

# **Rozproszone Systemy Informatyczne**

Raport - Ćwiczenie 6b

Paweł Kluska, 260391

Katsiaryna Ziatsikava, 245891

## Aplikacja

Aplikacja składa się z 2 producentów i jednego konsumenta. Producenci zostały zbudowane z użyciem .NET 6/ C#. Konsument został zbudowany z użyciem Javy. Aby uruchomić aplikację, dodaliśmy bibliotekę RabbitMQ Java Client i .NET/C# RabbitMQ Client Library, które dostarczają niezbędne klasy do obsługi komunikacji z RabbitMQ.

Producent miał działać wg następujących założeń. Definiujemy 2 producentów, których uruchamiamy jednocześnie. Wysyłają oni wiadomości do brokera RabbitMq. Pierwszy producent działa 10 sekund i ma przedział czasowy od 1 do 2 sekund, z którego losujemy liczbę, którą musi odczekać producent pomiędzy wysłaniem każdej z wiadomości. Drugi producent działa 12 sekund i ma przedział czasowy od 2 do 3 sekund. Te przedziały różnią się pomiędzy senderami, jednak mogą one zachodzić na siebie. Dzięki temu może się zdarzyć, że jeden z senderów wyśle 2 wiadomości pod rząd.

Senderzy podłączają się do brokera, który jest uruchomiony na trzeciej maszynie. Przy konfiguracji połączenia zostały podane: adres ip, nazwa użytkownika, hasło, oraz nazwa kolejki, do której sender chce wysłać wiadomości.

Kod **producenta 1** wygląda następująco:

```
1  using System;
2  using System.Text;
3  using RabbitMQ.Client;
4  using System.Threading;
5  using Newtonsoft.Json;
6
7  public class Program
8  {
9      private const int DurationSeconds = 10;
10     private const string EndMarkerMessage = "EndMarker";
11
12
13     public static void Main()
14     {
15         MyData.info();
16
17         var factory = new ConnectionFactory { HostName = "10.182.6.242", Port = 5672, UserName = "guest", Password = "guest" };
18         // var factory = new ConnectionFactory { HostName = "localhost" };
19
20         using var connection = factory.CreateConnection();
21         using var channel = connection.CreateModel();
22
23         DateTime endTime = DateTime.Now.AddSeconds(DurationSeconds);
24         int counter = 0;
25         while (DateTime.Now < endTime)
26         {
27
28             string message = JsonConvert.SerializeObject(new { name = "Pawel", time = DateTime.Now.TimeOfDay, counter = counter++ });
29
30             var body = Encoding.UTF8.GetBytes(message);
31
32             channel.BasicPublish(exchange: "",
33                                 routingKey: "01",
34                                 basicProperties: null,
35                                 body: body);
36
37             Console.WriteLine($" [x] Sent {message}");
38
39             Random rnd = new Random();
40             int sleep = rnd.Next(1000, 2000);
41             Thread.Sleep(sleep);
42         }
43     }
```

```

44     var endMarkerBody = Encoding.UTF8.GetBytes(EndMarkerMessage);
45     channel.BasicPublish(exchange: "",
46                         routingKey: "01",
47                         basicProperties: null,
48                         body: endMarkerBody);
49
50     Console.WriteLine($" [x] Sent end marker '{EndMarkerMessage}'");
51
52     Console.WriteLine(" Press [enter] to exit.");
53     Console.ReadLine();
54 }
55

```

Kod **producenta 2** wygląda następująco:

```

1  using System;
2  using System.Text;
3  using RabbitMQ.Client;
4  using System.Threading;
5  using Newtonsoft.Json;
6
7  public class Program
8  {
9      private const int DurationSeconds = 12;
10     private const string EndMarkerMessage = "EndMarker";
11
12
13     public static void Main()
14     {
15         MyData.info();
16
17         var factory = new ConnectionFactory { HostName = "10.182.6.242", Port = 5672, UserName = "guest", Password = "guest" };
18         // var factory = new ConnectionFactory { HostName = "localhost" };
19
20         using var connection = factory.CreateConnection();
21         using var channel = connection.CreateModel();
22
23         DateTime endTime = DateTime.Now.AddSeconds(DurationSeconds);
24         int counter = 0;
25         while (DateTime.Now < endTime)
26         {
27             string message = JsonConvert.SerializeObject(new { name = "Katya", time = DateTime.Now.TimeOfDay, counter = counter++ });
28
29             var body = Encoding.UTF8.GetBytes(message);
30
31             channel.BasicPublish(exchange: "",
32                                routingKey: "01",
33                                basicProperties: null,
34                                body: body);
35
36             Console.WriteLine($" [x] Sent {message}");
37
38             Random rnd = new Random();
39             int sleep = rnd.Next(2000, 3000);
40             Thread.Sleep(sleep);
41         }
42     }
43
44     var endMarkerBody = Encoding.UTF8.GetBytes(EndMarkerMessage);
45     channel.BasicPublish(exchange: "",
46                         routingKey: "01",
47                         basicProperties: null,
48                         body: endMarkerBody);
49
50     Console.WriteLine($" [x] Sent end marker '{EndMarkerMessage}'");
51
52     Console.WriteLine(" Press [enter] to exit.");
53     Console.ReadLine();
54 }
55

