Ask Question

RabbitMQ Message Queue Using .NET Core 6 Web API

Jaydeep Patil Updated date Jul 25, 2022 7.3k 0 4

RabitMqProductAPI.zip

In this article, we will discuss the RabbitMQ Message Queue and its implementation using .NET Core 6 API as Message Producer and Console Application as a Message Consumer.

Agenda:

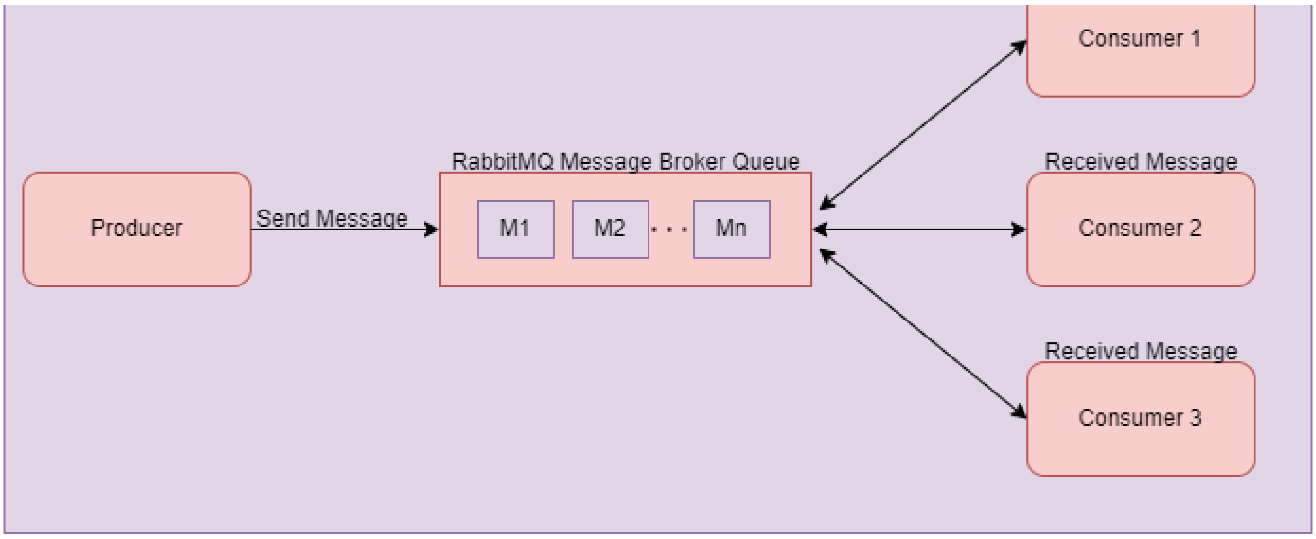
- Introduction of RabbitMQ
- Benefits of using RabbitMQ
- Implementation of RabbitMQ in .NET Core 6

Prerequisites:

- Visual Studio 2022
- Docker Desktop
- .NET Core 6 SDK

Introduction of RabbitMQ

- Rabbit MQ is the message broker that acts as a middleware while using multiple microservices.
- RabbitMQ is an open-source message broker software. It is sometimes also called message-oriented middleware.
- RabbitMQ is written in the Erlang programming language.
- RabbitMQ is used to reduce the load and delivery time of a web application when some of the resources have taken a lot of time to process the data.



- As you can see in the diagram above, there is one producer who sends a message to the RabbitMQ server. The server will store that message inside the queue in a FIFO manner.
- Once the producer has sent the message to the queue, there may be multiple consumers that want the message produced by the producer. In that case, consumers subscribe to the message and get that message from the Message Queue as you see in the above diagram.
- In this section, we will use one eCommerce Site as an example to understand more fully.
- There are multiple microservices running in the background while we are using the eCommerce website. There is one service that takes care of order details, and another service that takes care of payment details and receipts.
- Suppose we placed one order. At that time, the order service will start and process our order. After taking the order details, it will send data to the payment service, which takes the payment and sends the payment receipt to the end-users.
- In this case, there may be a chance of some technical issue occurring in the payment service. If the user did not receive the payment receipt due to this, the user will be impacted and connected with the support team ti try to learn the status of the order.
- There may be another scenario on the user(consumer) side. Perhaps due to some technical issue, the user is exited from the application when the payment is in process. But, he will not get any receipt details after payment is successfully processed from backend services.
- In these scenarios, the RabbitMQ plays an essential role to process messages in the message queue. So, when the consumer gets online, he will receive that order receipt message from the message queue, produced by the producer without impacting the web application.
- All these examples are just for understanding purpose. There are a lot of scenarios in which RabbitMQ may play an important role while using multiple microservices. Sometimes RabbitMQ is used fully to load balancing between multiple services, or for many other purposes.

Benefits of using RabbitMQ

There are many benefits to using a Message Broker to send data to the consumer. Below, we will discuss a few of these benefits.

Hight Availability

[illegible]
RabbitMQ server. After some time, when our service starts working, it will connect with RabbitMQ and take the pending message easily.

Scalability

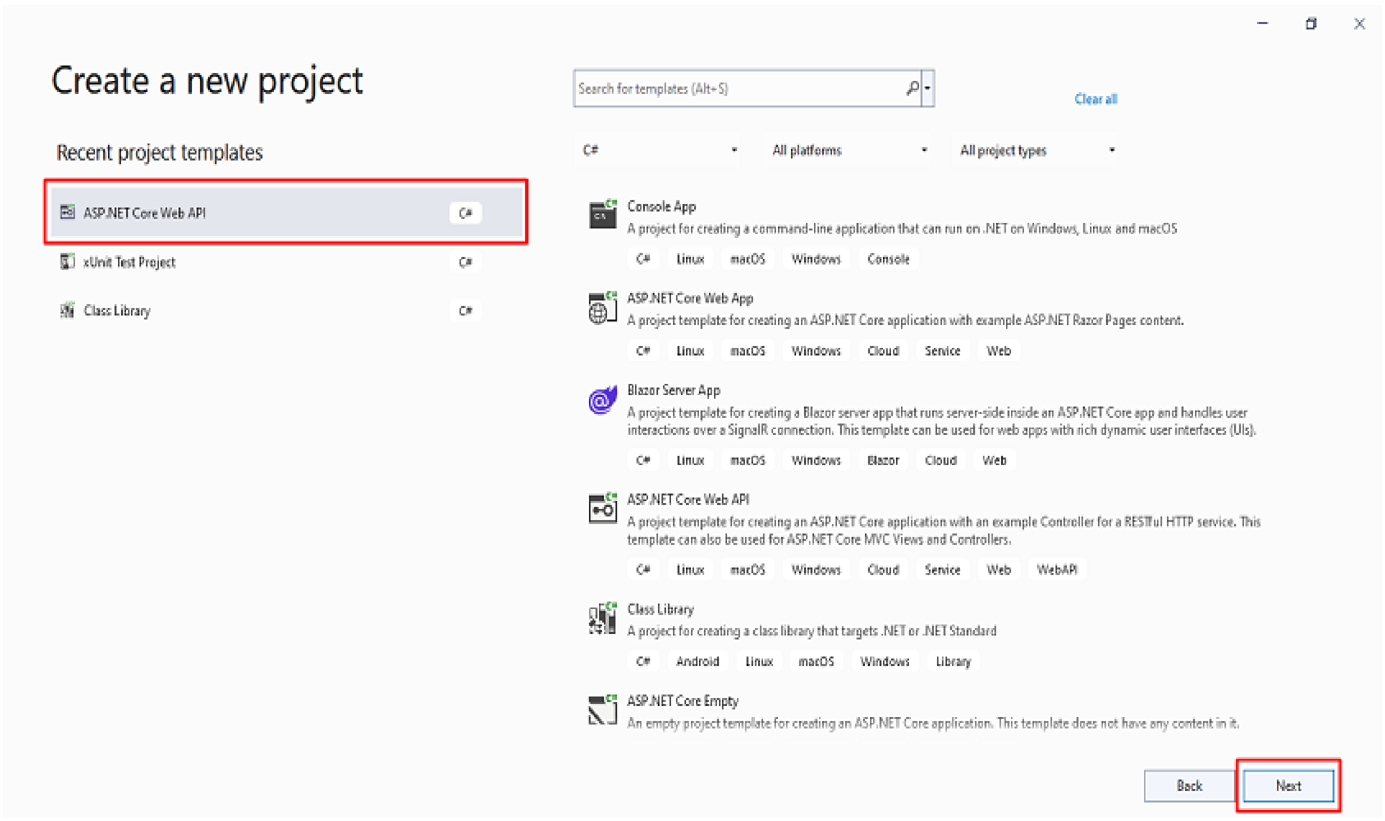
When we use RabbitMQ, at that time our application does not depend on only one server and virtual machine to process a request. If our server is stopped at that time, RabbitMQ will transfer our application load to another server that has the same services running in the background.

RabbitMQ Implementation with .NET Core 6

Let’s start with the practical implementation of RabbitMQ using .NET Core 6 Web API as producer and Console Application as a consumer.

Step 1

Create a .NET Core API project.


