

**WYDZIAŁ  
ELEKTROTECHNIKI  
I INFORMATYKI**  
POLITECHNIKI RZESZOWSKIEJ

**Paweł Wiącek**

RabbitMQ

**Projekt**

Usługi sieciowe w biznesie

Prowadzący:

Dr inż. Mariusz Borkowski

Rzeszów, 2022

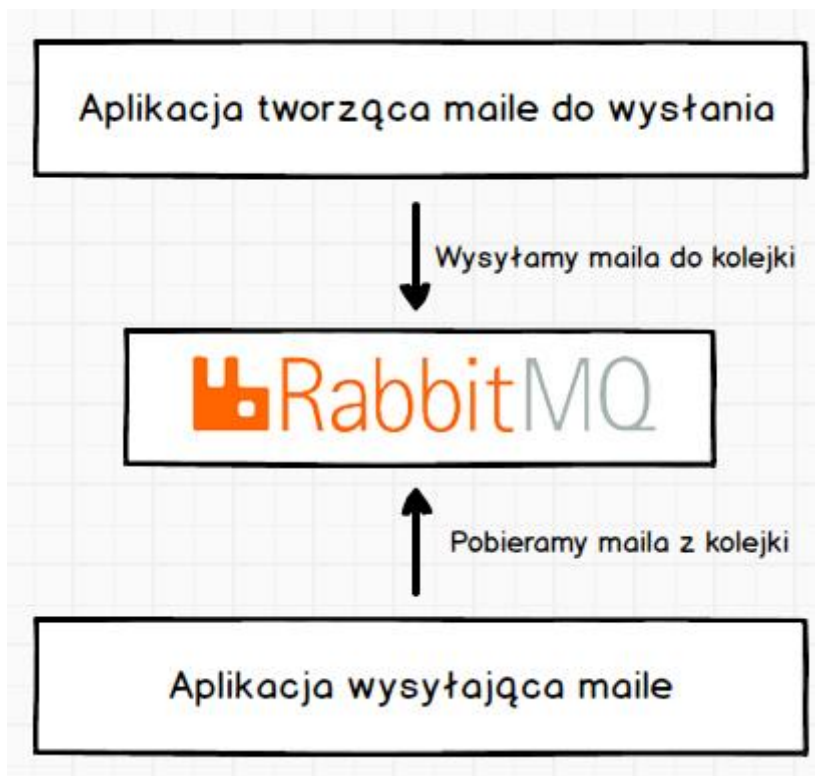
## Co to jest RabbitMQ?

RabbitMQ jest to zestaw narzędzi dystrybucji komunikatów na kolejki, które pozwalają na obsługę, zarządzanie dystrybucją oraz monitorowanie na kolejkach. Jest najbardziej rozpowszechnionym brokerem wiadomości o otwartym kodzie źródłowym. Pierwsza wersja RabbitMQ została w języku programowania Erlang i została wydana w 2007 roku, a od 2013 jest rozwijana przez Pivotal Software.

RabbitMQ jest oparty na protokole AMQP (Advanced Message Queuing Protocol). AMQP określa zachowanie usługi oraz klienta komunikacji w stopniu, który powoduje, że implementacje różnych dostawców są interoperacyjne.

AMQP to protokół ramek i transferu. Ramka oznacza, że zapewnia strukturę dla strumieni danych binarnych, które przepływa w dowolnym kierunku połączenia sieciowego. Struktura zapewnia rozdyskowanie odrębnych bloków danych, nazywanych ramkami, które mają być wymieniane między połączonymi stronami. Możliwości transferu zapewniają, że obie komunikujące się strony mogą ustalić wspólne informacje o tym, kiedy mają być przesyłane ramki i kiedy transfery należy uznać za ukończone.

RabbitMQ to narzędzie multiplatformowe i oficjalnie wspiera większość popularnych technologii i języków (JMS, Java, Spring, .NET, Python, Node.js, PHP, Ruby). Jego serwer można uruchomić na najpopularniejszych systemach operacyjnych takich jak: Windows, Linux czy MacOS. Serwer RabbitMQ można również postawić na chmurach takich jak Amazon AWS, Cloud Foundry czy Pivotal Cloud Foundry. Istnieje także gotowe rozwiązanie udostępniające RabbitMQ w chmurze czyli CloudAMQP.s



Uproszczony schemat działania RabbitMQ

## Jak wygląda architektura RabbitMQ?

Podstawowe pojęcia:

- Message – informacja wygenerowana przez producenta opakowana w dane niezbędne do obsługi przez RabbitMQ,
- Exchange – broker komunikatu, pierwszy “filtr” komunikatu
- Queue – bufor komunikatów, może być zapisywana albo in memory
- Routing-key – adres komunikatu,
- Binding – definicja mówiąca na jaką kolejkę (lub kolejki) ma trafić komunikat po trafieniu na dany exchange z danym routing-key (lub innymi atrybutami komunikatu)

Typy exchange:

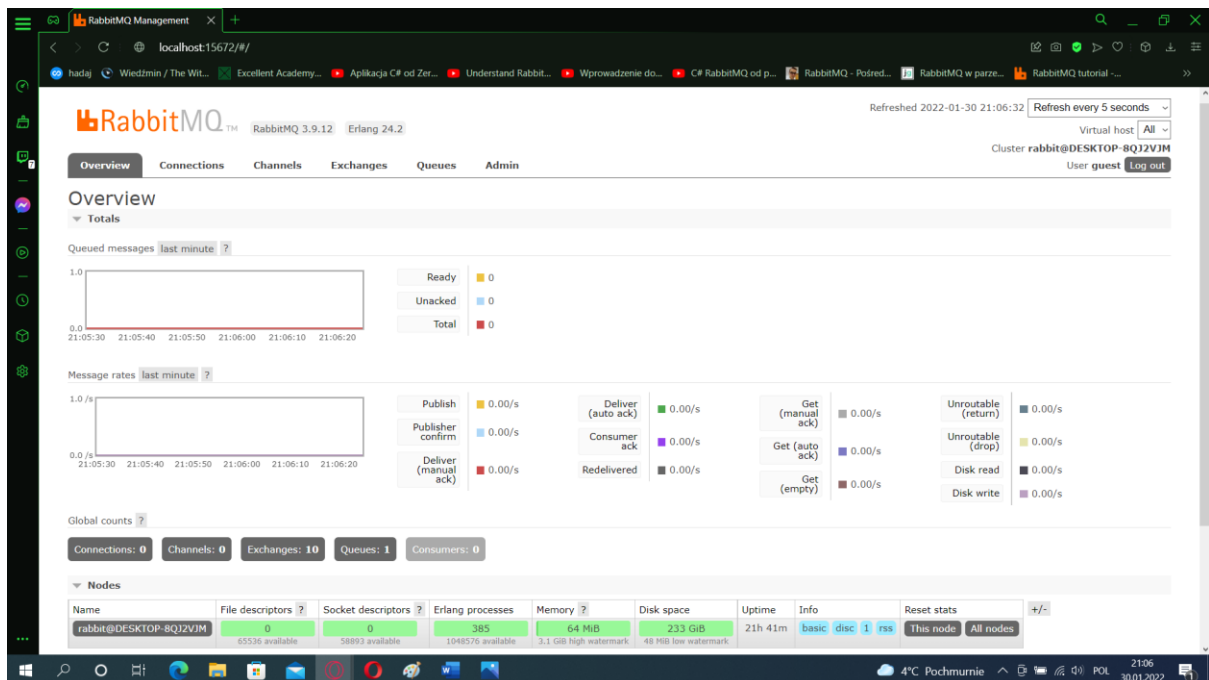
- Direct exchange – exchange dopasowuje routing key do kolejek na podstawie definicji binding (routing key musi być równy temu w definicji) i w ten sposób je propaguje,
- Fanout exchange – nie brany jest pod uwagę routing-key, komunikat po wejściu na exchange trafia na wszystkie kolejki,

- Topic exchange – exchange dopasowuje routing key do kolejek (routing key musi pasować do definicji) na podstawie definicji binding i w ten sposób je propaguje, można używać w bindingu \* i # celem dopasowania,
- Headers exchange – nie brany jest pod uwagę routing-key, komunikaty są propagowane na podstawie atrybutów w nagłówku, które mogą być dowolnego typu (string, liczba).

