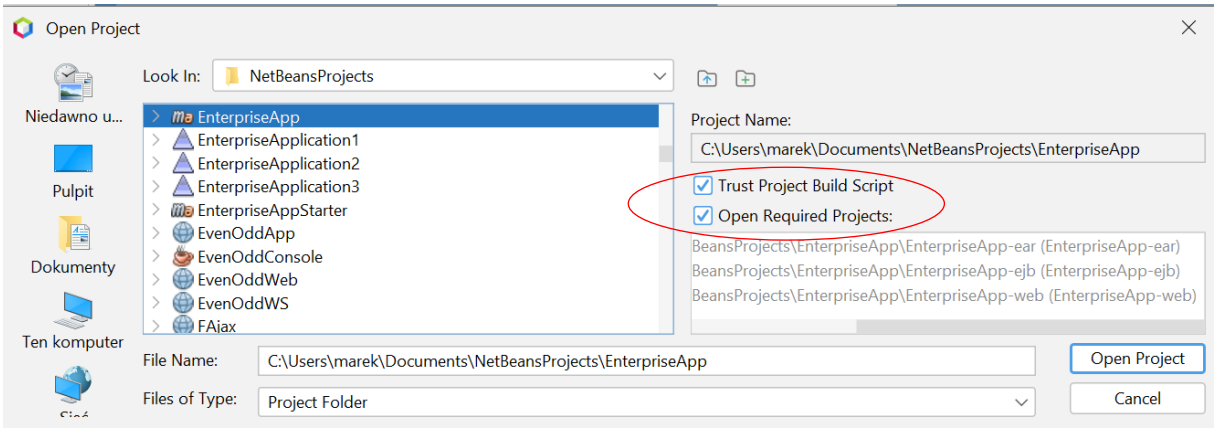
Message-Driven Beans

Celem ćwiczenia jest zapoznanie z komunikatowymi komponentami EJB w kontekście aplikacji Jakarta EE wykorzystującej również sesyjne EJB, encje JPA i strony JSF. Ćwiczenia zostały przygotowane dla środowiska NetBeans 17 i serwera Payara.

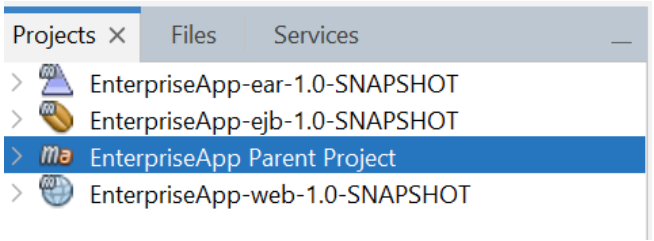
1. Pobierz archiwum zawierające hierarchię projektów Mavena EnterpriseApp i rozpakuj je na dysku.

2. Uruchom środowisko NetBeans.

3. Otwórz projekt nadrzędny o nazwie EnterpriseApp wraz z projektami składowymi. W tym celu wybierz z menu opcję File → Open Project. Wskaż projekt EnterpriseApp. Upewnij się, że zaznaczone są pola wyboru dotyczące ufania skryptom projektów i automatycznego otwarcia projektów składowych (powinny się pokazać trzy takie projekty w panelu po prawej stronie). Kliknij Open Project.

