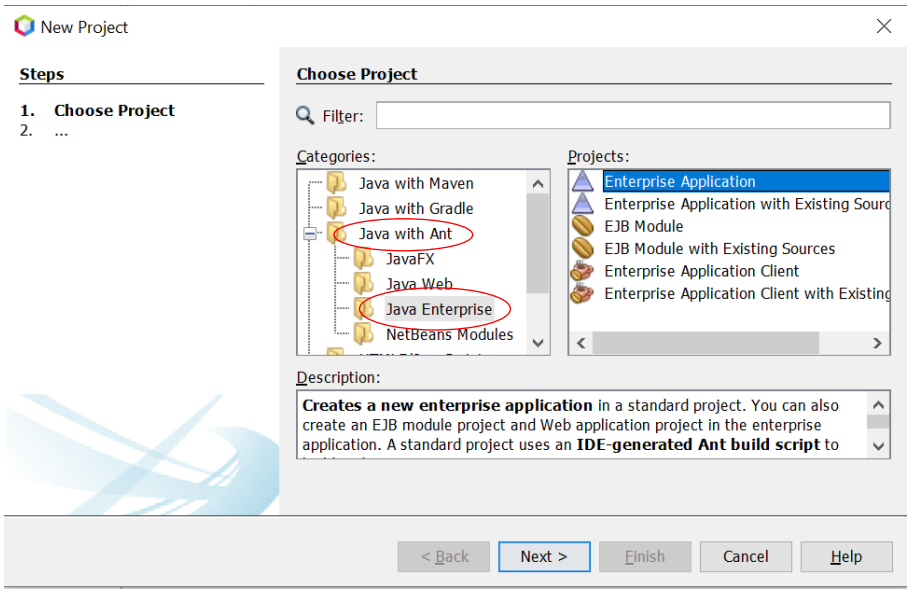
Message-Driven Beans

Celem ćwiczenia jest zapoznanie z komunikatowymi komponentami EJB w wersji 3.x w kontekście aplikacji Java EE wykorzystującej również sesyjne EJB, encje JPA i strony JSF.

Ćwiczenia zostały przygotowane dla środowiska NetBeans 11.2 i serwera GlassFish.

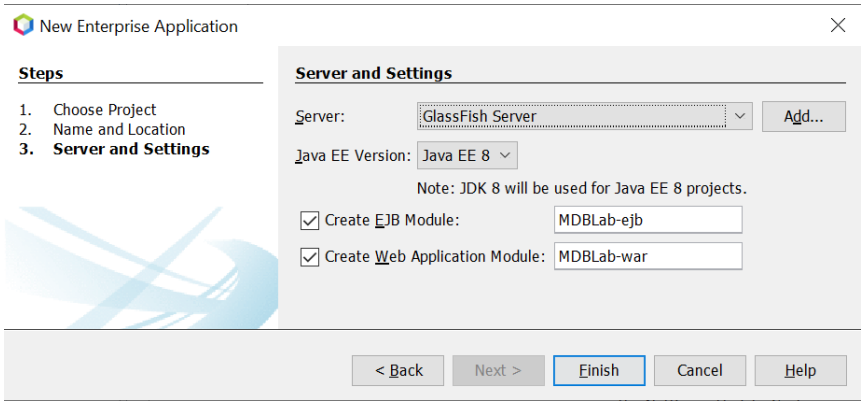
1. Uruchom środowisko NetBeans.

2. Utwórz nowy projekt typu Java with Ant/Java Enterprise/Enterprise Application:
   a) Wybierz z menu opcję File → New Project. Jako typ projektu wybierz Enterprise Application z kategorii Java with Ant/Java Enterprise.

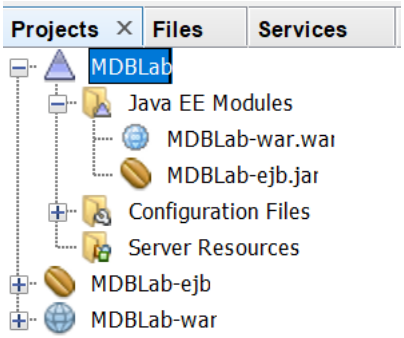


   b) W pierwszym kroku kreatora projektu jako nazwę projektu podaj **MDBLab**. Pozostałe ustawienia w tym oknie pozostaw domyślne.

