ЛАБОРАТОРНА РОБОТА 7

Тема: Технологія Enterprise JavaBeans (EJB)

Специфікація Enterprise JavaBeans (EJB) пережила три реалізації. Перша версія EJB представила індустрії інноваційні рішення, але обмеження функціональних можливостей призвели до нових вимог. У другій версії технологія EJB надала розробникам корпоративних застосунків підтримку віддалених взаємодій, механізми управління транзакціями, засоби забезпечення безпеки, обробки та зберігання інформації та веб-служби. Усі ці механізми є великоваговими, тому вимагають від розробників більше уваги приділяти особливостям взаємодій із самою інфраструктурою, а не реалізації бізнес-логіки. Наступна, 3 версія EJB стала простішою і легшою, введено використання анотацій, а компоненти EJB тепер можуть бути простими об'єктами Java (Plain Old Java Objects, POJO). EJB 3.2 був останньою версією перед переходом до Jakarta EE, де специфікації були перейменовані. Наступна специфікація отримала назву Jakarta Enterprise Beans, її версія 4.0 і вона стала частиною Jakarta EE 9. Jakarta Enterprise Beans 4.0 є, по суті, EJB 3.2 з основною зміною у просторі імен з javax.ejb на jakarta.ejb та технічними оновленнями:

- видалено методи java.security.Identity, оскільки цей клас був видалений у Java 14;
- видалено підтримку JAX-RPC, що є застарілим протоколом для веб-сервісів;
- видалено метод EJBContext.getEnvironment() як застарілий;
- підтримку розподіленої взаємодії (Distributed Interoperability, зокрема CORBA) було видалено для зменшення складності платформи;
- група API EJB 2.x тепер є опціональною, тому реалізації можуть не підтримувати її, якщо це не потрібно.

Тому Jakarta Enterprise Beans 4.0 вважається технічним оновленням задля підтримки екосистеми Jakarta EE.

Jakarta Enterprise Beans (або EJB) представляє колекцію фіксованих рішень типових проблем, що виникають при розробці серверних програм, а також перевірену часом схему реалізації серверних компонентів. Ці фіксовані рішення або служби надаються контейнером EJB. Для доступу до цих служб необхідно створити спеціалізовані компоненти, використовуючи декларативні та програмні EJB API, а також розгорнути їх у контейнері. Серверні компоненти EJB можна використовувати для побудови прикладних компонентів. Компонент EJB – це POJO з певними спеціальними можливостями, що залишаються невидимими, доки вони не стануть необхідні та не відволікають від головної ролі компонента.

Для того, щоб скористатися службами EJB, компонент повинен бути оголошений з типом, що розпізнається фреймворком EJB. EJB розпізнає два конкретні типи компонентів: сеансові компоненти (session beans) та компоненти, керовані повідомленнями (message-driven beans). Сеансові компоненти поділяються на сеансові компоненти без збереження стану (stateless session beans), сеансові компоненти із збереженням стану (stateful session beans) та компоненти-одинаки (синглетони – singletons). Компоненти кожного типу мають певне призначення, область видимості, стан, життєвий цикл та особливості використання на рівні прикладної логіки. Усі компоненти EJB є керованими. Керовані компоненти по суті є звичайними Java-об'єктами в середовищі Jakarta EE. Механізм управління контекстами та впровадження залежностей (Contexts and Dependency Injection, CDI) дозволяє здійснювати впровадження залежностей у будь-які керовані компоненти, включаючи компоненти EJB.

Практична частина

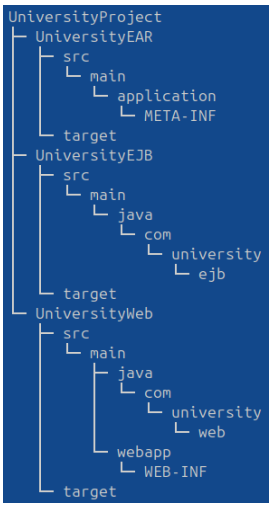
Загалом архітектура багаторівневого корпоративного застосунку складається з наступних рівнів:

- Веб-рівень, який містить логіку презентації програми та працює на сервері Jakarta EE. Веб-рівень часто представлено веб-модулем, містить JSF файли або сервлети, через які забезпечується доступ до бізнес-логіки в модулі EJB.
- Бізнес рівень - це програмний код, який працює на сервері Jakarta EE і містить бізнес-логіку застосунку. Бізнес-рівень забезпечується модулем EJB, код якого обробляє запити від клієнтів веб-рівня та керує транзакціями та тим, як об’єкти зберігаються в базі даних.
- Рівень КІС (корпоративних інформаційних систем) забезпечується серверами баз даних, системами планування ресурсів підприємства та представляє рівень постійного зберігання даних програми.

