Práctica 7: Capa de Negocio y Persistencia de Datos

Objetivo

En la anterior práctica nos familiarizamos con los conceptos y tecnologías del nivel web de Java EE, en esta práctica emplearemos tecnologías asociadas al nivel de negocio. En concreto migraremos la parte de lógica de negocio a EJBs y la persistencia en fichero a JPA.

Proyecto

En esta práctica ya es necesario un servidor Java EE completo, por lo que emplearemos JBoss en lugar del servidor Tomcat que hemos usado hasta ahora.

Creamos un "Dynamic Web Project" y como "Target Runtime" seleccionamos el servidor JBoss, creándolo previamente si no lo habíamos hecho ya. Una vez creado el proyecto vamos a "Project Facets" en las propiedades del mismo y habilitamos el soporte de JSF. Al usar JBoss, como librería para JSF no es necesario descargar nada si no que podemos seleccionar la proporcionada por el target runtime.

El soporte para JAX-RS no lo habilitamos ya que ahora usaremos EJB en su lugar.

Finalmente copiamos los ficheros java y xhtml de la práctica anterior.

EJB

En la práctica anterior la lógica de negocio estaba como servicios REST. Ahora vamos a migrarla a un EJB de tipo Singleton, en concreto los métodos para obtener el listado actual de productos y para dar de alta un nuevo producto.

Creamos el EJB haciendo click en "New/Other..." y seleccionando "EJB/Session Bean (EJB 3.X)". Damos un nombre al bean, indicamos el paquete en el que se creará, el tipo de bean y el tipo de acceso al mismo (ej. No-interface View).

A continuación migramos la lógica de negocio a este EJB:

1. Eliminamos las clases con los servicios REST moviendo los métodos correspondientes al EJB. Eliminamos también la parte del dendograma para simplificar la práctica.
2. Actualizamos los managed beans para que ahora utilicen el EJB en lugar del cliente REST.
3. Probad que la la aplicación web funcione correctamente.

Práctica 7: Capa de Negocio y Persistencia de Datos

Base de Datos

A continuación vamos a migrar el listado de productos de la tienda a una base de datos que se irá actualizando.

En primer lugar creamos la base de datos desde el cliente de MySQL:

1. Creamos una base de datos específica para la práctica (CREATE DATABASE).
2. Damos de alta un nuevo usuario con control total sobre esta base de datos (GRANT ALL PRIVILEGES ON).

A continuación damos de alta la base de datos en JBoss para que la puedan localizar las aplicaciones mediante JNDI:

1. Instalamos el conector de MySQL:
   a) Dentro del directorio de JBoss creamos el directorio modules/com/mysql/main.
   b) Copiamos en él el conector MySQL desde /usr/share/java/mysql-connector-java.jar
   c) Creamos en él un fichero module.xml con el siguiente contenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
    <resources>
        <resource-root path="mysql-connector-java.jar"/>
    </resources>
    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
        <module name="javax.servlet.api" optional="true"/>
    </dependencies>
</module>
```

2. Configuramos la conexión con la base de datos. Para ello en el fichero standalone.xml insertamos lo siguiente:

```
<datasource jndi-name="java:jboss/datasources/NombreDeTuBD" pool-name="NombreDeTuBD">
        <connection-url>jdbc:mysql://localhost:3306/NombreDeTuBD</connection-url>
        <driver>com.mysql</driver>
        <security>
```

Práctica 7: Capa de Negocio y Persistencia de Datos

```
                <user-name>usuario</user-name>
                <password>contraseña</password>
        </security>
</datasource>
<drivers>
        <driver name="com.mysql" module="com.mysql">
                <xa-datasource-class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-datasource-class>
        </driver>
</drivers>
```

De momento no creamos tablas ni insertamos productos ya que eso lo hará JPA automáticamente por nosotros, pero podemos ir siguiendo los cambios mediante SELECTs según vayamos operando con nuestra aplicación web (al iniciarla, al dar de alta un nuevo producto, etc.).

JPA

Ahora que ya está configurada en JBoss la conexión con la base de datos, actualizamos nuestro código Java EE para que utilice JPA.

En primer lugar debemos añadir soporte de JPA al proyecto de Eclipse:

1. Vamos a propiedades del proyecto y en “Project Facets” habilitamos JPA.
2. En la configuración disponible indicaremos que la plataforma es Hibernate y que la implementación de JPA la proporciona el target runtime.
3. En esta práctica no le indicamos ninguna conexión a la base de datos a Eclipse.

A continuación creamos una entidad que persistirá en la base de datos:

1. Eliminamos las anotaciones JAXB de nuestro bean producto, ya que las vamos a reemplazar por anotaciones de JPA. Además el bean con el listado de productos ya no es necesario y se puede eliminar.
2. Anotaremos este bean como entidad e indicaremos que la referencia es la clave primaria y que el resto de campos no pueden estar a NULL.
3. Creamos un @NamedQuery que devuelva el listado de todos los productos de la tienda.

Práctica 7: Capa de Negocio y Persistencia de Datos

Actualizamos el EJB para que lea la información de la base de datos:

1. Obtendrá una referencia de tipo EntityManager mediante @PersistenceContext.
2. Actualizamos el método que devuelve el listado de productos para que emplee el NamedQuery anterior en lugar de leer del fichero XML.
3. Actualizamos el método que añade un nuevo producto para que lo incluya en el contexto de persistencia en lugar de escribir en el fichero XML.

Finalmente editamos el fichero persistence.xml para referenciar la base de datos que hemos creado en el apartado anterior:

```
<jta-data-source>java:jboss/datasources/NombreDeTuBD</jta-data-source>
<properties>
        <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
        <property name="hibernate.hbm2ddl.auto" value="update"/>
</properties>
```

Ya podemos trabajar con la aplicación y ver cómo se van actualizando los datos en la base de datos.