

260389 Kamil Herbetko

243416 Jakub Szwedowicz

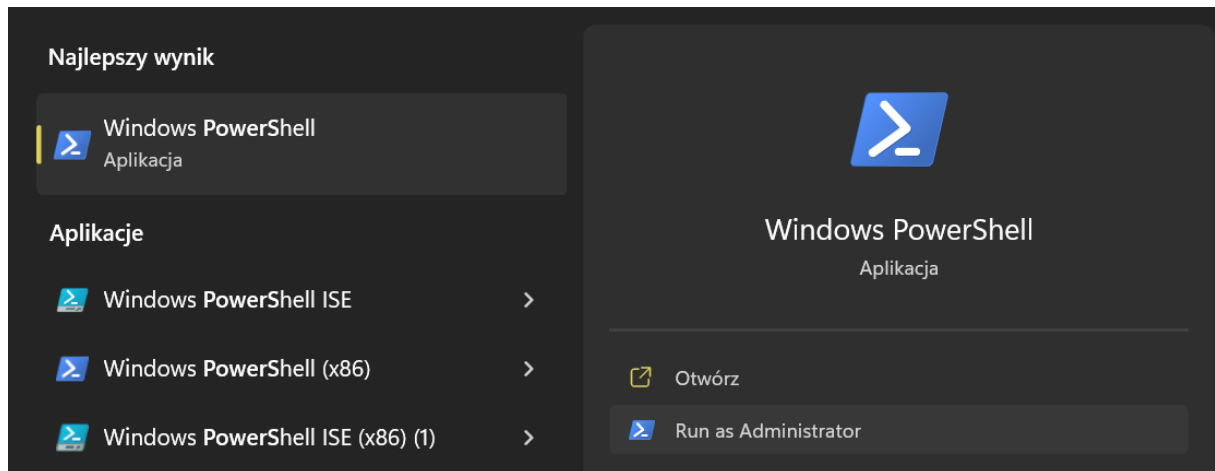
## Raport z listy nr 8 RSI

29.05.2023 r.

## 1. Konfiguracja RabbitMQ:

### 1.1. Instalacja Chocolatey.

W pierwszym kroku należy uruchomić powershell z uprawnieniami administratora.



Następnie sprawdź wynik polecenia `Get-ExecutionPolicy`:

```
PS C:\WINDOWS\system32> Get-ExecutionPolicy
Unrestricted
```

Jeśli wynikiem polecenia będzie napis „Restricted” to wówczas trzeba zmienić politykę wykonywania poprzez wywołanie komendy:

- `Set-ExecutionPolicy Bypass -Scope Process`

Po ustawieniu ExecutionPolicy na inną wartość niż „Restricted”, wywołaj komendę:

- `Set-ExecutionPolicy Bypass -Scope Process -Force;`  
`[System.Net.ServicePointManager]::SecurityProtocol =`  
`[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object`  
`System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))`

Następnie odczekaj kilka sekund aż komenda zakończy działanie. Jeśli nie będzie żadnych komunikatów o błędach to wpisz:

- `Choco`

Żeby się upewnić, że menadżer pakietów jest zainstalowany.

### 1.2. Instalacja RabbitMQ.

Jeśli Chocolatey z kroku 1.1 jest zainstalowane to wówczas można przejść do zainstalowania pakietu rabbitmq. W powershellu z uprawnieniami administratorскими wywołaj:

- `choco install rabbitmq`

## 2. Wygenerowanie dwóch projektów dla producenta i konsumenta w środowisku .NET

### 2.1. Najpierw trzeba zweryfikować czy ścieżka do .NET znajduje się w zmiennej środowiskowej PATH:

- `dotnet --help`

```
PS C:\WINDOWS\system32> dotnet --help
Użycie: dotnet [runtime-options] [path-to-application] [arguments]

Wykonaj aplikację platformy .NET.

runtime-options:
  --additionalprobingpath <path>    Ścieżka zawierająca zasady sondowania i zestawy do sondowania.
  --additional-deps <path>          Ścieżka do dodatkowego pliku deps.json.
  --depsfile <path>                 Ścieżka do pliku <aplikacja>.deps.json.
  --fx-version <version>            Wersja zainstalowanej platformy współużytkowanej służącej do uruchamiania aplikacji.
  --roll-forward <setting>          Przewiń do wersji platformy (LatestPatch, Minor, LatestMinor, Major, LatestMajor, Disable)
  --runtimeconfig <path>            Ścieżka do pliku <aplikacja>.runtimeconfig.json.

path-to-application:
  Ścieżka do pliku dll aplikacji do wykonania.

Użycie: dotnet [sdk-options] [command] [command-options] [arguments]

Wykonaj polecenie zestawu .NET SDK.

sdk-options:
  -d|--diagnostics Włącz diagnostyczne dane wyjściowe.
```