Create a new project

Recent project templates

ASP.NET Core Web API C#
xUnit Test Project C#
Class Library C#

Search for templates (Alt+S)
Clear all
C# All platforms All project types

Console App
A project for creating a command-line application that can run on .NET on Windows, Linux and macOS
C# Linux macOS Windows Console

ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

Blazor Server App
A project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Blazor Cloud Web

ASP.NET Core Web API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
C# Linux macOS Windows Cloud Service Web WebAPI

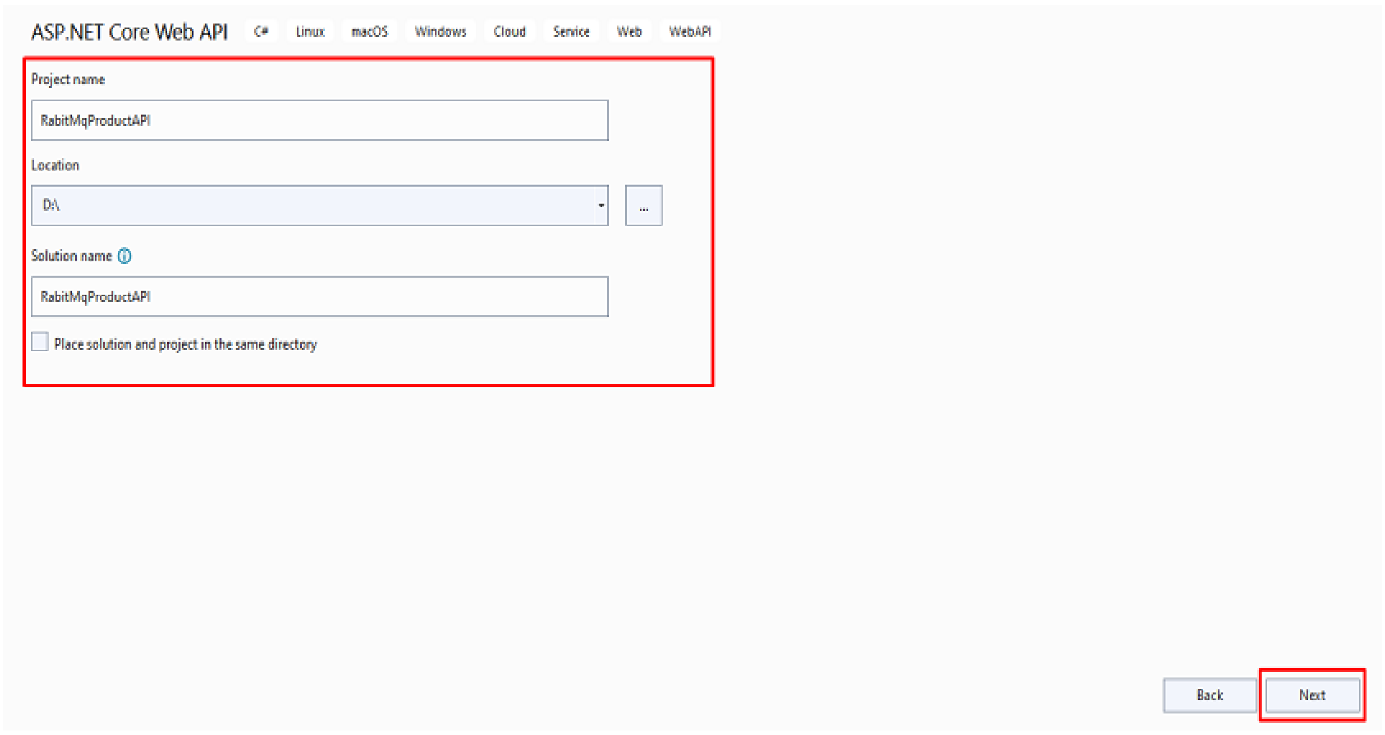
Class Library
A project for creating a class library that targets .NET or .NET Standard
C# Android Linux macOS Windows Library

ASP.NET Core Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

Back Next

Step 2

Configure your project.


ASP.NET Core Web API

Project name: RabitMqProductAPI

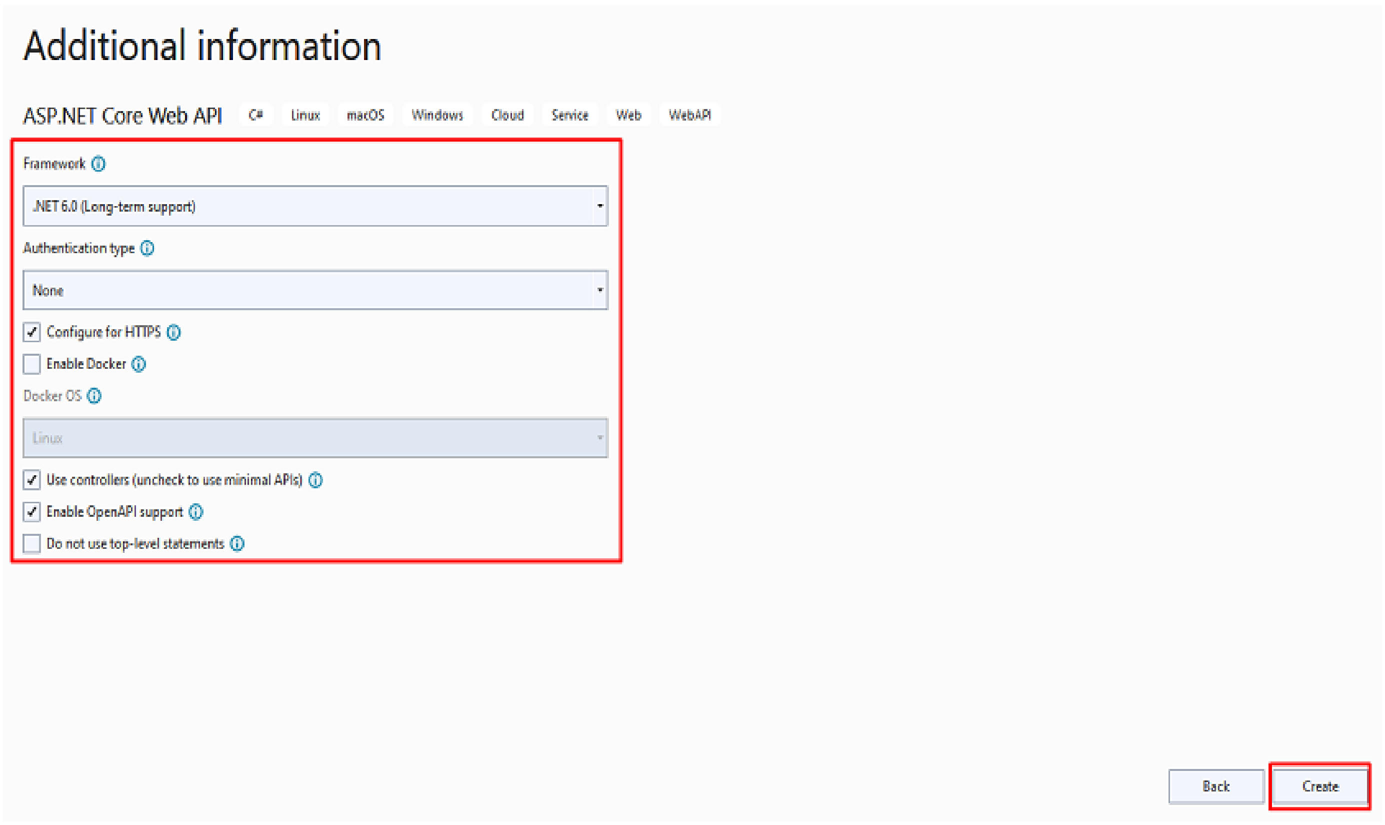
Location: D:\

Solution name: RabitMqProductAPI

☐ Place solution and project in the same directory

Step 3

Provide additional information about your project.


