AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

RabbitMQ

http://home.agh.edu.pl/~fmal/rabbitmq

Filip Malawski

fmal@agh.edu.pl

Modele komunikacji

- Komunikacja synchroniczna
- Komunikacja asynchroniczna

Modele komunikacji

- Komunikacja synchroniczna
  - Obie strony uczestniczące muszą być aktywne
  - Wywołania blokujące

- Komunikacja asynchroniczna
  - Obie strony uczestniczące nie muszą być aktywne jednocześnie
  - Wywołania nieblokujące
  - Potwierdzenia odbioru (opcjonalnie)

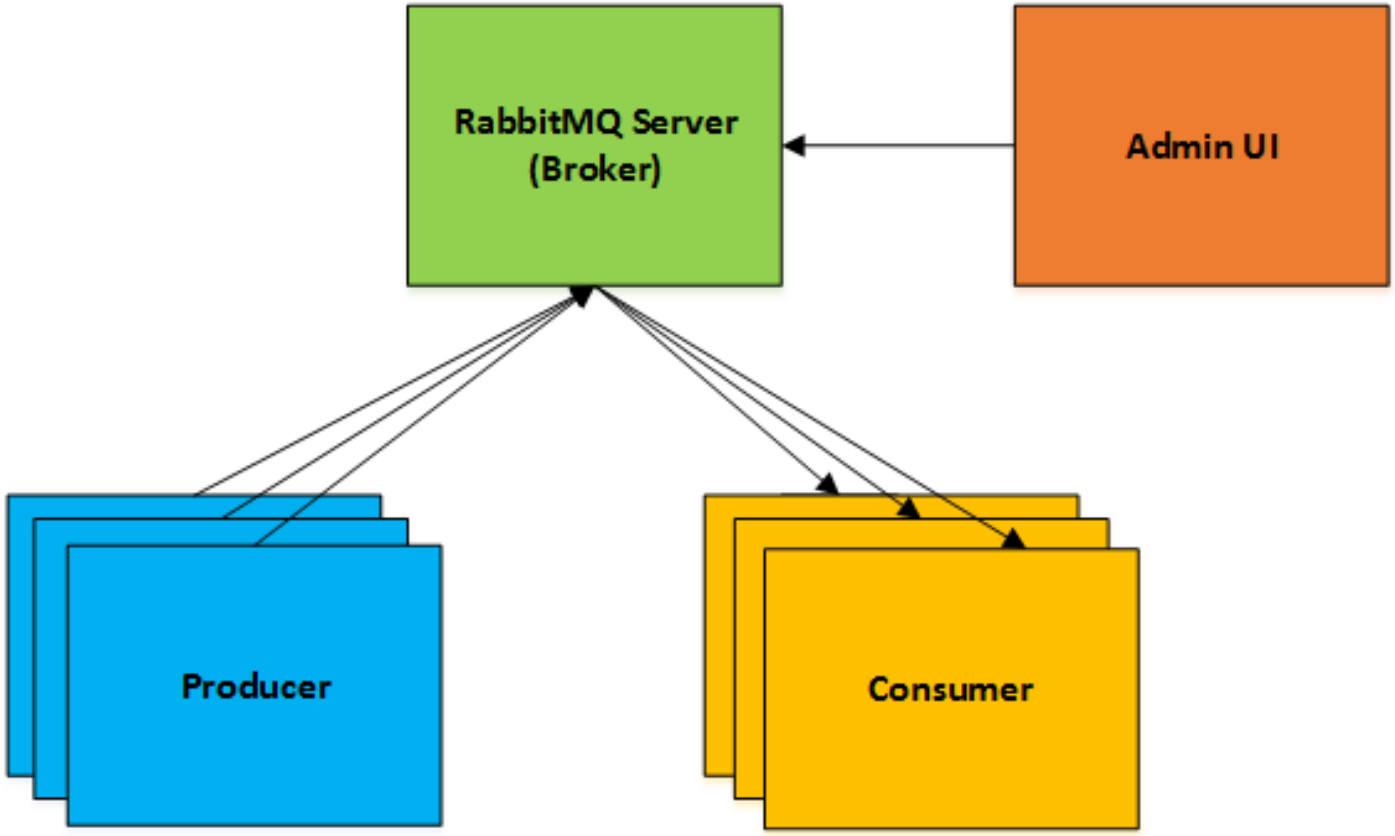
Wiadomości

- Alternatywa dla wywołań metod
  - Format wiadomości zamiast interfejsu
  - Ukierunkowane na zdarzenia
  - Brak sztywnych zależności czasowych
- Luźne powiązania komponentów
- Message Oriented Middleware
  - Warstwa pośrednia dostarczająca mechanizmów obsługi wiadomości