Polecenie powinno wygenerować pomoc dotyczącą użycia CLI dotnetu.

### 2.2. Następnie można przejść do generowania dwóch projektów. Żeby utworzyć projektu dla konsumenta i producenta to trzeba wywołać:

Dla producenta:

- `dotnet new console --name Send`
- `mv Send/Program.cs Send/Send.cs`

Dla klienta:

- `dotnet new console --name Receive`
- `mv Receive/Program.cs Receive/Receive.cs`

Utworzy to dwa osobne projekty, które będzie można następnie otworzyć przy użyciu Visual Studio.

### 2.3. W tym kroku należy dodać jeszcze zależności do obu projektów.

- `cd Send`
- `dotnet add package RabbitMQ.Client`
- `cd ../Receive`
- `dotnet add package RabbitMQ.Client`

```
PS C:\Users\kubas\Dev\University_RSI\lab5> ls

Directory: C:\Users\kubas\Dev\University_RSI\lab5

Mode                LastWriteTime         Length Name
----                -
d-----          29.05.2023    09:19             Receive
d-----          29.05.2023    09:18             Send
```

## Aplikacja klienta:

1. Utworzenie klasy Consumer, która będzie odbierać wiadomości.

```
public class Consumer
{
    private readonly string _exchangeName;
    private readonly string _queueName;
    private readonly string _routingKey;
    private readonly string _connectionString;
    private readonly ConnectionFactory _factory;
    private readonly IConnection _connection;
    private readonly IModel _channel;
    private readonly EventingBasicConsumer _consumer;

    public Consumer(string exchangeName, string queueName, string routingKey,
string connectionString)
    {
        _exchangeName = exchangeName;
        _queueName = queueName;
        _routingKey = routingKey;
        _connectionString = connectionString;
        _factory = new ConnectionFactory
        {
            HostName = "localhost"
        };

        _connection = _factory.CreateConnection();
        _channel = _connection.CreateModel();
        _channel.ExchangeDeclare(exchange: _exchangeName, type:
ExchangeType.Fanout);
        _queueName = _channel.QueueDeclare().QueueName;
        _channel.QueueBind(queue: _queueName, exchange: exchangeName, routingKey:
string.Empty);
        _consumer = new EventingBasicConsumer(_channel);
        _consumer.Received += (model, args) =>
        {
            var body = args.Body.ToArray();
            var message = Encoding.UTF8.GetString(body);
            Console.WriteLine("Message received: {0}", message);
        };
    }

    public void StartConsuming()
    {
        _channel.BasicConsume(queue: _queueName, autoAck: true, consumer:
_consumer);

        Console.WriteLine("Consumer started. Waiting for messages...");
        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
    }
}
```

2. Utworzenie głównej pętli programu, w której konsument będzie przetwarzał wiadomości:

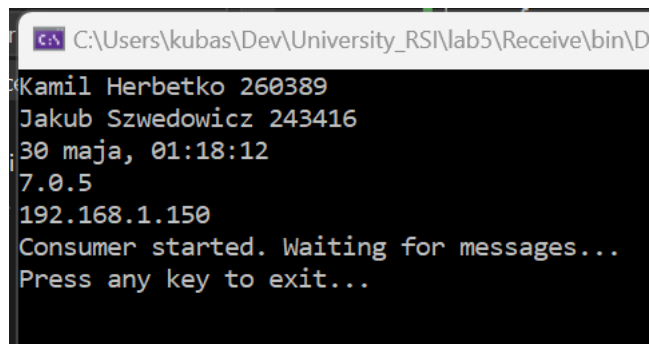
```
internal class Program
{
    static void Main()
    {
        MyData.Info();
        string exchangeName = "my_exchange";
        string queueName = "my_queue";
        string routingKey = "my_routing_key";
        string connectionString = "amqp://guest:guest@localhost:5672";

        Consumer consumer = new Consumer(exchangeName, queueName, routingKey,
connectionString);
        consumer.StartConsuming();

        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
    }
}
```

W obecnej implementacji routingKey oraz connectionString są ignorowane. Zamiast nich w konstruktorze zahardcodowane jest tworzenie połączenia przez fabrykę dla całego hosta: „localhost”.

