

**WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI**
POLITECHNIKI RZESZOWSKIEJ

Usługi sieciowe w biznesie

Projekt

Asynchroniczna komunikacja przy pomocy
ActiveMQ

Kamil Madej 161876

Rzeszów, 2022

Spis treści

1. Wstęp.....	3
2. Czym jest Active MQ.....	3
3. Zasada działania.....	3
3.1. Kolejka.....	3
3.2. Temat.....	4
3.3. Luźne powiązanie(loose coupling).....	4
3.3.1. Korzyści płynące z luźnego wiązania:.....	5
4. Komunikaty	5
4.1. Nagłówki	5
4.2. Właściwości.....	5
4.3. Treść	5
5. Zastosowanie ActiveMQ	5
6. Zalety brokerów wiadomości.....	6
7. Wady brokerów wiadomości.	7
8. Część praktyczna	8
8.1. Instalacja ActiveMQ.....	8
8.2. Program w Javie	11
8.2.1. Klasy programu:	11
8.3. Działanie programu	13
9. Podsumowanie	14
10. Linki.....	14

1. Wstęp

Celem projektu jest zaprezentowanie opisanie i zaprezentowanie działania komunikacji asynchronicznej przy użyciu brokera wiadomości ActiveMQ. Komunikacja asynchroniczna to metoda komunikacji gdzie uczestnicy obu stron konwersacji mają swobodę rozpoczynania, wstrzymywania i wznowiania wiadomości na własnych warunkach, eliminując potrzebę oczekiwania na bezpośrednie połączenie.

2. Czym jest Active MQ.

Apache ActiveMQ broker komunikatów o otwartym kodzie źródłowym napisany w Javie. Obsługuje standardowe protokoły, dzięki czemu użytkownicy mogą czerpać korzyści z wyboru klienta w szerokim zakresie języków i platform

ActiveMQ jest wykorzystywany jako pomost komunikacyjny pomiędzy dwiema lub kilkoma aplikacjami

3. Zasada działania

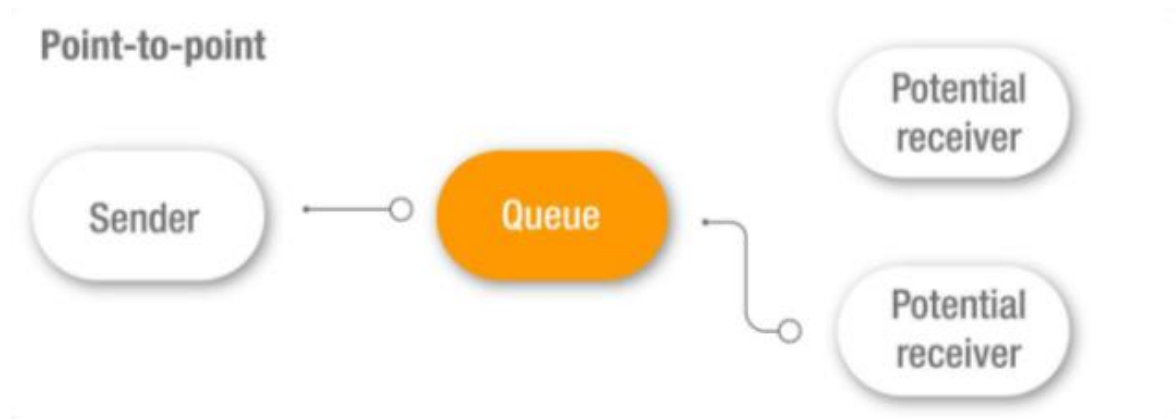
ActiveMQ wysyła wiadomości pomiędzy klientem aplikacji(producer), który tworzy wiadomości i zatwierdza je do wysłania, a konsumentem, który otrzymuje i przetwarza wiadomości.

ActiveMQ kieruje każdą wiadomość przez endpoint zwany miejscem docelowym(destination).

