

Jakarta EE: JSF + CDI + JPA + BV

Ćwiczenie 1

Celem ćwiczenia jest utworzenie prostej aplikacji bazodanowej umożliwiającej przeglądanie i dodawanie żądań serwisowych. Ćwiczenie pokazuje współpracę technologii JSF, CDI i JPA w ramach platformy Jakarta EE. Do realizacji ćwiczenia wymagane jest środowisko NetBeans 17 wraz z serwerem aplikacji Payara i serwerem bazy danych H2 (dostępnym razem z serwerem aplikacji Payara).

1. Uruchom NetBeans i utwórz nowy projekt opcją **File**→**New Project...** W kreatorze projektu z listy kategorii wybierz **Java with Maven**, a z listy projektów wybierz **Web Application**. Kliknij przycisk **Next >**. Jako nazwę projektu podaj **Requests**, wyczyść pole z nazwą pakietu i kliknij przycisk **Next >**. Wybierz wersję **Java EE Jakarta EE 10 Web** i upewnij się, że jako serwer aplikacji wybrany jest **Payara** (jeśli nie jest dostępny do wyboru, to kliknij **Add...** i pobierz oraz zainstaluj najnowszą dostępną wersję). Kliknij przycisk **Finish**.

Odszukaj w pliku `pom.xml` wersję `maven-war-plugin`. Jeśli podana jest wcześniejsza niż 3.3.2, to zmień ją na 3.3.2. Zapisz zmiany.

2. Utwórz w projekcie jednostkę trwałości, w ramach której obiekty aplikacji będą zachowywane w bazie danych. W tym celu:
 - a) Odszukaj w strukturze projektu plik `persistence.xml` (powinien zostać utworzony przez kreator projektu i znajdować się w podkatalogu `META-INF`). Przejdź do edycji jego źródła otwierając plik, a następnie przełączając edytor na zakładkę **Source**.
 - b) Zastąp w kodzie źródłowym pliku `persistence.xml` cały element `<persistence-unit>` poniższą wersją, specyfikującą jednostkę trwałości powiązaną ze źródłem danych na serwerze aplikacji i wykorzystującą transakcje w standardzie JTA.

```
<persistence-unit name="RequestsPU" 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>
```

3. Utworzenie klasy encji `Request` do reprezentowania żądań.
 - a) Kliknij prawym przyciskiem myszy na ikonie projektu i z menu kontekstowego wybierz **New** → **Entity Class**.
 - b) Jako nazwę klasy podaj `Request`, a jako nazwę pakietu `req.entities`. Pozostaw zaproponowany typ `Long` jako typ identyfikatora encji.
 - c) Popraw nazwy importowanych pakietów (`javax => jakarta`).

- d) Dodaj w klasie encji dwa prywatne pola: `requestDate` typu `LocalDate` i `requestText` typu `String`. Dodaj import klasy `java.time.LocalDate`. Wygeneruj publiczne gettery i settery dla dodanych pól (skorzystaj z kreatora **Refactor** → **Encapsulate Fields**).
4. Utwórz w projekcie klasę `EntityManagerProducer`, który będzie implementować bean CDI pozwalający uzyskać obiekt `EntityManager` w sposób pozwalający na jego wstrzyknięcie jak beana CDI z jednoczesnym zapewnieniem, że nie będzie on współdzielony przez różne żądania. Umieść ją w pakiecie `data`. W tym celu:
- Z menu kontekstowego projektu uruchom kreator **Java Class** z kategorii **Java**.
 - Wprowadź nazwę klasy i nazwę pakietu, po czym kliknij przycisk **Finish**.
 - Jako ciało klasy wklej poniższy kod. Zwróć uwagę na zwykłe wstrzyknięcie obiektu `EntityManager` i na metodę producenta, która „opakowuje” go jako bean CDI o zasięgu żądania.

```
@PersistenceContext
private EntityManager em;

@Produces
@RequestScoped
public EntityManager getEntityManager() {
    return em;
}
```

