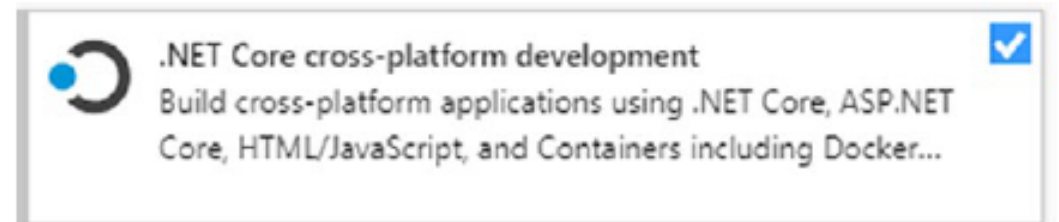


Figure 2-3. .NET Core cross-platform install component

MVC at a Glance

- MVC has three main components: the Model, View, and Controller.
- Combined with ASP.NET there is a fourth component, Routing
- Convention over Configuration,
- Model Binding,
- Anti-Forgery Token

MVC (2)

- The MVC pattern is a wheel of steps beginning with the request and ending with the response.
- First, the URL is parsed into sections and flows into the router.
- The router looks at the URL pieces and selects the correct controller and action (class and method) to satisfy the request.
- The controller's action method is responsible for calling into the rest of the application to process the data, harvest the results, and pass a model (class) to the view.
- The view is a templating engine, mapping HTML fragments together with the data to produce an HTTP response.
- The action method could also choose to forgo the view for API requests, choosing instead to hand data to a serialization engine that might produce JSON, XML, or gRPC.
- Once the response is constructed, the HTTP headers are attached, and the data is returned to the calling browser or application.
- This MVC pattern is indeed more complex than a simple URL to File path system like we had with WebForms or Classic ASP.

Controller flexibility

- The controller's action can make choices.
- Does the data requested not exist? Then let's send them to the not found page instead of the data page.
- Would they like JSON data instead of an HTML page? Let's forgo the view and choose a response method that serializes data instead.

Convention over Configuration

- Convention over Configuration refers to how ASP.NET MVC uses naming conventions to tie URLs to controller classes and methods, controller methods to views and URL segments and post bodies to C# classes and URL segments and post bodies to C# classes.
- It is not required to have every view programmatically added to any list or association.

Routing

- As a browser or service makes an HTTP connection, the URL pattern matching determines which controller and action receive the request.
- Using a default template, the Home controller is created and handles default requests.
- In the URL, the controller is defined right after the domain name, or port number if one is specified.
- For example, with the URL `http://localhost/Home/Index`, the controller class name is Home and the action method name is Index.
- The actions are the methods in the controller class that receive the requests, execute any business logic, and return information or just status

Routing (2)

- With ASP.NET Core, most typically the routes are configured with attributes on each class. Let's look at an example controller inside an ASP.NET Core MVC project:

```
[Route("[controller]")]
public class CustomerController : Controller
{
    [Route("get/{id}")]
    public ActionResult GetCustomer(int id)
    {
        return View();
    }
}
```

- Here the Route attribute at the top specifies that if the URL starts with <https://some-site/>
- Customer, it should run methods within this class. Next, the Route attribute on the method maps to the next part of the URL. If the URL is <https://some-site/Customer/get/7>, then the GetCustomer() method should be called.

Endpoints

- Beginning in ASP.NET Core 2.2, you can use endpoints instead of the route table.
- Endpoint mapping allows for routing rules to be defined without impacting other web frameworks, meaning that it is possible to leverage SignalR along with MVC, Razor, and others without them colliding in the middleware pipeline.

```
app.UseEndpoints(endpoints => {  
    endpoints.MapControllerRoute(name: "default",  
        pattern: "{controller=Home}/{action=Index}/{id?}");  
});
```

- Adding a Routing Attribute to the action helps control the routing logic as well.

Routing Example

- In the following example, the Index method is called with any of these URLs: /, /home/, or /home/index. The use of routing attributes also allows for method names not to match. Using the URL /home/about, in this example, will call the ShowAboutPage method.

```
[Route("")]
```

```
[Route("Home")]
```

```
[Route("Home/Index")]
```

```
public IActionResult Index()
```

```
{
```

```
    return View();
```

```
}
```

```
[Route("Home/About")]
```

```
public IActionResult ShowAboutPage()
```

```
{
```

```
    return View("About");
```

```
}
```

Controller

The job of a controller is

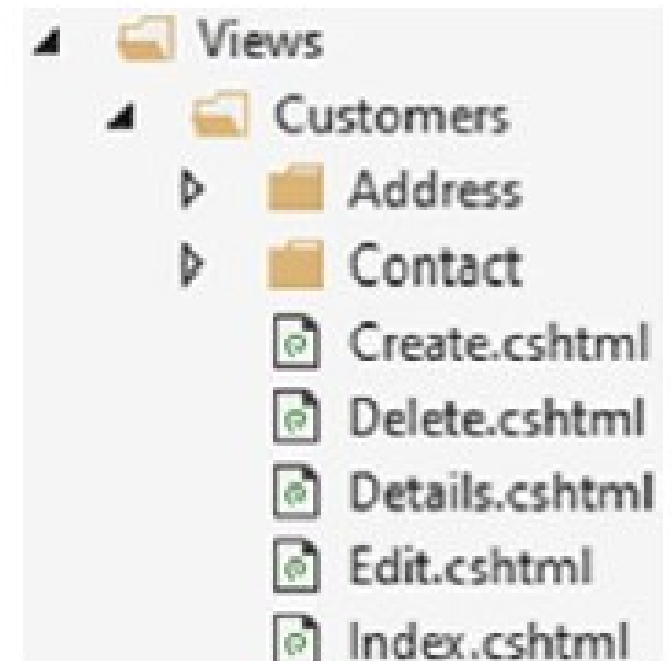
- to map request inputs to classes,
 - call the business classes that do the work, and
 - provide the results to the response
-
- They receive requests, via routing, to actions (methods), and work with model data and sending to the appropriate view.
-
- validation, security, and content types to customize the response sent to the caller.

Controller (2)

- simply a name and the word “Controller,” for example, CustomerController
- lightweight controllers.
- just enough logic to handle receiving and sending data.
- Other business logic and data access code should be in separate classes.

Views

- representation of a page with or without dynamic data
 - logic inside a view should be incredibly simple
 - complex logic, move this logic into the controller's action, where it can be correctly tested
 - *Inside the Views folder, we have a folder for each controller, and a View for each action*
-
- `http://localhost/Customers/Edit/3453`
 - with default routing configured,
 - we'd hit the CustomersController class, the Edit method, and find the vi in the Views/Customers/Edit.cshtml file.
 - the router could choose a different controller and action, and the action method could also choose a different view.

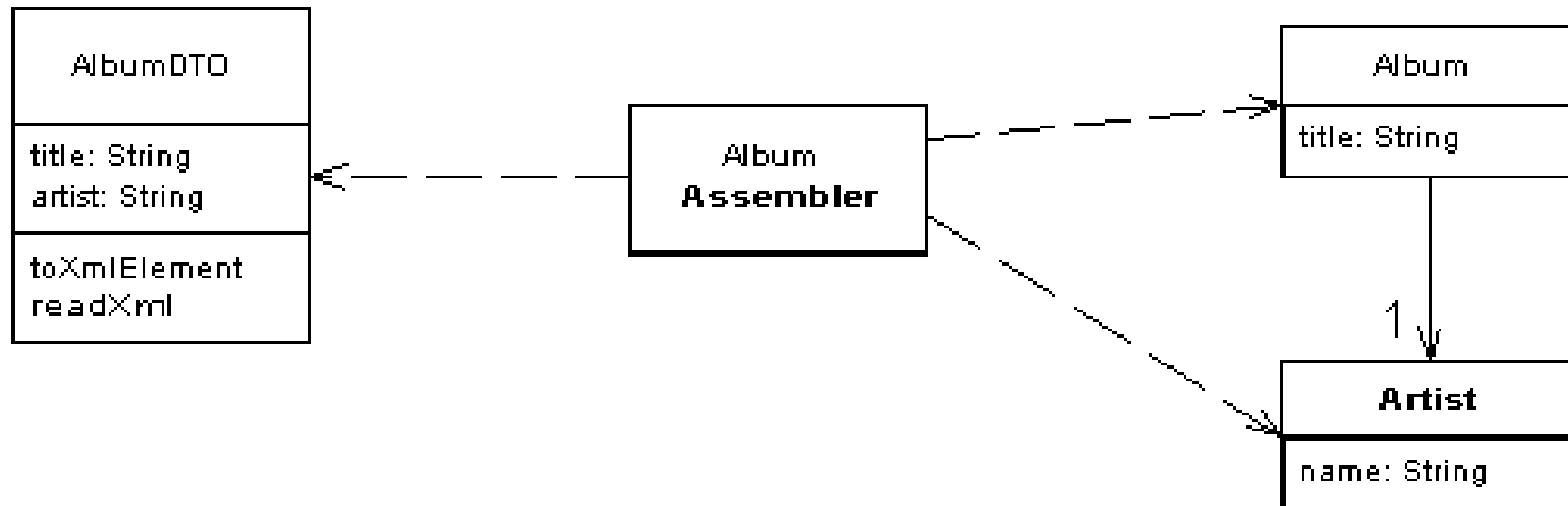


Model

- Models in ASP.NET Core are classes used to represent data
- model is a package holding the data the view renders.
- when retrieving a list of customers, each customer is an instance of a customer model.
- It has properties for the fields of that customer.
 - Name, Address, Phone Number, etc.
- A view's logic will loop through a collection of models and render each one.

Model (2)

- Entities that map to tables in our database
- data-transform objects, or DTOs, used to convey data between sections of an application
 - <https://martinfowler.com/bliki/LocalDTO.html>



- [Microsoft Cutting Edge - Pros and Cons of Data Transfer Objects](#)
- [Microsoft Create Data Transfer Objects \(DTOs\)](#)

View-Model

- When designing a class for use in a view, we could call this a view-model. This class is specifically tailored to keep the view rendering simple.
- Inside the controller or other business logic classes, we may map data from Entity classes into view-model classes.
- View-models are specific to the purpose it is being rendered.
- There may be a view-model for listing customers while another view-model is used for when editing customer details.

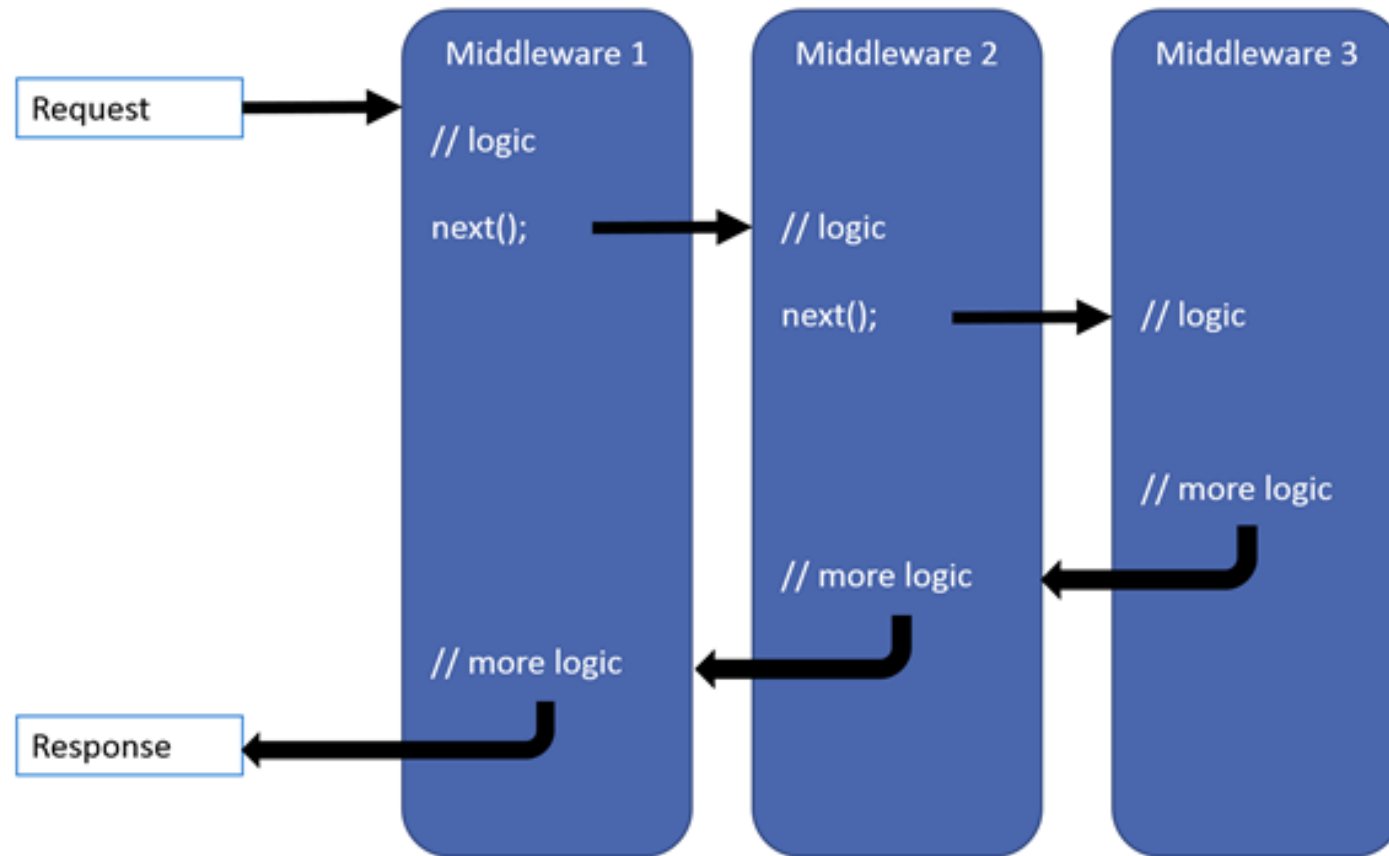
request-models

- Model binding can also use views to map request details into controller actions.
- For example, when saving a user's profile, we may have a request-model that includes their name, address, and phone number.
- This differs from the Entity class, which may also include the username and password
- We definitely don't want to include these properties on the request-model, because we don't want extra data in a request to become an injection attack that might change a user's password.