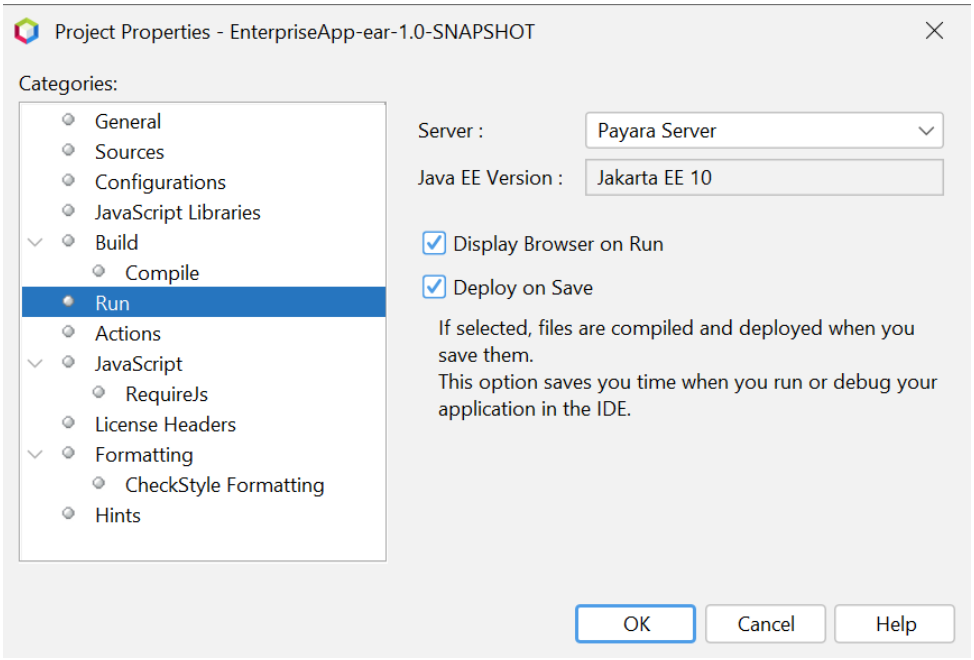


4. W panelu Projects powinny pojawić się 4 projekty:
   a) Projekt nadrzędny, z poziomu którego można będzie inicjować budowanie całej aplikacji.
   b) Projekt „ejb” z komponentami Enterprise Beans (moduł EJB).
   c) Projekt „web” z warstwą interfejsu użytkownika (moduł webowy).
   d) Projekt „ear” służący do utworzenia archiwum EAR łączącego moduły składowe aplikacji w archiwum typu Enterprise Archive. Z poziomu tego projektu będzie można w NetBeans uruchomić aplikację.



5. Otwórz i przeanalizuj zawartość plików pom.xml wszystkich 4 projektów, w kolejności podanej w poprzednim punkcie ćwiczenia. Szczególną uwagę zwróć na ustawienia <packaging> i konfigurację projektów składowych jako <dependencies> innych projektów.

6. Przejdź do właściwości (Properties) projektu „ear" i upewnij się, że do jego uruchomienia wybrany jest serwer Payara.



7. W projekcie modułu EJB znajduje się już plik `persistence.xml`. Odszukaj go i otwórz do edycji w trybie tekstowym. Umieść w nim poniższą definicję jednostki trwałości (Persistence Unit), zastępując istniejącą, jeśli plik jakąś już zawiera:

```
  <persistence-unit name="MDBLab-ejbPU" transaction-type="JTA">
    <jta-data-source>jdbc/__default</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="jakarta.persistence.schema-
generation.database.action" value="create"/>
    </properties>
  </persistence-unit>
```

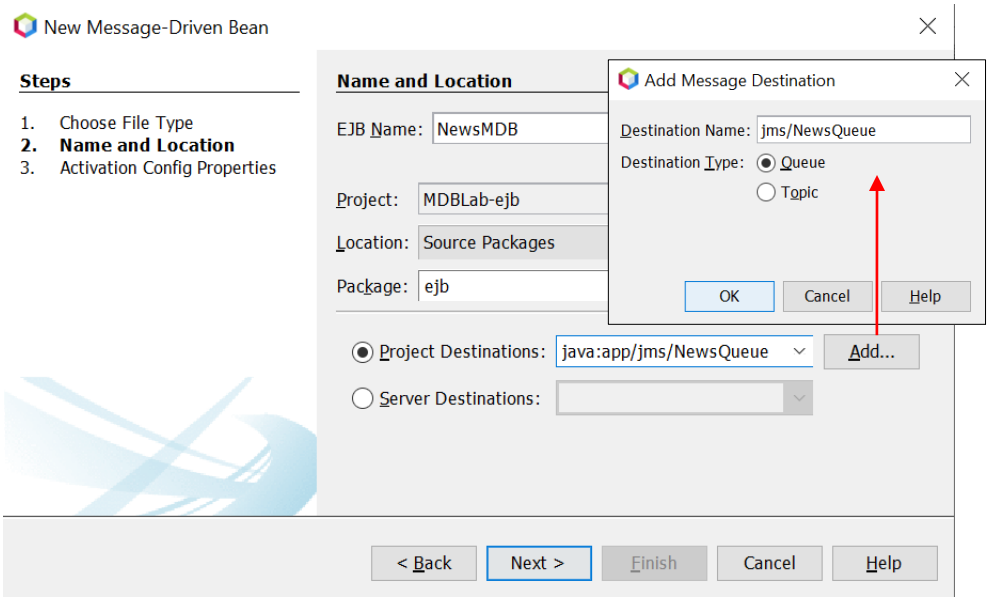
8. W projekcie modułu EJB utwórz nową klasę encji do reprezentowania wiadomości prasowych posiadających tytuł i treść:

   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) Popraw nazwy importowanych pakietów (javax => jakarta).
   d) 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).