Uruchom klienta (Ctrl + F5)



```
C:\Users\kubas\Dev\University_RSI\lab5\Receive\bin\Debug
Kamil Herbetko 260389
Jakub Szwedowicz 243416
30 maja, 01:18:12
7.0.5
192.168.1.150
Consumer started. Waiting for messages...
Press any key to exit...
```

## Aplikacja serwera:

1. Utworzenie klasy Producer, która będzie nadawać wiadomości.

```
public class Publisher
{
    private readonly string _publisherName;
    private readonly string _exchangeName;
    private readonly string _routingKey;
    private readonly string _connectionString;
    private readonly ConnectionFactory _factory;
    private readonly IConnection _connection;
    private readonly IModel _channel;

    public Publisher(string publisherName, string exchangeName, string
routingKey, string connectionString)
    {
        _publisherName = publisherName;
        _exchangeName = exchangeName;
        _routingKey = routingKey;
        _connectionString = connectionString;
        _factory = new ConnectionFactory
        {
            HostName = "localhost"
        };
        _connection = _factory.CreateConnection();
        _channel = _connection.CreateModel();
        _channel.ExchangeDeclare(exchange: _exchangeName, type:
ExchangeType.Direct);
    }

    public void PublishMessage(string message)
    {
        var body = Encoding.UTF8.GetBytes(message);
        _channel.BasicPublish(exchange: _exchangeName, routingKey: string.Empty,
basicProperties: null, body: body);
        Console.WriteLine("publishing message: {0}", message);
    }

    public void PublishMessageWithName(string message)
    {
        var body = Encoding.UTF8.GetBytes(message);
        _channel.BasicPublish(exchange: _exchangeName, routingKey: string.Empty,
basicProperties: null, body: body);
        Console.WriteLine(_publisherName + " - " + "publishing message: {0}",
message);
    }
}
```

2. Utworzenie głównej pętli programu, w której producent będzie nadawał wiadomości:

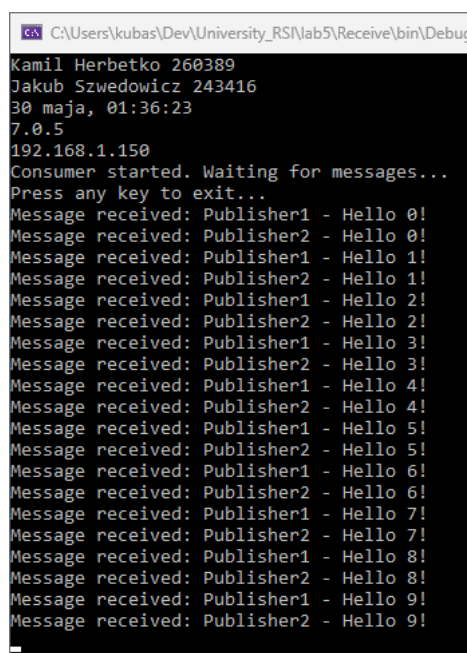
```
internal class Program
{
    static void Main()
    {
        MyData.Info();
        int max = 10;
        string exchangeName = "my_exchange";
        string routingKey = "my_routing_key";
        string connectionString = "amqp://guest:guest@localhost:5672";

        Publisher publisher1 = new Publisher("Publisher1", exchangeName,
routingKey, connectionString);
        Publisher publisher2 = new Publisher("Publisher2", exchangeName,
routingKey, connectionString);

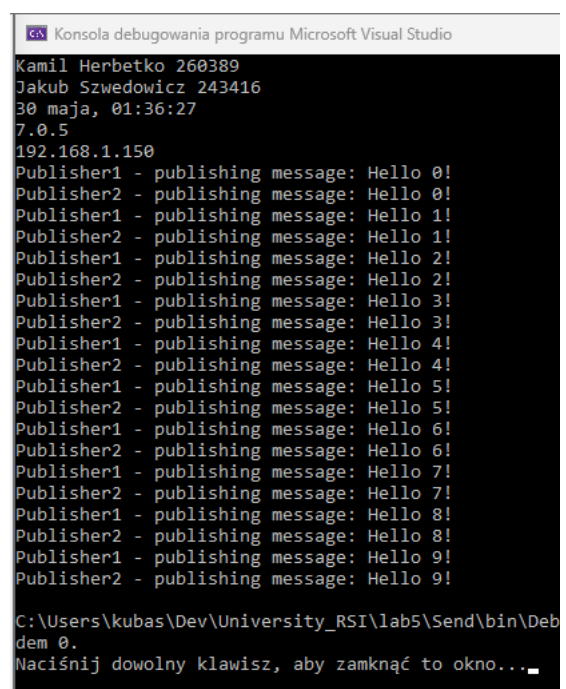
        var publishers = new List<Publisher>() { publisher1, publisher2 };
        var rand = new Random();
        for (int counter = 0; counter < max; counter++)
        {
            foreach (Publisher publisher in publishers)
            {
                string message = "Hello " + counter + "!";
                publisher.PublishMessageWithName(message);
                Thread.Sleep(rand.Next(1000, 3000));
            }
        }
    }
}
```