ASP.NET Core Middleware

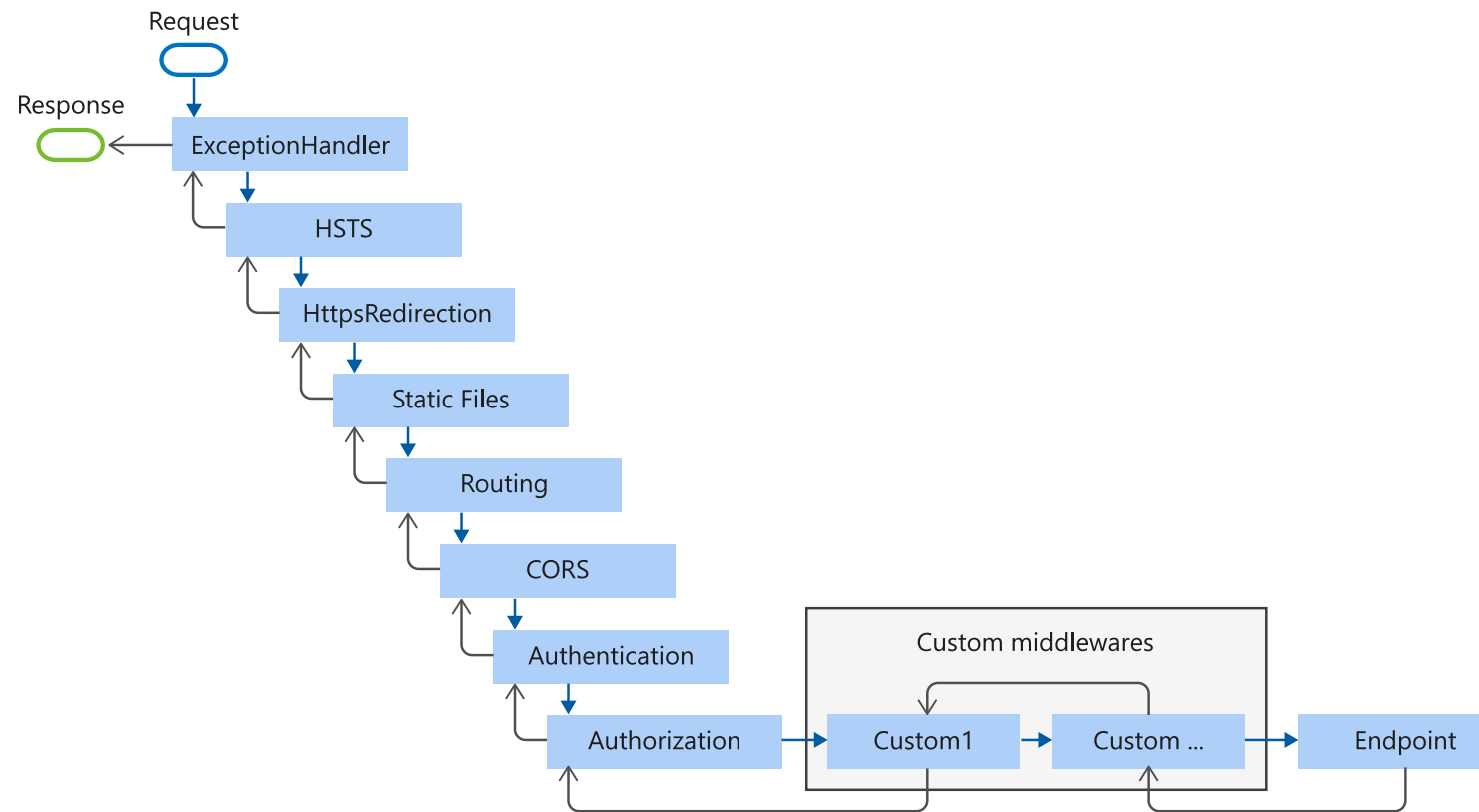
- Middleware pipeline. ([link to details for MS Middleware](#))
 - This pipeline is a set of steps that all requests flow through.
 - parses HTTP headers into C# classes or
 - JSON data from the request body into a request-model
 - captures and logs exceptions
 - validates the user is authenticated, returning an HTTP 401 status code if they're not
- Startup.cs file (Configure Method)
 - The IApplicationBuilder type is the main instance the configuration uses.
 - Leveraging IWebHostEnvironment will provide information such as if the web application is running in Development mode

Create a middleware pipeline with WebApplication



Middleware order

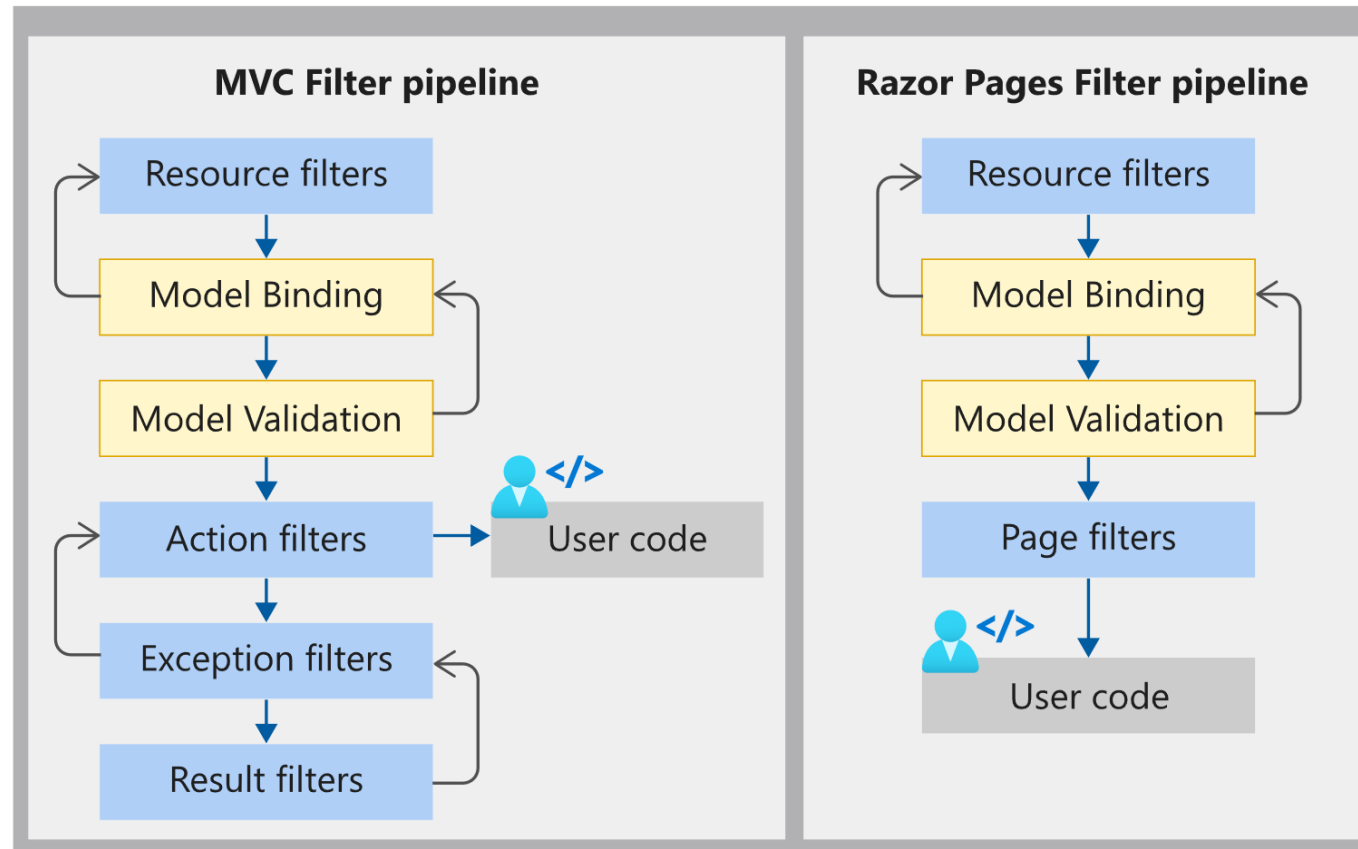
- The following diagram shows the complete request processing pipeline for ASP.NET Core MVC and Razor Pages apps.
- The **Endpoint** middleware in the preceding diagram executes the filter pipeline for the corresponding app type—MVC or Razor Pages.
- [HTTP Strict-Transport-Security](#)



MVC Endpoint

MVC Endpoint

(called by the Endpoint Middleware)



ASP.NET Core Web API

- Similar to the MVC project, ASP.NET Core includes a Web API template.
 - MVC template is geared toward the website
 - Web API template is geared toward microservices
- MVC pattern
- Web API project template, Views are removed, and Controllers typically return JSON data instead
- Web API project includes the Swashbuckle library. Swashbuckle provides a great way to browse through and explore an API using OpenAPI (formerly Swagger)

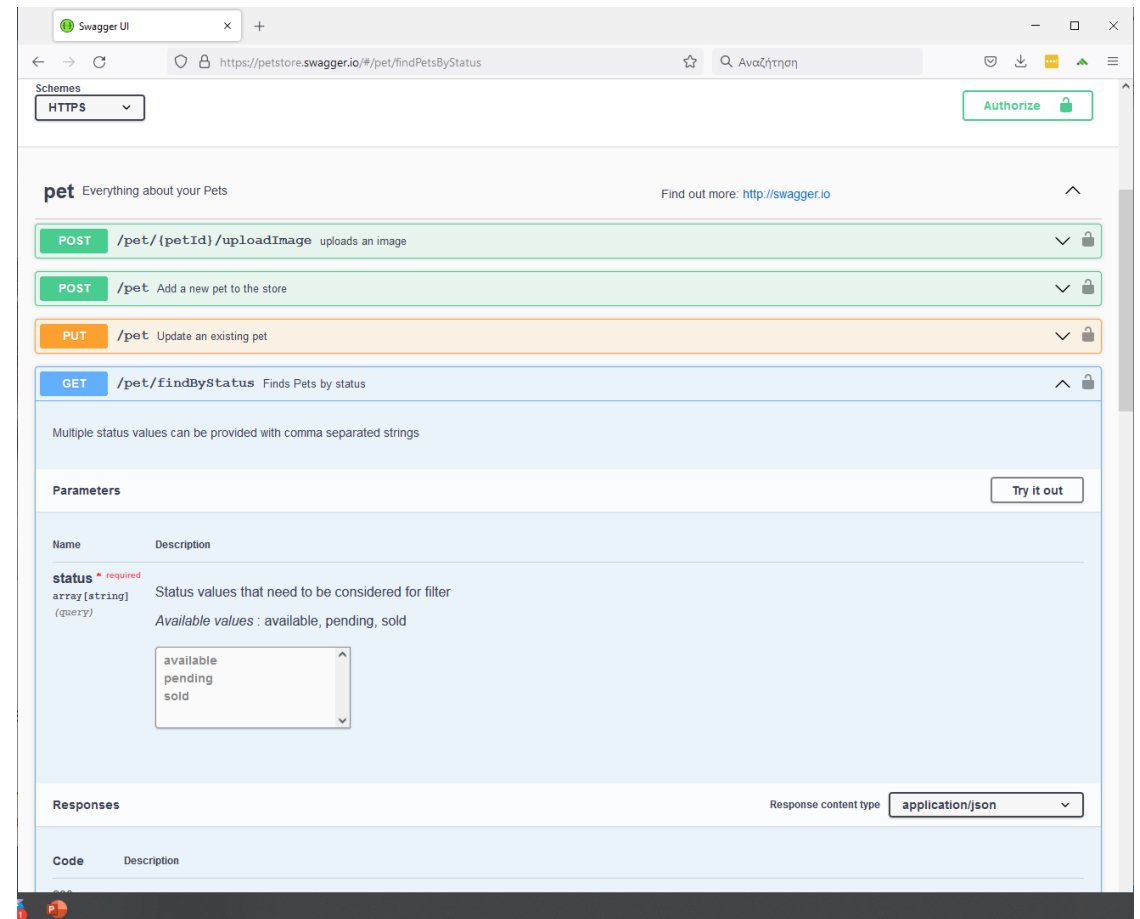
sample Swagger UI

A sample Swagger UI experience from the Swagger documentation at <https://petstore.swagger.io>

```
dotnet new webapi --name mywebapi  
dotnet run
```

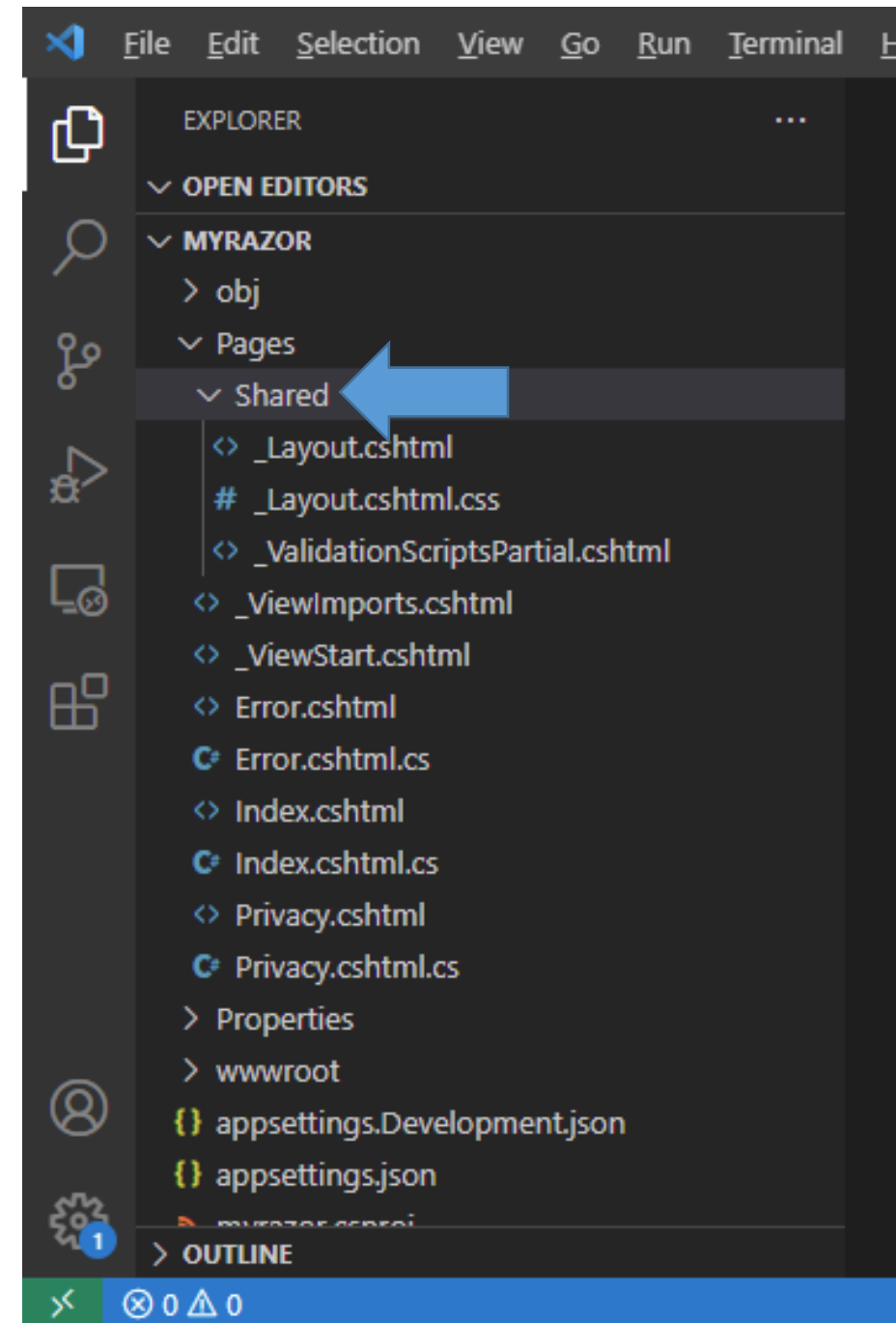
```
https://localhost:7000/  
https://localhost:7000/swagger/.
```

When you're done, hit Ctrl+C to stop the web server.



Razor Pages

- In ASP.NET Core 2.2, Razor pages became available.
- Razor pages don't use the MVC paradigm
- Classic ASP and WebForms: it maps URLs directly to files.
 - This can be great for building simple web pages that don't need the ability to choose different paths as the request navigates through the router, controller's action, and view.
- In a Razor Pages project, all pages exist inside a Pages folder.
- To map to the URL
- <https://localhost:7000/customer/detail>
- the view would be a file inside Pages/Customer/Detail.cshtml
- Shared is a common area to leverage [dependency injection](#)



DI

- **Dependency Inversion Principle:** it's a software design principle; it suggests a solution to the dependency problem but ***does not say how*** to implement it or which technique to use.
- Inversion of Control (IoC): this is a way to apply the Dependency Inversion Principle. ***Inversion of Control is the actual mechanism*** that allows your higher-level components to depend on abstraction rather than the concrete implementation of lower-level components.
- Inversion of Control is also known as the **Hollywood Principle**. This name comes from the Hollywood cinema industry, where, after an audition for an actor role, usually the director says, **don't call us, we'll call you**.

Άσκηση 1

- εκτελέστε DI για web και console με βάση το παράδειγμα σε .NET6
- <https://auth0.com/blog/dependency-injection-in-dotnet-core/>

Sample Razor

- `dotnet new razor`
- `code .`
- `dotnet new page -n Index`
 - `@page`
 - `@model MyApp.Namespace.IndexModel`
 - `@{`
 - `<p>This is the page for Customers.</p>`
 - `}`
- `dotnet run`

Minimal APIs (new in ASP.NET Core 6.0)

- C# 10 to create less code for simple APIs
- Minimal API techniques can be used in MVC, Web API, and Razor Page projects
- if your microservice is really simple, you may also see some performance gains from the minimal API approach.
- In the minimal API paradigm, the Startup.cs file is folded into the Program.cs file.
- familiar with Node.js and Express.js, you'll feel right at home with the minimal APIs.