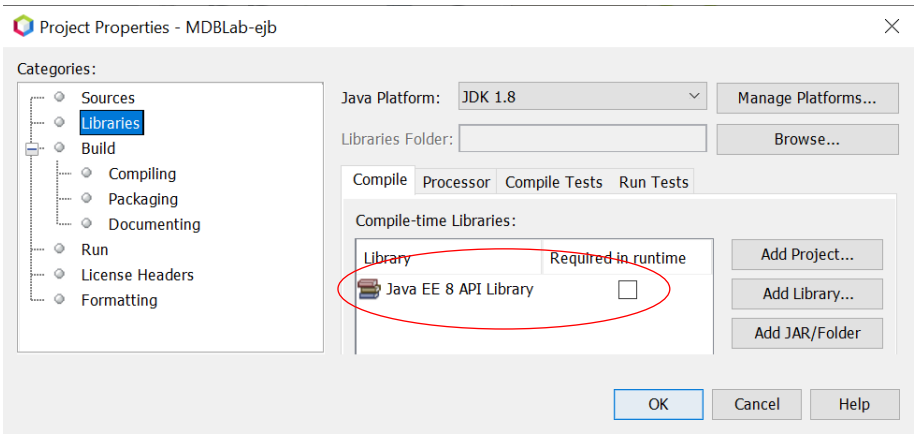
   c) W drugim kroku wybierz wersję Java EE Java EE 8 Web i upewnij się, że jako serwer aplikacji wybrany jest GlassFish (jeśli nie jest dostępny do wyboru, to kliknij Add... i pobierz oraz zainstaluj najnowszą dostępną wersję). Upewnij się czy zaznaczone jest tworzenie modułów EJB i webowego. Pozostaw zaproponowane przez kreator nazwy modułów.



d) Po zakończeniu działania kreatora obejrzyj organizację aplikacji w oknie Projects. Zwróć uwagę, że w środowisku NetBeans moduły EJB i webowy aplikacji Enterprise Application stanowią odrębne projekty, a główny projekt łączy je w całość wskazując jako swoje moduły składowe.



3. Sprawdź czy zarówno projekt EJB jak i projekt webowy mają dołączoną bibliotekę Java EE 8 API Library (opcją Properties z menu kontekstowego węzła projektu w panelu Projects). Jeśli nie, dodaj do obu tę bibliotekę, z odznaczonym polem wyboru wskazującym czy biblioteka ma być spakowana wraz z projektem by była dostępna w czasie pracy aplikacji. Spakowanie tej biblioteki do archiwum instalacyjnego projektu nie jest potrzebne, gdyż będzie ona dostępna na serwerze.



4. W projekcie modułu EJB utwórz nową jednostkę trwałości (Persistence Unit):

a) W oknie Projects wybierz z menu kontekstowego dla węzła projektu modułu EJB opcję New → Persistence Unit (z kategorii Persistence).
b) Jako dostawcę usług trwałości wybierz „EclipseLink”, a jako źródło danych - „jdbc/sample”. Pozostałe opcje pozostaw domyślne.

5. W projekcie modułu EJB utwórz nową klasę encji do reprezentowania wiadomości prasowych posiadających tytuł i treść:

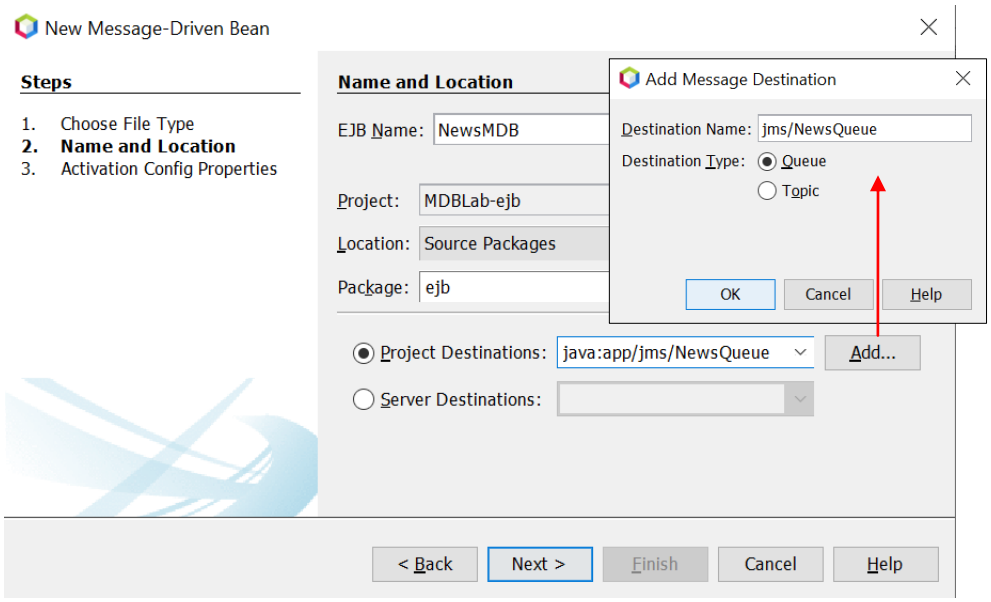
   a) W oknie Projects wybierz z menu kontekstowego dla węzła projektu modułu EJB opcję New → Entity Class.
   b) Jako nazwę klasy podaj `NewsItem`, a jako pakiet `ejb`. Pozostaw `Long` jako typ klucza głównego.