- d) Wstaw poniższe instrukcje `import`. (Uwaga: W tym wypadku należy zwrócić uwagę by zaimportować adnotacje z właściwych pakietów, gdyż niektóre z nich powtarzają się w różnych pakietach.)

```
import jakarta.enterprise.context.RequestScoped;
import jakarta.enterprise.inject.Produces;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
```

5. Utwórz w projekcie klasę `RequestRepository`, który będzie oferować funkcjonalność CRUD do obsługi encji `Request`. Umieść ją w pakiecie `data`. W tym celu:
- Z menu kontekstowego projektu uruchom kreator **Java Class** z kategorii **Java**.
 - Wprowadź nazwę interfejsu i nazwę pakietu, po czym kliknij przycisk **Finish**.
 - Oznacz klasę adnotacją `@ApplicationScoped`.
 - W ciele klasy umieść poniższą kod wstrzykujący obiekt `EntityManager` jako bean CDI.

```
@Inject
private EntityManager em;
```

Komentarz: Zalecany zasięg CDI dla beana implementującego funkcjonalność repozytorium to zasięg aplikacyjny, który sprawi że jedna instancja beana będzie współdzielona przez wszystkie żądania i sesje użytkowników. W przypadku gdyby do takiego beana `EntityManager` był wstrzyknięty zwyczajnie adnotacją `@PersistenceContext`, to z racji zasięgu aplikacyjnego beana repozytorium, byłby on również współdzielony przez wszystkie żądania, co stanowiłoby duży błąd, gdyż

obiekty `EntityManager` nie są „thread-safe”. W przyjętym rozwiązaniu `EntityManager` jest „produkowany” z zasięgiem pojedynczego żądania przez metodę producenta zawartą w beanie o domyślnym zasięgu `@Dependent`, co jest odpowiednie w tym wypadku. CDI obsługuje wstrzykiwanie beana o mniejszym zasięgu do beana o większym zasięgu (co ma miejsce w naszym rozwiązaniu) poprzez obiekty proxy.

- e) Umieść w ciele klasy poniżej wstrzyknięcia obiektu `EntityManager` poniższe metody repozytorium.

```
public void create(Request entity) {
    em.persist(entity);
}

public void edit(Request entity) {
    em.merge(entity);
}

public void remove(Request entity) {
    em.remove(em.merge(entity));
}

public Request find(Object id) {
    return em.find(Request.class, id);
}

public List<Request> findAll() {
    CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
    cq.select(cq.from(Request.class));
    return em.createQuery(cq).getResultList();
}
```

- f) Uzupełnij brakujące importy. Zwróć uwagę aby klasy/interfejsy JPA importować z pakietów o nazwach rozpoczynających się od „jakarta”.
6. Utwórz w projekcie nową stronę JSF (New File→JavaServer Faces/JSF Page). Jako jej nazwę podaj `requestsList`. Zwróć uwagę aby powstała strona ze składnią Facelets (a nie JSP). Rozszerzenie pliku strony `xhtml` zostanie dodane automatycznie.
7. Utwórz komponent backing bean obsługujący stronę JSF. W tym celu:
- a) Utwórz w projekcie nowy komponent CDI (New File→JavaServer Faces/JSF CDI Bean), który będzie pełnił rolę backing bean dla utworzonej przed chwilą strony JSF. W pierwszym kroku kreatora kliknij przycisk **Next >**.

- b) W drugim kroku kreatora podaj nazwę klasy (`RequestsList`), pakiet (`req.backing`), nazwę komponentu (`requestsList` – zgodna z konwencją, powinna ustawić się automatycznie) i jego zasięg (`request`), a następnie kliknij przycisk **Finish**.