Minimal APIs (2)

Program.cs

- Hello World program using the minimal APIs:

```
var builder = WebApplication.CreateBuilder(args);  
var app = builder.Build();  
app.MapGet("/", () => "Hello World");  
app.Run();
```

HelloWorld.csproj

```
<Project Sdk="Microsoft.NET.Sdk.Web">  
<PropertyGroup>  
<TargetFramework>net6.0</TargetFramework>  
</PropertyGroup>  
</Project>
```

Disadvantages of minimal API

- difficult to test a minimal API because the Program class is not public
 - making it difficult to reference from unit tests.
- loses the separation of concerns when one jams all the features into a single file that starts a web server.

Microservices Scenario

- **Searching for Microservices**
- client's problems with an application critical for their business.
- workshop-style meeting, called Event Storming
- Domain-Driven Design (DDD) and how developers can use it to prepare for decomposing a monolithic application into microservices
- brief introduction to the customer's business, which will aid with an example of analyzing the need for microservices

The Business

- Hyp-Log is a shipping logistics middleman coordinator for hotshot deliveries.
- Hotshot deliveries are point-to-point transfers of goods or materials that require a level of expedience that may not be attainable by larger carrier firms.
- Hyp-Log took all their rate spreadsheets and the analysis that staff were performing manually and built a “rate engine” to perform those same calculations
- previous employee of Hyp-Log created a custom application for them
- “Shipment Parcel Logistics Administration – Thing,” aka “SPLAT.”

Shipment Parcel Logistics Administration

- SPLAT has three main user types: customers, carriers, and administrators
- customers to submit load requests, pick a carrier based on a list of quotes, and provide payment.
- Carriers can manage their fleet and base costs per mile plus extra costs based on particular needs.
- Some loads require a trailer, are hazardous material, or even refrigerated.

Domain-Driven Design

- Domain,
- Ubiquitous Language,
- Bounded Contexts, and
- Aggregates with Aggregate Roots

Domain

- Hyp-Log, their domain is hotshot load management
- Other domains may exist in the company, like Human Resources and Insurance, but they are not relative to why Hyp-Log exists as a company.
- Eric Evans is the founder of DDD and author of Domain-Driven Design:
[Tackling Complexity in the Heart of Software.](#)
- DDD was not created for the use of microservices but can be leveraged for the development of microservices

Subdomains & Problem Space

- subdomain is a grouping of related business processes
- accounting domain
 - Accounts Payable,
 - Accounts Receivable, and
 - Payroll
- Payroll subdomain
 - Managing time sheets,
 - wages, and
 - federal forms
- **The groups of code** that provide functionality for their **subdomain** are called a **bounded context**

Subdomain type

- Core – Each core subdomain in an application contains one or more bounded contexts that are critical to the company.
- Supportive – The supportive subdomains are not deemed critical but contain code that is supportive of the business.
- Generic – Lastly, generic subdomains are those that are replaceable with off-the-shelf solutions.

Ubiquitous Language

- collection of phrases and terms that helps everyone involved to have a clear and concise
- user submit a request ->
 - customer provides a coupon code
 - the customer issues dispute request
- terms have overloaded meanings
- UL should also extend to the code
 - concise names for namespaces, classes, and methods helps

Bounded Contexts

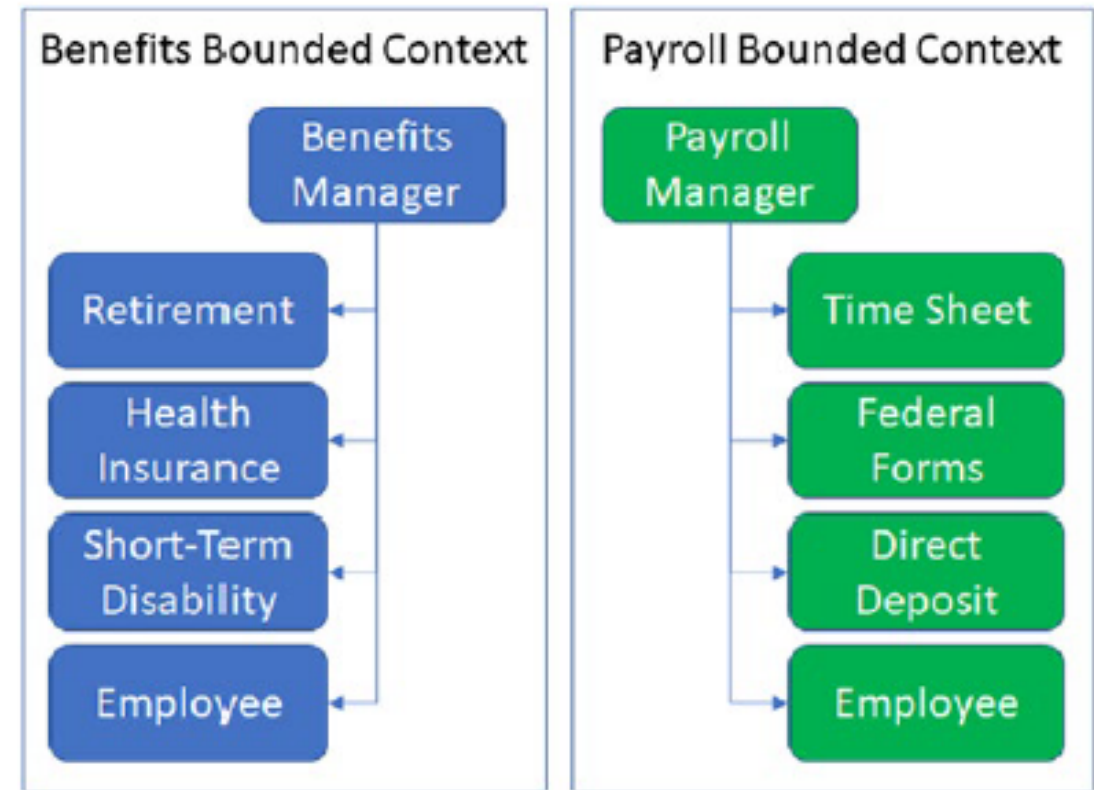
- A bounded context is a collection of codes that implements business processes in a subdomain distinctly different from other processes.
- using ubiquitous language here, the term “part” is distinguished to the various purposes to become
- “purchased part” vs. “manufactured part.”
- Thus, there is a bounded context, determined by the language around the term “part.”
- Functionality that uses specific language in the namespaces, class names, and method help identify a bounded context and its purpose.

Aggregates and Aggregate Roots

- Aggregate = group of related - depended classes
- top-level class = Aggregate Root
- Rule (Eric Evans)
- classes in an aggregate may call on each other
- classes in an aggregate may call an aggregate root of another aggregate even if it is in another bounded context.
- The idea is that no class in a bounded context may leverage a class in another bounded context without going through the root of an aggregate.
- *encapsulation*

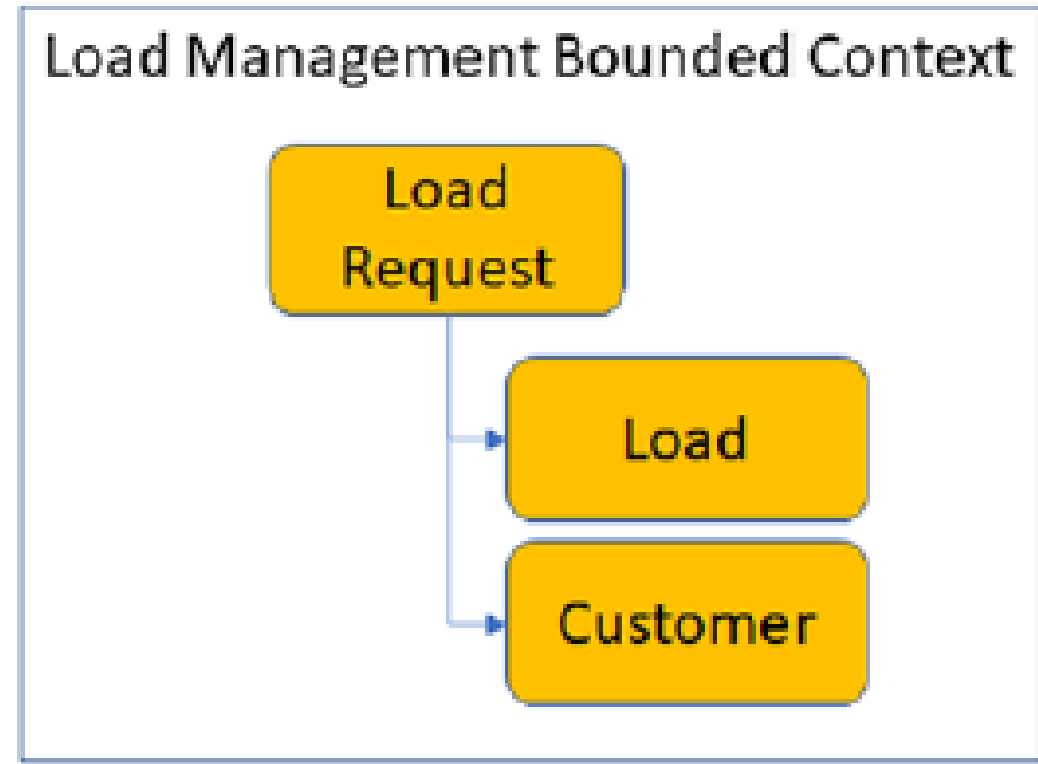
Example of aggregates

- Each aggregate, in the separate bounded contexts, has its own Employee class. This duplication of the Employee class allows employee functionality to be separate and not become a God class.
- Going only through the root allows for business rules to be in place to protect the integrity of the employee data and other related classes.
- Another example is that changes to an employee's benefits should not change their direct deposit information.



SPLAT: Load Request

- how the developers at Code Whiz analyze and create code



Event Storming

- Alberto Brandolini originally invented Event Storming to help identify aggregates in DDD.
- software developers learn from **subject matter experts**, known as **domain experts**, how the industry operates, and how that company operates
- help everyone involved gain a common understanding of how things currently work and to show processes that need to change



Setup - Color Coding

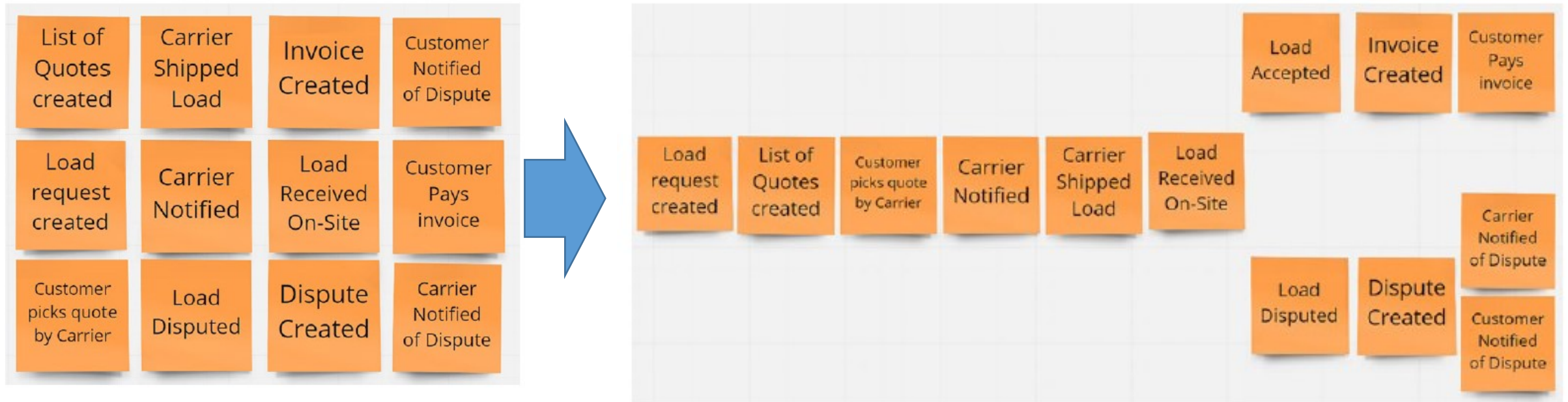
- Event Storming is best done on large walls using paper rolls, sticky notes, and pens. Since many people can work together, provide enough space for the sticky notes when working with multiple subdomains.
- Whatever color scheme you choose, make a legend
 - Orange – Domain Events (Business Processes)
 - Blue – Actions/Commands
 - Yellow – Actors (cause domain events or initiate action)
 - Purple – Policy (special business rule)
 - Green – Aggregate (note where domain object is)
 - Red – Questions (pending)





The Meeting

1. Note domain events (orange) by domain experts
2. Order of events (remove duplicates)

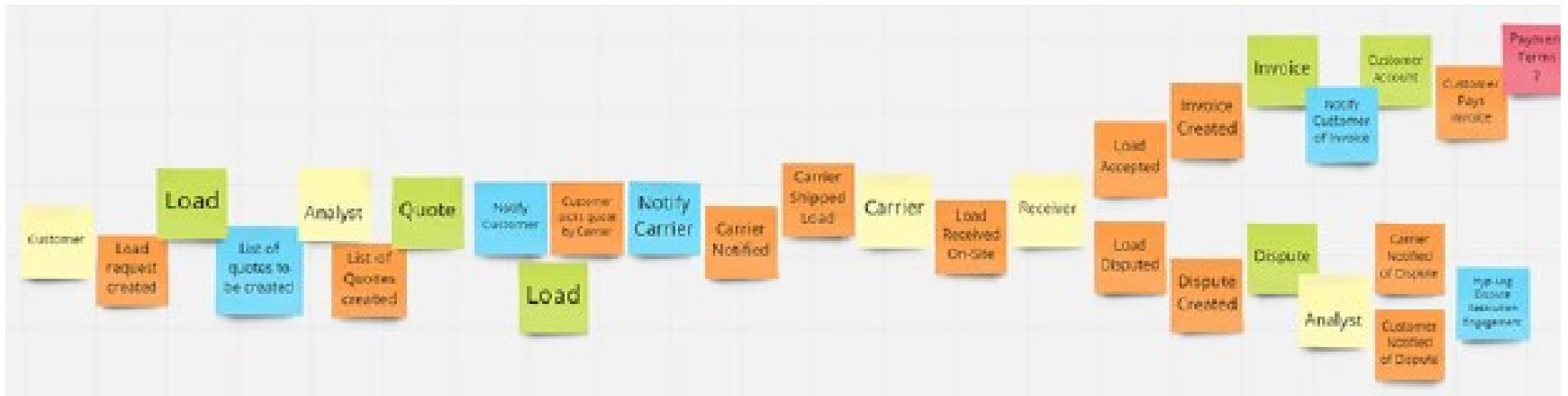




Add details

3. Add the actions/commands, aggregates, actors, and policies to their relative places showing a better picture of how business processes operate