**Atrybuty** są to dodatkowe elementy, które pozwalają na zidentyfikowanie odbiorców oraz sposobu ich dostarczenia. Atrybuty są ustawiane w momencie publikacji wiadomości, a następnie usuwane po ich wykorzystaniu na potrzeby routingu. Możemy wyróżnić następujące atrybuty wiadomości:

- **content type** - typ danych,
- **content encoding** - kodowanie danych,
- **routing key** - klucz routing-u,
- **delivery mode** - sposób dostarczenia wiadomości określający, czy wiadomość ma być utrwalona (ang. persistent),
- **message priority** - priorytet wiadomości,
- **message publishing timestamp** - czas publikacji wiadomości,
- **expiration period** - inaczej **TTL** czyli opóźnienie w dostarczeniu wiadomości podawane w milisekundach,
- **publisher application id** - identyfikator producenta,

Na serwer RabbitMQ można się zalogować poprzez przeglądarkę. Jeżeli pracujemy na swoim stanowisku to możemy się zalogować poprzez konto guest z takim samym hasłem.



The screenshot shows the RabbitMQ Management interface at localhost:15672, specifically the 'Exchanges' tab. It displays a table of all exchanges (7) with columns: Name, Type, Features, Message rate in, and Message rate out. The table lists several exchanges, including the default AMQP exchange and various built-in exchanges like amq.direct, amq.fanout, amq.headers, amq.match, amq.rabbitmq.trace, and amq.topic. Below the table is a link to 'Add a new exchange' and a footer with links to HTTP API, Server Docs, Tutorials, Community Support, Community Slack, Commercial Support, Plugins, Github, and Changelog.

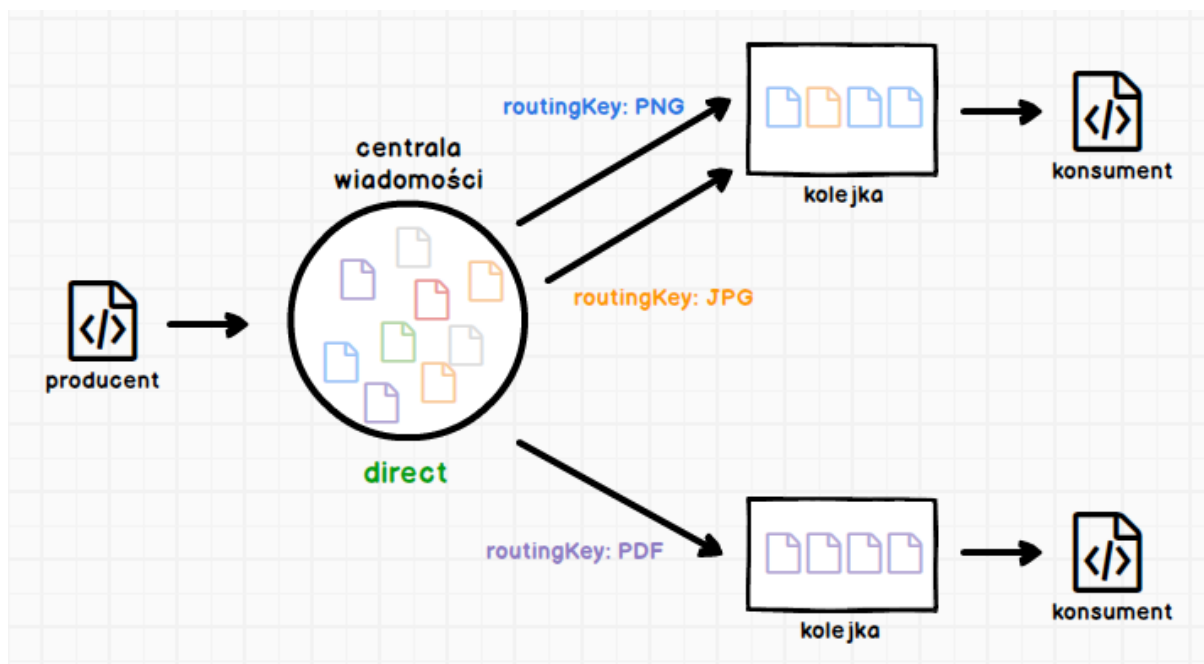
Name	Type	Features	Message rate in	Message rate out
(AMQP default)	direct	D		
amq.direct	direct	D		
amq.fanout	fanout	D		
amq.headers	headers	D		
amq.match	headers	D		
amq.rabbitmq.trace	topic	D I		
amq.topic	topic	D		

Utworzenie kolejki z atrybutami.

```
channel.QueueDeclare(queue: "Email",  
    durable: false,  
    exclusive: false,  
    autoDelete: false,  
    arguments: null);
```

Centrala wiadomości bezpośrednich (ang. direct exchange)

Bardzo często wykorzystywany typ centrali ze względu na prostotę działania. Centrala wiadomości na podstawie parametru `routingKey` wysyła wiadomości do określonej kolejki.

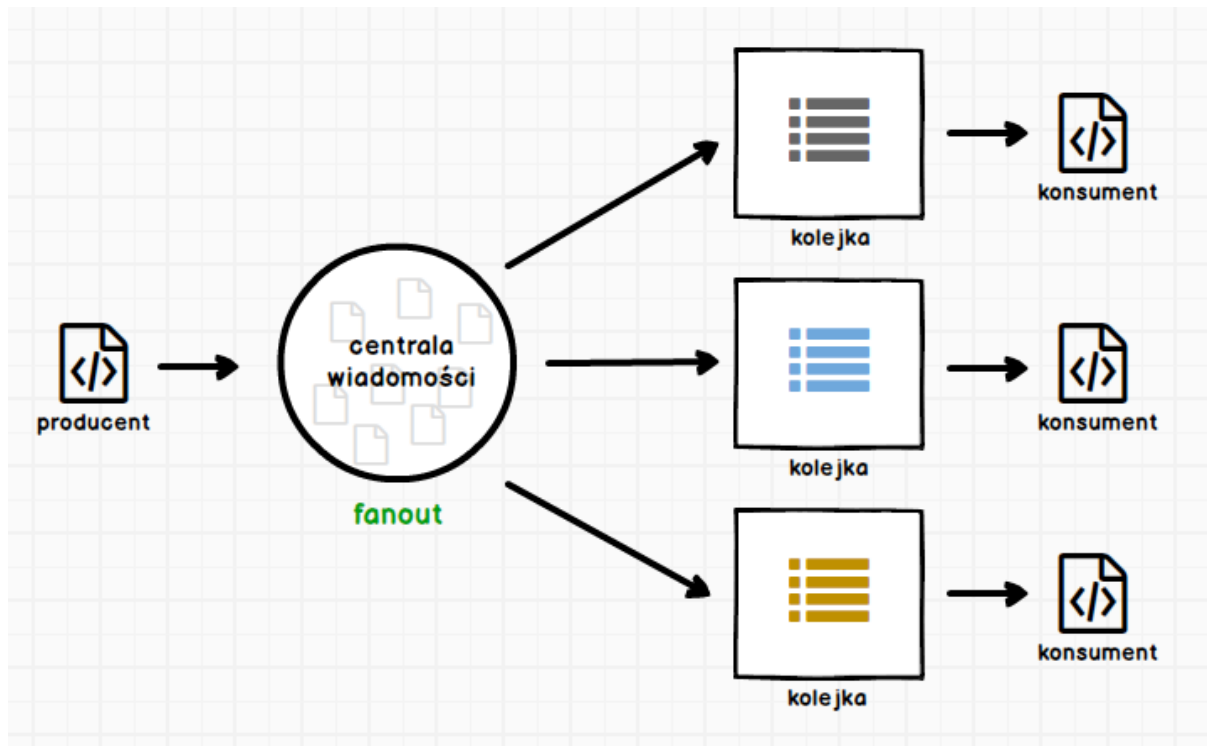


Deklaracja exchange na typ 'direct'.

```
channel.ExchangeDeclare(exchange: "direct_email", type: "direct");
```

Centrala rozgłośni wiadomości (ang. fanout exchange)

Najprostszy typ ze wszystkich dostępnych. Jego działanie sprowadza się do wysyłania wiadomości do wszystkich aktywnych, tymczasowych kolejek stworzonych przez konsumentów.