Podstawą działania ActiveMQ są topics(tematy) i queues(kolejki)

3.1. Kolejka

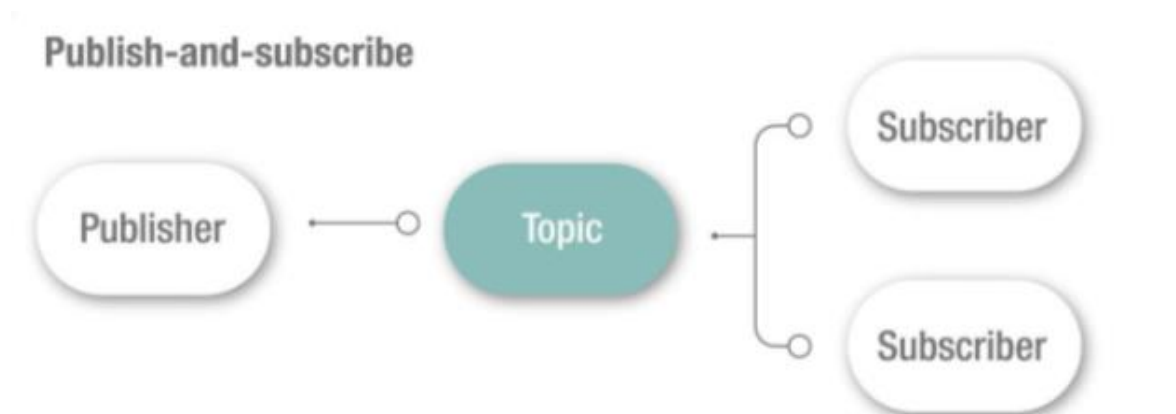
Kolejka JMS implementuje semantykę modułu równoważenia obciążenia. Pojedynczą wiadomość otrzyma tylko jeden konsument. Jeśli żaden konsument nie jest dostępny w danym czasie, wiadomość zostanie zachowana do czasu, gdy dostępny będzie konsument, który może przetworzyć wiadomość.



Rysunek 1. Schemat komunikacji point-to-point

3.2. Temat

Temat ActiveMQ implementuje semantykę publikowania i subskrybowania. Gdy wiadomość zostanie opublikowana, trafia do wszystkich zainteresowanych subskrybentów- więc zero lub wiele subskrybentów otrzyma kopie wiadomości. Jedynie subskrybenci, którzy mają aktywną subskrypcję w momencie odebrania komunikatu przez broker otrzymają kopię komunikatu.



Rysunek 2. Schemat komunikacji publish-and-subscribe

3.3. Luźne powiązanie(loose coupling)

ActiveMQ wysyła wiadomości asynchronicznie, więc konsumenci niekoniecznie otrzymują wiadomości od razu.

Utworzenie i wysłanie wiadomości przez producenta jest rozłączone od zadania konsumenta do jego pobrania.

ActiveMQ używa brokera jako pośrednika. Producenci i konsumenci są od siebie niezależni. Gdy tylko producent wyśle wiadomość do brokera jego zadanie jest zakończone, niezależnie od tego, czy i kiedy konsument otrzyma komunikat.

Podobnie w przypadku konsumenta, gdy tylko otrzymuje wiadomość od brokera, robi to bez wiedzy producenta, który utworzył wiadomość. Taki rodzaj układu, w którym klienci funkcjonują bez wzajemnej wiedzy, określa się mianem **luźnego wiązania**.

3.3.1. Korzyści płynące z luźnego wiązania:

- Wysoko przepustowość, ponieważ producenci nie muszą czekać na potwierdzenia od brokera lub konsumenta.
- Elastyczność: Klienci mogą być tymczasowo niedostępni, dynamicznie dodawani do środowiska, lub nawet przepisywani w nowym języku bez wpływu na innych klientów i bez powodowani błędów w procesie przesyłania wiadomości.
- Heterogeniczność: Klienci działają niezależnie, komunikują się z brokerem, nie bezpośrednio między sobą. W rezultacie mogą być napisani w dowolnym języku obsługiwanym przez ActiveMQ.

4. Komunikaty

Każda wiadomość, którą ActiveMQ wysyła bazowana jest na specyfikacji JMS i składa się z nagłówków oraz opcjonalnie właściwości i treści.

4.1. Nagłówki

Nagłówki zawierają metadane dotyczące wiadomości. Wartość nagłówków są ustawiane podczas tworzenia wiadomości przez producenta lub podczas wysyłania go jej przez ActiveMQ.

4.2. Właściwości

Właściwości umożliwiają dodawanie opcjonalnych metadanych do wiadomości.

4.3. Treść

Treść wiadomości może być tekstem lub danymi binarnymi. Wartość nagłówka, która jest jawnie ustawiana przez producenta podczas tworzenia komunikatu, określa, co może być przenoszone w treści komunikatu.