RabbitMQ

- Framework do obsługi wiadomości
- Główne cechy:
  - Mechanizmy wyboru ścieżek do przesyłu wiadomości (routing)
  - Mechanizmy zapewnienia niezawodności (potwierdzenia, ponowne wysłanie, itp.)
  - Wsparcie dla różnych protokołów
  - Wsparcie dla wielu języków programowania
  - Interfejs do zarządzania
  - Pluginy

RabbitMQ – Elementy składowe



Hello World

- Producent
  - wysyła wiadomość do kolejki
- Konsument
  - odbiera wiadomości z kolejki

Połączenie (Producer / Consumer)

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

(…)

// don’t close while listening (consumer)
channel.close();
connection.close();
```

Wysyłanie wiadomości (Producer)

```
String QUEUE_NAME = "queue1";
channel.queueDeclare(QUEUE_NAME, false, false, false, null);

String message = "Hello World!";
channel.basicPublish("", QUEUE_NAME, null, message.getBytes());

System.out.println("Sent: " + message);
```

Odbieranie wiadomości (Consumer)

```
String QUEUE_NAME = "queue1";
channel.queueDeclare(QUEUE_NAME, false, false, false, null);

Consumer consumer = new DefaultConsumer(channel) {
        @Override
        public void handleDelivery(String consumerTag,
          Envelope envelope, AMQP.BasicProperties properties,
                  byte[] body) throws IOException {
            String message = new String(body, "UTF-8");
            System.out.println("Received: " + message);
        }};

channel.basicConsume(QUEUE_NAME, true, consumer);
```

Uruchomienie przykładu

- Należy wystartować serwer RabbitMQ
  - Menu start -> RabbitMQ Service - start
- Kod dostępny na moodle
  - Projekt IntelliJ
- Uruchomić konsumenta **Z1_Consumer**
- Uruchomić producenta **Z1_Producer**
- Przesłać wiadomość

RabbitMQ

- Konsola administracyjna (web)
  - Uruchomić konsolę RabbitMQ Command Prompt (z menu start)
  - Wpisać:

  rabbitmq-plugins enable rabbitmq_management
  - Konsola dostępna pod adresem:

  http://localhost:15672/

  user: guest, password: guest

- Tutorial

  https://www.rabbitmq.com/getstarted.html

Mechanizmy obsługi kolejek

- Potwierdzenia
  - Potwierdzenie po otrzymaniu wiadomości
  - Potwierdzenie po przetworzeniu wiadomości
- Dystrybucja wiadomości do wielu konsumentów
  - Domyślnie round-robin
  - Możemy uzyskać load-balancing
- Trwałość
  - Możliwość zachowania wiadomości przy restarcie serwera

Zadanie 1 (2 pkt)

- Zaobserwować działanie mechanizmów:
  a) niezawodności
  b) load-balancing’u

Zadanie 1a

- Zmodyfikować producenta, aby wysyłał wiadomości wpisane z konsoli

```
BufferedReader br = new BufferedReader(new
    InputStreamReader(System.in));
```

- Zmodyfikować konsumenta, aby obsługiwał wiadomość zadaną ilość czasu (przesłaną w wiadomości)

```
int timeToSleep = Integer.parseInt(message);
Thread.sleep(timeToSleep * 1000);
```

Zadanie 1a

- Sprawdzić działanie potwierdzeń:
  - po **otrzymaniu wiadomości**

```
channel.basicConsume(QUEUE_NAME, true, consumer);
```

  - po **przetworzeniu wiadomości**

```
channel.basicConsume(QUEUE_NAME, false, consumer);
(...)
channel.basicAck(envelope.getDeliveryTag(), false);
```

  - bez potwierdzeń