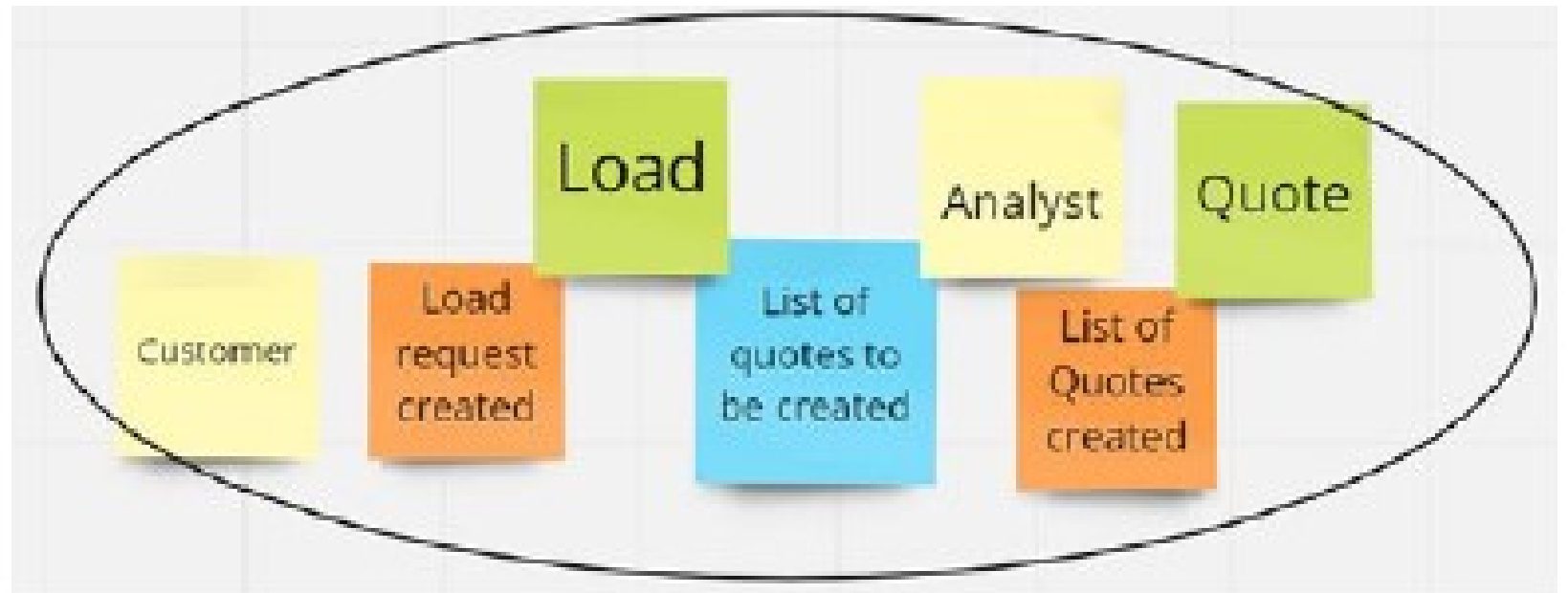
- Load, Customer, Carrier, and Invoice can have copies applied to differing contexts





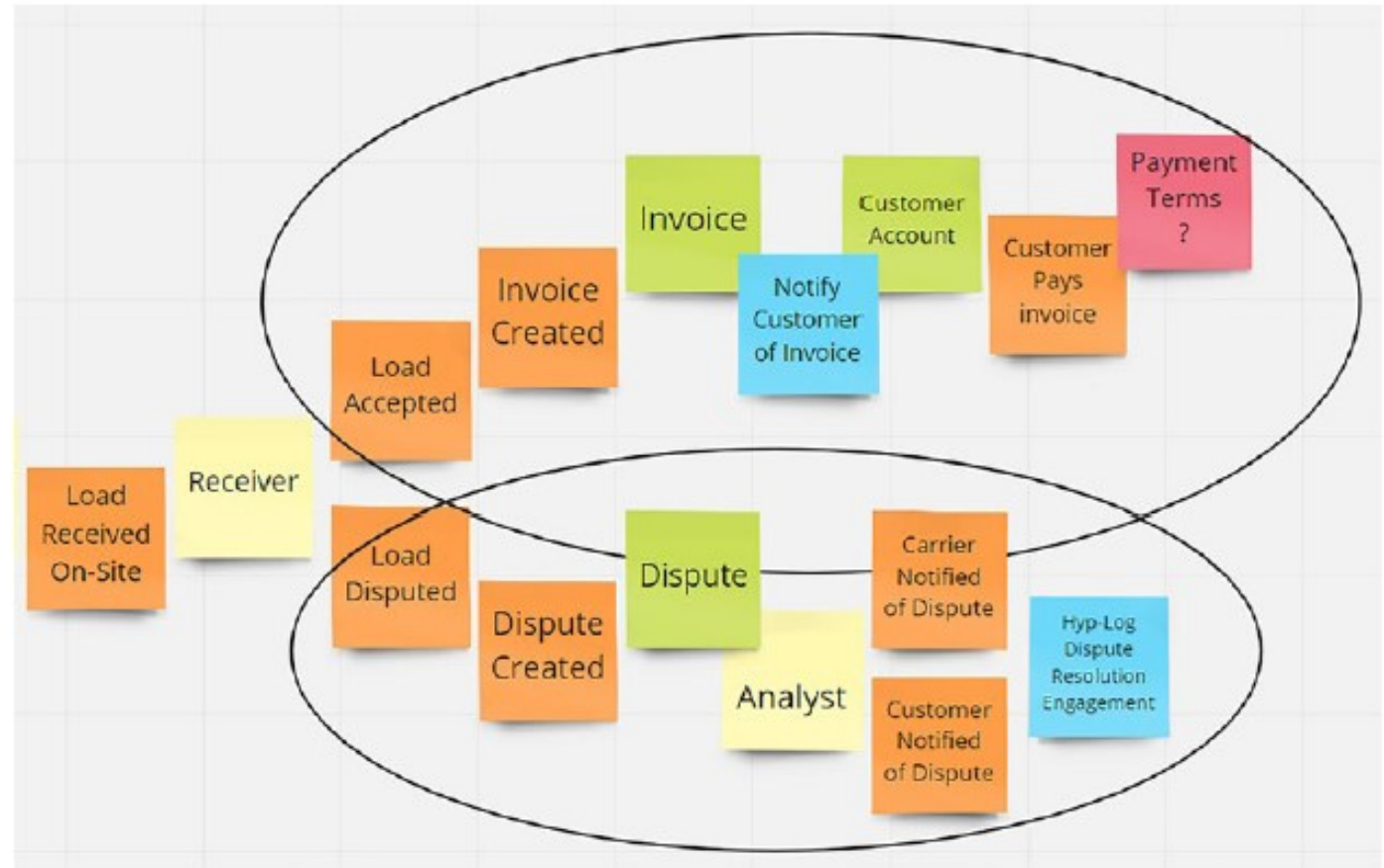
New context - Load Request Created

- the context for Load Requests and Quoting both rely on the Load aggregate.
- This reliance could be a case for what DDD calls a “Shared Kernel” relationship because different bounded contexts share code.



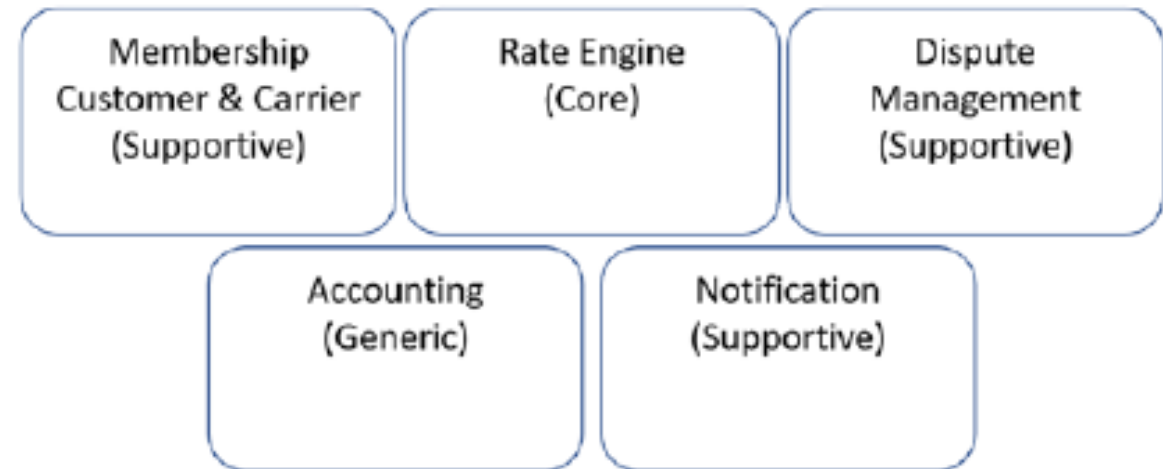
Invoicing and Dispute Management

- The code for invoicing should be completely separate from the code for handling disputes
- (Bounded Contexts)



Domains

- how can developers understand the worth a bounded context brings to a company?
- The developers group the bounded contexts into subdomains:
- Understanding the value of a subdomain that a bounded context fulfills helps **determine the cost of code change**.



Domain Models

- A domain model is a set of codes created to solve problems in their related subdomains.
- Domain models should focus on **behavior** instead of the state of an object. During the Event Storming sessions, pay close attention to the **verbs** in the UL

Decomposition

- The monolith has been detailed on the wall during the Event Storming sessions
- Key: Rate engine
- Hyp-Log analyst is responsible for the manual process of retrieving distance information to apply to the quotes.
 - Apply code changes than hire more people
- confirm the code is far from clean and is, in fact, a BBoM (Big Ball of Mud)
- Because of the amount of refactoring it would take to meet the customer's demands, the lead architect decides that retrieving distance information can be done using a microservice

Searching for microservices

- Using microservices will allow:
- the existing code to have minimal changes,
- keep the manual process intact, and
- have this feature deployed independently of other changes
- map information providers to be added without changing the monolith.
- having a microservice handle invoices

Becoming a Microservice

- Microservices should be considered independent applications free to evolve with little to no dependencies on other parts of a system.
- maintained or owned by a single team of developers.
- require networking for connectivity for either direct RPC-based calls or messaging-based communication.
- time-to-market factor
- adding microservices to solve their main pain points, the application will be able to scale easier over time and handle additional code projects in the future

Creating the first microservice

- solve the issue of retrieving map information from Google
- RPC-style communication between the monolith and the Web API endpoint.

First Microservice

- Microservices are more about software architecture to solve problems for the business than it is about code
- For each pickup and drop-off location, Annie, the analyst for Hyp-Log, gets the
- distance and enters the value in the SPLAT program.
- For Annie, she retrieves the distance information manually.
- However, this is time-consuming and impacts the generation of quotes.
- When looking at the business process to create quotes, where does a call to a microservice make sense?
- Before we can answer that question, we need to look at ways microservices communicate

Interprocess Communication

- IPC
- Direct: microservice directly using Remote Procedure Call (RPC)-style communication
- asynchronous messaging: messaging system like [MassTransit](#)
- Since microservices communicate over a network, there is inherent latency to the execution of a business process
- Using RPC-style communication for microservices has another drawback. The caller must know the network address of the microservices.

API First Design

- With Web API, the interfaces are only available via web-based communication protocols, usually Hypertext Transport Protocol (HTTP).
- We will build the API first.
- what this microservice will offer functionally.
- What is the shape/format of the request?
- Does the request use JSON, XML, or a binary format?
- Should the request use REST or gRPC?
- And what about the response back to the caller?

Transport Mechanisms

- REST and gRPC
- Roy Fielding created Representational State Transfer (REST) in 2000
- REST has operations called HTTP verbs. Here is a list of the most common:
 - GET – used for the retrieval of data
 - POST – used for the creation of data
 - PUT – used for updating data
 - DELETE – used for the deletion of data

gRPC

- “gRPC Remote Procedure Calls”(gRPC).
- Google created gRPC for faster communication with distributed systems by using a binary protocol called Protocol Buffers.
- Like REST, gRPC is language agnostic, so it can be used where the microservice and caller are using different programming languages.
- Unlike REST, gRPC is type specific and uses Protocol Buffers to serialize and deserialize data.
- **In gRPC, you must know the type definitions at design time for both parties to understand how to manage the data**

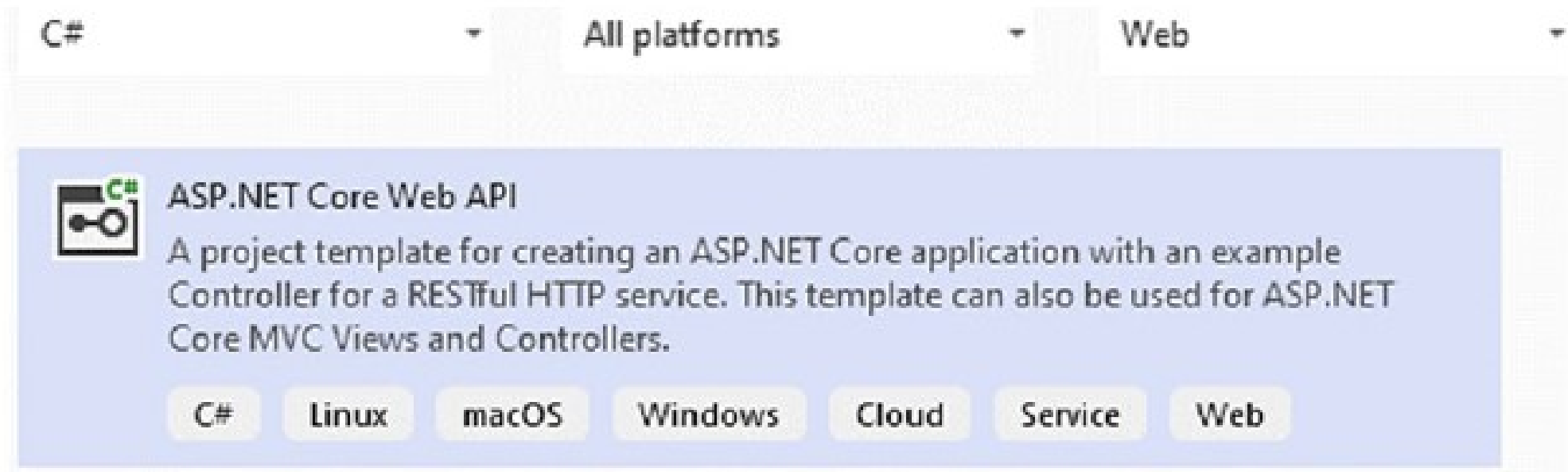
gRPC (2) !

- gRPC uses the HTTP/2 protocol
- If you know the microservices are public facing, either on the Internet or on a network deemed public, REST is the most likely chosen option.
- internal code such as a monolith or another microservice, you may consider gRPC.
- Using gRPC will help lower latency and is worth the extra time to set up

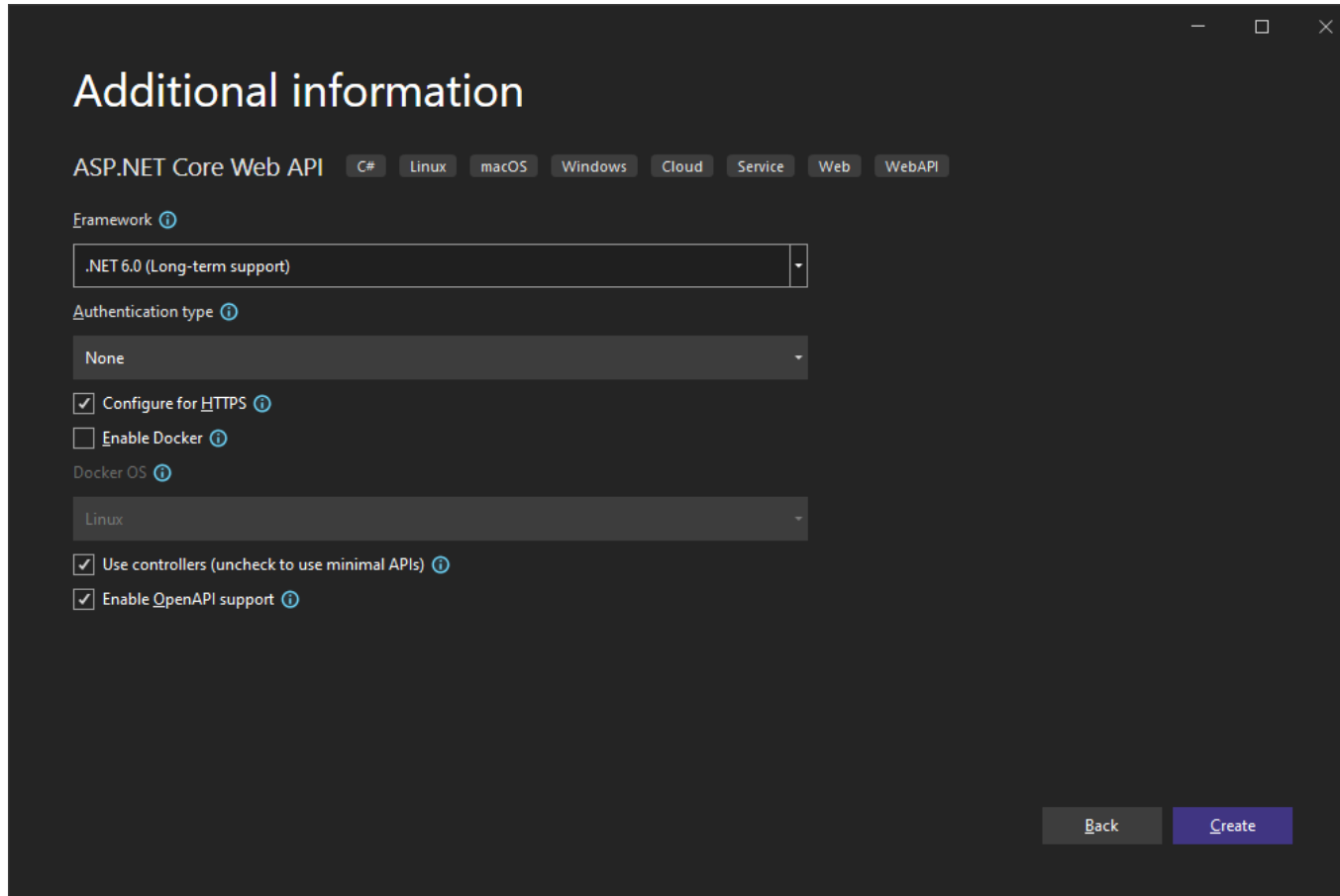
REST for starter

- we will start with REST for its simplicity and then show how to integrate gRPC into our project
- What should the request and response look like, JSON or XML? A brief look at the Google Distance API specification shows they are returning data with JSON.