### Wynik programu:

Zgodnie z wymaganiami wiadomości wysyłać będą naprzemiennie dwaj producenci o różnych nazwach: „Publisher1” oraz „Publisher2”. Komunikaty będą nadawane w zmiennych interwałach od 1s do 3 sekund. W sumie wysłane zostanie 20 wiadomości, po 10 na każdego Publishera.



```
C:\Users\kubas\Dev\University_RSI\lab5\Receive\bin\Debug
Kamil Herbetko 260389
Jakub Szwedowicz 243416
30 maja, 01:36:23
7.0.5
192.168.1.150
Consumer started. Waiting for messages...
Press any key to exit...
Message received: Publisher1 - Hello 0!
Message received: Publisher2 - Hello 0!
Message received: Publisher1 - Hello 1!
Message received: Publisher2 - Hello 1!
Message received: Publisher1 - Hello 2!
Message received: Publisher2 - Hello 2!
Message received: Publisher1 - Hello 3!
Message received: Publisher2 - Hello 3!
Message received: Publisher1 - Hello 4!
Message received: Publisher2 - Hello 4!
Message received: Publisher1 - Hello 5!
Message received: Publisher2 - Hello 5!
Message received: Publisher1 - Hello 6!
Message received: Publisher2 - Hello 6!
Message received: Publisher1 - Hello 7!
Message received: Publisher2 - Hello 7!
Message received: Publisher1 - Hello 8!
Message received: Publisher2 - Hello 8!
Message received: Publisher1 - Hello 9!
Message received: Publisher2 - Hello 9!
```



```
Konsola debugowania programu Microsoft Visual Studio
Kamil Herbetko 260389
Jakub Szwedowicz 243416
30 maja, 01:36:27
7.0.5
192.168.1.150
Publisher1 - publishing message: Hello 0!
Publisher2 - publishing message: Hello 0!
Publisher1 - publishing message: Hello 1!
Publisher2 - publishing message: Hello 1!
Publisher1 - publishing message: Hello 2!
Publisher2 - publishing message: Hello 2!
Publisher1 - publishing message: Hello 3!
Publisher2 - publishing message: Hello 3!
Publisher1 - publishing message: Hello 4!
Publisher2 - publishing message: Hello 4!
Publisher1 - publishing message: Hello 5!
Publisher2 - publishing message: Hello 5!
Publisher1 - publishing message: Hello 6!
Publisher2 - publishing message: Hello 6!
Publisher1 - publishing message: Hello 7!
Publisher2 - publishing message: Hello 7!
Publisher1 - publishing message: Hello 8!
Publisher2 - publishing message: Hello 8!
Publisher1 - publishing message: Hello 9!
Publisher2 - publishing message: Hello 9!

C:\Users\kubas\Dev\University_RSI\lab5\Send\bin\Debug
dem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```