```
channel.basicConsume(QUEUE_NAME, false, consumer);
```

Zadanie 1a

- Zaobserwować co się dzieje, gdy:
  - Konsument zostanie ponownie uruchomiony **po skończeniu przetwarzania** wiadomości
  - Konsument zostanie ponownie uruchomiony **w trakcie przetwarzania** wiadomości

Potwierdzenie	Ponowne uruchomienie po obsłużeniu wiad.	Ponowne uruchomienie w trakcie obsługi wiad.
Po otrzymaniu wiad.	?	?
Po obsłużeniu wiad.	?	?
Bez	?	?

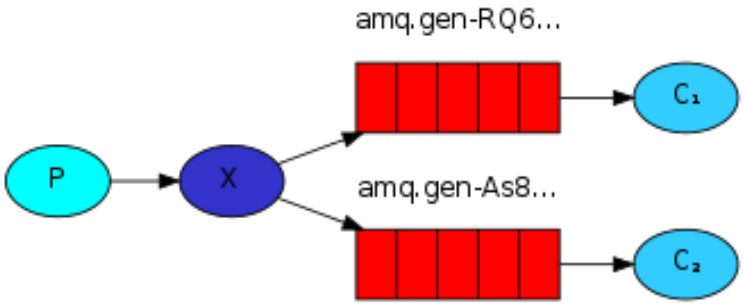
Zadanie 1b

- Zaobserwować co się dzieje, gdy:
  - Mamy uruchomionych dwóch konsumentów
  - Potwierdzenia po przetworzeniu wiadomości
  - Wysyłamy:
    - tylko krótkie zadania (1s)
    - na przemian krótkie i długie zadania (np. 1 s. i 5s.)
- Dodać obsługę QoS u konsumenta:

  `channel.basicQos(1);`
  - Zaobserwować działanie przy wysyłaniu na przemian długich i krótkich wiadomości

Routing

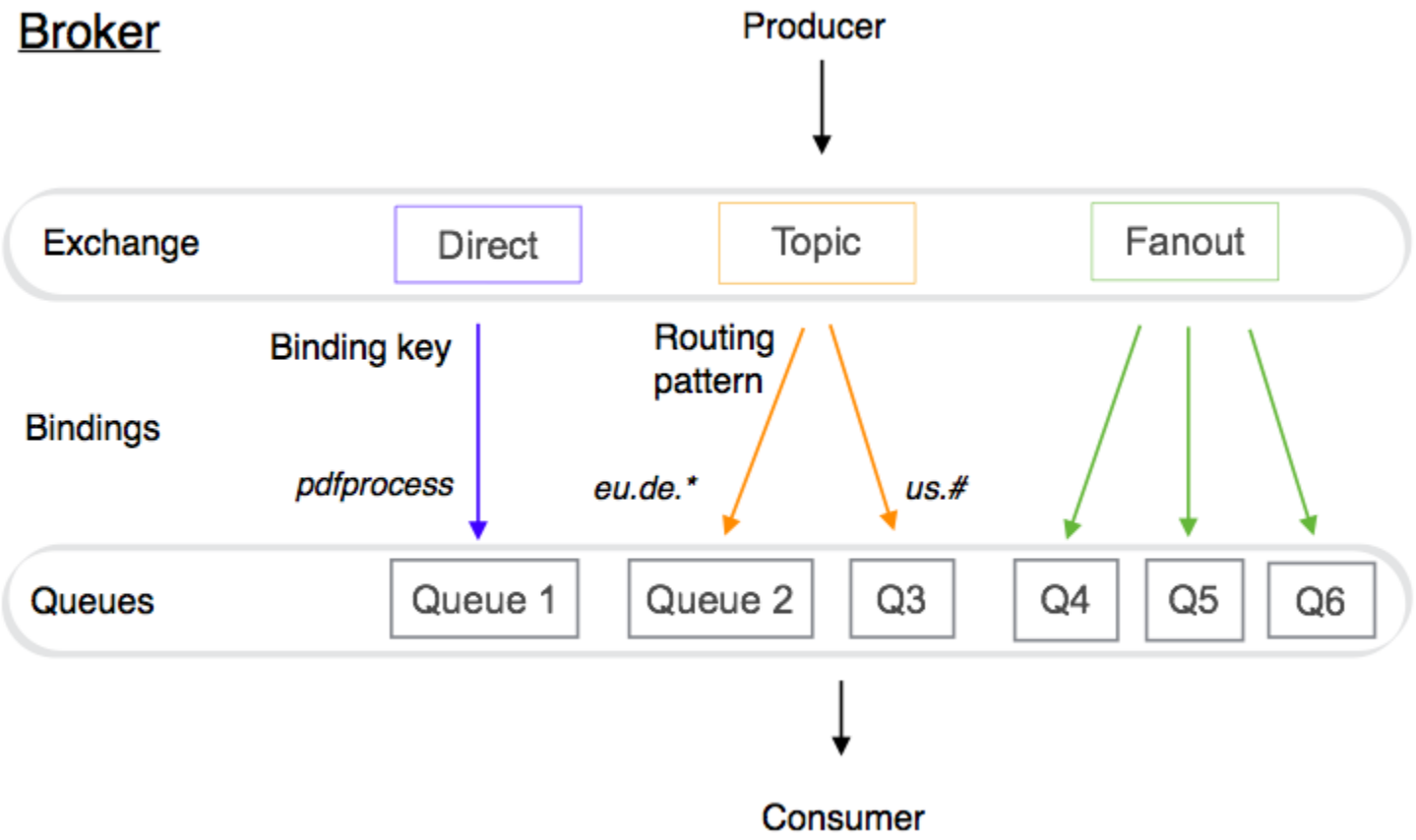
- Exchange
  - Producent nie wysyła wiadomości bezpośrednio do kolejki, lecz do Exchange
  - Exchange decyduje gdzie wysłać wiadomość
  - W poprzednim zadaniu korzystaliśmy z domyślnego Exchange (Nameless)



Routing

- Typy Exchange:
  - Direct (bezpośrednio wg. klucza)
  - Topic (dopasowanie wg. wzorca)
  - Fanout (do wszystkich zapisanych)
  - Headers
- Uwaga:
  - Kolejki muszą zostać związane (bind) z danym Exchange, aby otrzymywać z niego wiadomości

Routing

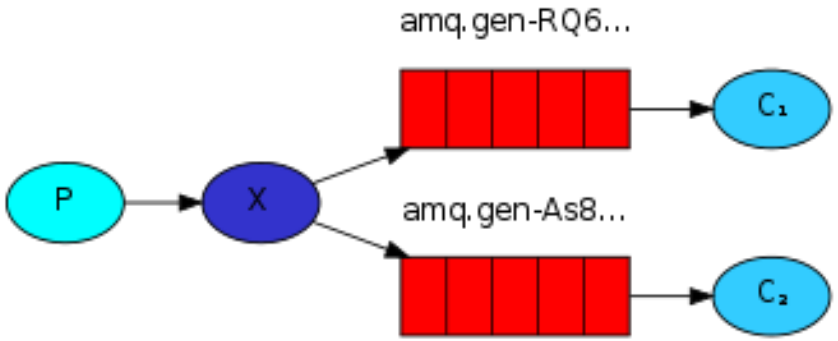

Broker
Producer
Exchange
Direct
Topic
Fanout
Binding key
Routing pattern
Bindings
pdfprocess
eu.de.*
us.#
Queues
Queue 1
Queue 2
Q3
Q4
Q5
Q6
Consumer

Routing Fanout

- Wiązanie kolejki z Exchange (model publish/subscribe)