Розглянемо принципи створення багаторівневого корпоративного застосунку на демонстраційному прикладі University з використанням Maven та команд в консолі.

1. Виконаємо налаштування структури проєкту

Загальна структура проєкту буде мати наступний вигляд:



В обраній локації створюємо кореневий каталог проєкту UniversityProject:

```
mkdir UniversityProject
```

У створеному каталогу відкриваємо кореневий pom.xml, який виконуватиме роль агрегатору модулів, з наступним вмістом:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.university</groupId>
    <artifactId>UniversityProject</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>pom</packaging>
    <modules>
        <module>UniversityEJB</module>
        <module>UniversityWeb</module>
        <module>UniversityEAR</module>
    </modules>
     <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.12.1</version>
                <configuration>
```

```
                    <source>21</source>
                    <target>21</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.4.0</version>
            </plugin>
            <plugin>
                <groupId>org.wildfly.plugins</groupId>
                <artifactId>wildfly-maven-plugin</artifactId>
                <version>5.1.2.Final</version>
                <configuration>
                    <hostname>localhost</hostname>
                    <port>9990</port>
                    <username>admin</username>
                    <password>admin</password>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Для виконання операції публікації (deploy) передбачено використання wildfly-maven-plugin, у конфігурації якого вказується login відповідного користувача. У сервера WildFly є можливість додавати користувачів для виконання ролі deployer, тому слід замінити зазначені параметри username та password на ті, що відповідають одному із користувачів з правами deployer.

В цьому каталогу також створюємо підкаталоги проєкту:

```
mkdir UniversityEJB UniversityWeb UniversityEAR
```

2. Створюємо файли модуля EJB

У корені каталогу UniversityEJB створюємо файл pom.xml з наступним вмістом:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <parent>
        <groupId>com.university</groupId>
        <artifactId>UniversityProject</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>UniversityEJB</artifactId>
    <packaging>ejb</packaging>

    <dependencies>
        <dependency>
            <groupId>jakarta.platform</groupId>
            <artifactId>jakarta.jakartaee-api</artifactId>
            <version>10.0.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-ejb-plugin</artifactId>
                <version>3.2.1</version>
                <configuration>
                    <ejbVersion>4.0</ejbVersion>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

У цьому каталогу створюємо підкаталог

```
mkdir -p src/main/java/com/university/ejb
```

в якому відкриваємо файл EJB компоненти GreetingService.java з наступним вмістом:

```
package com.university.ejb;

import jakarta.ejb.Stateless;

@Stateless
public class GreetingService {
    public String greet(String name) {
        return "Вітаємо, " + name + ", в системі університету!";
    }
}
```

3. Створюємо файли веб модуля (WAR)

У корені каталогу UniversityWeb створюємо файл pom.xml з наступним вмістом:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <parent>
        <groupId>com.university</groupId>
        <artifactId>UniversityProject</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>UniversityWeb</artifactId>
    <packaging>war</packaging>

    <dependencies>
        <dependency>
            <groupId>com.university</groupId>
            <artifactId>UniversityEJB</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>jakarta.platform</groupId>
            <artifactId>jakarta.jakartaee-api</artifactId>
            <version>10.0.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.glassfish</groupId>
            <artifactId>jakarta.faces</artifactId>
            <version>4.0.1</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.4.0</version>
            </plugin>
        </plugins>
    </build>
</project>
```

У цьому каталогу створюємо підкаталог:

```
mkdir -p src/main/java/com/university/web
```

в якому відкриваємо Managed Bean файл GreetingController.java з наступним вмістом:

```
package com.university.web;

import com.university.ejb.GreetingService;
import jakarta.ejb.EJB;
import jakarta.enterprise.context.RequestScoped;
import jakarta.inject.Named;
import jakarta.annotation.PostConstruct;

@Named
@RequestScoped
public class GreetingController {
    @EJB
    private GreetingService greetingService;

    private String message;

    @PostConstruct // Анотація, що вимагає автоматичний виклик методу
    public void init() {
        message = greetingService.greet("студент");
    }

    public String getMessage() {
        return message;
    }
}
```

Також у каталогу `UniversityWeb` створюємо підкаталог:

```
mkdir -p src/main/webapp
```

в якому відкриваємо файл JSF сторінки greeting.xhtml з наступним вмістом:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="jakarta.faces.html">
    <h:head><title>Студентський портал</title></h:head>
    <h:body>
        <h1>#{greetingController.message}</h1>
    </h:body>
</html>
```

У цьому підкаталогу створюємо ще один підкаталог WEB-INF, в якому відкриваємо файл web.xml з наступним вмістом:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd" version="6.0">

    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>jakarta.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.xhtml</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>greeting.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Блок `<welcome-file-list></welcome-file-list>` є опціональним і дозволяє за замовчуванням запустити файл із наданого списку. У даному випадку це єдиний файл greeting.xhtml. У разі

відсутності такої конфігурації файл greeting.xhtml до адресного рядку браузеру слід буде додавати власноруч.

4. Створюємо EAR модуль

У корені каталогу UniversityEAR створюємо файл pom.xml з наступним вмістом:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <parent>
        <groupId>com.university</groupId>
        <artifactId>UniversityProject</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>UniversityEAR</artifactId>
    <packaging>ear</packaging>

    <dependencies>
        <dependency>
            <groupId>com.university</groupId>
            <artifactId>UniversityEJB</artifactId>
            <version>1.0-SNAPSHOT</version>
            <type>ejb</type>
        </dependency>
        <dependency>
            <groupId>com.university</groupId>
            <artifactId>UniversityWeb</artifactId>
            <version>1.0-SNAPSHOT</version>
            <type>war</type>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-ear-plugin</artifactId>
                <version>3.3.0</version>
                <configuration>
                    <version>8</version>
                    <modules>
                        <webModule>
                            <groupId>com.university</groupId>
                            <artifactId>UniversityWeb</artifactId>
                            <contextRoot>/university</contextRoot>
                        </webModule>
                        <ejbModule>
                            <groupId>com.university</groupId>
                            <artifactId>UniversityEJB</artifactId>
                        </ejbModule>
                    </modules>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Також у каталогу UniversityEAR створюємо підкаталог:

```
mkdir -p src/main/application/META-INF
```

в якому відкриваємо файл application.xml з наступним вмістом:

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="https://jakarta.ee/xml/ns/jakartaee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/application_10.xsd" version="10">
  <display-name>UniversityEAR</display-name>
```

```
  <module>
    <web>
      <web-uri>com.university-UniversityWeb-1.0-SNAPSHOT.war</web-uri>
      <context-root>/university</context-root>
    </web>
  </module>
  <module>
    <ejb>com.university-UniversityEJB-1.0-SNAPSHOT.jar</ejb>
  </module>
</application>
```

Після виконання збірки проєкту слід впевнитись, що war та jar файли дійсно мають вказані імена і упаковані в ear файл проєкту. Якщо імена відрізняються від вказаних, то слід внести відповідне корегування до файлу application.xml.

5. Збірка та публікація проєкту

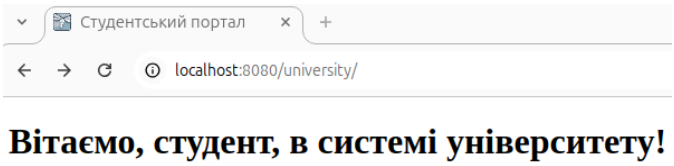
Збірку проєкту виконуємо за допомогою наступної команди в консолі:

```
mvn clean install
```

Цільовий ear файл буде знаходитись в каталогу UniversityEAR/target. Для його публікації на сервері WildFly необхідно його запустити (у режимі standalone) та виконати команду в локації центрального pom.xml проєкту:

mvn wildfly:deploy

Результат роботи проєкту отримаємо за допомогою браузеру:



Завдання

1. Вивчить та реалізуйте наведений приклад. Перевірте його роботу.
2. Ґрунтуючись на наведеному в практичній частині прикладі реалізуйте застосунок, опис якого надано за посиланням https://netbeans.apache.org/tutorial/main/kb/docs/javaee/javaee-entapp-ejb/. Зверніть увагу, що реалізацію застосунку показано для старих версій як платформи Java EE, сервера GlassFish, так і IDE NetBeans. Тому необхідно виконати його адаптацію до сучасної версії Jakarta EE та одного із серверів застосунків. Використання та тип IDE є опціональними.
3. Наведіть структуру проєкту та схему взаємодії компонент в ньому.
4. Розширте можливості прикладу: додайте список певних розділів новин (наприклад, «політика», «розваги», «наука», тощо); забезпечте показ новин за розділами.
5. Підготуйте звіт.