Additional information

ASP.NET Core Web API

Framework: .NET 6.0 (Long-term support)

Authentication type: None

☑ Configure for HTTPS

☐ Enable Docker

Docker OS: Linux

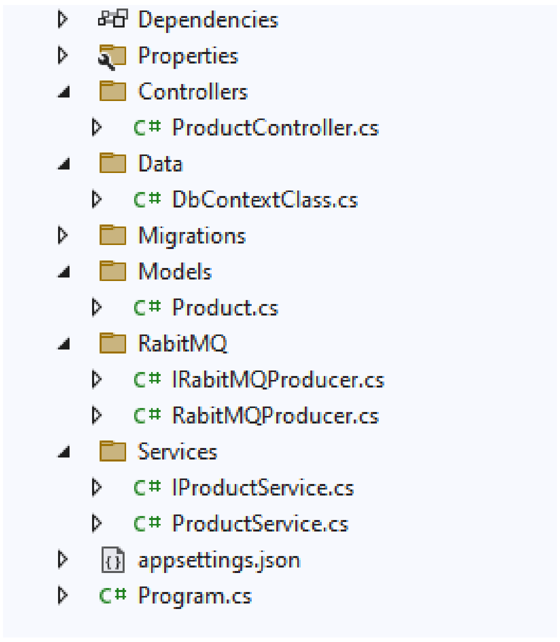
☑ Use controllers (uncheck to use minimal APIs)

☑ Enable OpenAPI support

☐ Do not use top-level statements

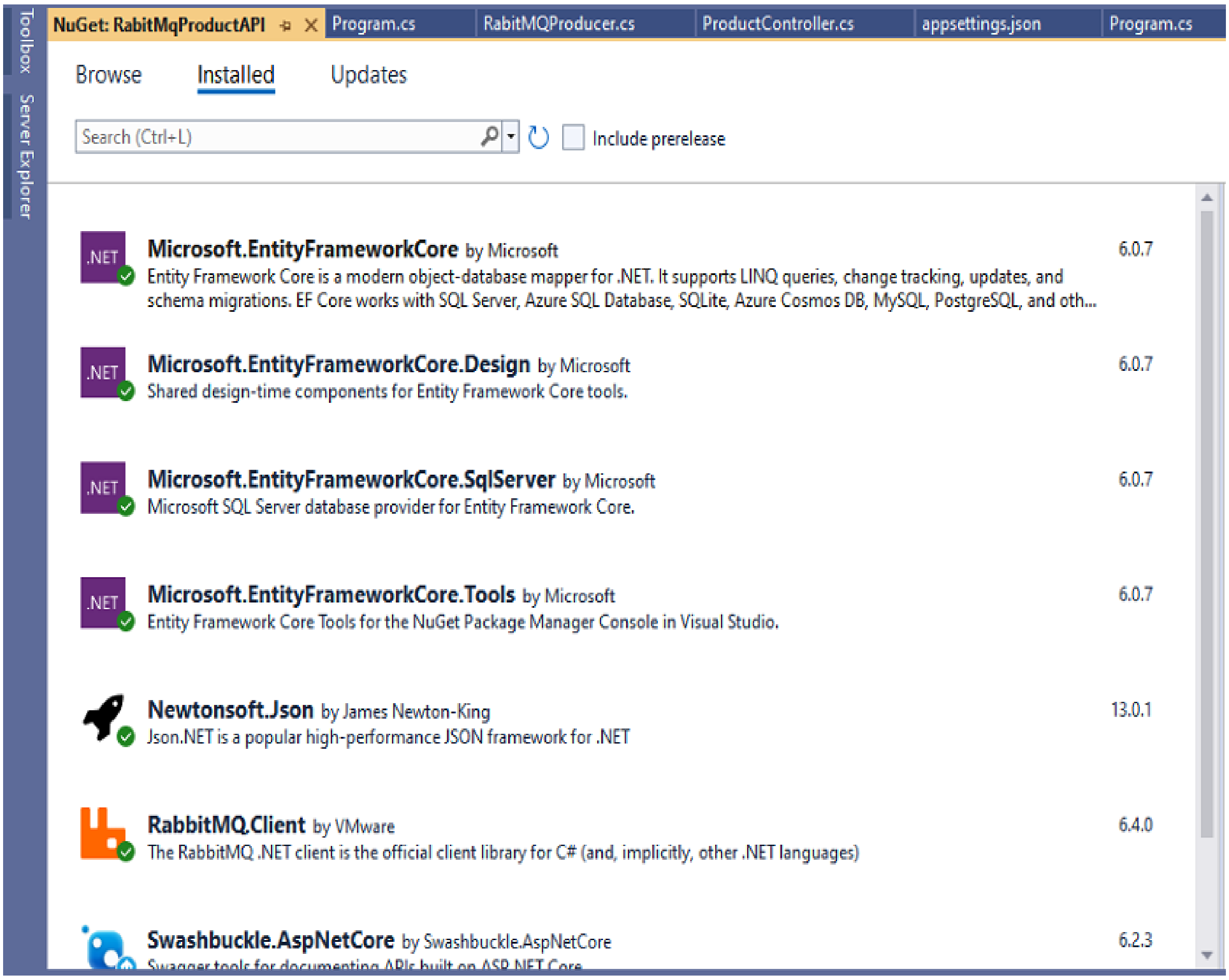
Step 4

Project Structure of Producer Application.



Step 5

Install a few NuGet Package.



Microsoft.EntityFrameworkCore by Microsoft 6.0.7
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and oth...

Microsoft.EntityFrameworkCore.Design by Microsoft 6.0.7
Shared design-time components for Entity Framework Core tools.

Microsoft.EntityFrameworkCore.SqlServer by Microsoft 6.0.7
Microsoft SQL Server database provider for Entity Framework Core.

Microsoft.EntityFrameworkCore.Tools by Microsoft 6.0.7
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

Newtonsoft.Json by James Newton-King 13.0.1
Json.NET is a popular high-performance JSON framework for .NET

RabbitMQ.Client by VMware 6.4.0
The RabbitMQ .NET client is the official client library for C# (and, implicitly, other .NET languages)

Swashbuckle.AspNetCore by Swashbuckle.AspNetCore 6.2.3
Swagger tools for documenting APIs built on ASP.NET Core

Step 6

```
namespace RabitMqProductAPI.Models {
    public class Product {
        public int ProductId {
            get;
            set;
        }
        public string ProductName {
            get;
            set;
        }
        public string ProductDescription {
            get;
            set;
        }
        public int ProductPrice {
            get;
            set;
        }
        public int ProductStock {
            get;
            set;
        }
    }
}
```

Step 7

Next, create a DbContextClass.cs class inside the Data folder.

```
using Microsoft.EntityFrameworkCore;
using RabitMqProductAPI.Models;
namespace RabitMqProductAPI.Data {
    public class DbContextClass: DbContext {
        protected readonly IConfiguration Configuration;
        public DbContextClass(IConfiguration configuration) {
            Configuration = configuration;
        }
        protected override void OnConfiguring(DbContextOptionsBuilder optic
            options.UseSqlServer(Configuration.GetConnectionString("Default
        }
        public DbSet < Product > Products {
            get;
            set;
        }

```

Step 8

Later on, create an IProductService.cs and ProductService.cs class inside the Services folder.

```
using RabitMqProductAPI.Models;
namespace RabitMqProductAPI.Services {
    public interface IProductService {
        public IEnumerable < Product > GetProductList();
        public Product GetProductById(int id);
        public Product AddProduct(Product product);
        public Product UpdateProduct(Product product);
        public bool DeleteProduct(int Id);
    }
}
```

Create a ProductService.cs.

```
using RabitMqProductAPI.Data;
using RabitMqProductAPI.Models;
namespace RabitMqProductAPI.Services {
    public class ProductService: IProductService {
        private readonly DbContextClass _dbContext;
        public ProductService(DbContextClass dbContext) {
            _dbContext = dbContext;
        }
        public IEnumerable < Product > GetProductList() {
            return _dbContext.Products.ToList();
        }
        public Product GetProductById(int id) {
            return _dbContext.Products.Where(x => x.ProductId == id).FirstC
        }
        public Product AddProduct(Product product) {
            var result = _dbContext.Products.Add(product);
            _dbContext.SaveChanges();
            return result.Entity;
        }
        public Product UpdateProduct(Product product) {
            var result = _dbContext.Products.Update(product);
            _dbContext.SaveChanges();
            return result.Entity;
        }
        public bool DeleteProduct(int Id) {
            var filteredData = _dbContext.Products.Where(x => x.Product
            var result = _dbContext.Remove(filteredData);
        }
    }
}
```

Step 9

Create IRabitMQProducer.cs and RabitMQProducer.cs classes for the message queue inside the RabbitMQ folder.

```
namespace RabitMqProductAPI.RabitMQ {
    public interface IRabitMQProducer {
        public void SendProductMessage < T > (T message);
    }
}
```

Next, create a RabitMQProducer.cs class.

```
using Newtonsoft.Json;
using RabbitMQ.Client;
using System.Text;
namespace RabitMqProductAPI.RabitMQ {
    public class RabitMQProducer: IRabitMQProducer {
        public void SendProductMessage < T > (T message) {
            //Here we specify the Rabbit MQ Server. we use rabbitmq docker
            var factory = new ConnectionFactory {
                HostName = "localhost"
            };
            //Create the RabbitMQ connection using connection factory detai
            var connection = factory.CreateConnection();
            //Here we create channel with session and model
            using
            var channel = connection.CreateModel();
            //declare the queue after mentioning name and a few property re
            channel.QueueDeclare("product", exclusive: false);
            //Serialize the message
            var json = JsonConvert.SerializeObject(message);
            var body = Encoding.UTF8.GetBytes(json);
            //put the data on to the product queue
            channel.BasicPublish(exchange: "", routingKey: "product", body:
        }
    }
}
```