```

Kod obu nadawców jest niemal identyczny, różnią się oni tylko przedziałami czasowymi w których wysyłają wiadomości. Do brokera przesyłany jest cały obiekt, składający się z 3 elementów: imienia, daty oraz licznika. W celu przesłania obiektu do klienta został on przekonwertowany najpierw do obiektu Json. Została tu użyta biblioteka Newtonsoft.Json. Po upływie zdefiniowanego wcześniej czasu zostanie wysłany marker końca przesyłania - jest to już zwykła wiadomość - string. W pierwszej linijce maina jest wywołana metoda info() z klasy MyData.

Kod konsumenta wygląda następująco:

```
public class Recv {  
  
    2 usages  
    private final static String QUEUE_NAME = "01";  
  
    2 usages  
    private final static int NO_SENDERS = 2;  
  
    2 usages  
    private static int endMarkerCount = 0;  
  
    no usages  
    public static void main(String[] argv) throws Exception {  
        MyData.info();  
        ConnectionFactory factory = new ConnectionFactory();  
        factory.setHost("10.182.6.242");  
        factory.setPort(5672);  
        factory.setUsername("guest");  
        factory.setPassword("guest");  
        Connection connection = factory.newConnection();  
        Channel channel = connection.createChannel();  
  
        channel.queueDeclare(QUEUE_NAME, b: false, b1: false, b2: false, map: null);  
        System.out.println(" [*] Waiting for messages. To exit press CTRL+C");  
  
        DeliverCallback deliverCallback = (consumerTag, delivery) -> {  
            String message = new String(delivery.getBody(), StandardCharsets.UTF_8);  
            System.out.println(" [x] Received '" + message + "'");  
  
            if (message.equals("EndMarker")) {  
                endMarkerCount++;  
                if (endMarkerCount >= NO_SENDERS) {  
                    System.out.println("Received " + NO_SENDERS + " end markers. Exiting...");  
                    channel.basicCancel(consumerTag);  
                    try {  
                        channel.close();  
                        connection.close();  
                    } catch (Exception e) {  
                        e.printStackTrace();  
                    }  
                }  
            } else {  
                Gson gson = new GsonBuilder()  
                    .registerTypeAdapter(LocalTime.class, new LocalTimeAdapter())  
                    .create();  
                Message messageDeserialized = gson.fromJson(message, Message.class);  
  
                System.out.println("Name: " + messageDeserialized.getName());  
                System.out.println("Time: " + messageDeserialized.getTime());  
                System.out.println("Counter: " + messageDeserialized.getCounter());  
            }  
        };  
        channel.basicConsume(QUEUE_NAME, b: true, deliverCallback, consumerTag -> {  
        });  
    }  
}
```

Kod implementuje konsumenta, który oczekuje na wiadomości w kolejce RabbitMQ. Została ustawiona odpowiednia nazwa kolejki i liczba nadawców. Zostało również skonfigurowane połączenie z RabbitMQ, ustawiony adres hosta, port, login i hasło. Po nawiązaniu połączenia można odbierać wiadomości. Za pomocą DeliverCallback wiadomość jest odbierana i wyświetlana w konsoli. Wiadomość jest konwertowana z JSONa do obiektu Javy - Message, następnie jego parametry są wyświetlone w konsoli. Jeśli jednak przesyłany jest znacznik końca od nadawców dodajemy wartość do licznika. Gdy wszyscy nadawcy prześlą znacznik końca, zamyka się kanał i połączenie. W pierwszej linijce maina jest wywołana metoda info() z klasy MyData.

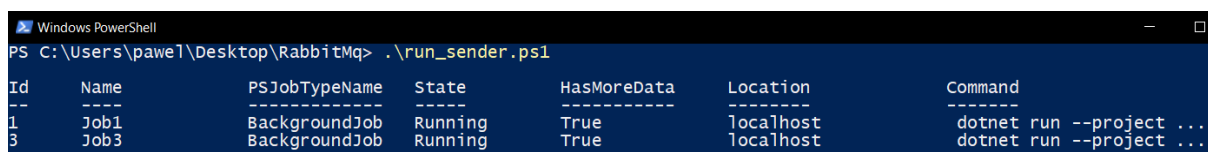
## Działanie aplikacji

Aplikacja została uruchomiona w konfiguracji trój maszynowej: producenci są uruchomieni na jednej maszynie, konsument na drugiej, serwer RabbitMQ był postawiony na trzeciej maszynie. Wszystkie maszyny były podłączone do tej samej sieci. Aby podłączyć się do serwera RabbitMQ w kodzie był ustawiony adres ip - 10.182.6.242, port - 5672, nazwa użytkownika i hasło oraz nazwa kolejki.