```
String queueName = channel.queueDeclare().getQueue();
channel.queueBind(queueName, EXCHANGE_NAME, "");
```

- Każdy kto jest zapisany do danego Exchange dostaje wiadomości

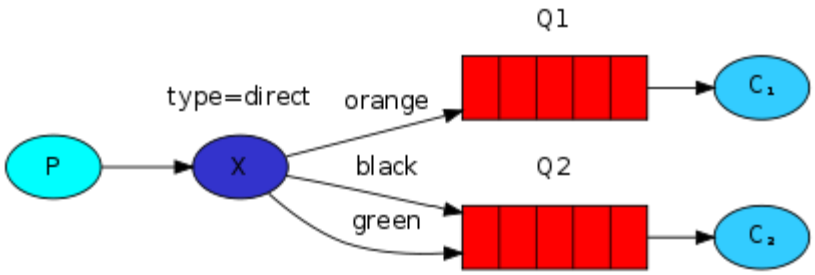


Routing Direct

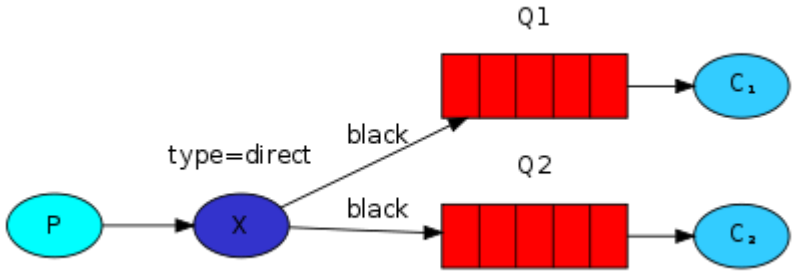
- Wiązanie kolejki z Exchange wg klucza