File – New – Project



.NET 6 + OpenAPI config



Additional information

ASP.NET Core Web API C# Linux macOS Windows Cloud Service Web WebAPI

Framework ⓘ
[.NET 6.0 (Long-term support)]

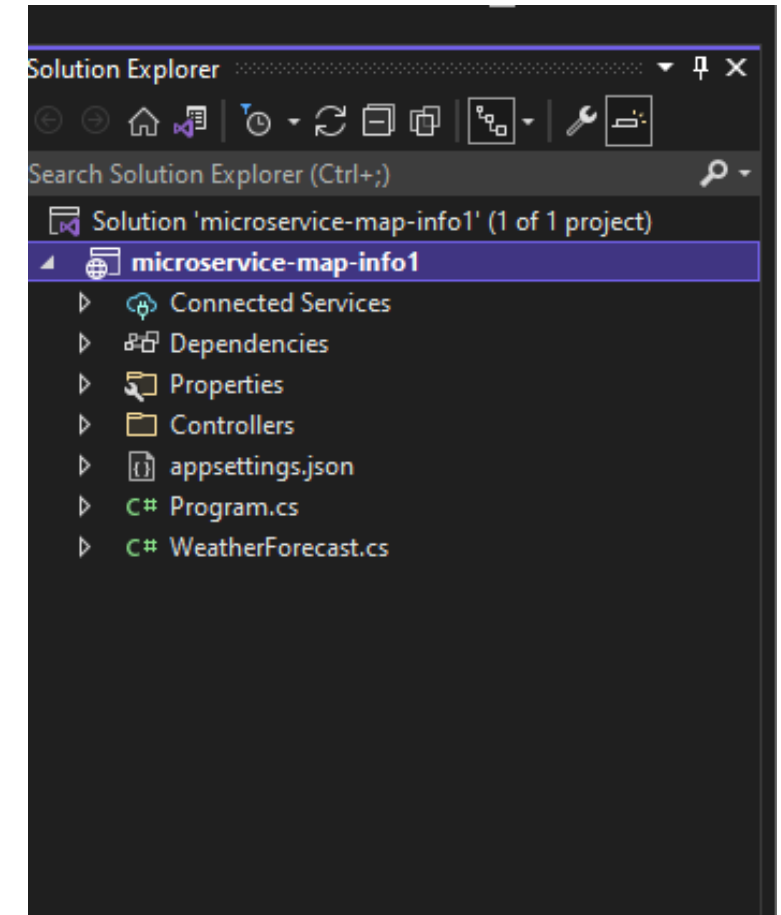
Authentication type ⓘ
[None]

☒ Configure for HTTPS ⓘ
☐ Enable Docker ⓘ

Docker OS ⓘ
[Linux]

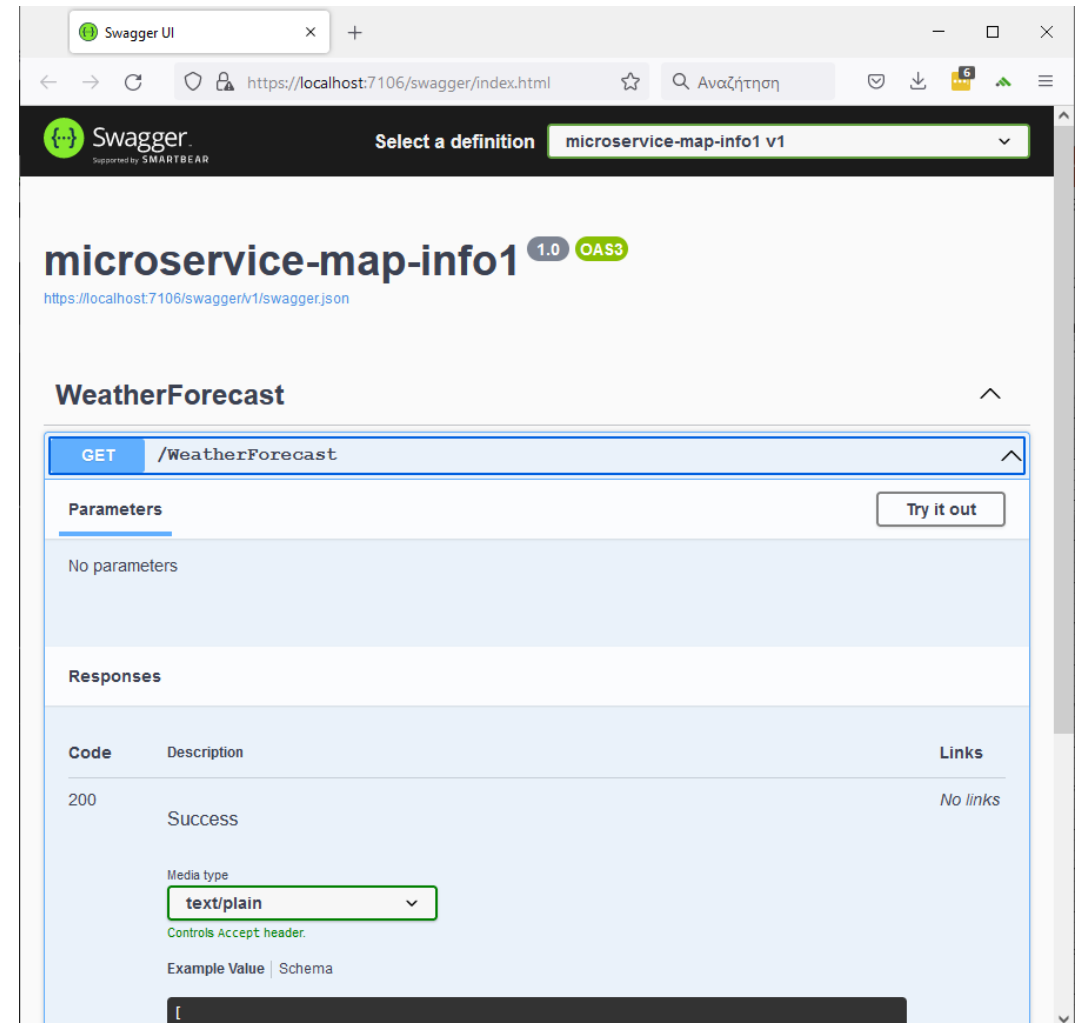
☒ Use controllers (unchecked to use minimal APIs) ⓘ
☒ Enable OpenAPI support ⓘ

Back Create



WeatherForecast

- After creating the project, you will see the files created from the template. It created a controller called WeatherForecastController and a class called WeatherForecast



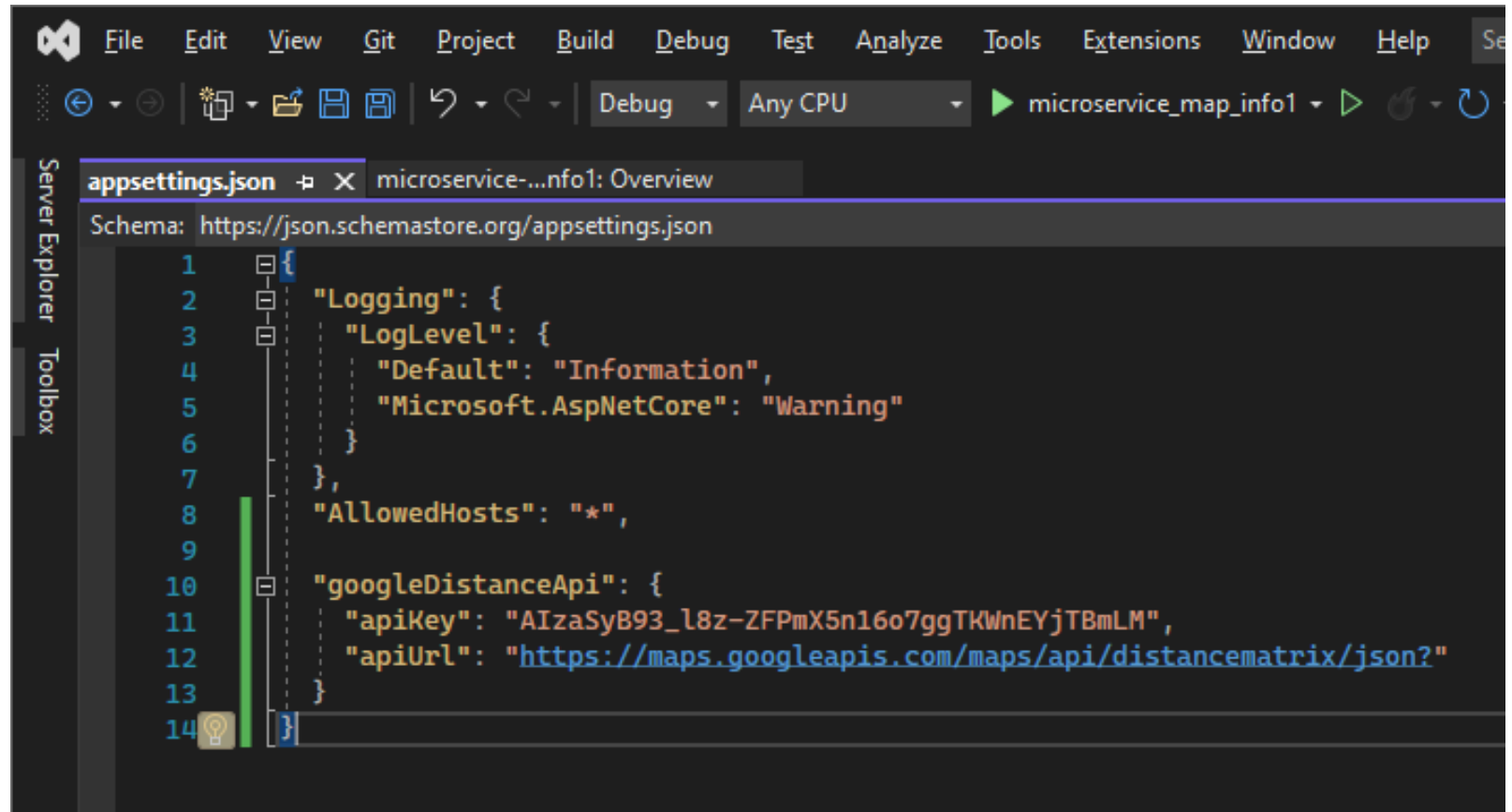
Contacting Google's Distance API

- calling Google's Distance API is not free
- reply with fake data.
 - Using fake data will at least get you up and running until you are in a position to pay for a service that gives the required information
- Start by going to <https://developers.google.com/maps/documentation/distance-matrix/>
- [start](#). You will need a Google account to obtain an API key. To
- get an API key, follow the instructions at <https://developers.google.com/maps/documentation/distance-matrix/get-api-key>
- it is best not to store any secrets or API keys in configuration files. It is very easy to deploy secrets to production accidentally. Instead, consider using the Secret Manager tool (<https://docs.microsoft.com/en-us/aspnet/core/security/app-secrets>).

App Settings

- open the file `appSettings.json`

```
"googleDistanceApi": {  
  "apiKey": "Enter your API Key here",  
  "apiUrl":  
    "https://maps.googleapis.com/maps/api/distancematrix/json?"  
}
```

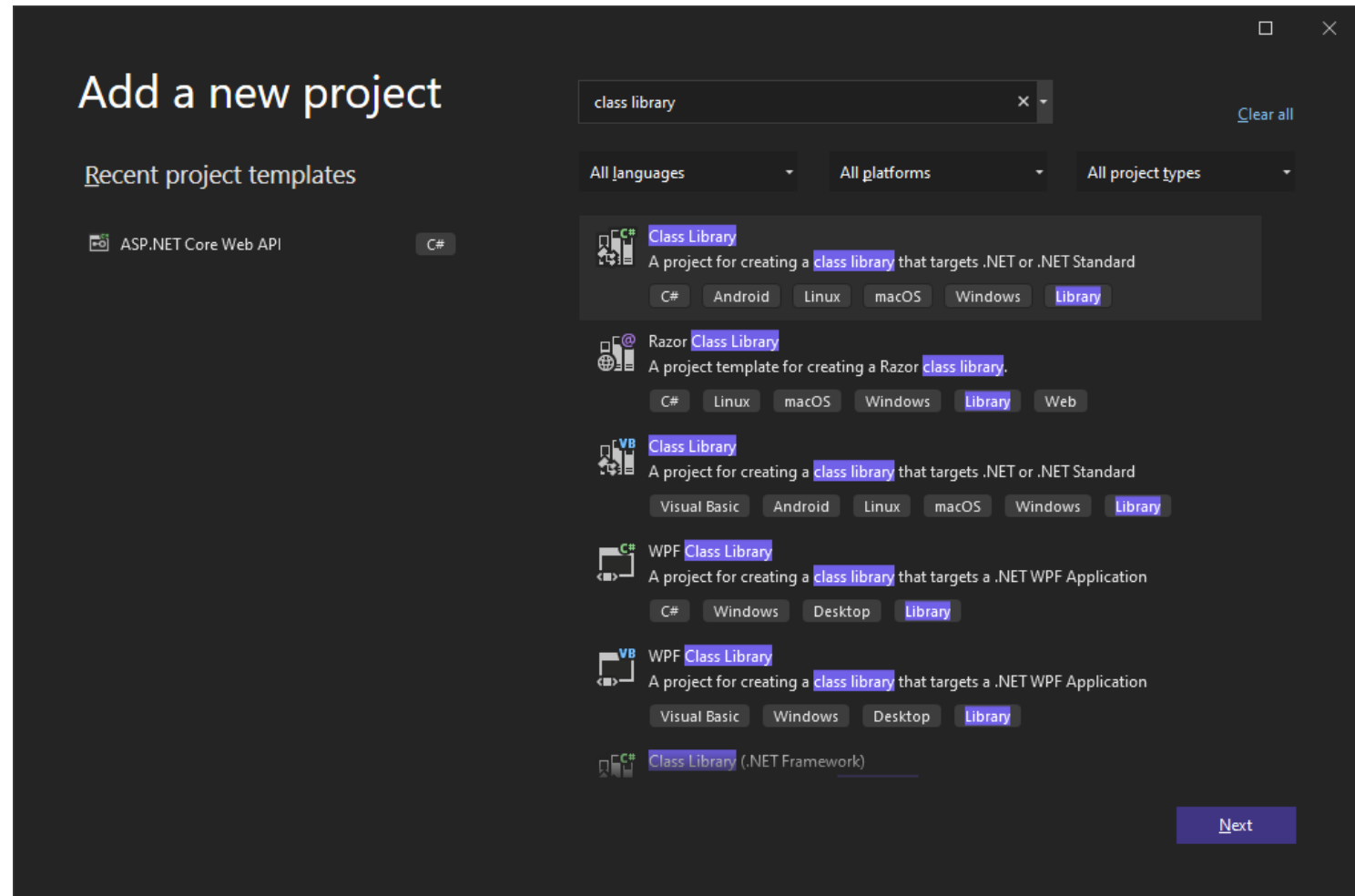


The screenshot shows the Visual Studio Code editor with the `appSettings.json` file open. The file content is as follows:

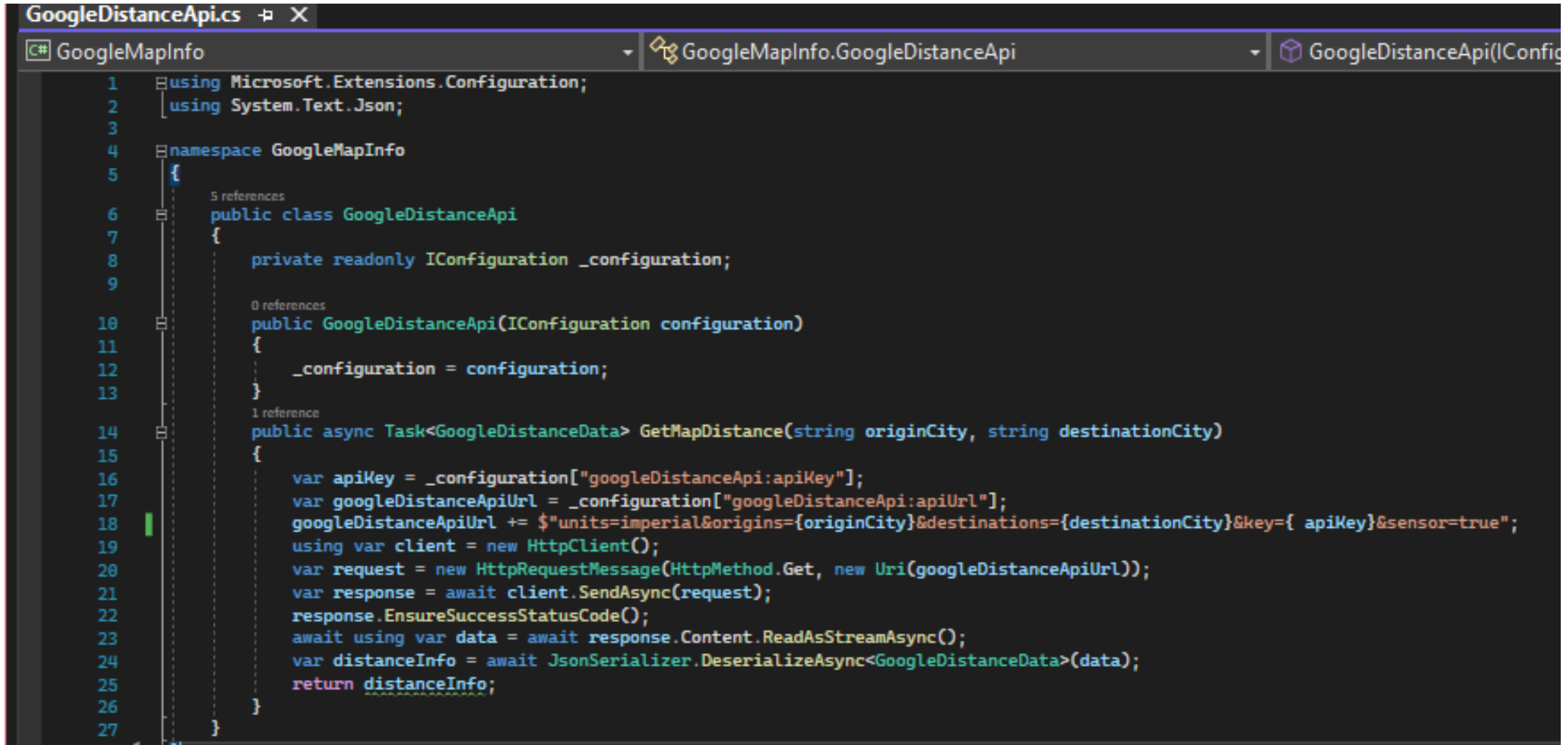
```
1  {  
2    "Logging": {  
3      "LogLevel": {  
4        "Default": "Information",  
5        "Microsoft.AspNetCore": "Warning"  
6      }  
7    },  
8    "AllowedHosts": "*",  
9  
10   "googleDistanceApi": {  
11     "apiKey": "AIzaSyB93_l8z-ZFPmX5n16o7ggTKWnEYjTBmLM",  
12     "apiUrl": "https://maps.googleapis.com/maps/api/distancematrix/json?"  
13   }  
14 }
```

The interface includes a menu bar (File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help), a toolbar with icons for navigation and development, and a sidebar with 'Server Explorer' and 'Toolbox'. The status bar at the bottom shows 'microservice_map_info1' and a 'Debug' button.

New Project Class library GoogleMapInfo



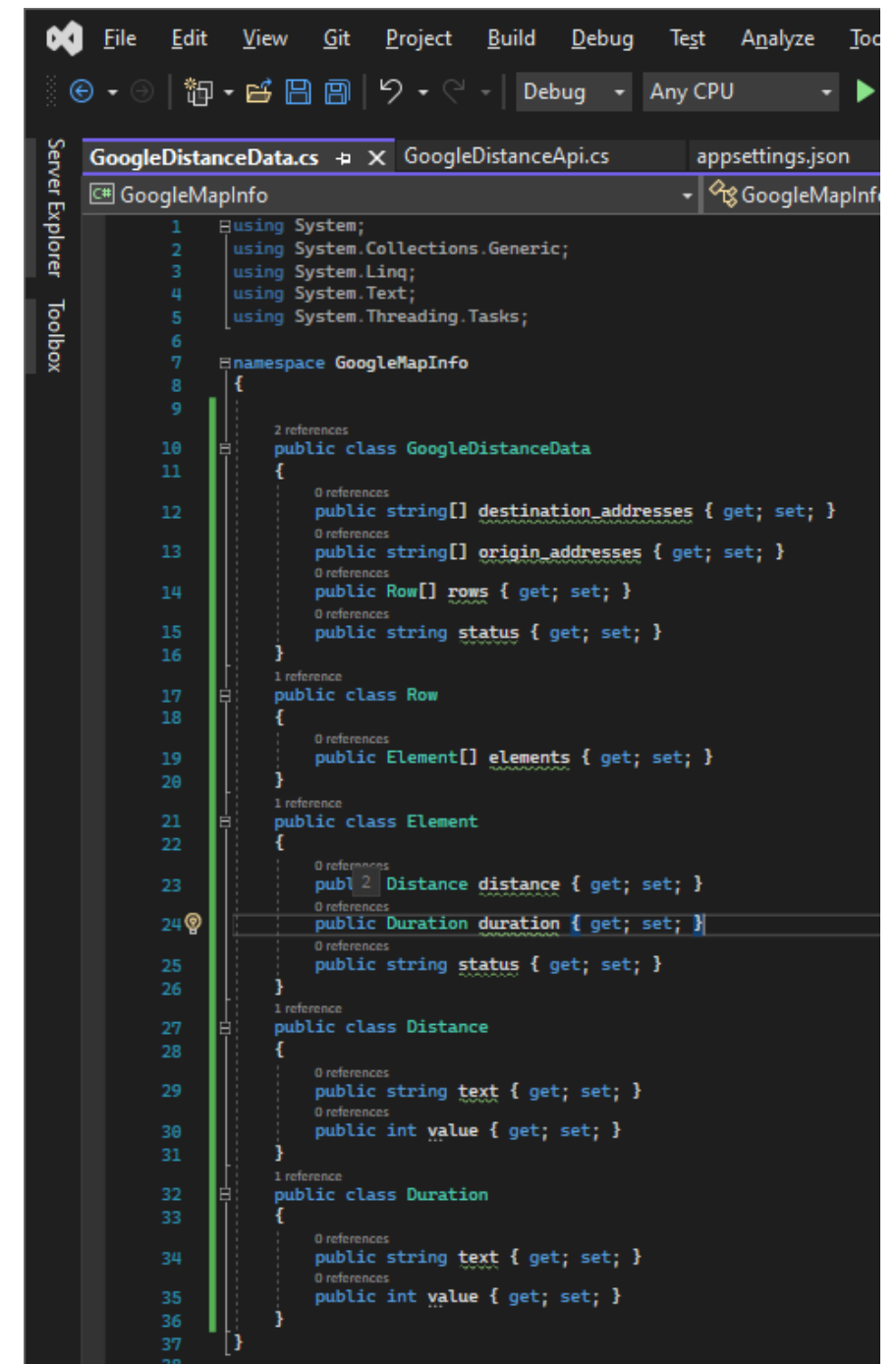
New class GoogleDistanceApi



```
GoogleDistanceApi.cs
C# GoogleMapInfo GoogleMapInfo.GoogleDistanceApi GoogleDistanceApi(IConfiguration)

1 using Microsoft.Extensions.Configuration;
2 using System.Text.Json;
3
4 namespace GoogleMapInfo
5 {
6     5 references
7     public class GoogleDistanceApi
8     {
9         private readonly IConfiguration _configuration;
10
11         0 references
12         public GoogleDistanceApi(IConfiguration configuration)
13         {
14             _configuration = configuration;
15         }
16
17         1 reference
18         public async Task<GoogleDistanceData> GetMapDistance(string originCity, string destinationCity)
19         {
20             var apiKey = _configuration["googleDistanceApi:apiKey"];
21             var googleDistanceApiUrl = _configuration["googleDistanceApi:apiUrl"];
22             googleDistanceApiUrl += $"units=imperial&origins={originCity}&destinations={destinationCity}&key={ apiKey}&sensor=true";
23             using var client = new HttpClient();
24             var request = new HttpRequestMessage(HttpMethod.Get, new Uri(googleDistanceApiUrl));
25             var response = await client.SendAsync(request);
26             response.EnsureSuccessStatusCode();
27             await using var data = await response.Content.ReadAsStreamAsync();
28             var distanceInfo = await JsonSerializer.DeserializeAsync<GoogleDistanceData>(data);
29             return distanceInfo;
30         }
31     }
32 }
```

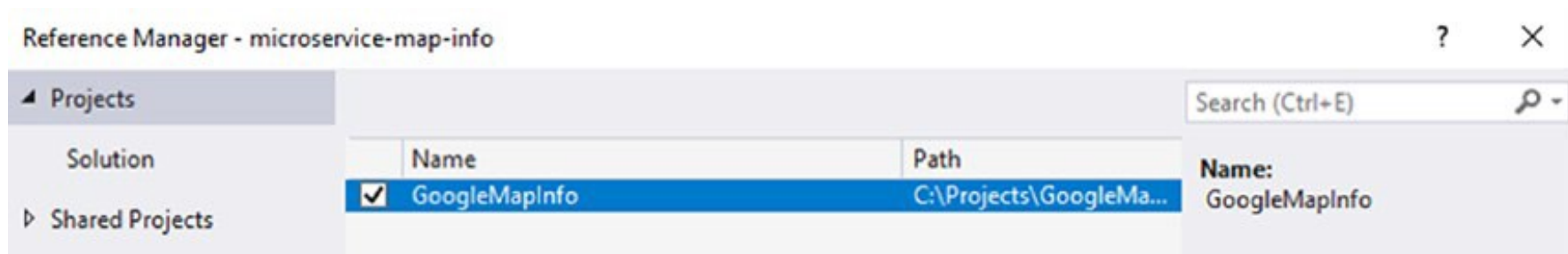
Google Distance Data class



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace GoogleMapInfo
8 {
9
10     2 references
11     public class GoogleDistanceData
12     {
13         0 references
14         public string[] destination_addresses { get; set; }
15         0 references
16         public string[] origin_addresses { get; set; }
17         0 references
18         public Row[] rows { get; set; }
19         0 references
20         public string status { get; set; }
21     }
22
23     1 reference
24     public class Row
25     {
26         0 references
27         public Element[] elements { get; set; }
28     }
29
30     1 reference
31     public class Element
32     {
33         0 references
34         public Distance distance { get; set; }
35         0 references
36         public Duration duration { get; set; }
37         0 references
38         public string status { get; set; }
39     }
40
41     1 reference
42     public class Distance
43     {
44         0 references
45         public string text { get; set; }
46         0 references
47         public int value { get; set; }
48     }
49
50     1 reference
51     public class Duration
52     {
53         0 references
54         public string text { get; set; }
55         0 references
56         public int value { get; set; }
57     }
58 }
```

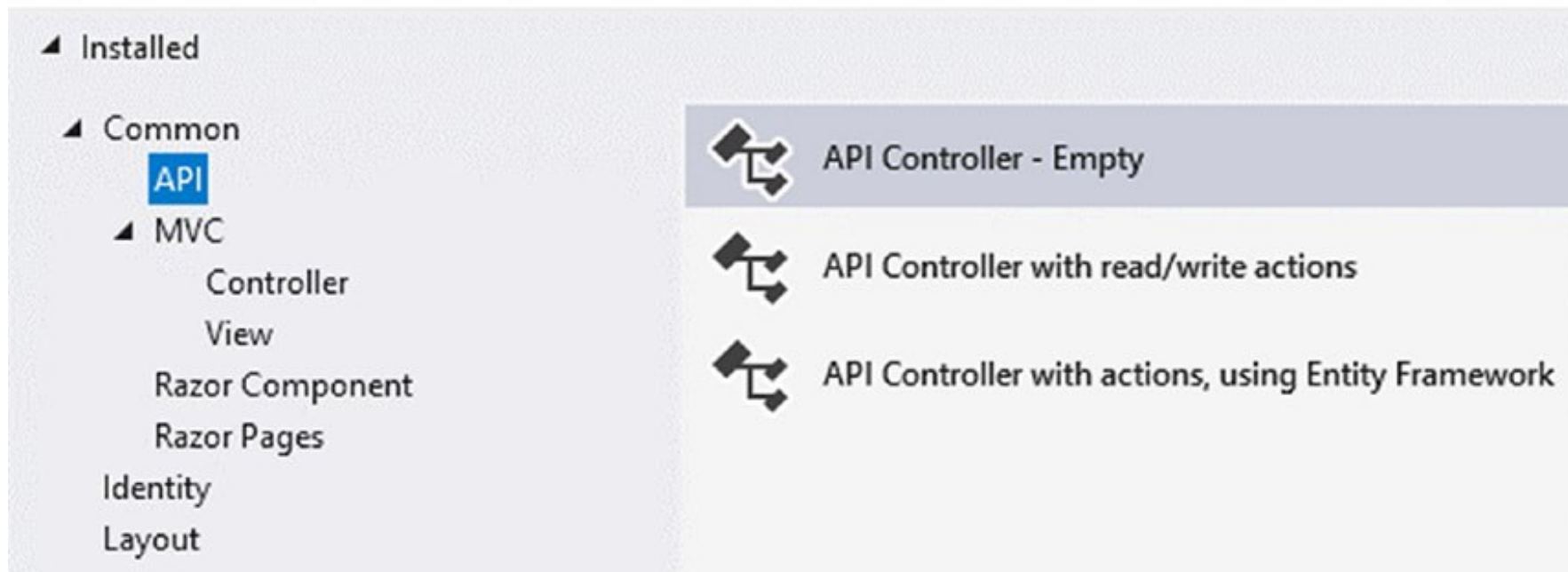
Add project reference

- With the code to call Google's Distance API in place, now you will reference this new project by the main project.



Map Info Controller

Add New Scaffolded Item



Startup + MapInfoController

- configure ASP.NET Core to know how to instantiate the MapInfoController to be ready for use

- Add program.cs

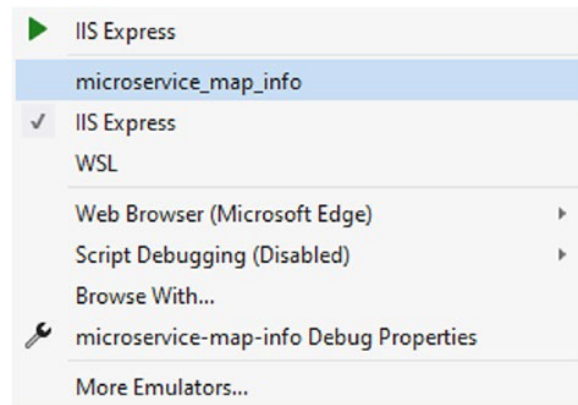
```
builder.Services.AddTransient<GoogleDistanceApi>();
```

- Add startup.cs

```
• public void ConfigureServices(IServiceCollection services)
•     {
•         services.AddTransient<GoogleDistanceApi>();
•         services.AddControllers();
•     }
```

Build succeed How to test?

- launchSettings.json file located in the Properties folder
- Choose swagger or iis express



```
https://json.schemastore.org/launchsettings.json
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:23996",
      "sslPort": 44335
    }
  },
  "profiles": {
    "microservice_map_info1": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "https://localhost:7106;http://localhost:5109",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "swagger",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```


Sample Calling the microservice

The screenshot displays the Swagger UI interface in a web browser. The address bar shows the URL `https://localhost:44335/swagger/index.html`. The main content area shows the details of a specific API endpoint. The request URL is `https://localhost:44335/MapInfo/GetDistance?originCity=athens%2Cgreece&destinationCity=patras%2Cgreece`. The server response is a 200 status code with a JSON body. The response body contains the following JSON structure:

```
{
  "destination_addresses": [
    "Patras, Greece"
  ],
  "origin_addresses": [
    "Athens, Greece"
  ],
  "rows": [
    {
      "elements": [
        {
          "distance": {
            "text": "131 mi",
            "value": 210656
          },
          "duration": {
            "text": "2 hours 27 mins",
            "value": 8791
          },
          "status": "OK"
        }
      ]
    }
  ],
  "status": "OK"
}
```

The response headers are also displayed:

```
content-type: application/json; charset=utf-8
date: Sat, 02 Apr 2022 17:54:12 GMT
server: Microsoft-IIS/10.0
x-firefox-spy: h2
x-powered-by: ASP.NET
```

At the bottom, the 'Responses' section shows a table with a single entry for status 200, labeled 'Success'. The media type is set to 'text/plain'.

