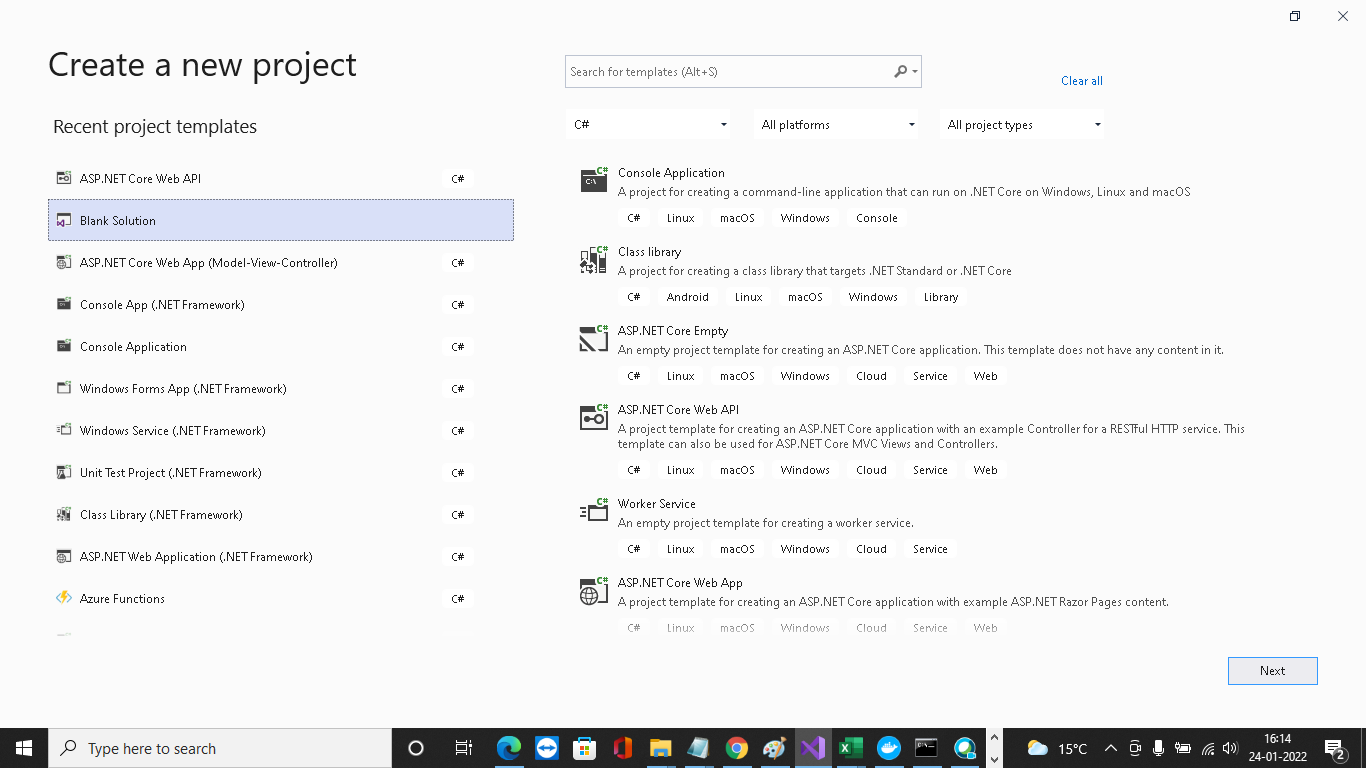
Microservices With Ocelot API Gateway In ASP.NET Core

A **microservice** architecture – a variant of the [service-oriented architecture](https://en.wikipedia.org/wiki/Service-oriented_architecture) (SOA) structural style – arranges an application as a collection of [loosely-coupled](https://en.wikipedia.org/wiki/Loose_coupling) services. In a microservices architecture, services are fine-grained and the protocols are [lightweight](https://en.wikipedia.org/wiki/Lightweight_protocol). The goal is that teams can bring their services to life independent of others. Loose coupling reduces all types of dependencies and the complexities around it, as service developers do not need to care about the users of the service, they do not force their changes onto users of the service. Therefore, it allows organizations developing software to grow fast, and big, as well as use off the shelf services easier. Communication requirements are less. But it comes at a cost to maintain the decoupling. Interfaces need to be designed carefully and treated as a public API. Techniques like having multiple interfaces on the same service, or multiple versions of the same service, to not break existing users code [Wikipedia](https://en.wikipedia.org/wiki/Microservices)

Steps to Microservice architecture and its implementation using Ocelot API Gateway in ASp.Net Core – 5.0.