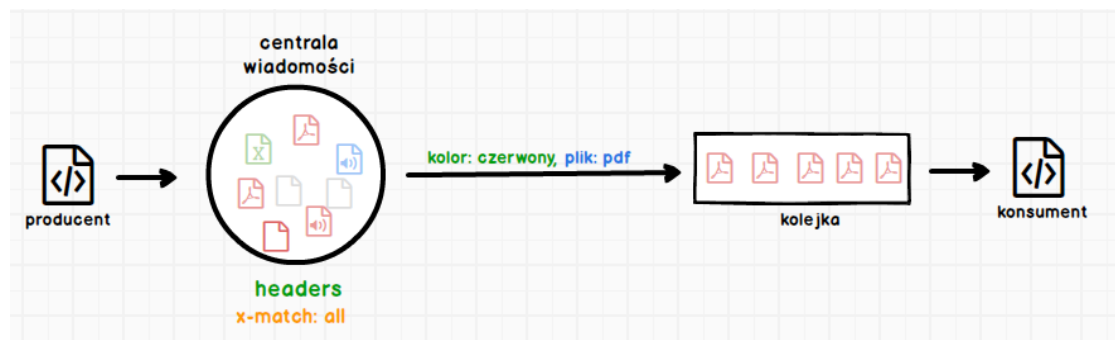


Deklaracja exchange na typ 'fanout'.

```
channel.ExchangeDeclare(exchange: "Email", type: "fanout");
```

## Centrala wiadomości z nagłówkami (ang. headers exchange)

Można powiedzieć że jest to rozbudowana wersja typu direct, z tą różnicą, że nie wykorzystywany jest parametr routingKey, a wykorzystywane są nagłówki. Dzięki wykorzystaniu nagłówków możemy stosować liczby, łańcuchy znaków oraz wyniki działania funkcji skrótu (ang. hash). Dodatkowo wykorzystując nagłówek x-match mamy kontrolę nad sposobem weryfikacji nagłówków.

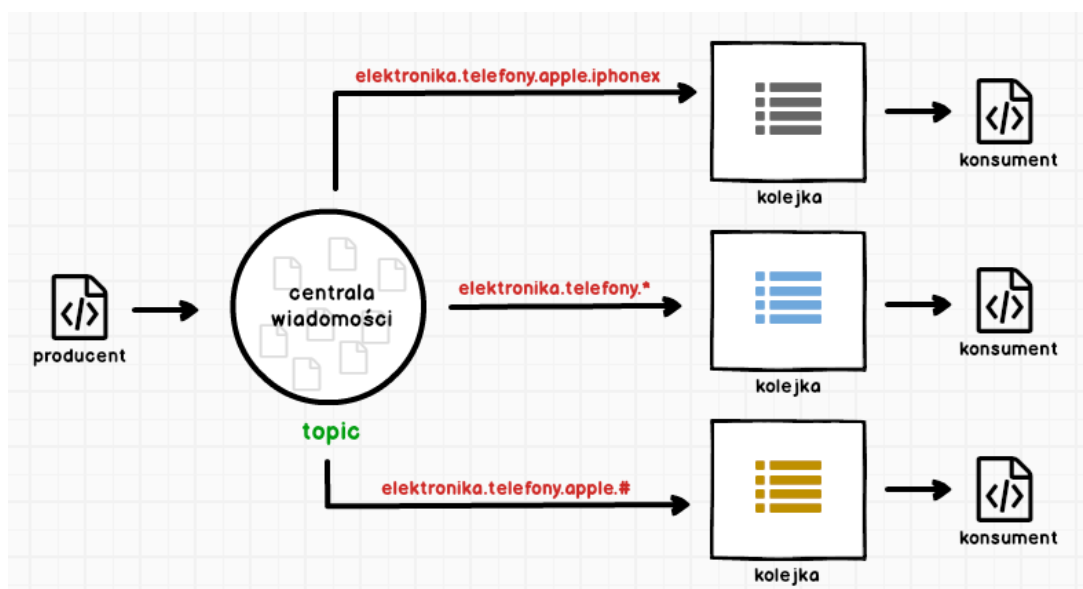


## Centrala wiadomości tematycznych (ang. topic exchange)

Ostatni typ centrali bazujący na rozwinięciu parametru routingKey. W przypadku tej centrali wiadomości parametr ten musi składać się ze słów oddzielonych kropkami. Pojedyncze słowa mogą zostać zastąpione przez:

- \* (nag. asterix), która zastąpi dokładnie jedno słowo,

- # (ang. hash), który zastępuje dowolną liczbę słów oraz oddzielające je kropki,



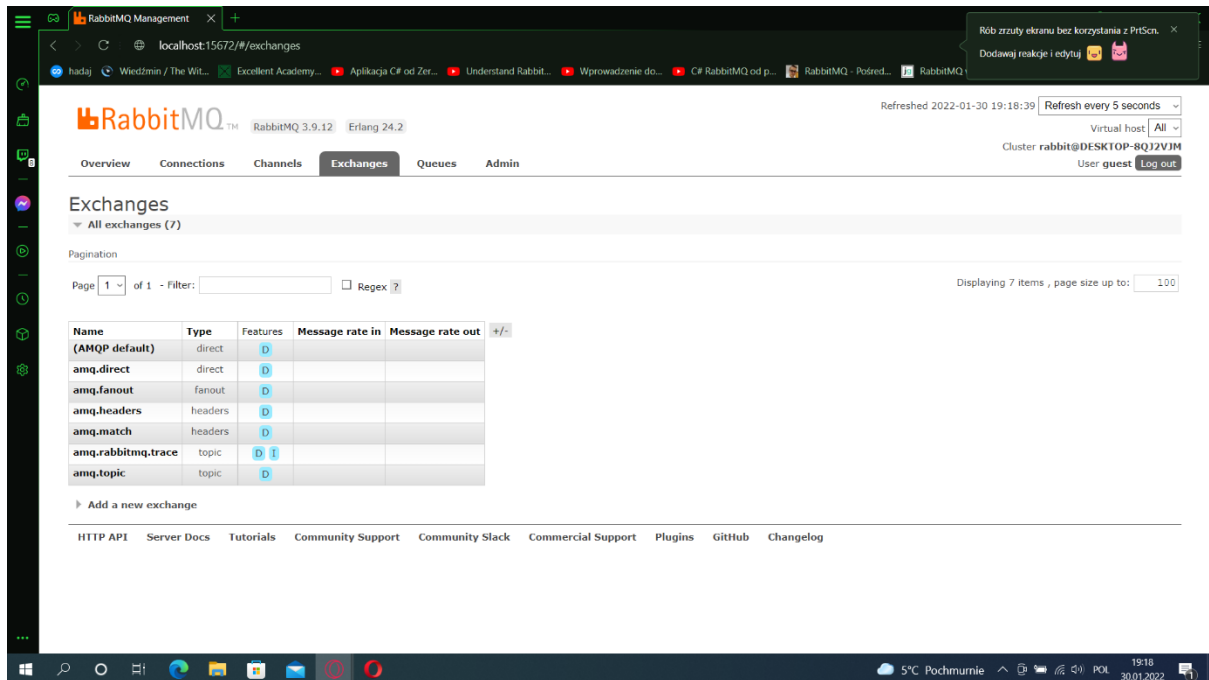


Deklaracja exchange na typ 'topic'.