Code	Description	Links
200	Success	No links

Media type:

Controls: Accept header

Example Value | Schema

Swagger UI

https://localhost:44358/swagger/index.html

Αναζήτηση

Swagger.
Supported by SMARTBEAR

Select a definition

My microservice for map information.

My map API v1 OAS3
[/swagger/v1/swagger.json](#)

MapInfo

GET /MapInfo

GET /MapInfo/GetDistance

Parameters

Cancel

Name	Description
originCity string (query)	<input type="text" value="athens"/>
destinationCity string (query)	<input type="text" value="patras"/>

Execute

Clear

Responses

Leveraging gRPC

- Up to now REST and JSON
- binary transport mechanism for faster communication
- gRPC, communication may be faster depending on the type of data sent.
- For small, simple message payloads, you may be better off with JSON.
- gRPC uses HTTP/2 for transport and Protocol Buffers for the interface definition language (IDL).

Leveraging gRPC

- With JSON, converting an instance of a class to a stream of text relies on many rules and uses reflection
- With gRPC conversion effort is not as straightforward
- Developers must use customizations even on simplistic data types.
- In this section, you will modify the microservice to leverage gRPC to learn how to trim latency with the network calls.

gRPC uses an IDL definition file, or “contract,”

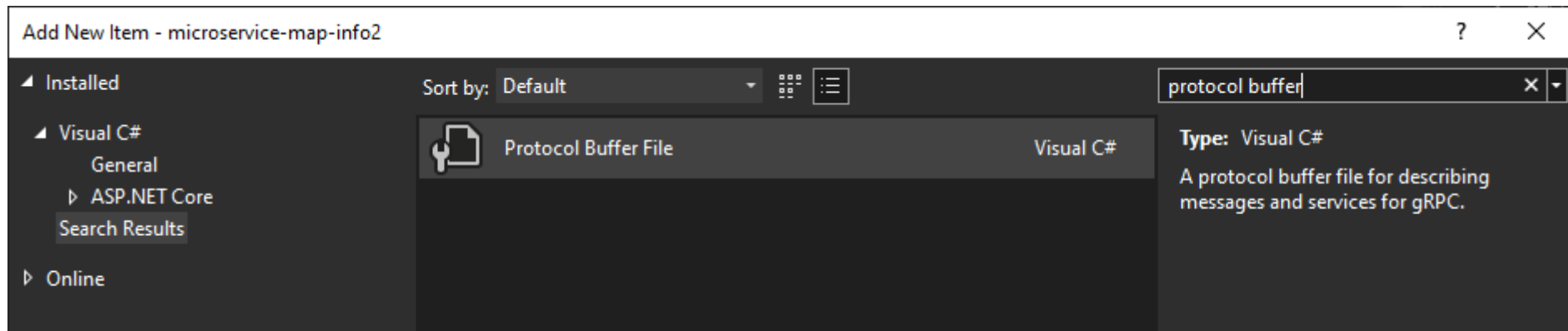
```
syntax = "proto3";  
service SubscribeToEvent {  
  rpc Subscribe (EventSubscription) returns (Subscribed)  
}  
message EventSubscription {  
  int32 eventId = 123;  
  string name = foo;  
}  
message Subscribed {  
  string message = bar;  
}
```

In this example, a service “SubscribeToEvent” is defined with the method “Subscribe.”

The “Subscribe” method takes in a parameter type “EventSubscription,” which returns the type “Subscribed.”

Incorporating gRPC

- add new folders and files and modify the project file.
- New Folder. Name the folder “Protos” & Services
- Right-click the Protos folder and select Add ➤ New Item.
- In the menu, select “Protocol Buffer File.” Name it “distance.proto.”



distance.proto

- `syntax = "proto3";`
- `option csharp_namespace = "microservice_map_info.Protos";`
- `package distance;`
- `service DistanceInfo {`
- `rpc GetDistance (Cities) returns (DistanceData);`
- `}`
- `message Cities {`
- `string originCity = 1;`
- `string destinationCity = 2;`
- `}`
- `message DistanceData {`
- `string miles = 1;`
- `}`

Services/DistanceInfoService.cs

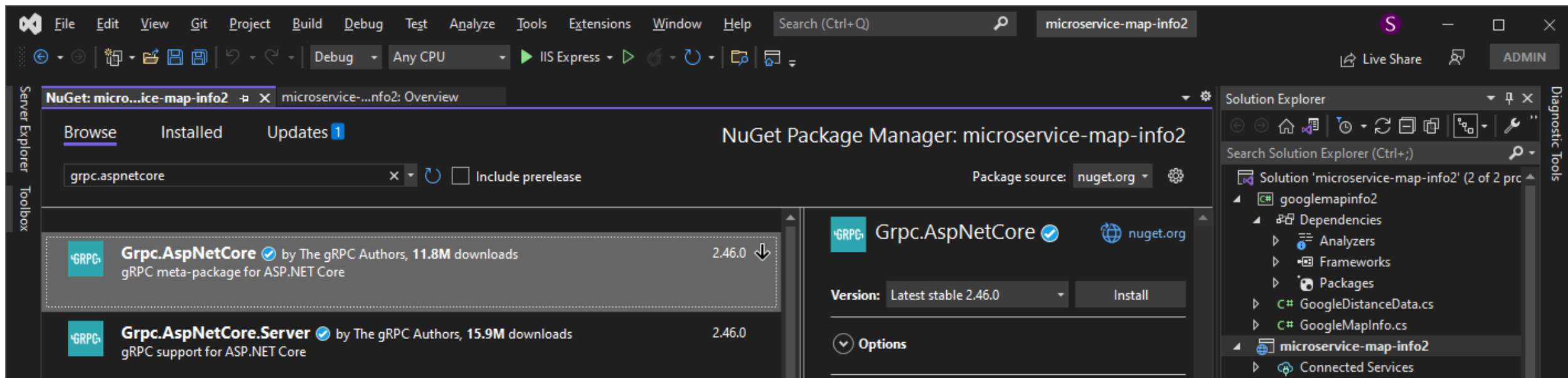
```
public class DistanceInfoService : DistanceInfo.DistanceInfoBase
{
    private readonly ILogger<DistanceInfoService> _logger;
    private readonly GoogleDistanceApi _googleDistanceApi;
    public DistanceInfoService(ILogger<DistanceInfoService> logger,
        GoogleDistanceApi googleDistanceApi)
    {
        _logger = logger;
        _googleDistanceApi = googleDistanceApi;
    }
    public override async Task<DistanceData> GetDistance(Cities cities,
        ServerCallContext context)
    {
        var totalMiles = "0";
        var distanceData = await _googleDistanceApi.GetMapDistance(cities.
            OriginCity, cities.DestinationCity);
        foreach (var distanceDataRow in distanceData.rows)
        {
            foreach (var element in distanceDataRow.elements)
            {
                totalMiles = element.distance.text;
            }
        }
        return new DistanceData { Miles = totalMiles };
    }
}
```

you see errors and
it will not compile
yet.

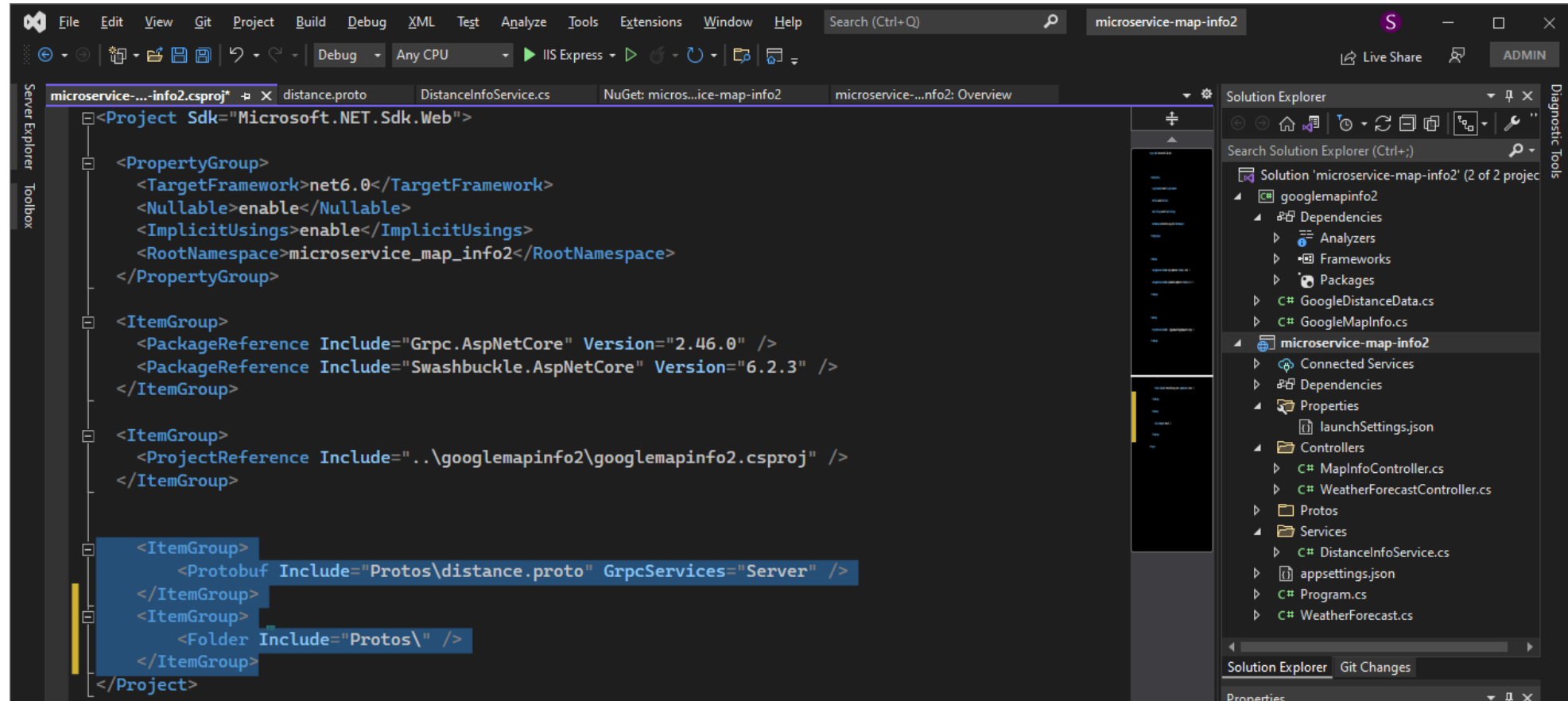
After a couple
more steps, the
solution will
compile.

NuGet Packages

- install the gRPC NuGet package
- Install-Package Grpc.AspNetCore



Modify project (left click!)