Producenci zostali uruchomieni jednocześnie przy pomocy skryptu powershell (Wystartowanie 2 procesów w tle)

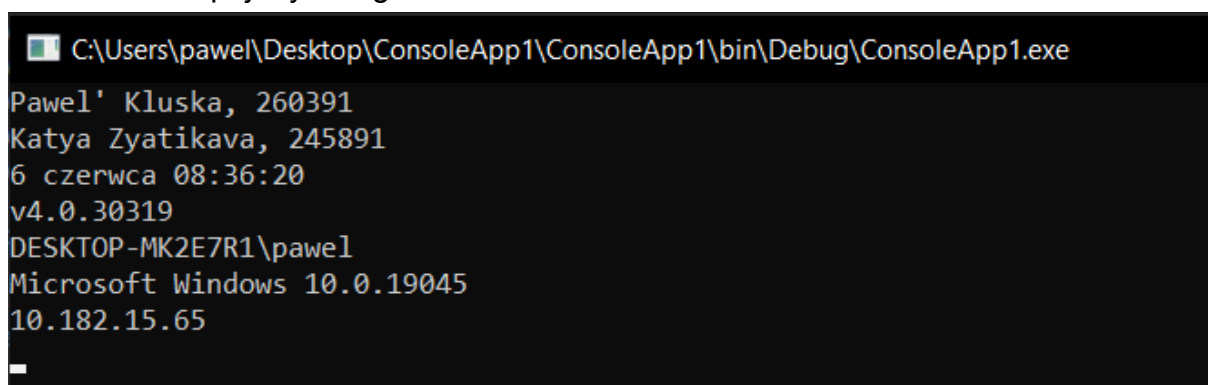
```
run_sender.ps1 x
1  #!/usr/bin/env powershell
2
3  Start-Job -ScriptBlock { dotnet run --project "C:\Users\pawel\Desktop\RabbitMq\Send\" }
4  Start-Job -ScriptBlock { dotnet run --project "C:\Users\pawel\Desktop\RabbitMq\Send2\" }
5
```

## Uruchomienie skryptu senderów



Id	Name	PSJobTypeName	State	HasMoreData	Location	Command
1	Job1	BackgroundJob	Running	True	localhost	dotnet run --project ...
3	Job3	BackgroundJob	Running	True	localhost	dotnet run --project ...

## Uruchomienie pojedynczego sendera



```
C:\Users\pawel\Desktop\ConsoleApp1\ConsoleApp1\bin\Debug\ConsoleApp1.exe
Pawel' Kluska, 260391
Katya Zyatikava, 245891
6 czerwca 08:36:20
v4.0.30319
DESKTOP-MK2E7R1\pawel
Microsoft Windows 10.0.19045
10.182.15.65
```

Działanie **konsumenta** prezentuje się następująco:

```
Paweł Kluska, 260391
Katya Zyatikava, 245891
6 czerwca 08:36:31
18.0.2
katya
Windows 10
10.182.14.61
[*] Waiting for messages. To exit press CTRL+C
[x] Received '{"name":"Paweł","time":"08:36:38.2031810","counter":0}'
Name: Paweł
Time: 08:36:38.203181
Counter: 0
[x] Received '{"name":"Paweł","time":"08:36:39.7893160","counter":1}'
Name: Paweł
Time: 08:36:39.789316
Counter: 1
[x] Received '{"name":"Katya","time":"08:36:41.2438020","counter":0}'
Name: Katya
Time: 08:36:41.243802
Counter: 0
[x] Received '{"name":"Paweł","time":"08:36:41.5627130","counter":2}'
Name: Paweł
Time: 08:36:41.562713
Counter: 2
```

```
[x] Received '{"name":"Paweł","time":"08:36:42.6879495","counter":3}'
Name: Paweł
Time: 08:36:42.687949500
Counter: 3
[x] Received '{"name":"Paweł","time":"08:36:43.8764332","counter":4}'
Name: Paweł
Time: 08:36:43.876433200
Counter: 4
[x] Received '{"name":"Katya","time":"08:36:44.1105213","counter":1}'
Name: Katya
Time: 08:36:44.110521300
Counter: 1
[x] Received '{"name":"Paweł","time":"08:36:45.6595601","counter":5}'
Name: Paweł
Time: 08:36:45.659560100
Counter: 5
[x] Received '{"name":"Katya","time":"08:36:47.0008759","counter":2}'
Name: Katya
Time: 08:36:47.000875900
Counter: 2
[x] Received '{"name":"Paweł","time":"08:36:47.3896513","counter":6}'
Name: Paweł
Time: 08:36:47.389651300
Counter: 6
```

```
[x] Received 'EndMarker'  
[x] Received '{"name":"Katya","time":"08:36:49.5288487","counter":3}'  
Name: Katya  
Time: 08:36:49.528848700  
Counter: 3  
[x] Received '{"name":"Katya","time":"08:36:52.2959706","counter":4}'  
Name: Katya  
Time: 08:36:52.295970600  
Counter: 4  
[x] Received 'EndMarker'  
Received 2 end markers. Exiting...
```