Step 10

```
using Microsoft.AspNetCore.Mvc;
using RabitMqProductAPI.Models;
using RabitMqProductAPI.RabitMQ;
using RabitMqProductAPI.Services;
namespace RabitMqProductAPI.Controllers {
    [Route("api/[controller]")]
    [ApiController]
    public class ProductController: ControllerBase {
        private readonly IProductService productService;
        private readonly IRabitMQProducer _rabitMQProducer;
        public ProductController(IProductService _productService, IRabitMQP
                productService = _productService;
                _rabitMQProducer = rabitMQProducer;
            }
            [HttpGet("productlist")]
        public IEnumerable < Product > ProductList() {
                var productList = productService.GetProductList();
                return productList;
            }
            [HttpGet("getproductbyid")]
        public Product GetProductById(int Id) {
                return productService.GetProductById(Id);
            }
            [HttpPost("addproduct")]
        public Product AddProduct(Product product) {
                var productData = productService.AddProduct(product);
                //send the inserted product data to the queue and consumer
                _rabitMQProducer.SendProductMessage(productData);
                return productData;
            }
            [HttpPut("updateproduct")]
        public Product UpdateProduct(Product product) {
                return productService.UpdateProduct(product);
            }
            [HttpDelete("deleteproduct")]
        public bool DeleteProduct(int Id) {
            return productService.DeleteProduct(Id);
        }
    }
}
```

Here, you can see that we inject the IRabbitMQProducer service inside the constructor and use it in the add product API endpoint to send data into the message queue. This inserts product det

Step 11

Add the connection string inside the appsetting.json file.

```
{
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "AllowedHosts": "*",
    "ConnectionStrings": {
        "DefaultConnection": "Data Source=DESKTOP-***;Initial Catalog=Rabit
    }
}
```