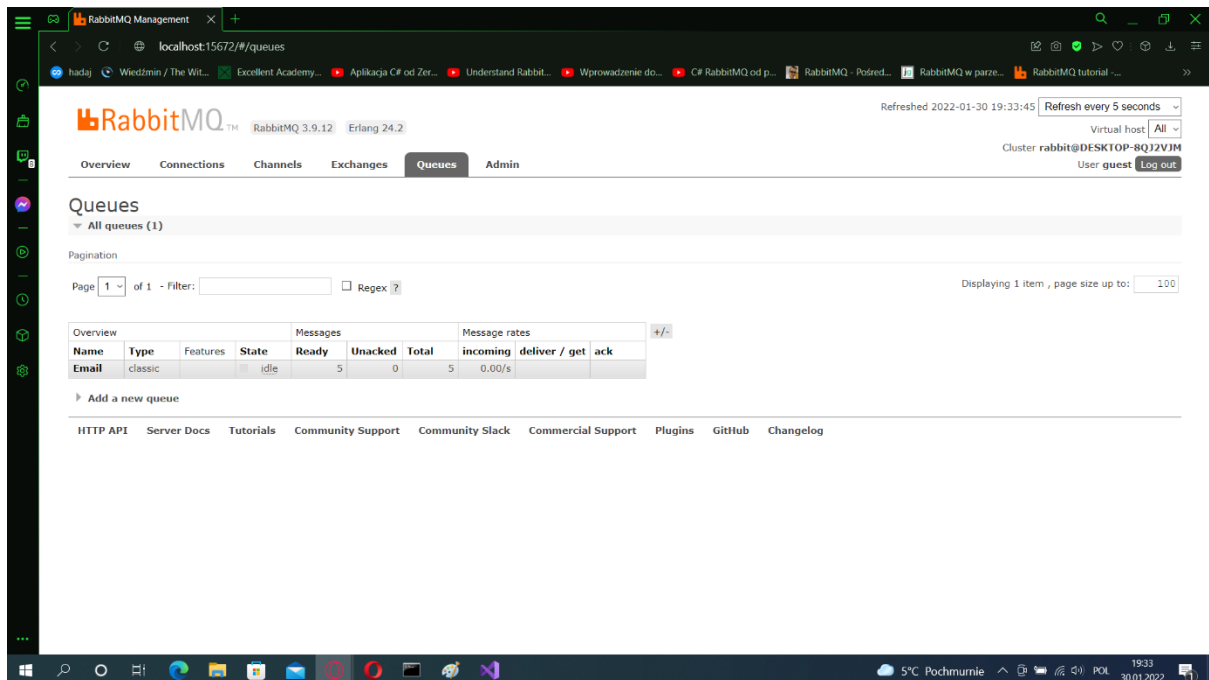
```
channel.ExchangeDeclare(exchange: "topic_logs", type: "topic");
```

Poniżej umieszczam kilka zrzutów ekranu, które pokazują działanie RabbitaMQ w praktyce.

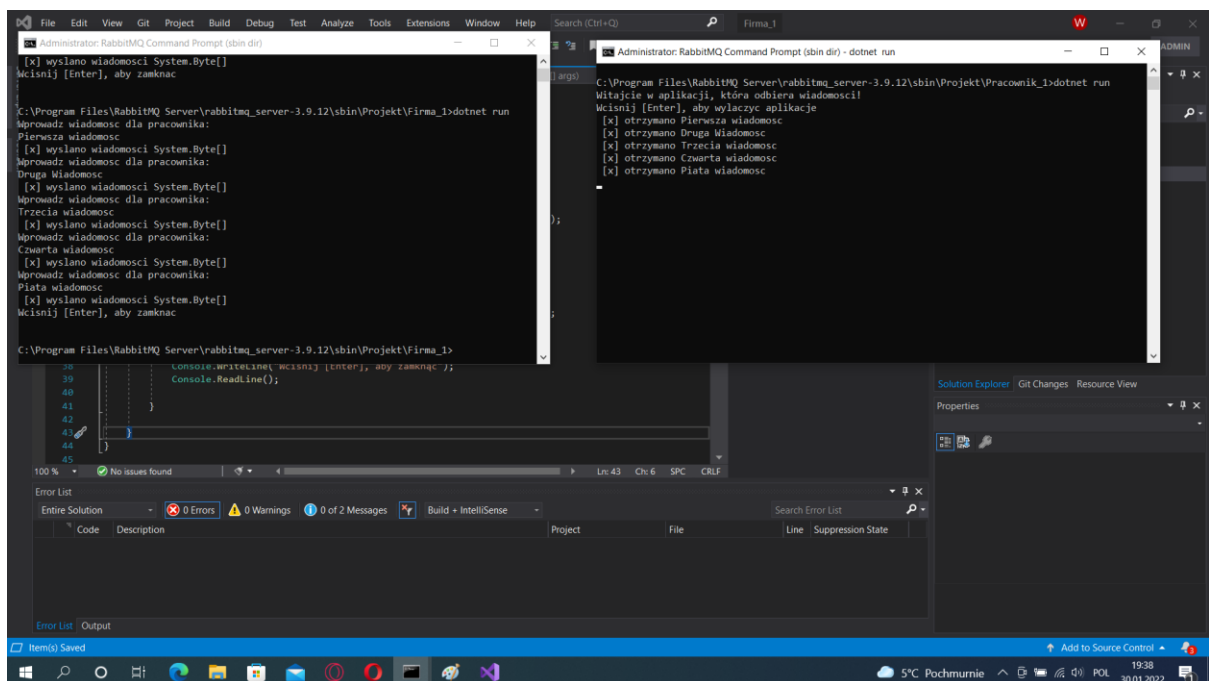
Strona RabbitaMQ po zalogowaniu.