   c) Dodaj w klasie encji dwa prywatne pola typu `String` o nazwach `heading` i `body`. Wygeneruj dla nich publiczne metody setter/getter (wykorzystaj do tego celu kreator Refactor → Encapsulate Fields).

6. W projekcie modułu EJB utwórz komunikatowy komponent EJB, którego zadaniem będzie utrwalanie w bazie danych nadesłanego obiektu klasy `NewsItem`:

   a) W oknie Projects wybierz z menu kontekstowego dla węzła projektu modułu EJB opcję New → Message-Driven Bean.

   b) Jako nazwę klasy komponentu podaj `NewsMDB`, a jako pakiet `ejb`. Następnie kliknij przycisk Add obok pola Project Destinations, aby utworzyć nowe miejsce przeznaczenia wiadomości. Jako nazwę miejsca przeznaczenia podaj „`jms/NewsQueue`”, a jako typ miejsca przeznaczenia wybierz kolejkę.



   c) Zakończ tworzenie komponentu komunikatowego przechodząc do ostatniego ekranu kreatora, pozostawiając w nim zaproponowane ustawienia konfiguracyjne i klikając Finish. Obejrzyj wygenerowany kod klasy zwracając szczególną uwagę na adnotacje `@JMSDestinationDefinition` i `@MessageDriven`. Pierwsza definiuje kolejkę jako zasób na poziomie aplikacji. Druga sprawia, że klasa staje się komponentem komunikatowym i wskazuje nazwę JNDI miejsca przeznaczenia JMS i jego typ (kolejka lub temat). Zwróć uwagę na zgodność nazwy JNDI kolejki jako zasobu w obu adnotacjach i prefiks „java:app/”, od którego nazwa ta się rozpoczyna. Jest to prefiks dla zasobów lokalnych dla aplikacji.

Uwaga: Jeśli NetBeans „zapomni” o adnotacji `@JMSDestinationDefinition`, dodaj ją ręcznie w poniższej formie, bezpośrednio przed adnotacją `@MessageDriven`:

```
@JMSDestinationDefinition(name = "java:app/jms/NewsQueue",
interfaceName = "javax.jms.Queue", resourceAdapter = "jmsra",
destinationName = "NewsQueue")
```

7. Zmodyfikuj kod komponentu komunikatowego w taki sposób, aby reagował na wiadomość zawierającą obiekt `NewsItem`, zapisując go do bazy danych:

   a) Wstrzyknij zarządcę encji JPA do komponentu komunikatowego wstawiając poniższy kod przed konstruktorem:

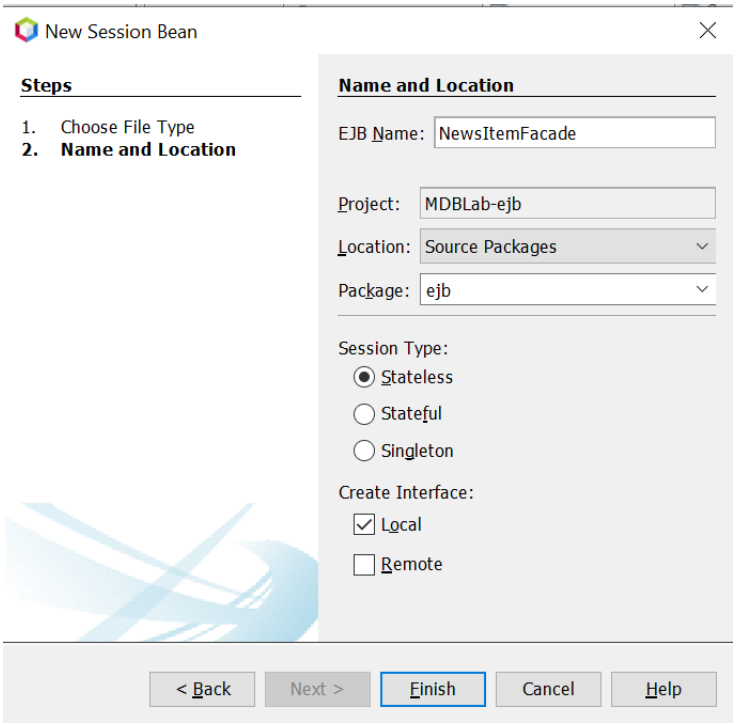
```
    @PersistenceContext
    private EntityManager em;
```