Step 12

Next, register a few services inside the Program.cs class.

```
using RabitMqProductAPI.Data;
using RabitMqProductAPI.RabitMQ;
using RabitMqProductAPI.Services;
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
builder.Services.AddScoped < IProductService, ProductService > ();
builder.Services.AddDbContext < DbContextClass > ();
builder.Services.AddScoped < IRabitMQProducer, RabitMQProducer > ();
builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcor
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
var app = builder.Build();
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment()) {
    app.UseSwagger();
    app.UseSwaggerUI();
}
app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

Add migration and update using the following entity framework command after executing that into the package manager console under the main project.

add-migration “first”

update-database

Step 14

Install Rabbitmq docker file using the following command (Note- docker desktop is in running mode):

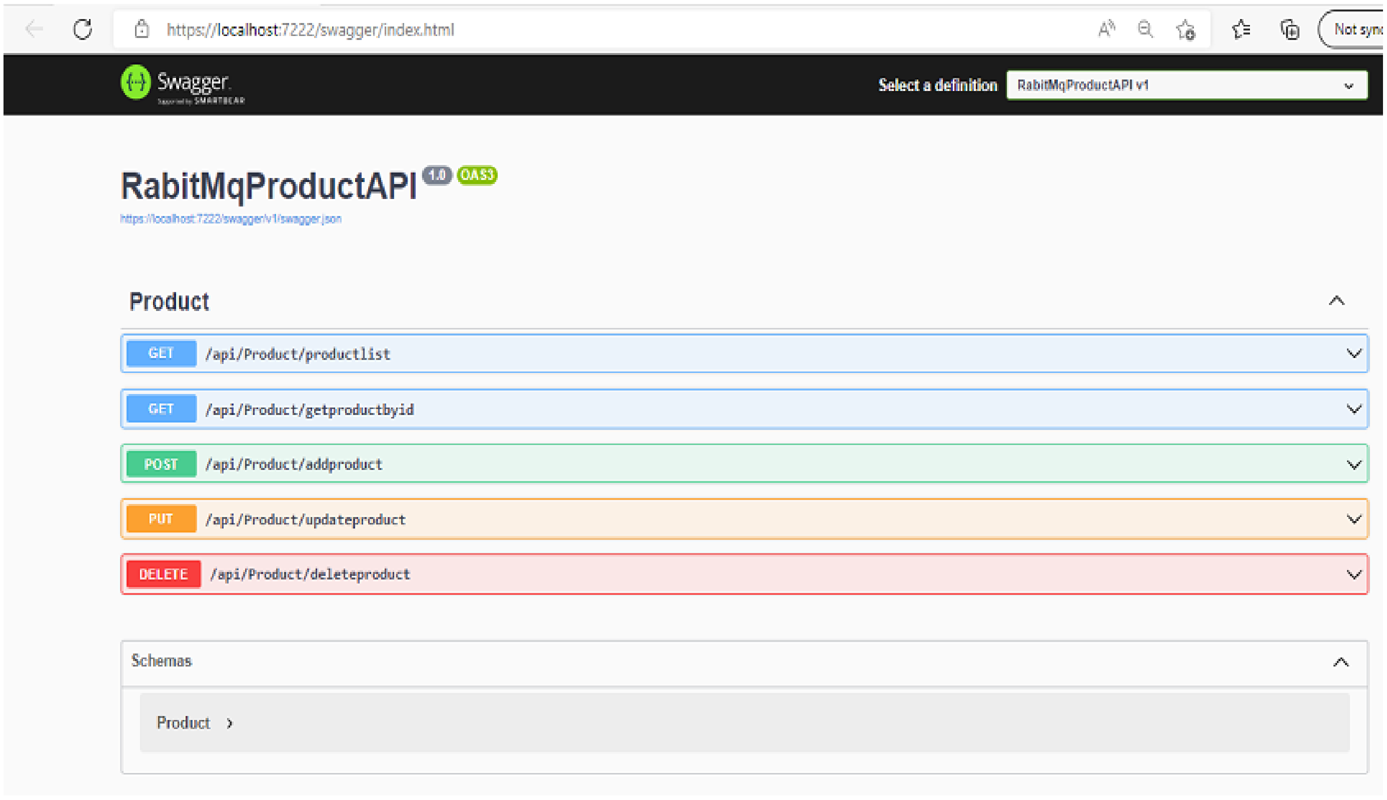
docker pull rabbitmq:3-management

Next, create a container and start using the Rabbitmq Dockerfile that we downloaded:

docker run --rm -it -p 15672:15672 -p 5672:5672 rabbitmq:3-management

Step 15

Finally, run your application and you will see the swagger UI and API endpoints.


https://localhost:7222/swagger/index.html

Select a definition RabitMqProductAPI v1

RabitMqProductAPI 1.0 OAS3

https://localhost:7222/swagger/v1/swagger.json

Product

GET /api/Product/productlist

GET /api/Product/getproductbyid

POST /api/Product/addproduct

PUT /api/Product/updateproduct

DELETE /api/Product/deleteproduct

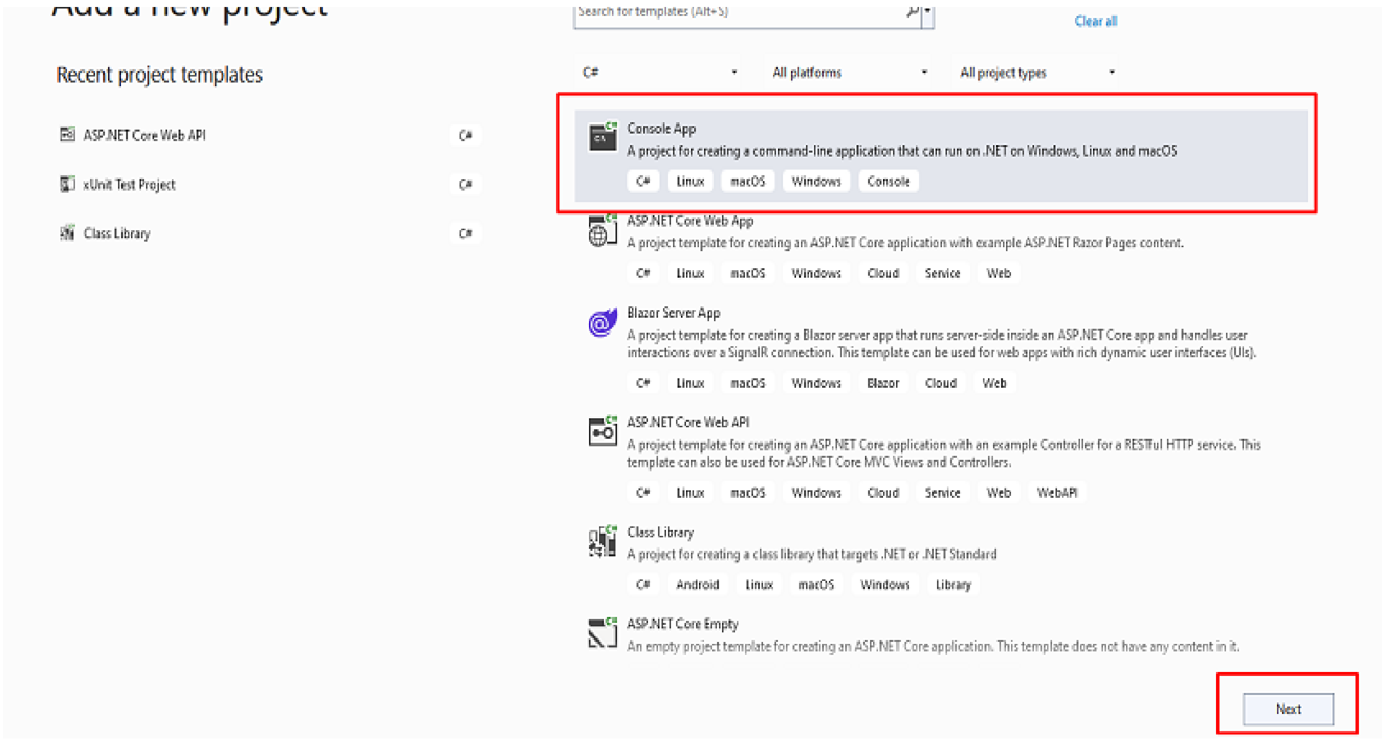
Schemas

Product >

This is all about Product Web API as producer. Now, let's create a new console application as the consumer consumes the messag sent by the producer.

Step 1

Add a new console application inside the same solution.



Console App
A project for creating a command-line application that can run on .NET on Windows, Linux and macOS
C# Linux macOS Windows Console

ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

Blazor Server App
A project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Blazor Cloud Web

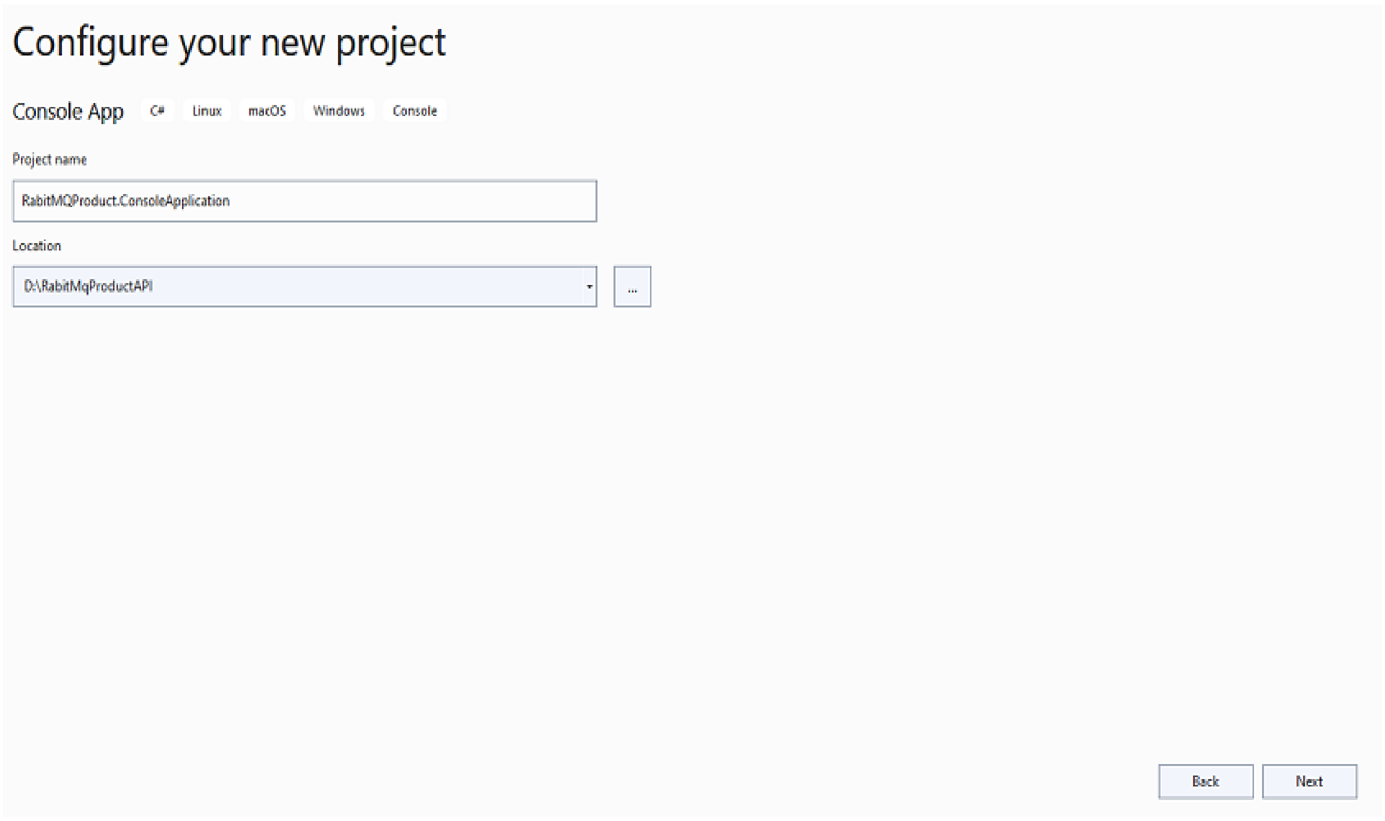
ASP.NET Core Web API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
C# Linux macOS Windows Cloud Service Web WebAPI

Class Library
A project for creating a class library that targets .NET or .NET Standard
C# Android Linux macOS Windows Library

ASP.NET Core Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

Step 2

Configure your new project.

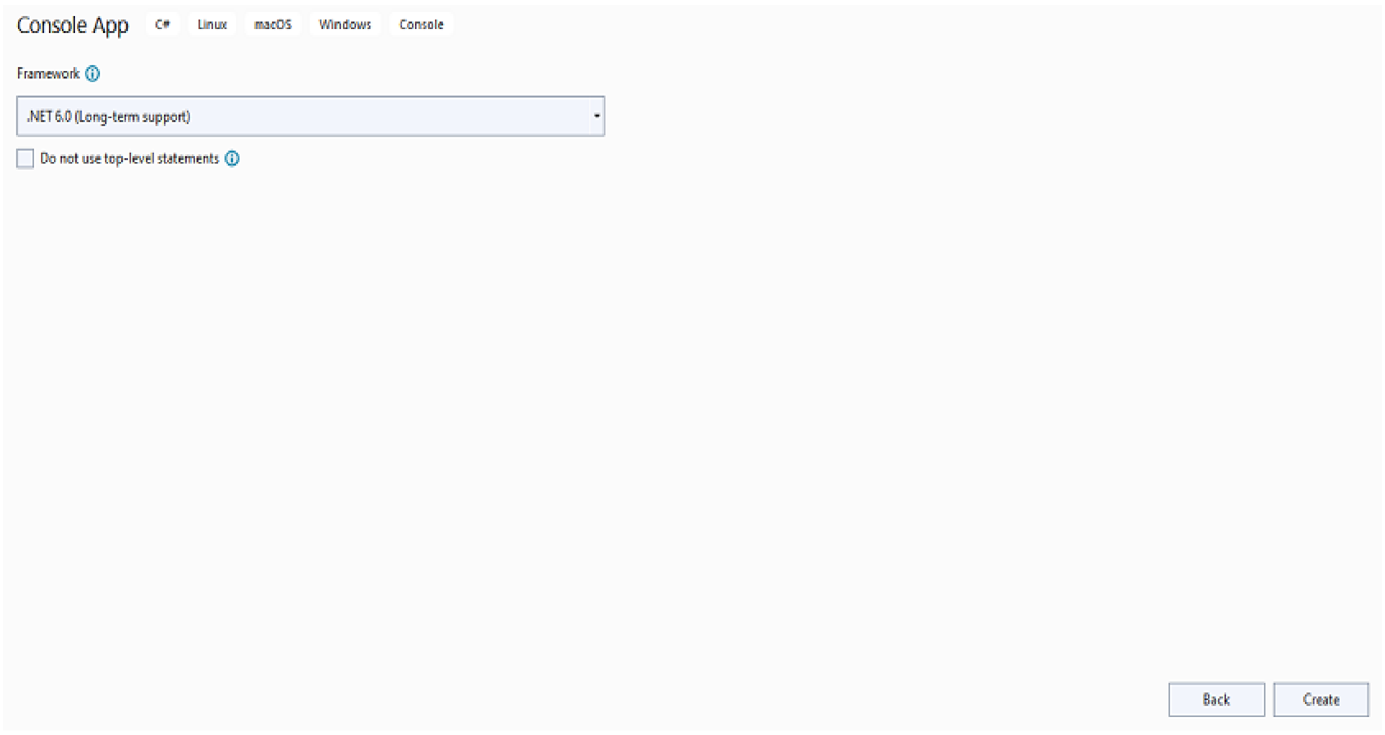

Configure your new project
Console App C# Linux macOS Windows Console
...
Back
Next

Project name: RabitMQProduct.ConsoleApplication

Location: D:\RabitMqProductAPI

Step 3

Provide additional information.

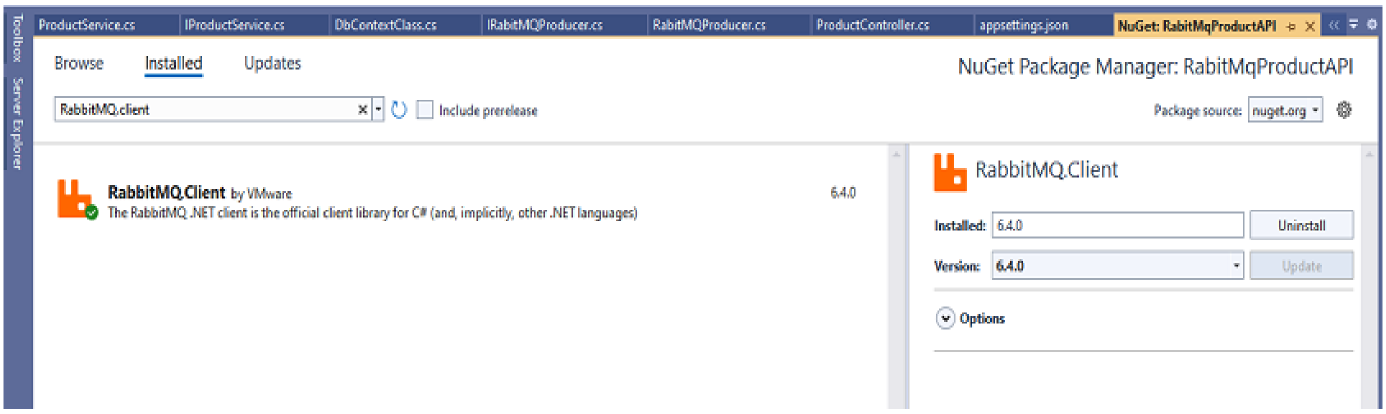


Framework: .NET 6.0 (Long-term support)

☐ Do not use top-level statements

Step 4

Install a few NuGet packages.



NuGet Package Manager: RabitMqProductAPI

RabbitMQ.client

RabbitMQ.Client by VMware 6.4.0
The RabbitMQ .NET client is the official client library for C# (and, implicitly, other .NET languages)

RabbitMQ.Client
Installed: 6.4.0
Version: 6.4.0

Step 5

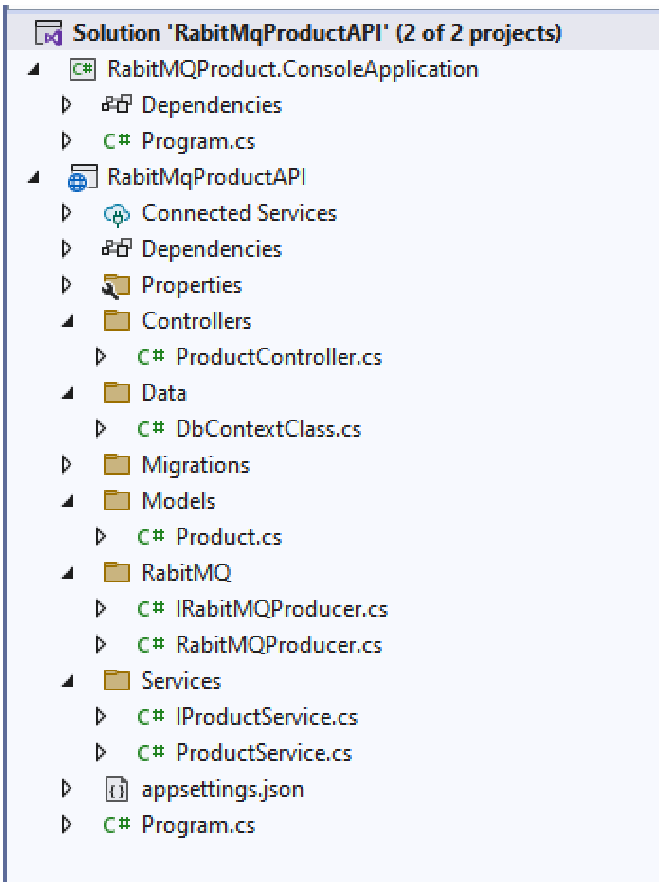
Add the following code inside the Program.cs class:

```
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using System.Text;
//Here we specify the Rabbit MQ Server. we use rabbitmq docker image and us
var factory = new ConnectionFactory {
    HostName = "localhost"
};
//Create the RabbitMQ connection using connection factory details as i ment
var connection = factory.CreateConnection();
//Here we create channel with session and model
using
```

```
channel.QueueDeclare("product", exclusive: false);
//Set Event object which listen message from chanel which is sent by produc
var consumer = new EventingBasicConsumer(channel);
consumer.Received += (model, eventArgs) => {
    var body = eventArgs.Body.ToArray();
    var message = Encoding.UTF8.GetString(body);
    Console.WriteLine($ "Product message received: {message}");
};
//read the message
channel.BasicConsume(queue: "product", autoAck: true, consumer: consumer);
Console.ReadKey();
```

Step 6

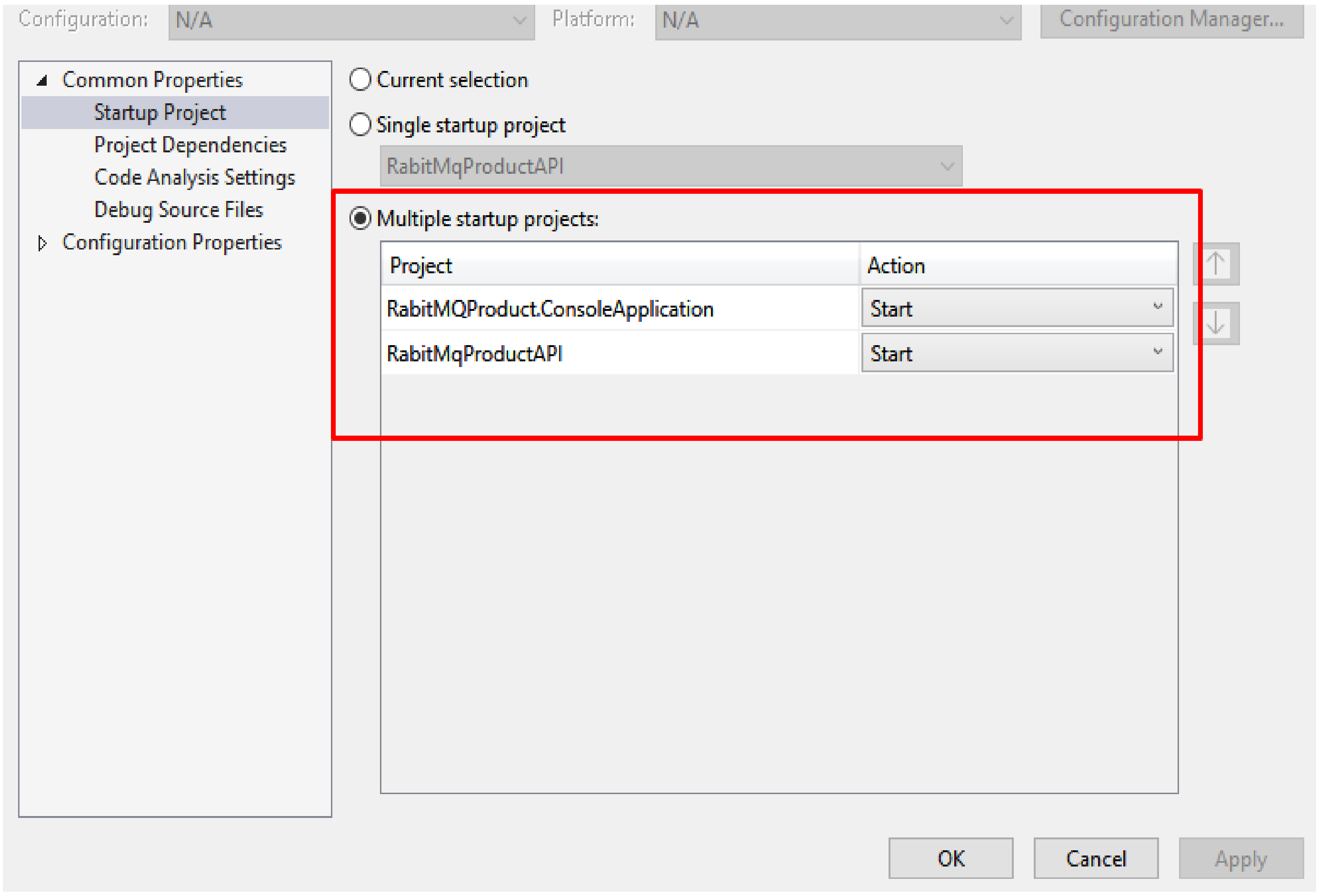
Final Project Structure:



```
Solution 'RabitMqProductAPI' (2 of 2 projects)
  RabitMQProduct.ConsoleApplication
    Dependencies
    Program.cs
  RabitMqProductAPI
    Connected Services
    Dependencies
    Properties
    Controllers
      ProductController.cs
    Data
      DbContextClass.cs
    Migrations
    Models
      Product.cs
    RabitMQ
      IRabitMQProducer.cs
      RabitMQProducer.cs
    Services
      IProductService.cs
      ProductService.cs
    appsettings.json
    Program.cs
```

Step 7

Go to the solution property and configure both the producer and the consumer project as a starting project as shown below:



Multiple startup projects: [✓]

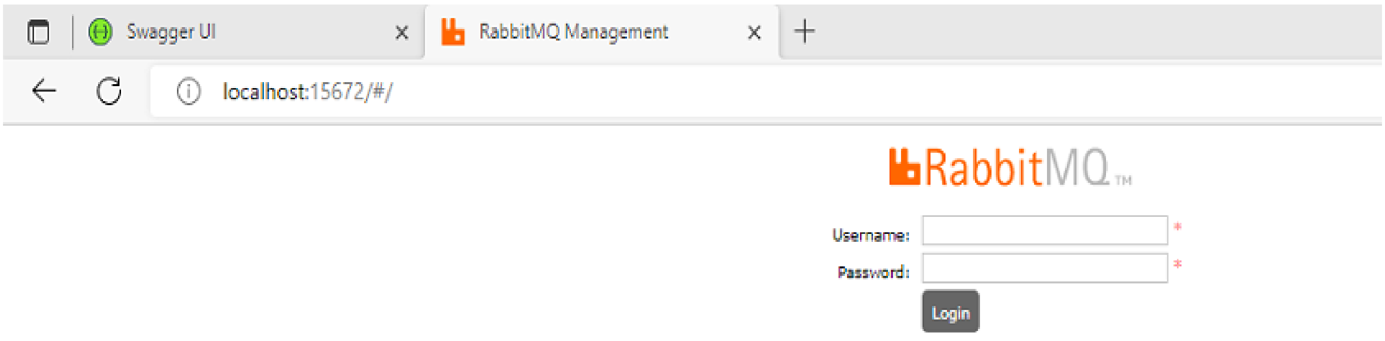
Project	Action
RabitMQProduct.ConsoleApplication	Start
RabitMqProductAPI	Start

Step 8

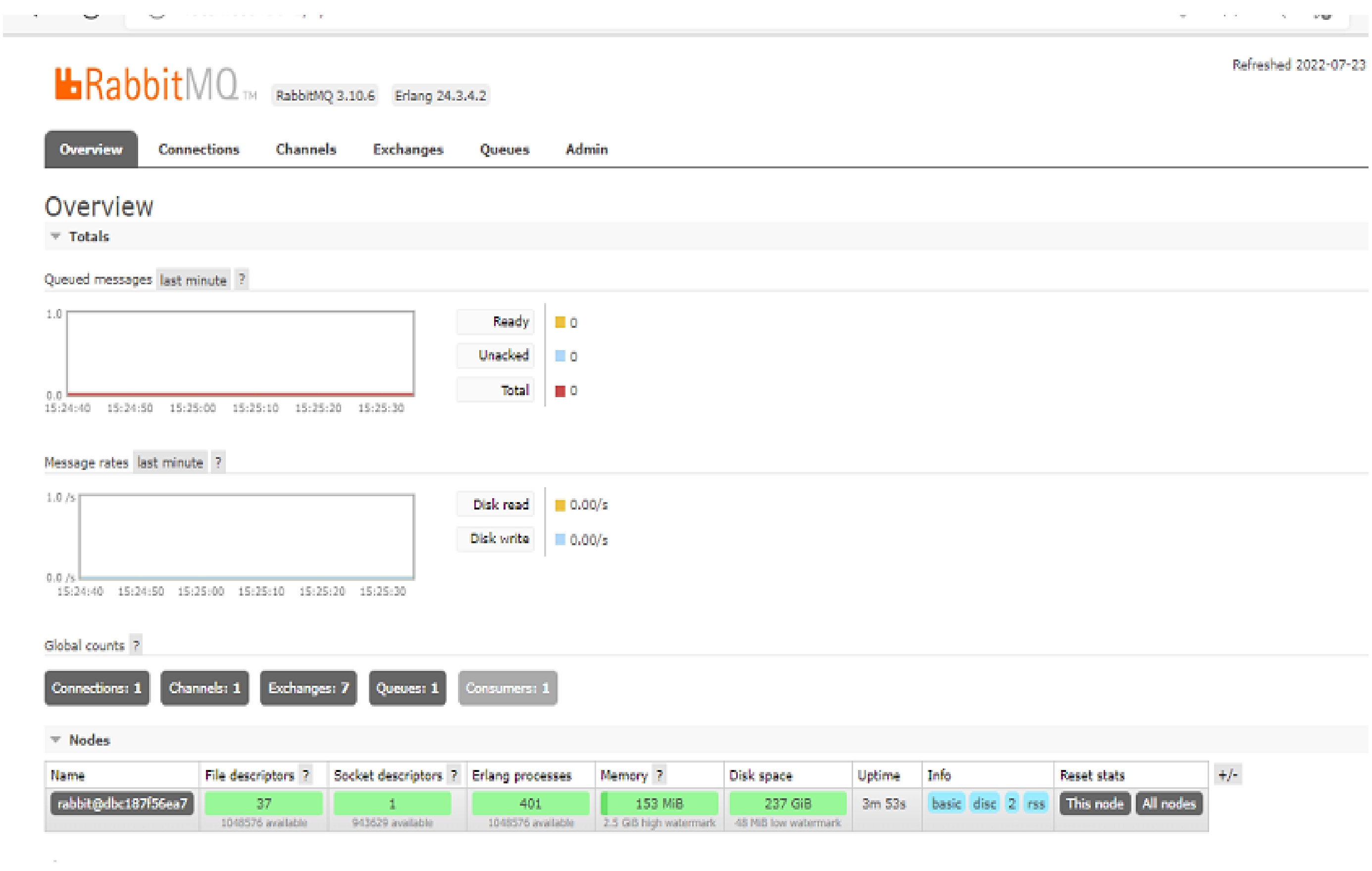
Open the following URL to open the RabbitMQ dashboard on the port we set while running docker:

http://localhost:15672/

When you click the URL, the login page will open.



Enter default username ("guest") and password (also "guest"), and next you will see the dashboard.


RabbitMQ™ RabbitMQ 3.10.6 Erlang 24.3.4.2

Refreshed 2022-07-23

Overview | Connections | Channels | Exchanges | Queues | Admin

Overview

Totals

Queued messages last minute ?

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last minute ?

Disk read	🟨 0.00/s
Disk write	🟦 0.00/s

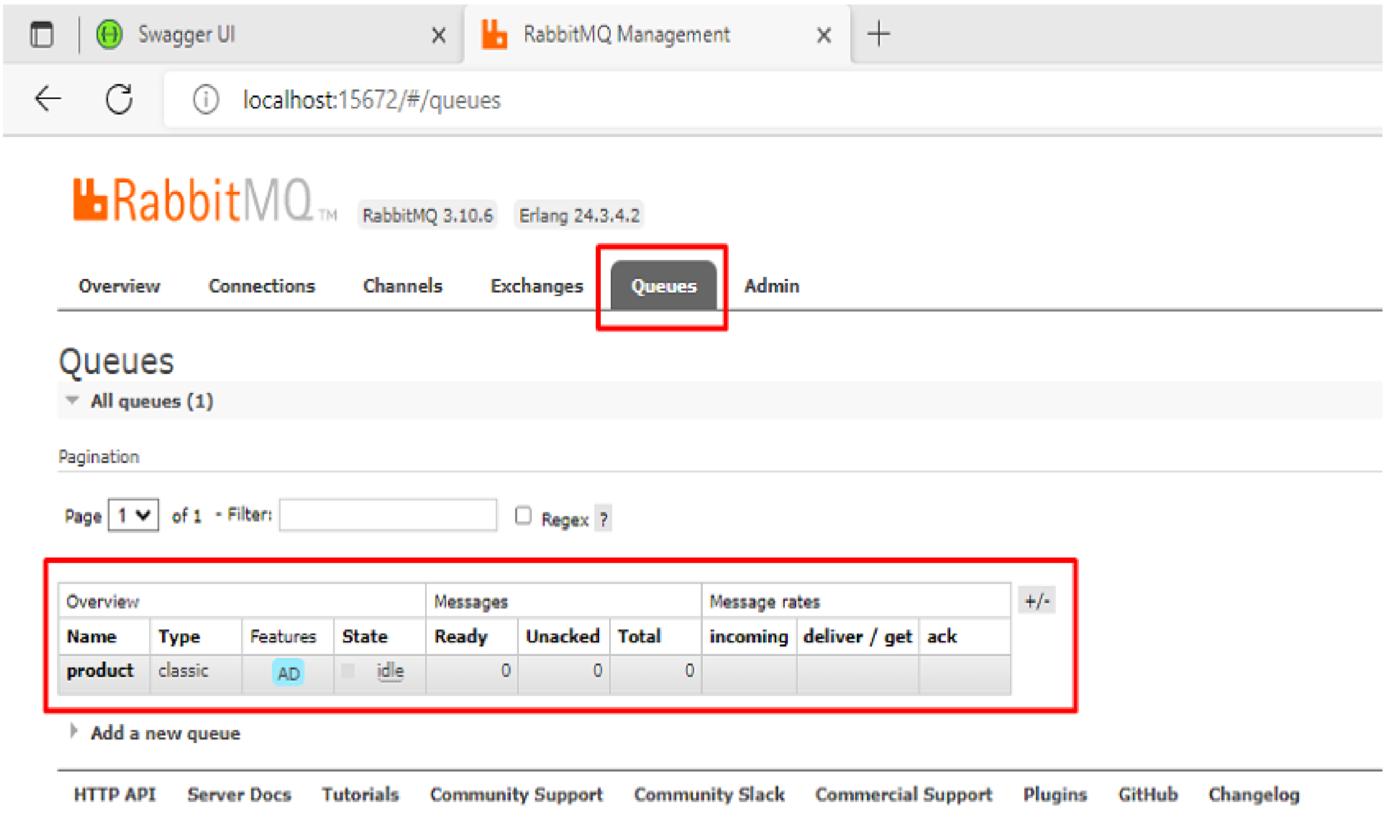
Global counts ?

Connections: 1 | Channels: 1 | Exchanges: 7 | Queues: 1 | Consumers: 1

Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats
rabbit@dbc187f56ea7	37 1048576 available	1 943629 available	401 1048576 available	153 MiB 2.5 GiB high watermark	237 GiB 48 MiB low watermark	3m 53s	basic disc 2 rss	This node All nodes

Open the queue tab, in which you will see our product queue.


Swagger UI | RabbitMQ Management

localhost:15672/#/queues

RabbitMQ™ RabbitMQ 3.10.6 Erlang 24.3.4.2

Overview | Connections | Channels | Exchanges | Queues | Admin

Queues

All queues (1)

Pagination

Page 1 of 1 - Filter: Regex ?

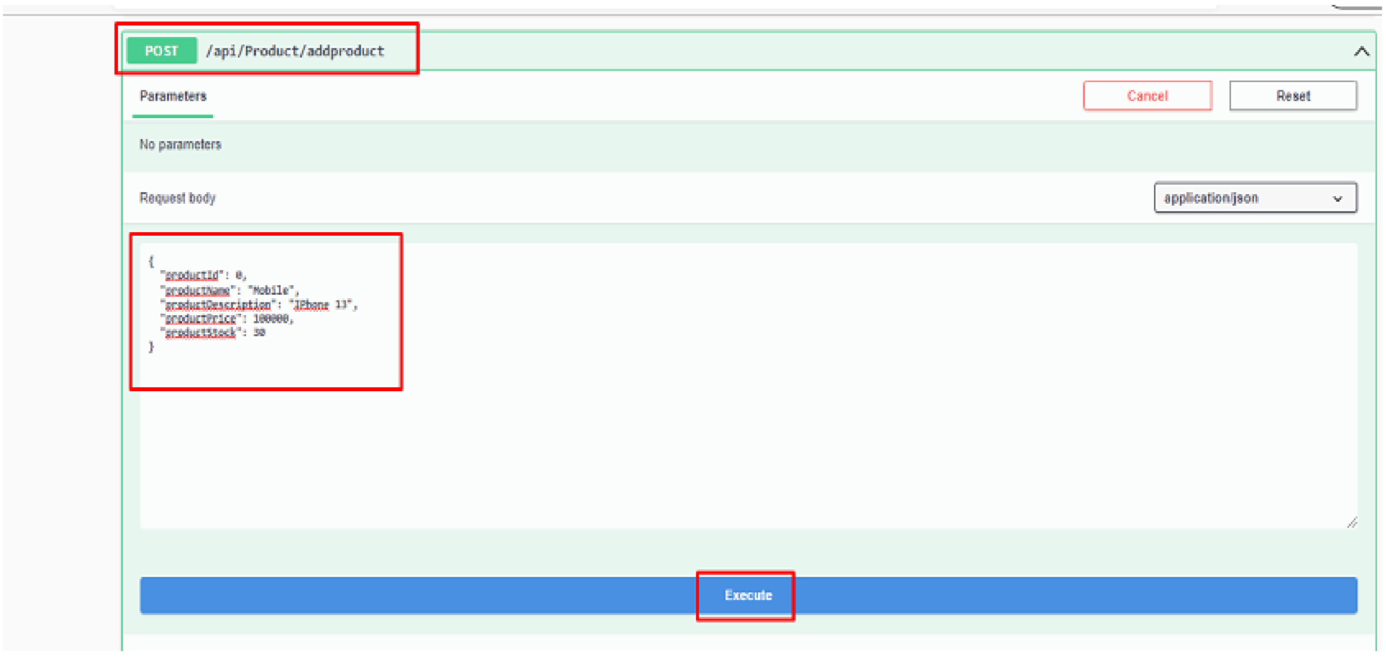
Overview				Messages			Message rates		
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
product	classic	AD	idle	0	0	0			

Add a new queue

HTTP API | Server Docs | Tutorials | Community Support | Community Slack | Commercial Support | Plugins | GitHub | Changelog

Step 9

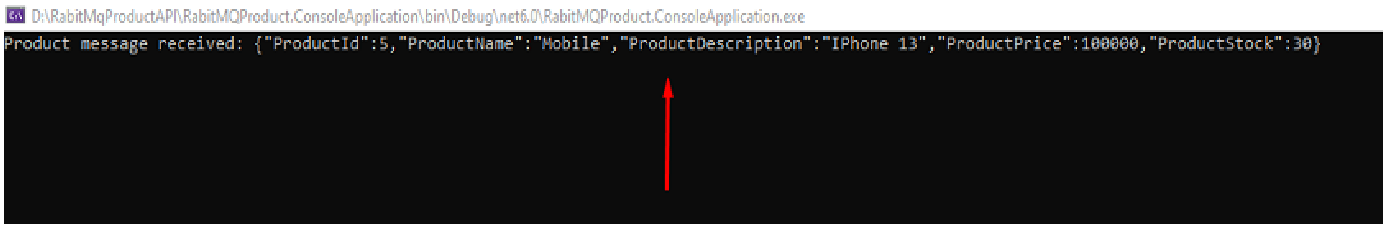
Enter the one product details and execute the API.