5. Zastosowanie ActiveMQ

Istnieje wiele sytuacji gdzie ActiveMQ i asynchroniczna komunikacja może mieć znaczący wpływ na architekturę systemu. Poniżej kilka z głównych przypadków użycia brokera:

- **W przypadku integracji niejednorodnej aplikacji** – ActiveMQ jest napisany przy pomocy Javy, więc oczywiście istnieje API dla klienta Javy. Jednak ActiveMQ dostarcza również klientów dla min. C/C++, Perl, PHP, Python, Ruby i wielu innych języków. Jest to duża zaleta, gdy chcemy zintegrować aplikacje napisane w różnych językach na różnych platformach. W takim przypadku różne API klienta umożliwia wysyłanie i odbieranie wiadomości za pośrednictwem ActiveMQ bez względu na używany język.
- **Jako zamiennik RPC** – systemy bazujące na rządaniach synchronicznych zazwyczaj mają ograniczoną możliwość skalowania, ponieważ ostatecznie rządania zaczęłyby tworzyć kopie apasowe, spalnając w ten sposób cały system. Komunikacja asynchroniczna w tym przypadku może łatwo dodać dodatkowych odbiorców komunikatów, aby były zużywane jednocześnie, a tym samym obsługiwane szybciej.
- **Do zmniejszenia powiązania pomiędzy aplikacjami** – jak już zostało wspomniane, architektura luźnego wiązania wskazuje na mniej zależności, dzięki czemu aplikacje lepiej radzą sobie z nieprzewidzianymi zmianami. Zmiana w jednym komponencie systemu nie wpłynie na cały system, ale także znacznie uproszczona zostanie interakcja między komponentami. Zamiast używać synchronicznej komunikacji, komponenty wykorzystują komunikację asynchroniczną (wysyłają wiadomość bez oczekiwania na odpowiedź).

6. Zalety brokerów wiadomości

- Zapewnia komunikację pomiędzy serwisami, które mogą nie działać w tym samym czasie. Producent może wysyłać wiadomości niezależnie od tego, czy konsument jest aktywny, czy nie. Wszystko czego potrzebuje to działający broker wiadomości. To samo dotyczy konsumenta.
- Poprawia wydajność systemu dzięki wprowadzeniu przetwarzania asynchronicznego. Zadania wymagające dużej ilości czasu można rozdzielić na osobne procesy, co przyczynia się do przyspieszenia aplikacji.
- Zwiększa niezawodność poprzez zagwarantowanie transmisji wiadomości. Brokery wiadomości oferują mechanizm ponownego dostarczenia. W przypadku awarii konsumenta może ponownie dostarczyć wiadomość natychmiast lub po określonym czasie.

7. Wady brokerów wiadomości.

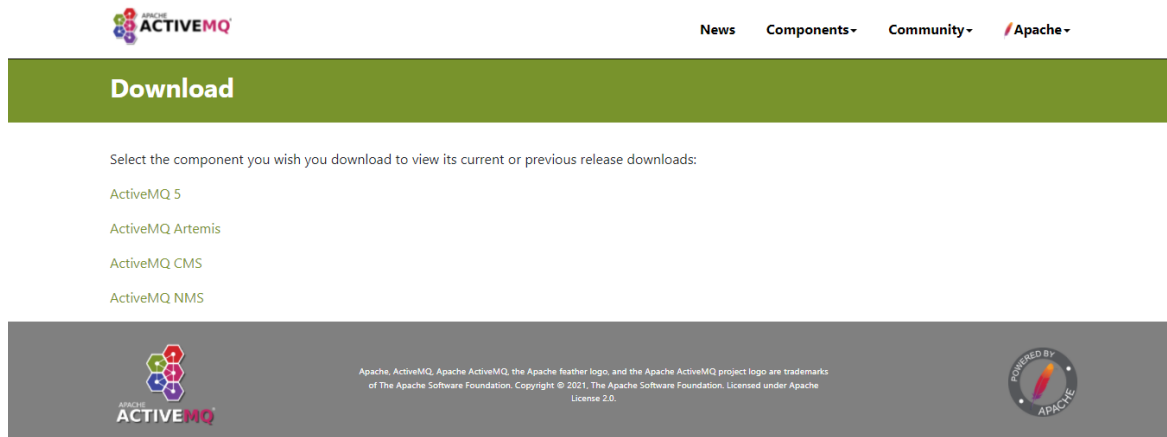
Używanie brokerów wiadomości wiąże się z przetwarzaniem asynchronicznym. Wady związane z użyciem brokera wiążą się z wyzwaniami jakie napotkamy używając komunikacji asynchronicznej.