## **Microservices Creation**

**Create a Blank Solution APIGateWayWithOcelotDemo**

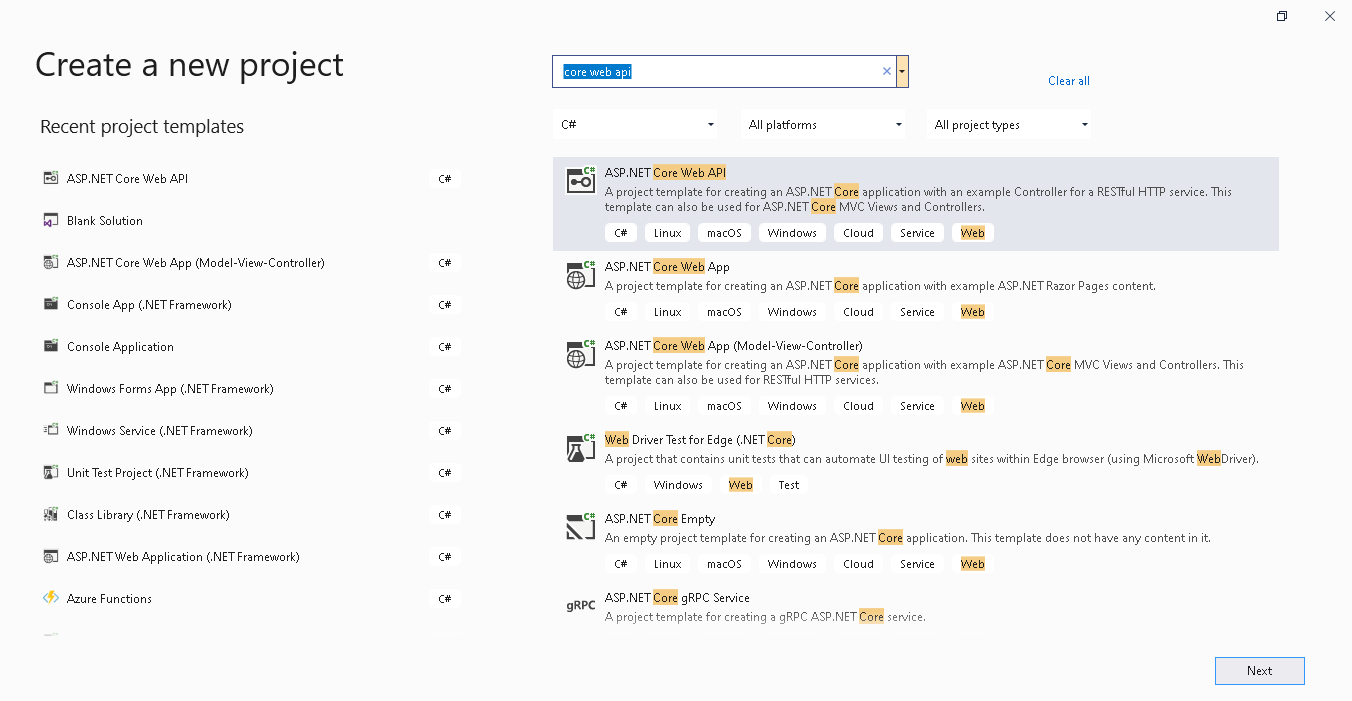


Create a New Solution Folder with the name as Microservices

**Right Click on Microservices Folder**

**Click on Add**

**Click on New Project**

****

Choose API as template and we are going with .Net Core 5.0 and above Version.

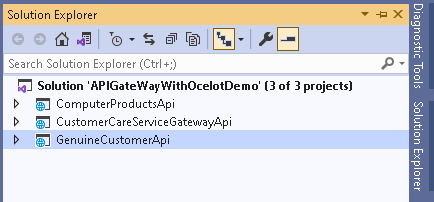
Enter the project name  **ComputerProductsApi**

Click on Create

Repeat the same step to add another new project ASP.NET Core Web API

Enter the project name  **GenuineCustomerApi**

Click on Create



Do the following modification in the **ComputerProductsApi**

1. **Add class**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace ComputerProductsApi

{

public class Product

{

public int ProductCode { get; set; }

public string ProductName { get; set; }

public decimal ProductPrice { get; set; }

}

}

1. **Add Controller ComputerProducts**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace ComputerProductsApi.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class ComputerProductsController : ControllerBase

{

public static List<Product> ProductsList = new List<Product>()

{

new Product(){ ProductCode=1,ProductName="Speakers",ProductPrice=30000.00m},

new Product(){ ProductCode=2,ProductName="Desktops",ProductPrice=30000.00m},

new Product(){ ProductCode=3,ProductName="Laptops",ProductPrice=25000.00m},

new Product(){ ProductCode=4,ProductName="Gaming PC's",ProductPrice=60000.00m},

new Product(){ ProductCode=5,ProductName="Servers",ProductPrice=100000.00m},

new Product(){ ProductCode=6,ProductName="Tablet",ProductPrice=5000.00m},

new Product(){ ProductCode=7,ProductName="PC's Monitors",ProductPrice=5000.00m},

new Product(){ ProductCode=8,ProductName="Memory",ProductPrice=2500.00m},

new Product(){ ProductCode=9,ProductName="Printers",ProductPrice=4000.00m},

new Product(){ ProductCode=10,ProductName="Projectors",ProductPrice=30000.00m}

};