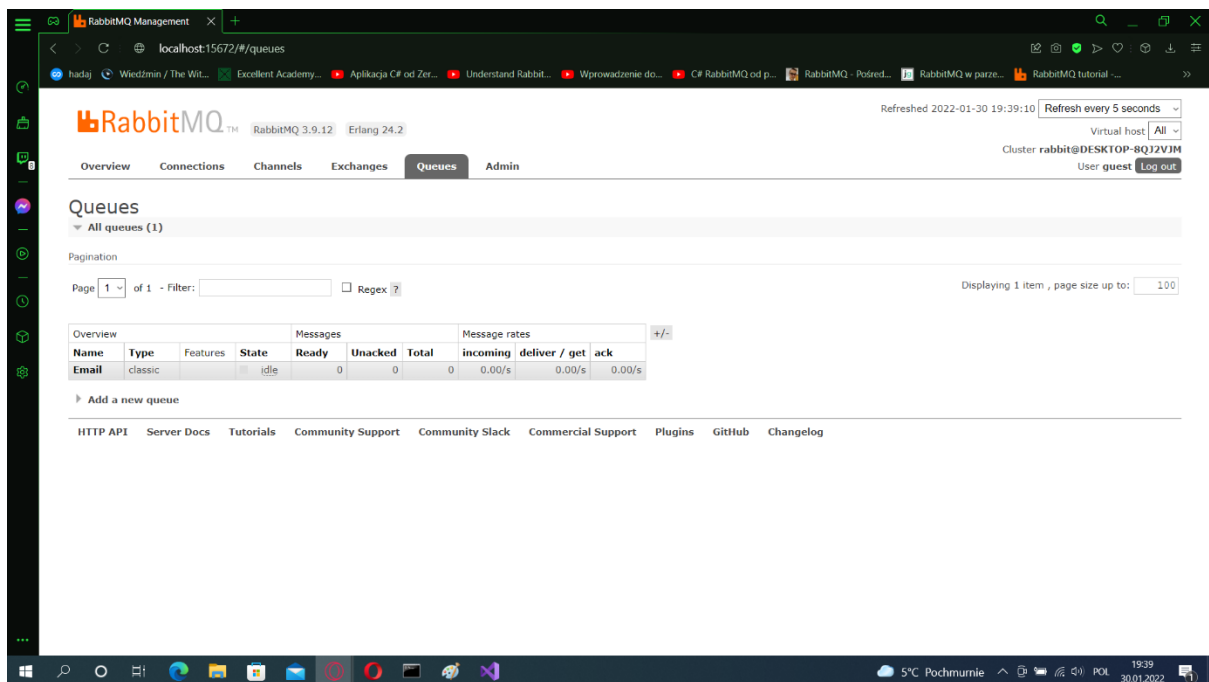
Poniżej widać, że została utworzona kolejka 'Email' w której jest 5 zakolejkowanych wiadomości.



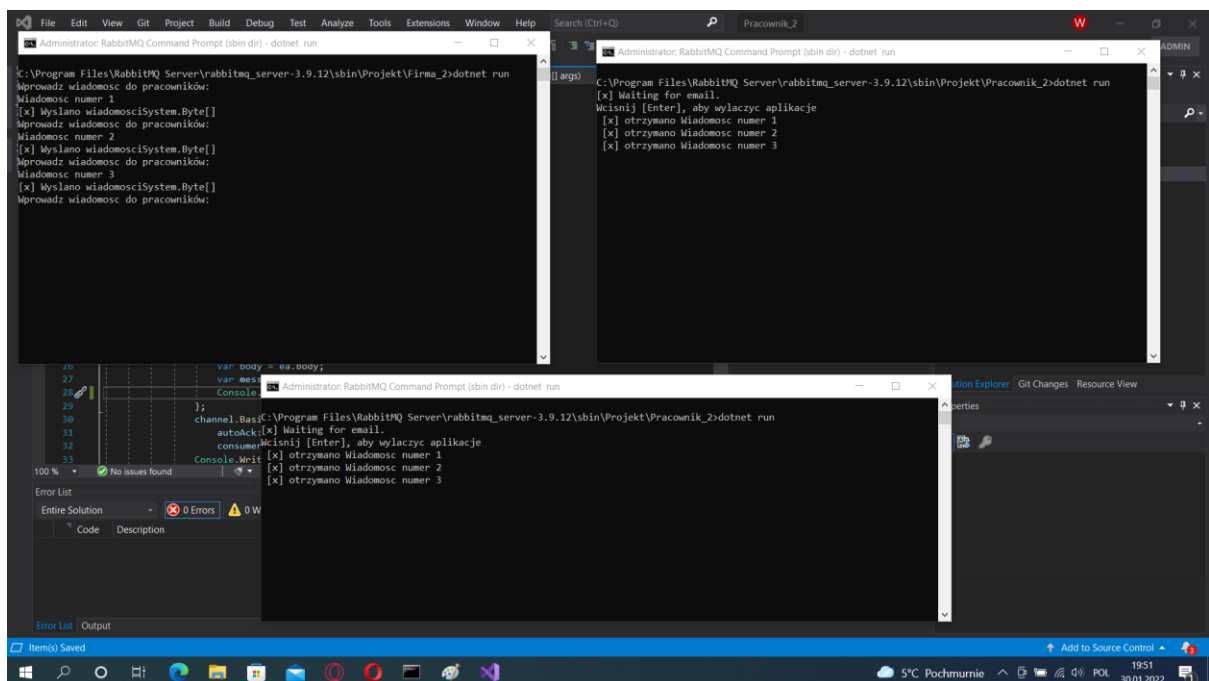
Na poniższym screenie widać uruchomioną aplikację, która wysyła wiadomości, oraz 'pracownika' który odbiera wiadomości. Exchange jest typu 'direct'.



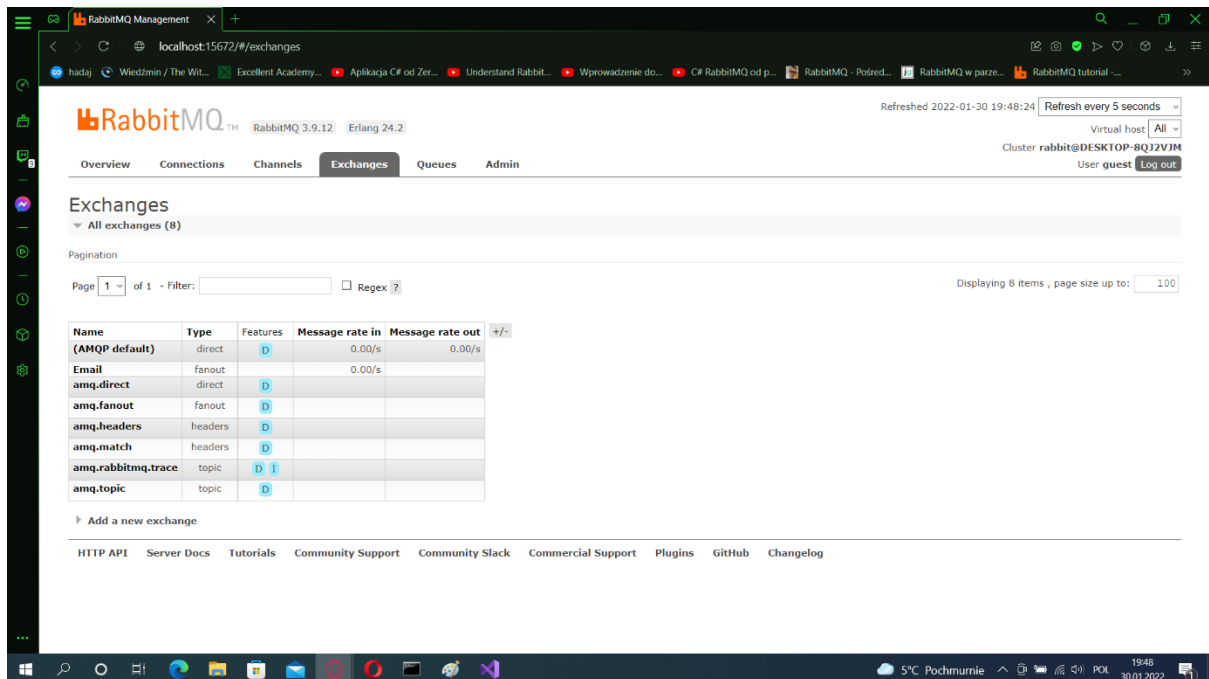
Po odebraniu wiadomości przez pracownika, wiadomości z kolejki zniknęły.



Odbieranie wiadomości przez dwóch pracowników. Exchange typu fanout.