- Zwiększona złożoność systemu. Wprowadzenie brokera komunikatów do systemu wiąże się między innymi z utrzymaniem sieci między komponentami oraz kwestiami bezpieczeństwa. Dodatkowo pojawia się problem z ostateczną spójnością. Niektóre komponenty mogą nie mieć aktualnych danych dopóki komunikaty nie zostaną rozpropagowane i przetworzone.
- Debugowanie może być trudniejsze.

8. Część praktyczna

8.1. Instalacja ActiveMQ

1. Wchodzimy na stronę <https://activemq.apache.org/download>



Rysunek 3. Strona z linkiem do pobrania ActiveMQ

2. Wybieramy ActiveMQ 5
3. Pobieramy paczkę zip z najnowszą dystrybucją programu



Rysunek 4. Plik zip, który należy pobrać.

4. Wypakowujemy archiwum do dowolnego folderu.

Nazwa	Data modyfikacji	Typ	Rozmiar
bin	26.01.2022 11:49	Folder plików	
conf	26.01.2022 11:49	Folder plików	
data	26.01.2022 11:51	Folder plików	
docs	26.01.2022 11:49	Folder plików	
examples	26.01.2022 11:49	Folder plików	
lib	26.01.2022 11:49	Folder plików	
webapps	26.01.2022 11:49	Folder plików	
webapps-demo	26.01.2022 11:49	Folder plików	
activemq-all-5.16.3.jar	26.01.2022 11:49	Executable Jar File	18 189 KB
LICENSE	26.01.2022 11:49	Plik	41 KB
NOTICE	26.01.2022 11:49	Plik	4 KB
README.txt	26.01.2022 11:49	Dokument tekstowy	3 KB

Rysunek 5. Zawartość folderu po wypakowaniu pliku zip.

5. Uruchamiamy wiersz poleceń i przechodzimy do katalogu, gdzie wypakowaliśmy paczkę zip a dokładnie do folderu bin

```

C:\Users\Kamil>cd C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin

```

Rysunek 6. Komenda przejścia do katalogu z ActiveMQ

6. Wpisujemy komendę „activemq start”

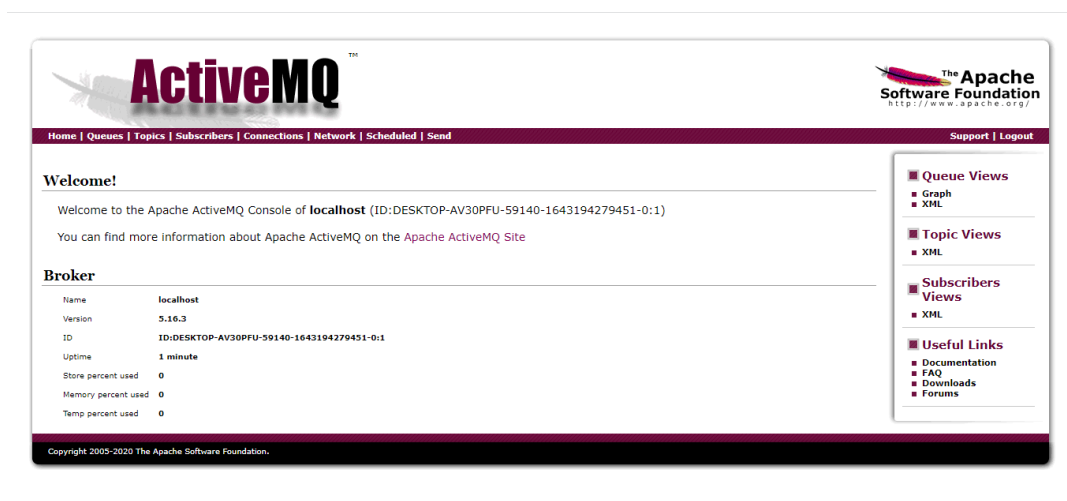
```

C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin>activemq start
Java Runtime: Oracle Corporation 11.0.12 C:\Program Files\Java\jdk-11.0.12
Heap sizes: current=1048576k free=1041918k max=1048576k
JVM args: -Dcom.sun.management.jmxremote -Xms1G -Xmx1G -Djava.util.logging.config.file=logging.properties -Djava.security.a
uth.login.config=C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\conf\login.config -Dactivemq.classpath
=C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\conf;C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-
activemq-5.16.3\bin\..\conf;C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\conf; -Dactivemq.home=C:\Us
ers\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\ -Dactivemq.base=C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\ap
ache-activemq-5.16.3\bin\..\ -Dactivemq.conf=C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\conf -Dacti
vemq.data=C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\data -Djava.io.tmpdir=C:\Users\Kamil\Desktop\
Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\data\tmp
Extensions classpath:
[C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\lib;C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache
-activemq-5.16.3\bin\..\lib\camel;C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\lib\optional;C:\Users
\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\lib\web;C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activem
q-5.16.3\bin\..\lib\extra]
ACTIVEMQ_HOME: C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..
ACTIVEMQ_BASE: C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..
ACTIVEMQ_CONF: C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\conf
ACTIVEMQ_DATA: C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\data
Loading message broker from: xbean:activemq.xml
INFO | Refreshing org.apache.activemq.xbean.XBeanBrokerFactory$1@25ce9dc4: startup date [Wed Jan 26 11:51:18 CET 2022]; root o
f context hierarchy
INFO | Using Persistence Adapter: KahaDBPersistenceAdapter[C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bi
n\..\data\kahadb]
INFO | PListStore[C:\Users\Kamil\Desktop\Semestr 7\ActiveMQ\apache-activemq-5.16.3\bin\..\data\localhost\tmp_storage] started
INFO | Apache ActiveMQ 5.16.3 (localhost, ID:DESKTOP-AV30PFU-59140-1643194279451-0:1) is starting
INFO | Listening for connections at: tcp://DESKTOP-AV30PFU:61616?maximumConnections=1000&wireFormat.maxFrameSize=104857600
INFO | Connector openwire started
INFO | Listening for connections at: amqp://DESKTOP-AV30PFU:5672?maximumConnections=1000&wireFormat.maxFrameSize=104857600
INFO | Connector amqp started
INFO | Listening for connections at: stomp://DESKTOP-AV30PFU:61613?maximumConnections=1000&wireFormat.maxFrameSize=104857600
INFO | Connector stomp started
INFO | Listening for connections at: mqtt://DESKTOP-AV30PFU:1883?maximumConnections=1000&wireFormat.maxFrameSize=104857600
INFO | Connector mqtt started
INFO | Starting Jetty server
INFO | Creating Jetty connector
WARN | ServletContext@0.e.j.s.ServletContextHandler@5d1b9c3d{/null,STARTING) has uncovered http methods for path: /
INFO | Listening for connections at ws://DESKTOP-AV30PFU:61614?maximumConnections=1000&wireFormat.maxFrameSize=104857600
INFO | Connector ws started
INFO | Apache ActiveMQ 5.16.3 (localhost, ID:DESKTOP-AV30PFU-59140-1643194279451-0:1) started
INFO | For help or more information please see: http://activemq.apache.org
INFO | ActiveMQ WebConsole available at http://127.0.0.1:8161/
INFO | ActiveMQ Jolokia REST API available at http://127.0.0.1:8161/api/jolokia/

```

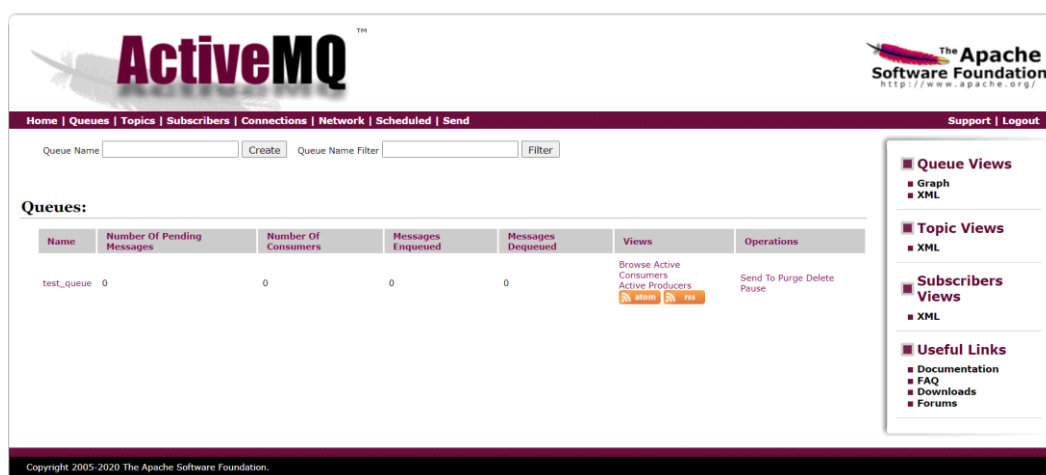
Rysunek 7. Widok po zainstalowaniu ActiveMQ.