```
{
  "productId": 0,
  "productName": "Mobile",
  "productDescription": "IPhone 13",
  "productPrice": 100000,
  "productStock": 30
}
```

Step 10

When you execute the above API using swagger, then the message is sent in the queue. You can see inside the console window of the consumer the product details that he listened to from the queue immediately.


D:\RabitMqProductAPI\RabitMQProduct.ConsoleApplication\bin\Debug\net6.0\RabitMQProduct.ConsoleApplication.exe

```
Product message received: {"ProductId":5,"ProductName":"Mobile","ProductDescription":"IPhone 13","ProductPrice":100000,"ProductStock":30}
```

This is all about RabbitMQ, you may use this as per your requirement and purpose.

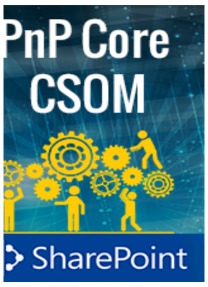
Conclusion

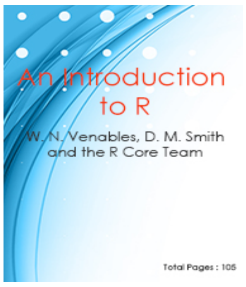
We discussed all the things related to RabbitMQ beginning from the introduction, working, and then set up a docker image and a few functionalities related to the producer and the consumer using the product application.

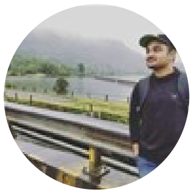
Happy learning!

.NET Core .NET Framework ASP.NET C# Docker

Next Recommended Reading
.NET Core Web API Logging Using NLog In RabbitMQ







Jaydeep Patil TOP 1000

C# Corner MVP | Fullstack Developer with 2.5+ yrs of experience in .NET Core API, Angular 8+, SQL Server, Docker, Kubernetics and Azure

https://www.linkedin.com/in/jaydeep-patil-b13a54134

516 97.2k 1

4 0

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

FEATURED ARTICLES

How To Receive Real-Time Data In .NET 6 Client Application Using SignalR

Another Tool For Analyzing RDB Files Of Redis

Making Faster API Calls Using Refit Rest Library

Microservice Architecture, Its Design Patterns And Considerations

SignalR Introduction And Implementation Using The .NET Core 6 Web API And Angular 14

TRENDING UP

01 Why SharePoint Framework (SPFx) Is Best for SharePoint Development

02 Getting Started With Angular Electron Application Development

03 Using List Data Structures In Python😋

04 Basic Authentication For Azure Functions (Open API) .Net 6

05 Typescript Express Node Server

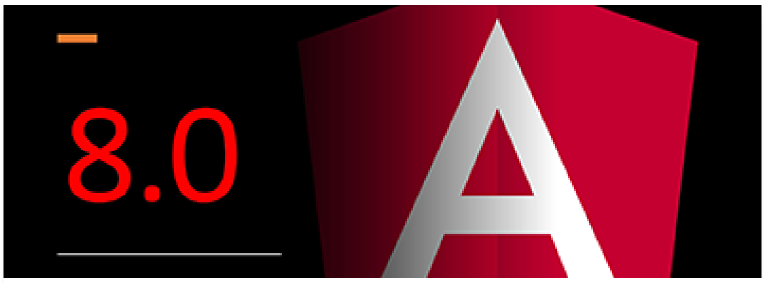
06 Onion Architecture In ASP.NET Core 6 Web API

07 The A - Z Guide Of SQL Views

08 Using Tuples Data Structure In Python 😋

09 JWT Token Authentication In Angular 14 And .NET Core 6 Web API

10 Introducing Carbon - Google's New Programming Language



CHALLENGE YOURSELF

Blockchain Basics Skill

Python Developer