9. 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.

   d) Popraw nazwy importowanych pakietów (javax => jakarta).

   e) Obejrzyj wygenerowany kod klasy zwracając szczególną uwagę na adnotację `@MessageDriven`. Sprawia ona, że klasa staje się komponentem komunikatowym i wskazuje nazwę JNDI miejsca przeznaczenia JMS i jego typ (kolejka lub temat). Popraw nazwę pakietu (javax => jakarta) w jednym z atrybutów adnotacji.

10. Komponent komunikatowy oczekuje odpowiedniej kolejki skonfigurowanej na serwerze. Kreator NetBeans przygotował plik `glassfish-resources.xml` (w folderze setup), który sprawi że podczas instalacji aplikacji na serwerze GlassFish/Payara kolejka wraz z fabryką połączeń zostaną skonfigurowane. Odszukaj ten plik w drzewie projektu „ejb" i obejrzyj jego zawartość. Popraw nazwę pakietu (javax => jakarta) w atrybutach elementów XML konfigurujących oba zasoby JMS.

11. 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;
```

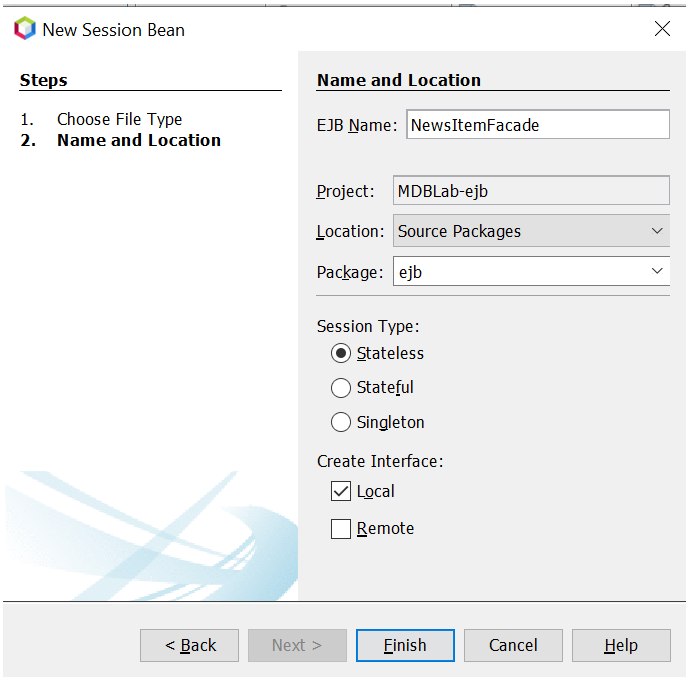
    Dodaj potrzebne importy z pakietu jakarta.persistence.

    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();
         }
```

12. 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. Popraw import. Następnie wstrzyknij do klasy komponentu 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 (uważaj by klasy JPA importować z pakietu jakarta.persistence).

```
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. Przy okazji popraw import w pliku z interfejsem.

13. 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.

14. 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`. Po utworzeniu komponentu popraw importy w jego klasie.

    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 `jakarta.annotation.Resource`):

```
    @Inject
    private JMSContext context;
    @Resource(lookup="jms/NewsQueue")
    private  jakarta.jms.Queue queue;
```

Uwaga: Powyższa adnotacja `@Resource` wskazuje kolejkę, która zostanie automatycznie skonfigurowana na serwerze przy instalacji 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 ...;
  }
```

15. 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.)

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

17. 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;
}
```

18. Na końcu formularza na stronie JSF umieść komponent tabelki do wyświetlania wiadomości zapisanych w bazie danych:

```
          <h:dataTable value="#{newsBean.newsItems}" var="item"
                       border="1">
                <h:column>
                    <f:facet name="header">
                        <h:outputText value="Id"/>
                    </f:facet>
                    <h:outputText value="#{item.id}"/>
                </h:column>
                <h:column>
                    <f:facet name="header">
                        <h:outputText value="Heading"/>
                    </f:facet>
                    <h:outputText value="#{item.heading}"/>
                </h:column>
                <h:column>
                    <f:facet name="header">
                        <h:outputText value="Body"/>
                    </f:facet>
                    <h:outputText value="#{item.body}"/>
                </h:column>
          </h:dataTable>
```

19. W elemencie `<html>` dodaj atrybut z kolejną przestrzenią nazw (JSF Core): `xmlns:f="http://xmlns.jcp.org/jsf/core"`.

20. Zbuduj aplikację (z poziomu projektu nadrzędnego). Możesz odszukać na dysku powstałe archiwum EAR i ewentualnie podejrzeć jego zawartość.

21. Uruchom i przetestuj aplikację (uruchamiając projekt „ear"). Popraw adres w przeglądarce na http://localhost:8080/EnterpriseApp/faces/news.xhtml. 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.