ActiveMQ zostało zainstalowane, teraz możemy korzystać z konsoli ActiveMQ na serwerze lokalnym pod adresem 8161.



Rysunek 8. Konsola ActiveMQ

Tworzenie kolejki, do której będziemy wysyłać komunikaty



Rysunek 9. Widok kolejek w konsoli ActiveMQ.

Kolejka składa się z kilku właściwości:

Number of Pending Messages - liczba wiadomości, które nie zostały jeszcze dostarczone.

Number Of Customers – liczba konsumentów.

Messages Enqueued - liczba wiadomości, które zostały opublikowane od startu serwera.

Messages Dequeued - liczba wiadomości, które zostały usunięte od czasu startu serwera.

8.2. Program w Javie

Do stworzenia programu w Javie został użyty SpringBoot i Maven z zależnościami : SpringWeb i Spring for Apache ActiveMQ 5

8.2.1. Klasy programu:

8.2.1.1. *JmsConfig*



```
@Configuration
@EnableJms
public class JmsConfig {

    @Bean
    public DefaultJmsListenerContainerFactory jmsListenerContainerFactory(ConnectionFactory connectionFactory){
        DefaultJmsListenerContainerFactory jmsListenerContainerFactory =
            new DefaultJmsListenerContainerFactory();

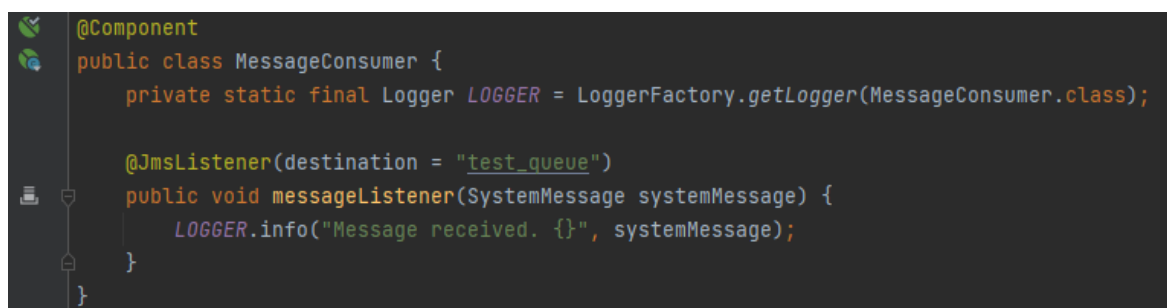
        jmsListenerContainerFactory.setConnectionFactory(connectionFactory);
        jmsListenerContainerFactory.setConcurrency("5-10");

        return jmsListenerContainerFactory;
    }
}
```

Rysunek 10. Zawartość klasy *JmsConfig*

To klasa konfiguracyjna, służąca do nawiązania połączenia z kolejką ActiveMQ, przetwarzania komunikatów z kolejki przy pomocy @JmsListener

8.2.1.2. *MessageConsumer*



```
@Component
public class MessageConsumer {
    private static final Logger LOGGER = LoggerFactory.getLogger(MessageConsumer.class);

    @JmsListener(destination = "test_queue")
    public void messageListener(SystemMessage systemMessage) {
        LOGGER.info("Message received. {}", systemMessage);
    }
}
```

Rysunek 11. Zawartość klasy *MessageConsumer*

W tej klasie zdefiniowany jest Logger, który będzie pobierał logi o komunikatach pobranych z kolejki ActiveMQ.

Metoda messageListener nasłuchuje kolejki „test_queue” i wyświetla dane o otrzymanym komunikacie używając wcześniej zdefiniowanego Loggera.