```
channel.queueBind(queueName, EXCHANGE_NAME, routingKey);
```

- Możliwe wiele kluczy dla danej kolejki

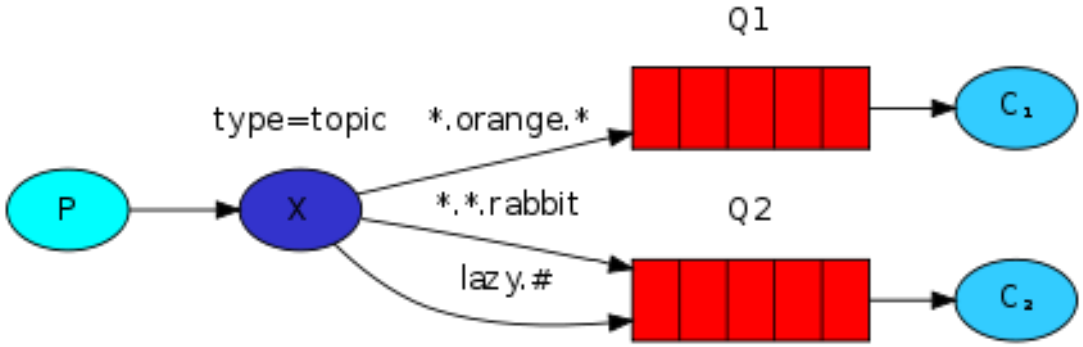


- Możliwe wiele kolejek z tym samym kluczem



Routing Topic

- Dopasowanie do wzorca
  - np. blue.fast.sedan
  - * to dokładnie jedno dowolne słowo
  - # to zero lub więcej dowolnych słów



Uruchomienie przykładu (Fanout)

- Uruchomić producenta **Z2_Producer**
- Uruchomić dwóch konsumentów **Z2_Consumer**
- Przesłać wiadomość
- Każdy konsument powinien dostać wiadomość

Zadanie 2 (2 pkt)

- Pokazać działanie routingu:
  - Direct
  - Topic

Zadanie 2a – Direct (1 pkt)

- Zmodyfikować producenta, aby:
  - przyjmował z konsoli klucz routingu oraz wiadomość
  - wysyłał do Exchange typu *Direct*
- Zmodyfikować konsumenta tak, aby:
  - przy uruchamianiu przyjmował klucz routingu
  - odczytywał wiadomości wysłane z danym kluczem z Exchange typu *Direct*
- Pokazać działanie routingu typu Direct

Zadanie 2b – Topic (1 pkt)

- Zmodyfikować producenta tak, aby:
  - wysyłał do Exchange typu *Topic*
- Zmodyfikować konsumenta tak, aby:
  - odczytywał wiadomości wysłane z pasującym wzorcem z Exchange typu *Topic*
- Pokazać działanie routingu typu Topic

Zadanie 2 - wskazówki

```
channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.DIRECT);
channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.TOPIC);