   b) Zmodyfikuj metodę `onMessage` umieszczając w niej poniższy kod (zaimportuj również wykorzystywane klasy biblioteczne JMS):

```
ObjectMessage msg = null;
 try {
     if (message instanceof ObjectMessage) {
         msg = (ObjectMessage) message;
         NewsItem e = (NewsItem) msg.getObject();
         em.persist(e);
     }
 } catch (JMSException e) {
     e.printStackTrace();
 }
```

8. Utwórz sesyjną fasadę do obsługi encji z poziomu wyższych warstw aplikacji:

   a) Wywołaj z poziomu projektu modułu EJB kreator Session Bean. Jako nazwę beana podaj „NewsItemFacade”, a jako pakiet „ejb”. Wskaż, że komponent ma być bezstanowy i zleć utworzenie tylko interfejsu lokalnego.



b) Przejdź do edycji klasy utworzonego komponentu sesyjnego i wstrzyknij do niego zarządcę encji JPA w ten sam sposób co wcześniej w komponencie komunikatowym.

c) Dodaj w komponencie sesyjnym poniższą metodę zwracającą wynik zapytania zbudowanego z wykorzystaniem Criteria API JPA, pobierającego z bazy danych wszystkie newsy. Zaimportuj wszystkie wykorzystywane klasy.

```
public List<NewsItem> getAllNewsItems() {
  CriteriaBuilder cb = em.getCriteriaBuilder();
  CriteriaQuery<NewsItem> cq = cb.createQuery(NewsItem.class);
  Root<NewsItem> rootEntry = cq.from(NewsItem.class);
  CriteriaQuery<NewsItem> ct = cq.select(rootEntry);
  TypedQuery<NewsItem> allNewsItemsQuery = em.createQuery(ct);
  return allNewsItemsQuery.getResultList();
}
```

d) Udostępnij metodę dodaną do komponentu sesyjnego przez jego interfejs lokalny, edytując ręcznie kod interfejsu.

e) Sprawdź czy NetBeans nie utworzył pliku `ejb-jar.xml` w gałęzi Configuration Files projektu EJB. Plik ten jest opcjonalny, a w postaci utworzonej przez NetBeans może powodować błędy. Jeśli ten plik istnieje, usuń go prawym klawiszem myszy z poziomu panelu Projects.

9. W projekcie modułu webowego utwórz nową stronę JSF (JSF Page). Jako jej nazwę podaj `news`, pozostałe opcje pozostaw domyślne. Zwróć uwagę na domyślny wybór Facelets zamiast JSP.

10. Dodaj do projektu plik konfiguracyjny JSF (`faces-config.xml`) korzystając z kreatora New File→JavaServer Faces/JSF Faces Configuration. Nie będziemy edytować zawartości tego pliku. Celem dodania go jest aktywacja technologii JSF

w projekcie, która jest aktualnie wymagana by można było korzystać na stronach JSF z beanów CDI.