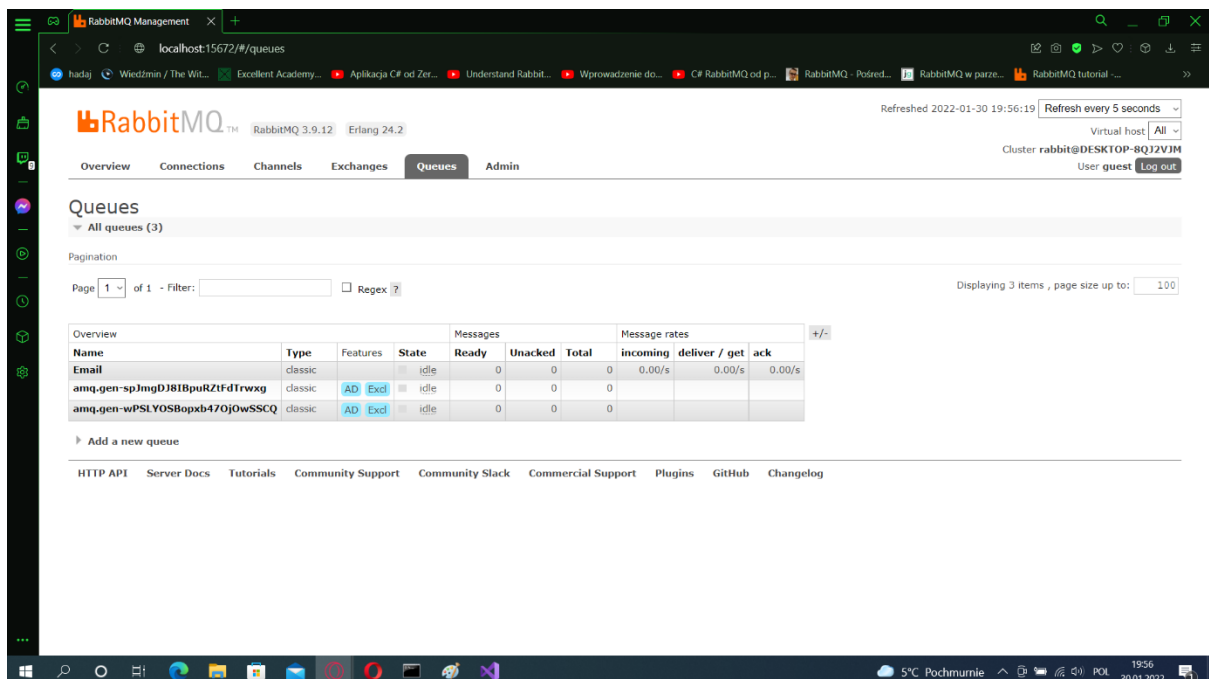
## Utworzenie exchange 'Email' typu fanout.



The screenshot shows the RabbitMQ Management interface at localhost:15672/#/exchanges. The 'Exchanges' tab is selected, displaying a list of exchanges. The 'Email' exchange is highlighted, showing it is of type 'fanout'.

Name	Type	Features	Message rate in	Message rate out	+/-
(AMQP default)	direct	D	0.00/s	0.00/s	
Email	fanout		0.00/s		
amq.direct	direct	D			
amq.fanout	fanout	D			
amq.headers	headers	D			
amq.match	headers	D			
amq.rabbitmq.trace	topic	D I			
amq.topic	topic	D			

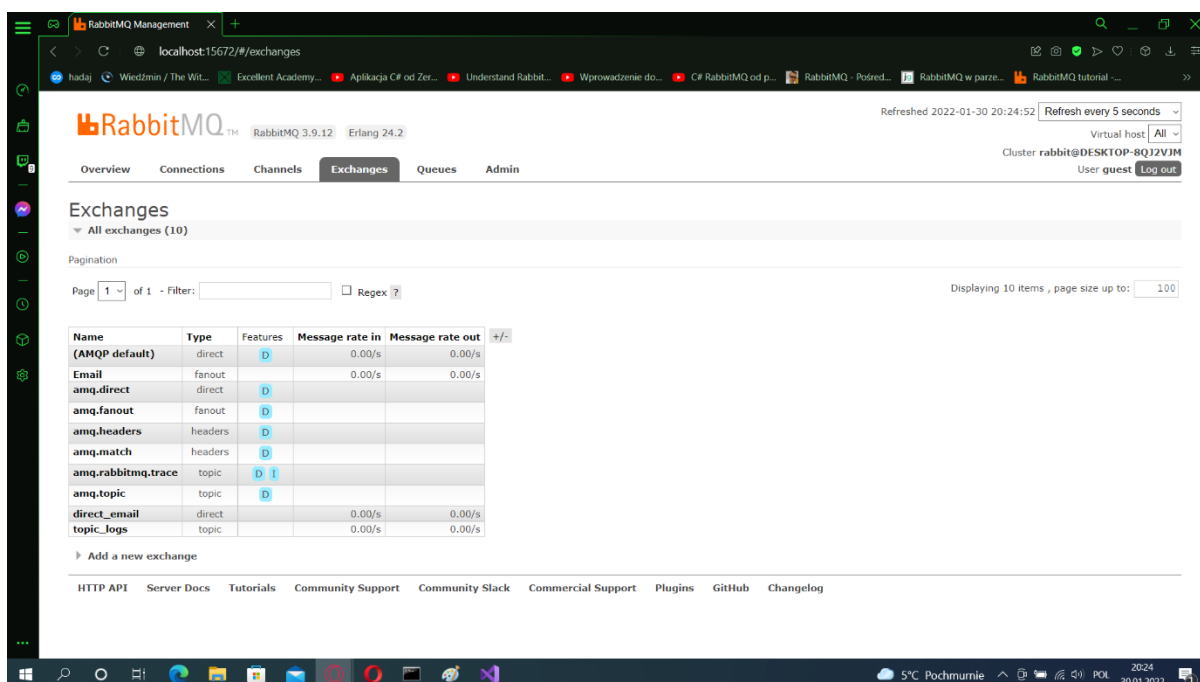
## Utworzenie kolejek dla dwóch pracowników.



The screenshot shows the RabbitMQ Management interface at localhost:15672/#/queues. The 'Queues' tab is selected, displaying a list of queues. Two queues are listed under the 'Email' exchange: 'amq-gen-spJmgD81BpuRZfFdTrwxg' and 'amq-gen-wPSLYOSBopxb470jOwSSCQ'.

Overview				Messages			Message rates				+/-
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack		
Email	classic		idle	0	0	0	0.00/s	0.00/s	0.00/s		
amq-gen-spJmgD81BpuRZfFdTrwxg	classic	AD Excl	idle	0	0	0					
amq-gen-wPSLYOSBopxb470jOwSSCQ	classic	AD Excl	idle	0	0	0					

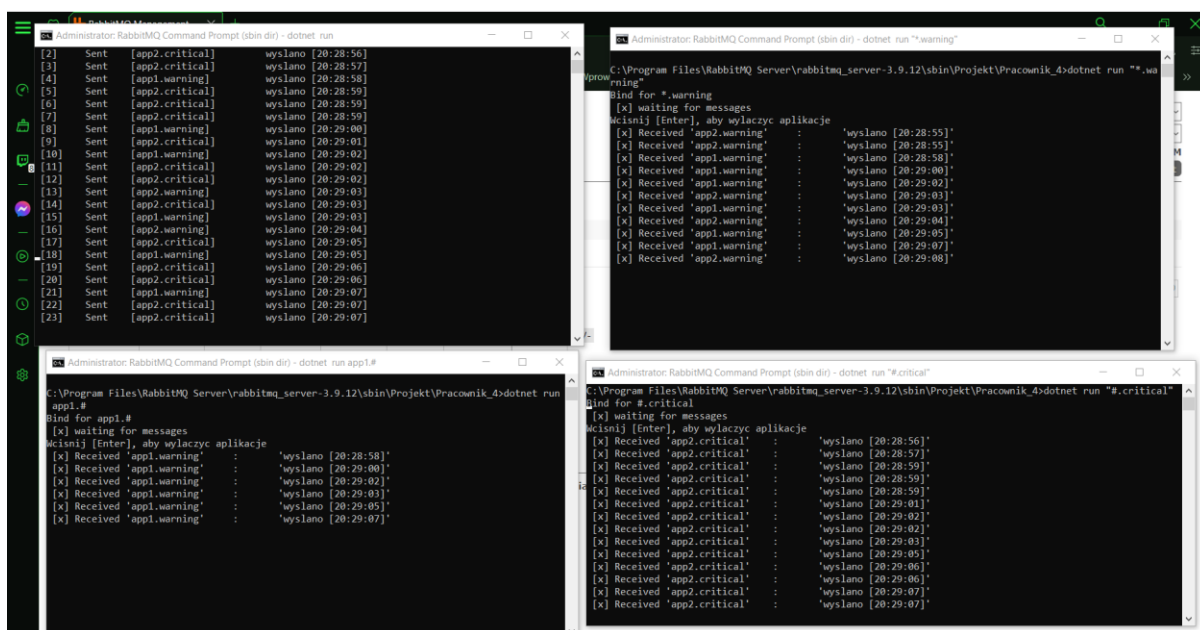
## Utworzenie exchange 'topic\_logs' typu topic.