channel.basicPublish(EXCHANGE_NAME, key, null, message.getBytes("UTF-8"));
channel.queueBind(queueName, EXCHANGE_NAME, key);
```

Zadanie domowe

- Scenariusz: Obsługujemy oddział ortopedyczny w szpitalu
- Mamy 3 typy użytkowników:
  - Lekarz (zleca badania, dostaje wyniki)
  - Technik (wykonuje badania, wysyła wyniki)
  - Administrator (loguje całą aktywność, może wysyłać informacje do wszystkich)
- Szpital przyjmuje pacjentów z kontuzjami:
  - Biodra (hip), kolana (knee) lub łokcia (elbow)

Zadanie domowe

- Lekarz:
  - Wysyła zlecenie badania podając typ badania (np. knee) oraz nazwisko pacjenta, do dowolnego technika, który umie wykonać takie badanie
  - Otrzymuje wyniki asynchronicznie

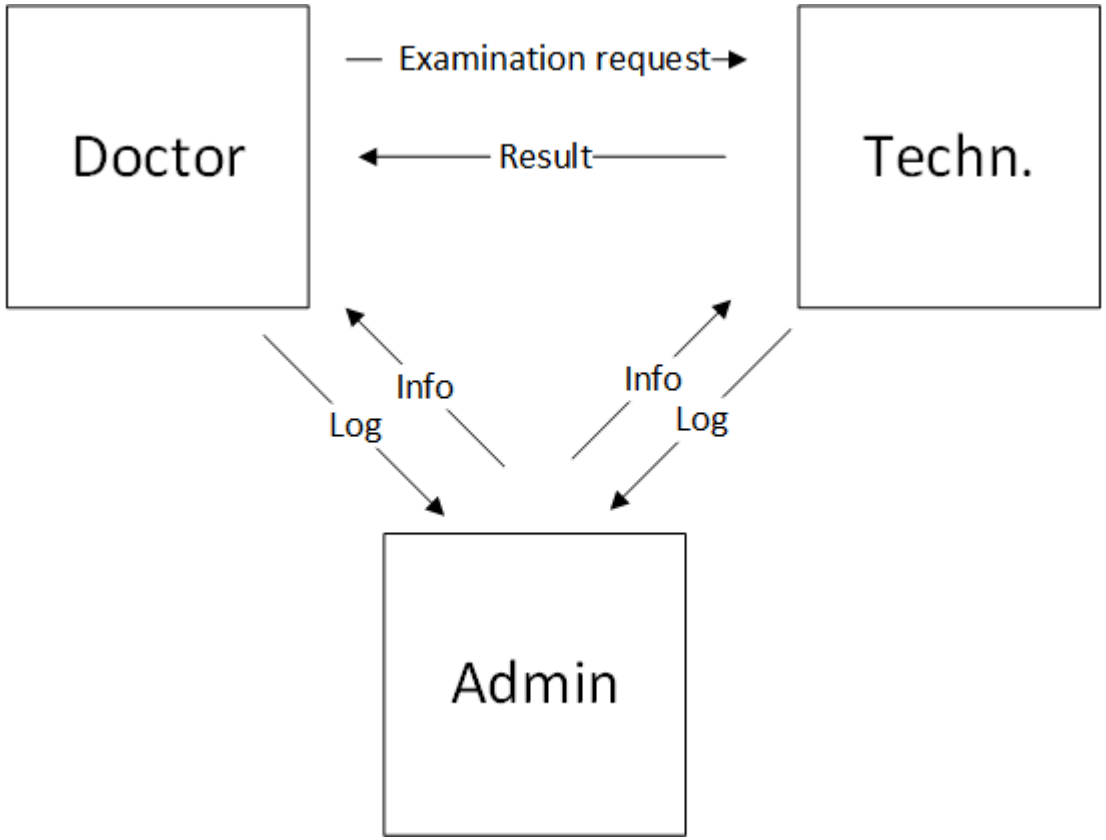
Zadanie domowe

- Technik
  - Każdy technik umie wykonać 2 typy badań, które podawane są przy starcie (np. knee, hip)
  - Przyjmuje zgłoszenia danego typu i odsyła wyniki do lekarza zlecającego (wynik to nazwa pacjenta + typ badania + „done”)
  - Uwaga: jeśli jest dwóch techników z tym samym typem badania (np. knee) to wiadomość powinna być obsłużona tylko przez jednego (ale nie zawsze tego samego)

Zadanie domowe

- Administrator
  - Loguje całą aktywność (dostaje kopie wszystkich wiadomości – zleceń oraz odpowiedzi)
  - Ma możliwość przesłania wiadomości (info) do wszystkich

Zadanie domowe



Prezentacja zadania

- Proszę przygotować rysunek ze schematem (użytkownicy, exchange, kolejki, wiadomości)
- Schemat musi być przygotowany w wersji elektronicznej (skan rysunku nie jest dozwolony)
- Scenariusz testowy:
  - 2 lekarzy
  - 2 techników
    - np. (knee, hip) oraz (knee, elbow)
  - 1 administrator

Zadanie domowe

- Punktacja
  - Schemat 2 pkt
  - Lekarz + Technik 5 pkt
  - Administrator 3 pkt (wymaga zrobienia najpierw Lekarza i Technika)
- Zadanie można zaimplementować w dowolnym języku obsługiwanym przez RabbitMQ

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Dziękuję