11. Utwórz w projekcie modułu webowego komponent CDI, który będzie pełnił funkcję backing bean dla utworzonej w poprzednim kroku strony JSF. Komponent będzie udostępniał m.in. metody do odczytu wiadomości z bazy danych i do wysyłania wiadomości do kolejki JMS:

    a) Uruchom w projekcie webowym kreator JSF CDI Bean. Jako nazwę klasy podaj „NewsBean" a jako nazwę pakietu „web". Koniecznie zmień zaproponowany zasięg komponentu na `request`.

    b) Dodaj w klasie komponentu CDI (`NewsBean.java`) wstrzyknięcie referencji do komponentu fasadowego EJB poprzez jego interfejs lokalny. Wykorzystaj poniższy kod. Nie zapomnij o wymaganych importach.

```
    @Inject
    private NewsItemFacadeLocal facade;
```

    c) Dodaj w klasie komponentu CDI podane poniżej wstrzyknięcie zależności dla kontekstu JMS oraz kolejki wiadomości (wykorzystana w kodzie adnotacja to `javax.annotation.Resource`):

```
    @Inject
    private JMSContext context;
    @Resource(lookup="java:app/jms/NewsQueue")
    private  javax.jms.Queue queue;
```

    Uwaga: Powyższa adnotacja `@Resource` wskazuje kolejkę zdefiniowaną wcześniej jako zasób aplikacji (tę samą, z której wiadomości pobierać ma utworzony wcześniej komunikatowy EJB).

    d) Dodaj w klasie komponentu CDI poniższą metodę, wysyłającą wiadomość z obiektem `NewsItem` do kolejki (zaimportuj wykorzystywane klasy biblioteczne JMS i klasę encji):

```
void sendNewsItem(String heading, String body) {
  try {
    ObjectMessage message = context.createObjectMessage();
    NewsItem e = new NewsItem();
    e.setHeading(heading);
    e.setBody(body);
    message.setObject(e);

    context.createProducer().send(queue, message);
  } catch (JMSException ex) {
     ex.printStackTrace();
  }
}
```

e) Dodaj w klasie komponentu CDI metodę (niekompletny kod metody poniżej) odczytującą listę wszystkich obiektów `NewsItem` z bazy danych poprzez wstrzyknięty komponent fasadowy:

```
public List<NewsItem> getNewsItems()
{
    return ...;
}
```

12. Umieść na stronie JSF formularz umożliwiający wysyłanie nowych wiadomości do kolejki. Formularz powinien zawierać następujące komponenty JSF: dwa pola tekstowe (znacznik `<h:inputText>`, identyfikatory pól: `headingInputText` i `bodyInputText`) do wprowadzania danych poprzedzone etykietami (znacznik `<h:outputText>`) oraz przycisk (znacznik `<h:commandButton>`, identyfikator: `submitButton`) uruchamiający akcję. (W celu utworzenia formularza ręcznie edytuj kod strony wspomagając się podpowiedziami w edytorze.)

13. Dla obu pól formularza do wprowadzania danych utwórz odpowiadające im właściwości typu `String` w klasie komponentu CDI (o nazwach `headingText` i `bodyText`) i powiąż je z wartościami komponentów.

```
...
<h:inputText id="headerInputText"
             value="#{newsBean.headingText}"> </h:inputText>
...
<h:inputText id="bodyInputText"
             value="#{newsBean.bodyText}"></h:inputText>
...
```

14. Dodaj w kodzie komponentu CDI metodę akcji (niekompletny kod metody poniżej) i powiąż ją z przyciskiem formularza. Metoda akcji powinna wywoływać metodę `sendNewsItem`, przekazując jej jako parametry wartości pochodzące z pól formularza.