The screenshot shows the RabbitMQ Management interface. The 'Exchanges' tab is selected, displaying a table of all exchanges. The table has columns for Name, Type, Features, Message rate in, and Message rate out. The 'topic\_logs' exchange is listed with a type of 'topic'.

Name	Type	Features	Message rate in	Message rate out
(AMQP default)	direct	D	0.00/s	0.00/s
Email	fanout		0.00/s	0.00/s
amq.direct	direct	D		
amq.fanout	fanout	D		
amq.headers	headers	D		
amq.match	headers	D		
amq.rabbitmq.trace	topic	D I		
amq.topic	topic	D		
direct_email	direct		0.00/s	0.00/s
topic_logs	topic		0.00/s	0.00/s

Przechwytywanie różnych wiadomości przez różnych pracowników.  
Przykładowo jeden pracownik odbiera wszystko co jest związane z app1 a drugi odbiera wszystkie krytyczne błędy.



The image shows four terminal windows running the RabbitMQ Command Prompt. The windows demonstrate message consumption from the 'topic\_logs' exchange. The messages are categorized by application (app1, app2) and severity (warning, critical). The workers 'Pracownik\_4' and 'Pracownik\_5' are shown receiving these messages.

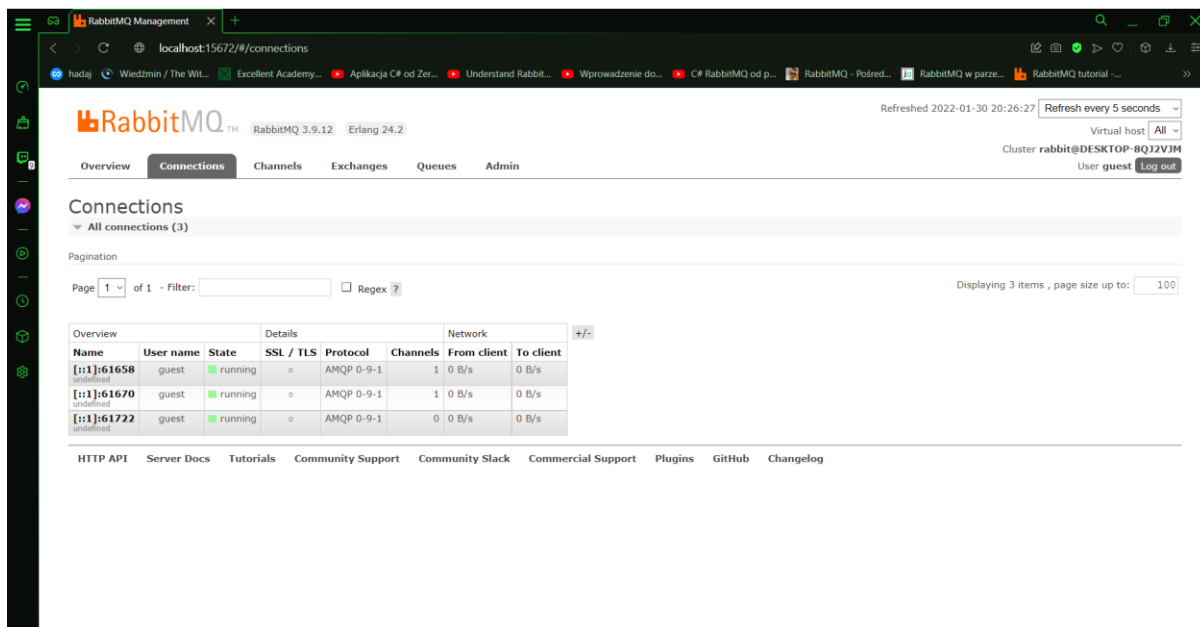
```
Administrator: RabbitMQ Command Prompt (sbin dir) - dotnet run
[2] Sent [app2.critical] wyslano [20:28:56]
[3] Sent [app2.critical] wyslano [20:28:57]
[4] Sent [app1.warning] wyslano [20:28:58]
[5] Sent [app2.critical] wyslano [20:28:59]
[6] Sent [app2.critical] wyslano [20:28:59]
[7] Sent [app2.critical] wyslano [20:28:59]
[8] Sent [app1.warning] wyslano [20:29:00]
[9] Sent [app2.critical] wyslano [20:29:01]
[10] Sent [app1.warning] wyslano [20:29:02]
[11] Sent [app2.critical] wyslano [20:29:02]
[12] Sent [app2.critical] wyslano [20:29:02]
[13] Sent [app2.warning] wyslano [20:29:03]
[14] Sent [app2.critical] wyslano [20:29:03]
[15] Sent [app1.warning] wyslano [20:29:03]
[16] Sent [app2.warning] wyslano [20:29:04]
[17] Sent [app2.critical] wyslano [20:29:05]
[18] Sent [app1.warning] wyslano [20:29:05]
[19] Sent [app2.critical] wyslano [20:29:06]
[20] Sent [app2.critical] wyslano [20:29:06]
[21] Sent [app1.warning] wyslano [20:29:07]
[22] Sent [app2.critical] wyslano [20:29:07]
[23] Sent [app2.critical] wyslano [20:29:07]
```

```
Administrator: RabbitMQ Command Prompt (sbin dir) - dotnet run "app1"
C:\Program Files\RabbitMQ Server\rabbitmq_server-3.9.12\sbin\Projekt\Pracownik_4>dotnet run "app1"
Bind for app1.#
[x] waiting for messages
Wcisnij [Enter], aby wyłaczyć aplikację
[x] Received 'app1.warning' : 'wyslano [20:28:58]'
[x] Received 'app1.warning' : 'wyslano [20:29:00]'
[x] Received 'app1.warning' : 'wyslano [20:29:02]'
[x] Received 'app1.warning' : 'wyslano [20:29:03]'
[x] Received 'app1.warning' : 'wyslano [20:29:05]'
[x] Received 'app1.warning' : 'wyslano [20:29:07]'
```