8.2.1.3. PublishController

```
@RestController
public class PublishController {

    @Autowired
    private JmsTemplate jmsTemplate;

    @PostMapping("/publishMessage")
    public ResponseEntity<String> publishMessage(
        @RequestBody SystemMessage systemMessage){

        try{
            jmsTemplate.convertAndSend( destinationName: "test_queue", systemMessage);
            return new ResponseEntity<>( body: "Sent", HttpStatus.OK);
        }catch (Exception e){
            return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

Rysunek 12. Zawartość klasy PublishController

Ta klasa to kontroler REST'fullowy.

Na początku wstrzykujemy klasę pomocniczą JmsTemplate, która pomaga wysyłać i otrzymywać wiadomości.

Następnie zdefiniowana jest metoda POST, która będzie wyzwalala publikowanie komunikatów do naszej kolejki „test_queue”.

8.2.1.4. SystemMessage

```
public class SystemMessage implements Serializable {

    private static final long serialVersionUID = 1L;

    private String source;
    private String message;

    public String getSource() { return source; }

    public void setSource(String source) { this.source = source; }

    public String getMessage() { return message; }

    public void setMessage(String message) { this.message = message; }

    @Override
    public String toString(){
        return "SystemMessage{" + "source='" + source + '\'' + ", message='" + message + '\'' + '}';
    }
}
```

Rysunek 13. Zawartość klasy SystemMessage

To klasa, której obiekty będą wysyłane do kolejki, a następnie odbierane i wyświetlane w konsoli przez klasę MessageConsumer.

8.3. Działanie programu

Uruchamiamy aplikację

```
INFO 976 --- [main] com.example.projekt.ProjektApplication : Starting ProjektApplication using Java 17.0.1 on DESKTOP-AV30PFU with PID 976
INFO 976 --- [main] com.example.projekt.ProjektApplication : No active profile set, falling back to default profiles: default
INFO 976 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8090 (http)
INFO 976 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
INFO 976 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.56]
INFO 976 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
INFO 976 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 789 ms
INFO 976 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8090 (http) with context path ''
INFO 976 --- [main] com.example.projekt.ProjektApplication : Started ProjektApplication in 1.69 seconds (JVM running for 2.255)
```

Rysunek 14. Zawartość konsoli po uruchomieniu programu

Jak widzimy działa ona na porcie 8090:

Następnie korzystamy z aplikacji Postman do wysłania rządania POST. Ciało rządania składa się z dwóch atrybutów: source i message, tak jak wcześniej zdefiniowana klasa SystemMessage.

<http://localhost:8090/publishMessage>



Rysunek 15. Widok z programu Postman.

Kolejka przed wysłaniem rządania:



Queues:

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
test_queue	0	0	0	0	Browse Active Consumers Active Producers atom rss	Send To Purge Delete Pause

Rysunek 16. Zawartość kolejki przed wysłaniem rządanie POST

Kolejka po wysłaniu 3 rządań

Queues:

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
test_queue	0	5	3	3	Browse Active Consumers Active Producers  atom  rss	Send To Purge Delete Pause

Rysunek 17. Zawartość kolejki po wysłaniu 3 rządań POST

Na konsoli została wyświetlone wiadomości pobrane z kolejki za pomocą klasy MessageConsumer

```
INFO 18372 --- [ntContainer#0-5] c.e.p.c.component.MessageConsumer : Message received. SystemMessage{source='Test 1', message='Test message 1.'}
INFO 18372 --- [ntContainer#0-1] c.e.p.c.component.MessageConsumer : Message received. SystemMessage{source='Test 2', message='Test message 2.'}
INFO 18372 --- [ntContainer#0-4] c.e.p.c.component.MessageConsumer : Message received. SystemMessage{source='Test 3', message='Test message 3.'}
```

Rysunek 18. Komunikaty pobrane z kolejki

9. Podsumowanie

Celem projektu było pokazania zastosowania brokera ActiveMQ.

W projekcie w prosty sposób został zaprezentowane działanie takiego brokera, poprzez wysłanie do kolejki prostego komunikatu w formacie JSON, a następnie pobranie go z kolejki i wyświetlenie w konsoli.

10. Linki

<https://github.com/KamilMadej/Projekt-us-ugi-sieciowe>

<https://activemq.apache.org/>