// GET: api/<ComputerProductsController>

[HttpGet]

public IEnumerable<Product> Get()

{

return ProductsList.ToList();

}

// GET api/<ComputerProductsController>/5

[HttpGet("{id}")]

public IActionResult Get(int id)

{

Product product = ProductsList.FirstOrDefault(p => p.ProductCode == id);

if (product != null)

{

return StatusCode(StatusCodes.Status200OK, product);

}

else

{

return BadRequest();

}

}

// POST api/<ComputerProductsController>

[HttpPost]

public IActionResult Post(Product product)

{

if (product != null)

{

ProductsList.Add(product);

return StatusCode(StatusCodes.Status200OK, product);

}

else

{

return BadRequest();

}

}

// PUT api/<ComputerProductsController>/5

[HttpPut("{id}")]

public void Put(int id, [FromBody] string value)

{

}

// DELETE api/<ComputerProductsController>/5

[HttpDelete("{id}")]

public void Delete(int id)

{

}

}

}

1. **Change the port number in launchSettings**

 "ComputerProductsApi": {

"commandName": "Project",

"dotnetRunMessages": "true",

"launchBrowser": true,

"launchUrl": "swagger",

"applicationUrl": "http://localhost:7001",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

Do the following modification in the **ComputerProductsApi**

Create configuration.Json File under the Gateway.WebAPI to define the Routes which are necessary for Microservices.

**configuration.Json**

{

  "Routes": [

    {

      "DownstreamPathTemplate": "/api/product",

      "DownstreamScheme": "https",

      "DownstreamHostAndPorts": [

        {

          "Host": "localhost",

          "Port": 44337

        }

      ],

      "UpstreamPathTemplate": "/gateway/product",

      "UpstreamHttpMethod": [ "POST", "PUT", "GET" ]

    },

    {

      "DownstreamPathTemplate": "/api/product/{id}",

      "DownstreamScheme": "https",

      "DownstreamHostAndPorts": [

        {

          "Host": "localhost",

          "Port": 44337

        }

      ],

      "UpstreamPathTemplate": "/gateway/product/{id}",

      "UpstreamHttpMethod": [ "GET", "DELETE" ]

    },

    {

      "DownstreamPathTemplate": "/api/customer",

      "DownstreamScheme": "https",

      "DownstreamHostAndPorts": [

        {

          "Host": "localhost",

          "Port": 44373

        }

      ],

      "UpstreamPathTemplate": "/gateway/customer",

      "UpstreamHttpMethod": [ "POST", "PUT", "GET" ]

    },

    {

      "DownstreamPathTemplate": "/api/customer/{id}",

      "DownstreamScheme": "https",

      "DownstreamHostAndPorts": [

        {

          "Host": "localhost",

          "Port": 44373

        }

      ],

      "UpstreamPathTemplate": "/gateway/customer/{id}",

      "UpstreamHttpMethod": [ "GET", "DELETE" ]

    }

  ],

  "GlobalConfiguration": {

    "BaseUrl": "http://localhost:44382"

  }

}

**DownstreampathTemplate -**Defines the route of actual endpoint of Microservice

**DownstreamScheme -** scheme of Microservice, HTTPS

**DownstreamHostsandPorts -**Host and Port of Microservice will define here.

**UpstreampathTemplate -**The path at which the client will request the Ocelot API Gateway

**UpstreamHttpmethod -**The Supported HTTP Methods to the API Gateway. Based on the incoming method,Ocelot sends a  similar HTTP method request to microservices as well.

Let's test the application and this will run under the Gateway.WebAPI Port number which we already defined in the configuration.json file

**launchsettings.json (Gateway API)**

{

  "iisSettings": {

    "windowsAuthentication": **false**,

    "anonymousAuthentication": **true**,

    "iisExpress": {

      "applicationUrl": "http://localhost:51733",

      "sslPort": 44382

    }

  },

  "profiles": {

    "IIS Express": {

      "commandName": "IISExpress",

      "launchBrowser": **true**,

      "environmentVariables": {

        "ASPNETCORE\_ENVIRONMENT": "Development"

      }

    },

    "Gateway.WebApi": {

      "commandName": "Project",

      "launchBrowser": **true**,

      "applicationUrl": "https://localhost:5001;http://localhost:5000",

      "environmentVariables": {

        "ASPNETCORE\_ENVIRONMENT": "Development"

      }

    }

  }

}