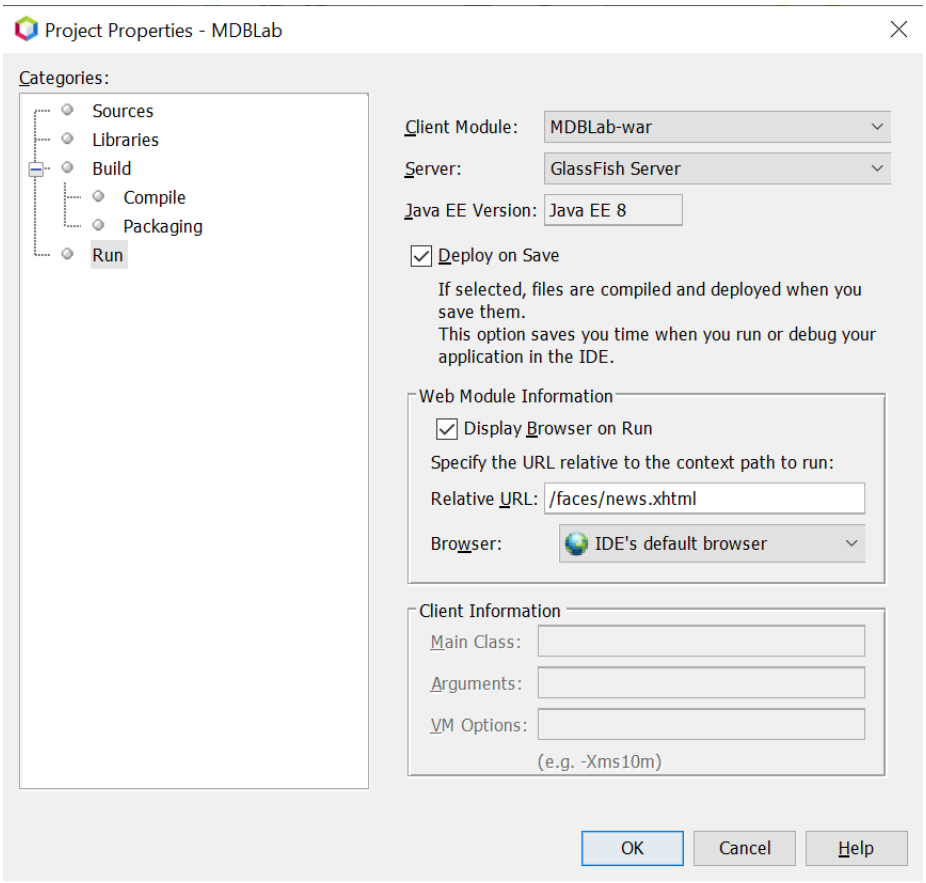
```
public String submitNews()
{
    ...;
    return null;
}
```

15. Na końcu formularza na stronie JSF umieść komponent tabelki do wyświetlania wiadomości zapisanych w bazie danych. Możesz w tym celu posłużyć się kreatorem wywoływanym poprzez wybranie komponentu JSF Data Table From Entity z okienka, wyświetlanego po wybraniu z menu kontekstowego pozycji Insert Code. (Uwaga: Jeśli lista encji do wyboru jest pusta, to zamknij środowisko NetBeans, otwórz je ponownie i ponownie uruchom kreator…)

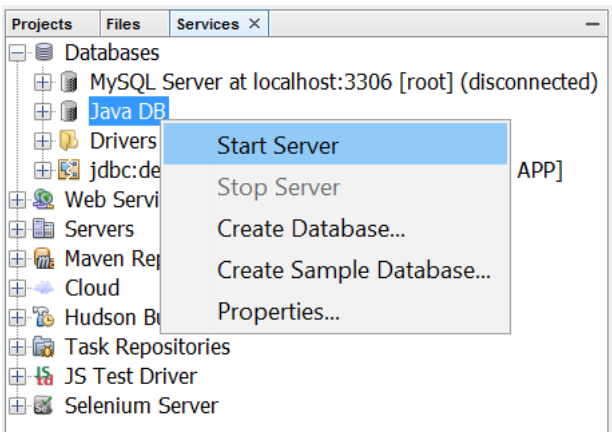
16. Popraw kod wygenerowany przez kreator:

    a) Usuń znaczniki, który kreator wstawił przez znacznikiem otwierającym `<h:dataTable>` oraz po zamykającym `</h:dataTable>`.
    b) W elemencie `<html>` dodaj atrybut z kolejną przestrzenią nazw (JSF Core): `xmlns:f="http://xmlns.jcp.org/jsf/core"`. (Tylko jeśli wcześniej nie został dodany automatycznie!)

17. Ustaw stronę `news.xhtml` jako stronę startową aplikacji, modyfikując właściwości głównego projektu aplikacji.



18. Uruchom serwer bazy danych Java DB (jeżeli jeszcze nie działa).



19. Uruchom i przetestuj aplikację (uruchamiając główny projekt aplikacji). Wyślij kilka wiadomości. Spróbuj zaobserwować efekt komunikacji asynchronicznej.

    Komentarz: Ze względu na asynchroniczną realizację dodawania nowych informacji, może się zdarzyć, że zawartość tabelki na stronie będzie odświeżona zanim nowy wiersz trafi do bazy danych.

Zadanie do samodzielnego wykonania

Zmodyfikuj aplikację tak, aby do kolejki była wysyłana wiadomość `TextMessage` (zamiast `ObjectMessage`). Treścią wiadomości powinien być tekst składający się z nagłówka i treści newsa, oddzielonych pionową kreską. Komponent komunikatowy powinien podzielić odebrany tekst na nagłówek i treść newsa, utworzyć obiekt encji `NewsItem` i utrwalić go w bazie danych.