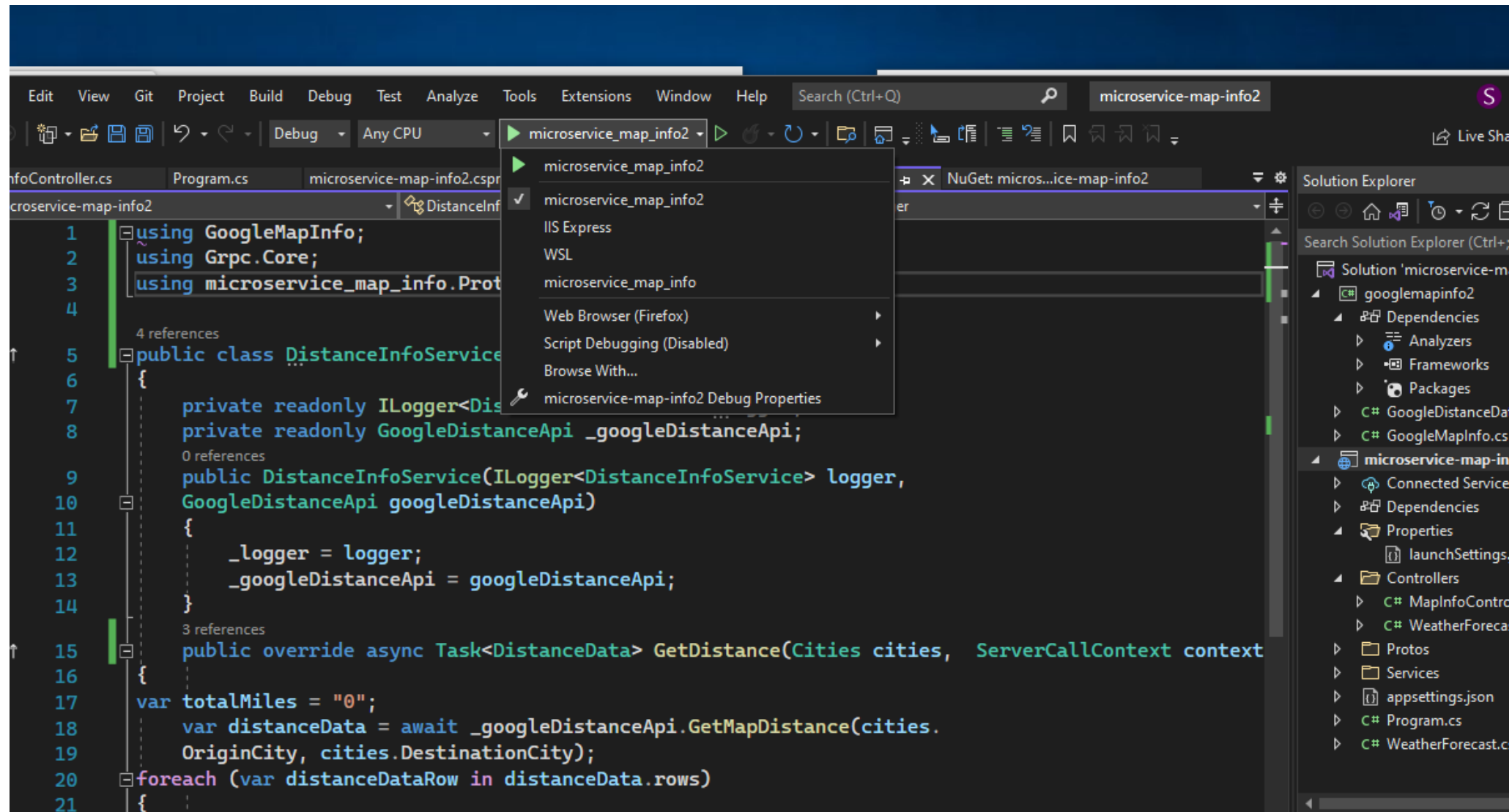


Add in Program.cs

- `builder.Services.AddGrpc();`
- `app.MapGrpcService<DistanceInfoService>();`

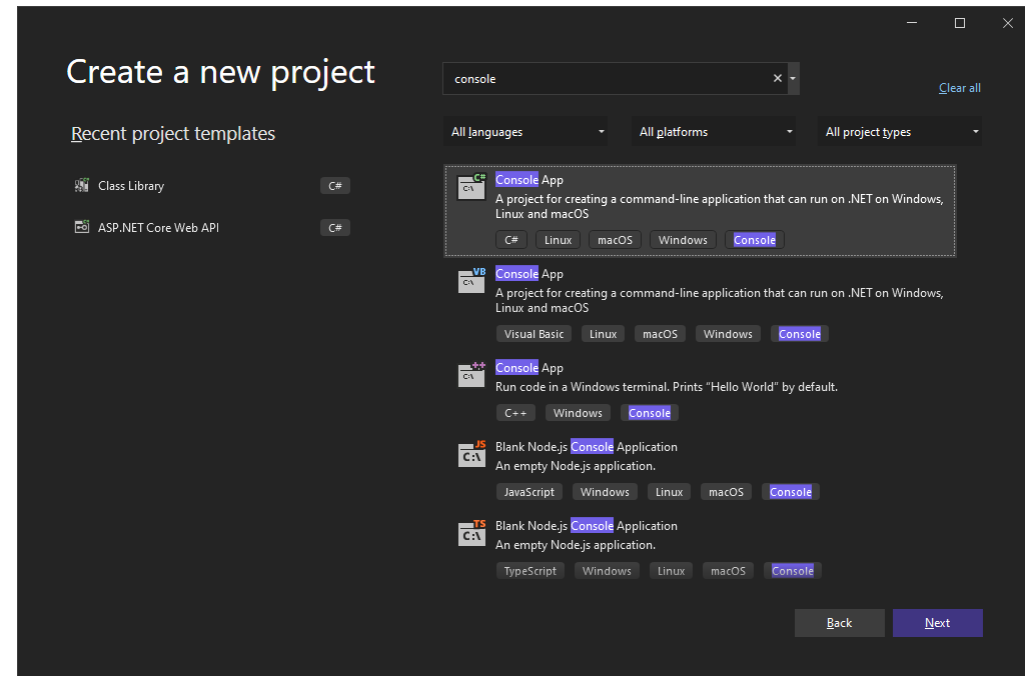
Kestrel

- Now that gRPC has been brought into this project, a restriction arises. gRPC will not
 - run with IIS. So, you must debug and run in production using Kestrel instead of IIS.
-
- Kestrel is a cross-platform web server for ASP.NET Core applications. For more information, go to <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-6.0>.



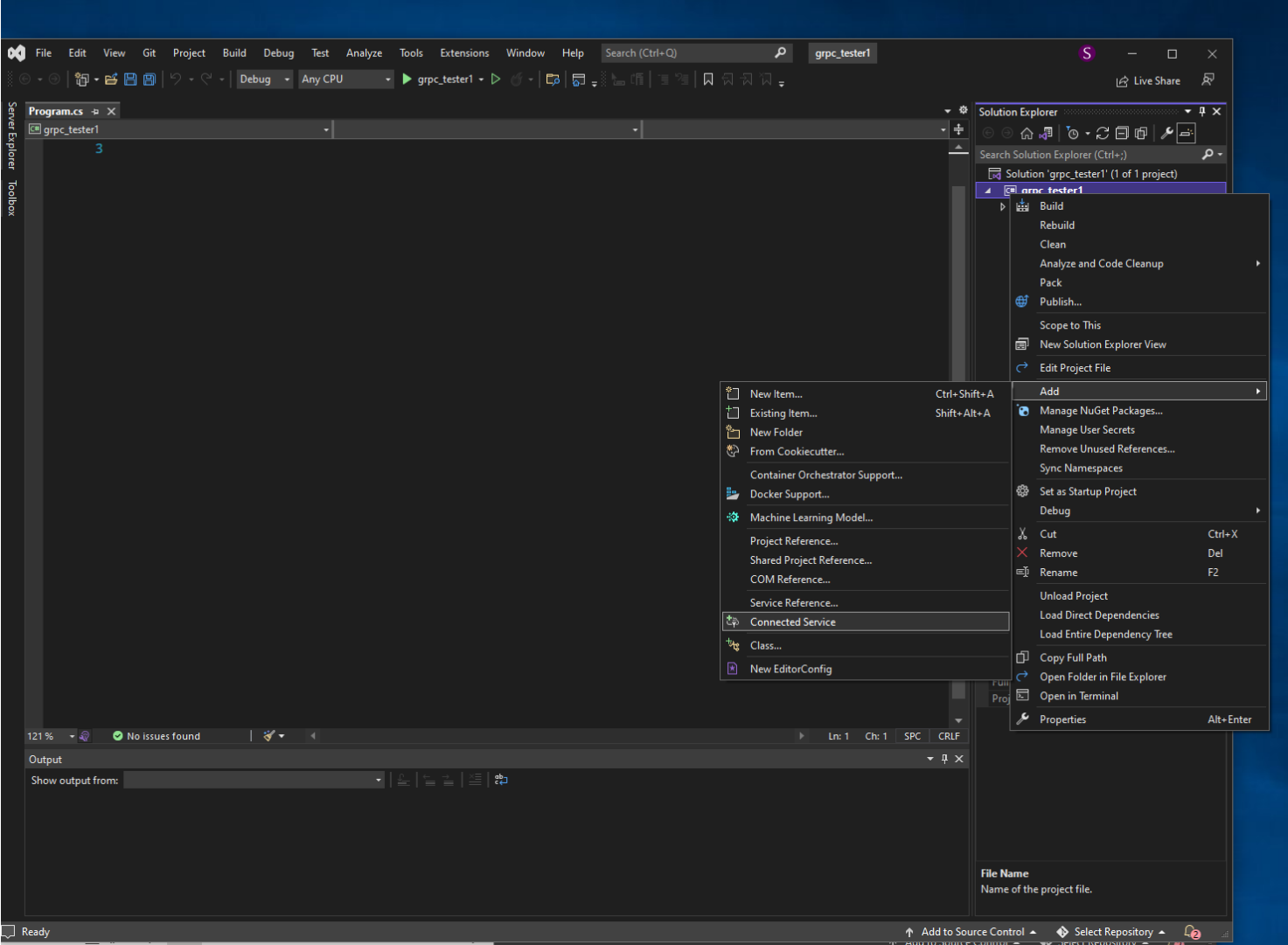
Testing gRPC Endpoint

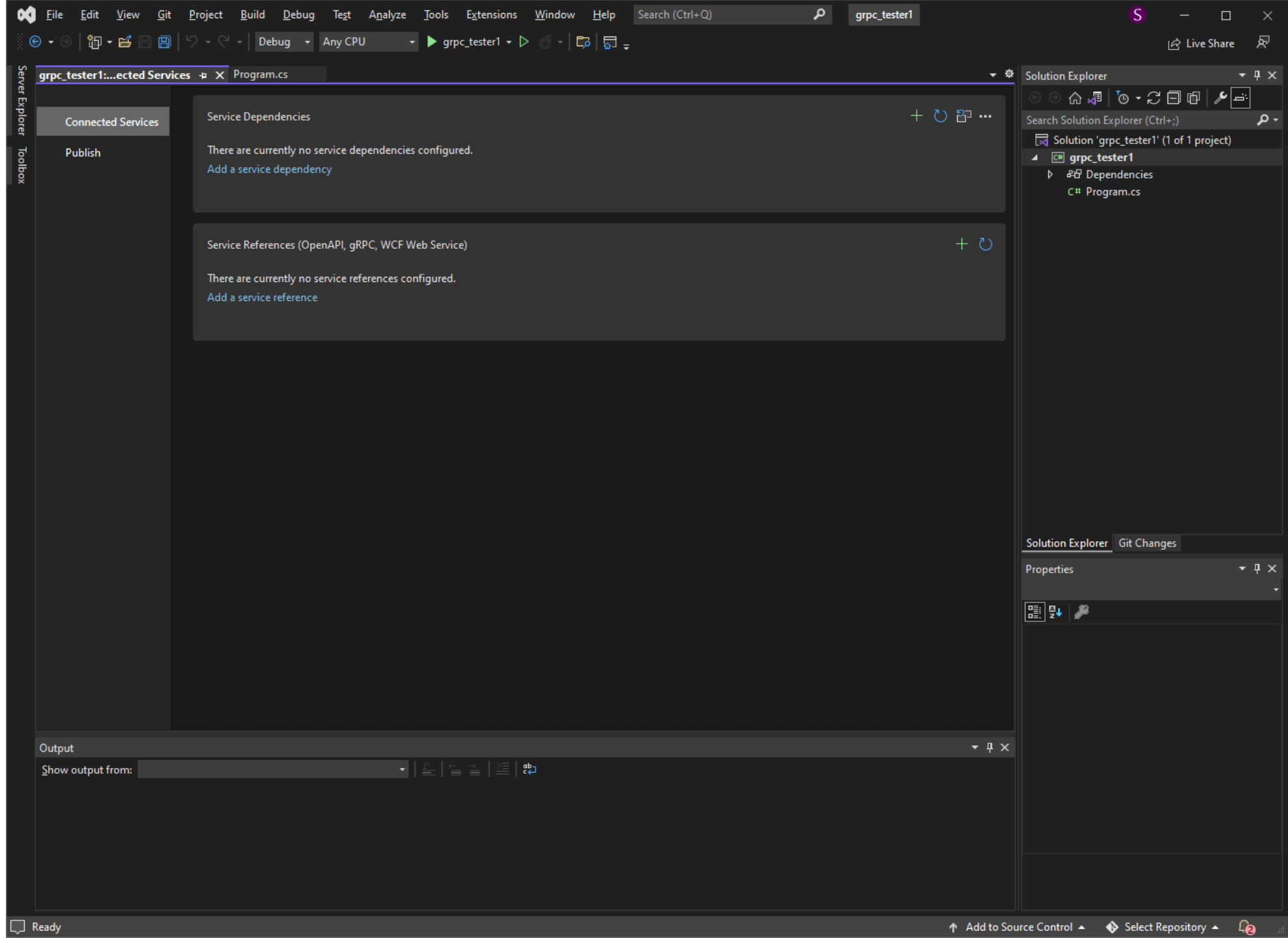
- how can you test the gRPC service endpoint?
- You can test with a simple console application.



Service Reference

- After creating the project, you need to make a Service Reference. Much like adding a project reference, this will add a reference based on the proto file created earlier.
- Right-click the project and select “Add ► Connected Service.”
- When the next window appears, find the “Service References (OpenAPI, gRPC, WCF Web Service)” section and select the plus symbol.







Add service reference

Select a service reference to add to your application



OpenAPI

Consume web services which conform to the OpenAPI Specification



gRPC

Consume and produce web services which conform to the gRPC open source universal RPC framework



WCF Web Service

Add a WCF web service reference to your project.

Back

Next

Finish

Cancel



Add new gRPC service reference

Select a file or URL

☐ File

C:\Users\refoman\Source\Repos\microservice-map-info2\microservice-map-info2\Protos\distance.proto

Browse...

☒ URL

Select the type of class to be generated

Client



Back


Next

Finish

Cancel



Service reference configuration progress

Checking project for required NuGet packages...
Installing NuGet packages to project...
Installing package 'Grpc.Net.ClientFactory' with version '2.40.0'.
Installing package 'Google.Protobuf' with version '3.18.0'.
Installing package 'Grpc.Tools' with version '2.40.0'.
Adding service reference distance to the project...
 Successfully added service reference(s)

☐ Automatically close when succeeded

Back

Next

Close

Cancel

Notice: URL of microservice must match

- Notice the URL is set to `https://localhost:5001`.
- This path needs to match that of the microservice. Looking in the microservice project,
- in the `launchSettings.json` file, you see that when running with Kestrel, for HTTPS, the
- URL is `https://localhost:5001`. Since those two URLs match, we are good for testing.

Program.cs in console

```
// See https://aka.ms/new-console-template for more information
```

```
using Grpc.Net.Client;
```

```
using microservice_map_info.Protos;
```

```
var channel = GrpcChannel.ForAddress(new Uri("https://localhost:7271"));
```

```
var client = new DistanceInfo.DistanceInfoClient(channel);
```

```
var response = await
```

```
client.GetDistanceAsync(new Cities
```

```
{ OriginCity = "Topeka,KS", DestinationCity = "Los Angeles,CA" });
```

```
Console.WriteLine(response.Miles);
```

```
Console.ReadKey();
```

Run from console

- { OriginCity = "Topeka,KS", DestinationCity = "Los Angeles,CA" });
- OriginCity = "Patras", DestinationCity = "Athens"

```
C:\Users\refoman\Source\Rep
```

```
1,555 mi
```

```
C:\Users\refoman\Source
```

```
131 mi
```

Modify Monolith

- the microservice is working and ready to be called
- let's modify the monolith to use it
- This section will alter the monolith to call the microservice to retrieve the distance between two cities

Add services/distanceinfosvc.cs class

- The DistanceInfoSvc class is responsible for calling the microservice and deserializing the results.
- JSON is the data format for the returned distance data.
- The data deserializes into MapDistanceInfo class.
- generate the classes to match JSON data in Visual Studio by capturing the JSON payload in the clipboard

DistanceInfoSvc

- alter the quote service to call the DistanceInfoSvc to retrieve the distance for the quote.
- In the QuoteSvc class, add the following code alterations.

QuoteSvc

- Now we need to alter the quote service to call the DistanceInfoSvc to retrieve the
- distance for the quote. In the QuoteSvc class, add the following code alterations.
- First, alter the constructor to have the DistanceInfoSvc injected in.

```
private readonly IDistanceInfoSvc _distanceInfoSvc;  
public QuoteSvc(IDistanceInfoSvc distanceInfoSvc)  
{  
    _distanceInfoSvc = distanceInfoSvc;  
}
```

Create the quote

- Now alter the method that creates the quote to leverage DistanceInfoSvc class.

```
public async Task<Quote> CreateQuote(string originCity, string destinationCity)
{
    var distanceInfo = await _distanceInfoSvc
        .GetDistanceAsync(originCity, destinationCity);
    // other code here for creating a quote
    var quote = new Quote {Id = 123,
        ExpectedDistance = distanceInfo.Item1,
        ExpectedDistanceType = distanceInfo.Item2};
    return quote;
}
```

Location of microservice

- Update the appSettings.json file to hold the location of the microservice.

```
"DistanceMicroservice": {  
  "Location": "https://localhost:6001/mapinfo"  
}
```

update the Startup.cs

- We need to update the Startup.cs file to register the DistanceInfoSvc and provide the
- HttpClient. In the ConfigureServices method, add the following code:

```
services.AddScoped(typeof(IDistanceInfoSvc), typeof(DistanceInfoSvc));  
var distanceMicroserviceUrl =  
Configuration.GetSection("DistanceMicroservice:Location").Value;  
services.AddHttpClient("DistanceMicroservice", client =>  
{  
client.BaseAddress= new Uri(distanceMicroserviceUrl);  
});
```

Final Touch

- The line that registers IDistanceInfoSvc will inject instances of DistanceInfoSvc where used in constructors.
- In this case, when instantiating the QuoteSvc class, an instance of DistanceInfoSvc is injected.
- The next section gets the URL to the microservice and provides it to the HttpClient base address.
- Using the AddHttpClient method allows for retrieving an instance of HttpClient.
- The key benefit of doing it this way is that another instance of the HttpClient is not generated with each call to the microservice.

Service Discovery

- the caller (monolith in our case) requires knowing the IP address of where the microservice instance is hosted
- The caller should not have the IP because another instance could replace the microservice with a new IP address.
- However, there should never be just one microservice instance in production.
- Callers of a microservice should not have the responsibility to know about every instance of microservice

Service Discovery

- Instead, the caller should only know about one IP address, which is a load balancer.
- Using a load balancer, you have a separation of concerns between the caller and the microservice instances.
- For this purpose, many products are available, such as Apache's ZooKeeper, HashiCorp's Consul, Eureka by Netflix, and even Nginx.

- Based on Elsevier Book .NET 6 Pro Microservices 2022
- Based on Modul 7.11 course public slides by Paul Leis