New JSF CDI Bean

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name:

Project:

Location:

Package:

Created File:

☐ Add data to configuration file

Configuration File:

Name:

Scope:

< Back Next > Finish Cancel Help

8. Przejdź do edycji kodu klasy komponentu backing bean (`RequestsList`). Wprowadź w nim poniższe modyfikacje:

- a) Popraw nazwy importowanych pakietów (`javax` => `jakarta`).
- b) Wstrzyknij referencję do komponentu repozytorium wstawiając w klasie przed konstruktorem poniższy kod (pamiętaj o niezbędnych importach!).

```
@Inject
private RequestRepository requestRepository;
```

- c) Poniżej konstruktora wklej poniższą metodę, która ma pośredniczyć w dostępie do metody komponentu repozytorium i udostępnić listę wszystkich obiektów encji w formie właściwości komponentu zarządzanego. Samodzielnie uzupełnij kod metody i zaimportuj wykorzystywane klasy (`java.util.List` oraz naszą klasę encji `Request`). Zapisz wszystkie zmiany.

```
public List<Request> getAllRequests() {
    ...
}
```

9. Przejdź do edycji źródła strony JSF (`requestsList.xhtml`). Usuń treść zawartą w elemencie `<h:body>` i zastąp ją poniższym kodem.

```
<h:form>
    <h:dataTable value="#{requestList.allRequests}"
        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="RequestDate"/>
            </f:facet>
            <h:outputText value="#{item.requestDate}"/>
        </h:column>
        <h:column>
            <f:facet name="header">
                <h:outputText value="RequestText"/>
            </f:facet>
            <h:outputText value="#{item.requestText}"/>
        </h:column>
    </h:dataTable>
</h:form>
```

10. Uzupełnij brakującą deklarację przestrzeni nazw.
11. Otwórz do edycji plik `web.xml`. Odszukaj w nim wskazanie strony startowej aplikacji i popraw ją na: `faces/requestsList.xhtml`.
12. Zapisz wszystkie zmiany i uruchom aplikację. W przeglądarce powinna wyświetlić się strona z tabelką z listą żądań. (Na razie bez żadnych danych.)
13. Przejdź do edycji źródła utworzonej strony JSF i wstaw wewnątrz elementu `<h:form>` przed elementem `<h:dataTable>` poniższe trzy elementy (korzystając z podpowiedzi i autouzupełniania edytora tekstowego):
- element `<h:outputText>` z atrybutem `value` o wartości „New request”
 - element `<h:inputText>` z atrybutem `id` o wartości „newReqInputText”
 - element `<h:commandButton>` z atrybutem `value` o wartości „Add”

14. Przejdź do edycji kodu klasy komponentu backing bean (`RequestsList`). Wywołaj poprzez menu kontekstowe edytora kreator `Insert Code` → `AddProperty`. Dodaj prywatną właściwość `newRequest` typu `String` z publicznymi metodami `getter` i `setter`. Zwróć uwagę, aby dodana właściwość nie zaburzyła poprawnej struktury klasy (aby własność nie została wstawiona bezpośrednio po adnotacji).

Add Property

Name: = ;

Type:

☒ private ☐ package ☐ protected ☐ public

☐ static ☐ final

☒ Generate getter and setter ☐ Generate getter ☐ Generate setter

☒ Generate javadoc

☒ Bound

☐ Vetoable

☐ Indexed

☐ Generate Property Change Support ☐ Generate Vetoable Change Support

Preview:

```
private String newRequest;

/**
 * Get the value of newRequest
```

15. Analogicznie do poprzedniego punktu ćwiczenia dodaj w klasie backing bean właściwość `requestsDataTable` typu `javax.faces.component.html.HtmlDataTable`.
16. Przejdź do edycji strony JSF. Dokonaj powiązania jej komponentów z komponentem backing bean dodając następujące atrybuty znaczników:
- w elemencie `<h:inputText>`:
`value="#{requestsList.newRequest}"`
 - w elemencie `<h:commandButton>`:
`action="#{requestsList.addRequest}"`
 - w elemencie `<h:dataTable>`:
`binding="#{requestsList.requestsDataTable}"`

17. Dodaj w klasie komponentu backing bean metodę, do której odnosi się przycisk na stronie JSF. Uzupełnij kod metody o operacje:

- utworzenia instancji klasy Request
- ustawienia w nowo utworzonym obiekcie daty żądania na bieżącą i tekstu żądania na wprowadzony do formularza
- utrwalenia obiektu poprzez repozytorium

```
public String addRequest()
{
    ...
    setNewRequest("");
    return null;
}
```

18. Zapisz wszystkie zmiany. Uruchom aplikację. Przetestuj czy działa dodawanie nowych żądań serwisowych.

19. Przy próbie dodania nowego żądania powinien pojawić się wyjątek informujący o tym, że operacje modyfikujące bazę danych powinny być realizowane w transakcji. W przeciwieństwie do metod beanów starszej technologii EJB, metody beanów CDI nie są domyślnie transakcyjne. Oznacz więc metodę dodającą nowe żądanie w klasie komponentu backing bean (nie w klasie repozytorium!) adnotacją @Transactional. Dodaj stosowny import.

20. Dodaj w klasie komponentu backing bean metodę do usuwania obiektu encji Request reprezentowanego przez bieżący wiersz w komponencie Data Table. Uzupełnij kod metody o operację usunięcia trwałej reprezentacji obiektu za pomocą fasady.

```
@Transactional
public String deleteRequest() {
    Request req =
        (Request) getRequestsDataTable().getRowData();
    ...
    return null;
}
```

21. Przejdź do edycji strony JSF. Dodaj w komponencie Data Table kolumnę z przyciskami umożliwiającymi usunięcie żądania wyświetlanego w bieżącym wierszu (jako ostatnią kolumnę tabeli). Użyj w odpowiedni sposób znacznika <h:commandButton> i jego własności value i action.

Uwaga: W rzeczywistej aplikacji kliknięcie przycisku powinno prowadzić do strony z prośbą o potwierdzenie decyzji o usunięciu żądania.

22. Zapisz wszystkie zmiany. Uruchom aplikację i przetestuj ją usuwając kilka żądań.

Ćwiczenie 2

Celem ćwiczenia jest pokazanie sposobu walidacji danych z wykorzystaniem technologii Bean Validation.

1. Oznacz pole w beanie CDI podpięte pod pole formularza do wprowadzania nowego żądania adnotacją `@Size` standardu Bean Validation. Ustaw minimalną długość tekstu na 3 znaki, a maksymalną na 60 znaków.
2. Na stronie JSF z formularzem dodaj komponent `<h:message>` (obok przycisku do dodawania wpisów) do wyświetlania komunikatu o błędzie walidacji dla pola do wprowadzenia nowego żądania.
3. Przetestuj dodawanie nowych żądań zwracając uwagę na standardowy komunikat o błędzie walidacji.
4. Zastąp standardowy komunikat komunikatem „Request text must be from 3 to 60 characters long.” wykorzystując odpowiedni atrybut adnotacji `@Size`.
5. Sprawdź czy nadal poprawnie działa usuwanie żądań.
6. Spraw aby możliwe było usuwanie żądań gdy pole z treścią nowego żądania jest puste.
7. Zawarcie komunikatu o błędzie w atrybucie adnotacji jest złą praktyką, m.in. z punktu widzenia możliwości internacjonalizacji aplikacji. W celu poprawy tego mankamentu wykonaj poniższe kroki:
 - a) Utwórz projekcie, w węźle `Other Sources / src/main/resources` plik `properties` korzystając z kreatora `Properties File` z kategorii `Other`. Nazwij plik `ValidationMessages.properties`.
 - b) Umieść w utworzonym pliku `properties` poniższy wpis klucz=wartość:

```
request.size=Request text must be from {min} to {max} characters long.
```
 - c) W adnotacji `@Size` w kodzie klasy komponentu backing bean zastąp treść komunikatem jego kodem w nawiasach klamrowych (`{request.size}`).
 - d) Jeśli masz czas, możesz przetestować internacjonalizację komunikatu o błędzie walidacji przygotowując drugi, zlokalizowany, plik komunikatów.