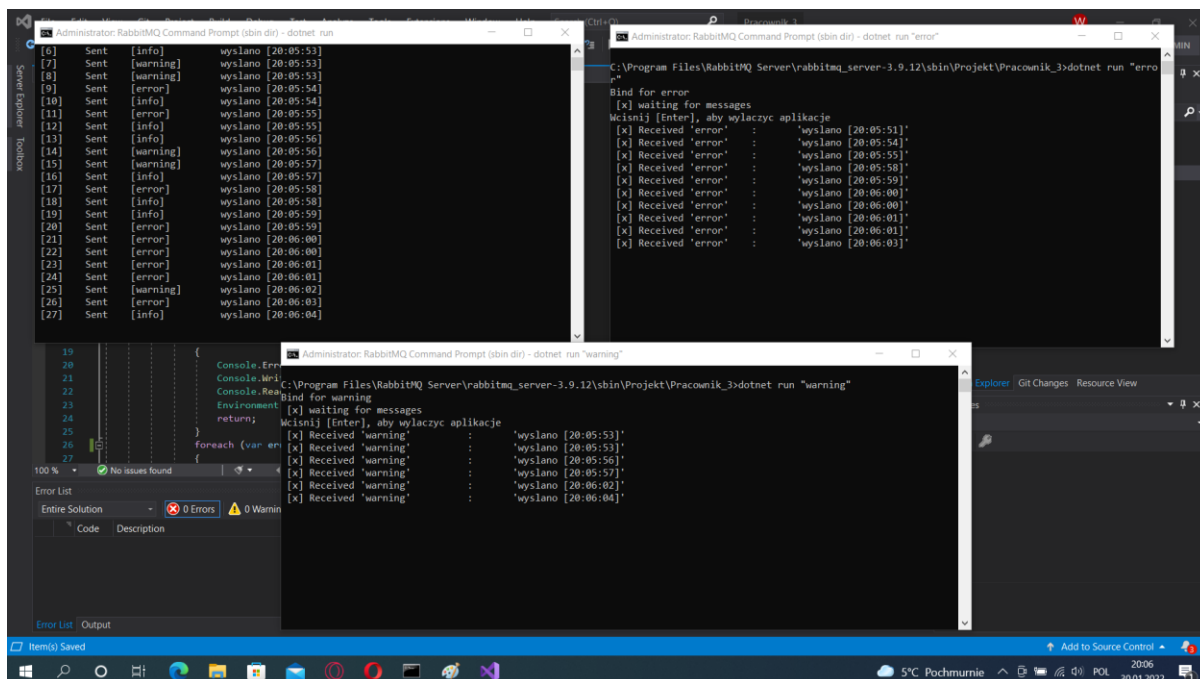
```
Administrator: RabbitMQ Command Prompt (sbin dir) - dotnet run "app2.critical"
C:\Program Files\RabbitMQ Server\rabbitmq_server-3.9.12\sbin\Projekt\Pracownik_4>dotnet run "app2.critical"
Bind for app2.critical
[x] waiting for messages
Wcisnij [Enter], aby wyłaczyć aplikację
[x] Received 'app2.critical' : 'wyslano [20:28:56]'
[x] Received 'app2.critical' : 'wyslano [20:28:57]'
[x] Received 'app2.critical' : 'wyslano [20:28:59]'
[x] Received 'app2.critical' : 'wyslano [20:28:59]'
[x] Received 'app2.critical' : 'wyslano [20:29:01]'
[x] Received 'app2.critical' : 'wyslano [20:29:02]'
[x] Received 'app2.critical' : 'wyslano [20:29:02]'
[x] Received 'app2.critical' : 'wyslano [20:29:03]'
[x] Received 'app2.critical' : 'wyslano [20:29:04]'
[x] Received 'app2.critical' : 'wyslano [20:29:05]'
[x] Received 'app2.critical' : 'wyslano [20:29:06]'
[x] Received 'app2.critical' : 'wyslano [20:29:06]'
[x] Received 'app2.critical' : 'wyslano [20:29:07]'
[x] Received 'app2.critical' : 'wyslano [20:29:07]'
```

```
Administrator: RabbitMQ Command Prompt (sbin dir) - dotnet run "app2.warning"
C:\Program Files\RabbitMQ Server\rabbitmq_server-3.9.12\sbin\Projekt\Pracownik_4>dotnet run "app2.warning"
Bind for app2.warning
[x] waiting for messages
Wcisnij [Enter], aby wyłaczyć aplikację
[x] Received 'app2.warning' : 'wyslano [20:28:55]'
[x] Received 'app2.warning' : 'wyslano [20:28:55]'
[x] Received 'app1.warning' : 'wyslano [20:28:58]'
[x] Received 'app1.warning' : 'wyslano [20:29:00]'
[x] Received 'app1.warning' : 'wyslano [20:29:02]'
[x] Received 'app2.warning' : 'wyslano [20:29:03]'
[x] Received 'app1.warning' : 'wyslano [20:29:04]'
[x] Received 'app2.warning' : 'wyslano [20:29:05]'
[x] Received 'app1.warning' : 'wyslano [20:29:07]'
```

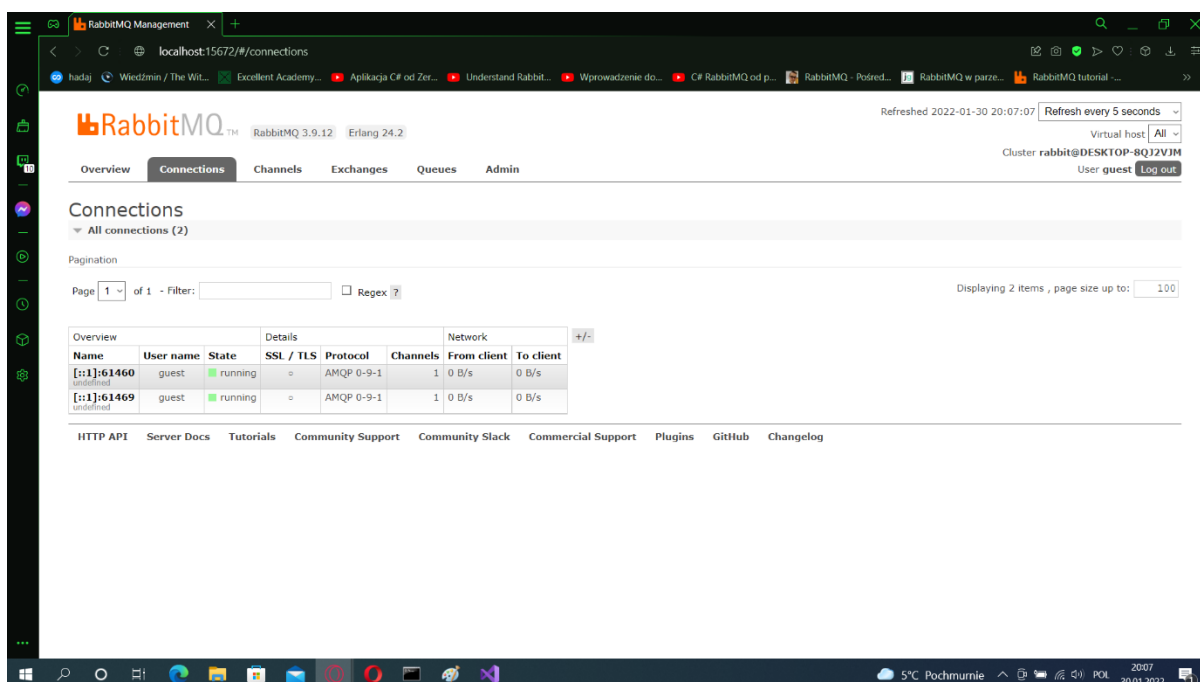
Utworzenie kolejek dla pracowników.



Odbieranie różnych wiadomości przez pracowników.



Utworzone połączenia dla pracowników.



## Podsumowanie

Celem mojego projektu było przedstawienie zasady działania serwera RabbitMQ. Jest to broker wiadomości z otwartym kodem źródłowym, który można zaimplementować w wielu popularnych językach programowania. Swój projekt wykonałem w języku C#. Przedstawiłem kilka sposobów rozsyłania wiadomości pomiędzy serwerem a klientem.

Odnosiniki:

<https://patikod.pl/index.php/2018/05/30/rabbitmq-posrednik-przetwarzania-komunikatow-wstep-do-rabbitmq/>

<https://czterytygodnie.pl/rabbitmq.html>

<https://www.rabbitmq.com>