|  |
| --- |
| **Nombre Integrante 1: Walter Javier Alonso Roa.**  **Nombre Integrante 2: Juan Sebastian Paz Prieto.**  **Nombre Integrante 3:** |

Con respecto a la inspección del código fuente de ejemplo, responda las siguientes preguntas:

1. Ubique la plantilla *vendedores.xhtml*, revise las invocaciones que se realizan al managed bean *VendedorBean*, ubique la acción para agregar un vendedor ¿Cómo se realiza esta invocación?

Los beans son conectados a los componentes Usando sintaxis JSF EL expressions, en este caso usando la expresion “#{vendedorBean.agregarVendedor}” la evalua y busca en el managed bean VendedorBean el método “agregarVendedor()”.

Dado el modelo MVC, es el controlador en JSF “Faces Servlet” quien siguiendo el ciclo de vida de JSF, en la actualización de los valores del modelo los transfiere al managed bean que esta referenciado.

1. En la acción borrar un vendedor en la plantilla *vendedores.xhtml*, ¿Cómo se pasa el parámetro para identificar el vendedor que debe ser eliminado?

Dentro de la librería de JSF se dispone del tag “<f:param >”, el cual envía su valor por parámetros de request, estos valores son tomados por medio del FacesContext el cual permite acceder desde el bean al entorno de JSF y en general al entorno de ejecución en el que la clase se está ejecutando, este es un puente al que también le permite acceder al contexto HTTP, de esta forma obtiene el parámetro que se envió desde la vista con el tag “<f:param”, tomado por el controlador para que en el bean sea encontrado.

En este caso, el código usado fue:

1. En la vista “vendedores.xhtml” se colocó dentro del componente “<h:commandButton id="DDRtn" action="#{vendedorBean.eliminarVendedor}" value=’Eliminar’>” el cual es el botón para la acción de borrar el vendedor, el parámetro: “<f:param name="vendedorId" value="#{employee.identificacion}"/>”, esto indica que del modelo toma el número de identificación del vendedor y lo coloca como valor del parámetro “vendedorId”.
2. En el bean administrado “VendedorBean”, en el método “eliminarVendedor” se obtiene el valor del parámetro dado por medio de FacesContext:

FacesContext context = FacesContext.getCurrentInstance();

Map map = context.getExternalContext().getRequestParameterMap();

long vendedorId=Long.parseLong((String)map.get("vendedorId"));

1. Revise las interfaces *IServicioVendedoresMockLocal* e *IServicioVendedoresMockRemote* ¿Qué tipo de interfaces son?

EJB 3, dentro de sus características define 2 tipos de interfaces, estas permiten al cliente (componente web u otro EJB) realizar la invocación al session bean dependiendo de:

* Remotas: Puede ejecutarse en una máquina diferente y una JVM diferente al enterprise bean que accede.
* Locales: En este caso el cliente reside en el mismo lugar que la instancia del session bean, es decir está ejecutándose en la misma JVM en la cual se ejecutan los EJB a los que accede.

Para el caso del ejercicio cada interfaz corresponde a cada tipo nombrado anteriormente:

* *IServicioVendedoresMockLocal:* Es la interfaz local, la cual es definida por la anotación @Local.
* *IServicioVendedoresMockRemote:* Es la interfaz remota, la cual es definida por la anotación @Remote.

1. ¿Si usted cambia la anotación *@Stateful* a *@Stateless* del session bean *ServicioVendedoresMock* qué consecuencias habrían la aplicación? ¿Qué consecuencias genera la anotación *@Singleton*?

EJB 3, maneja 3 tipos de session beans, cada uno da características diferentes para el estado conversacional, estos 3 son:

* **Stateful**: Mantiene el estado durante la sesión del cliente con el bean, es decir reserva el bean para un cliente, este dejara de existir cuando el cliente remueva el bean o finalice la sesión. Cada vez que invoque métodos, se le asignara la misma instancia con la que comenzó.
* **Stateless**: No mantiene un estado conversacional con el cliente, cada vez que invoca un método, la intancia del bean que atiende la solicitud puede ser otra distinta con la que comenzó.
* **Singleton**: Es un bean único, instanciado una vez por la aplicación, no mantiene el estado conversacional, asegura que todo cliente que la consulte tenga la misma instancia.

Para el caso del ejercicio:

1. ¿Qué sucede si se cambia de *@Stateful* a *@Stateless* el session bean *ServicioVendedoresMock*?

La clase *ServicioVendedoresMock* no mantiene estado para los clientes que la usan, cada uno de los métodos que ofrece esta clase pueden ser consultados por varios clientes, dado que son sin estado.

Sin embargo al cambiarlo de estado @Stateful a @Stateless se mejora el rendimiento dado que se está tomando del pool una instancia se esa clase, a diferencia de stateful que estaría creando instancias por cada cliente.

1. ¿Qué consecuencias genera la anotación *@Singleton*?

Si se cambia la anotación, se tendría una única instancia en toda la aplicación, con ello se estarían generando accesos concurrentes a métodos CRUD, los cuales sin controles pueden generar duplicidad y mal manejo de la información, por ejemplo, si un cliente elimina un vendedor, y otro cliente elimina otro vendedor

Son diseñados para casos en los que una misma instancia de un bean es

compartida

y

accedida concurrentemente

por los clientes de la aplicación

Seria una instancia única para todos los clientes, esto conllevaría a que todo cliente tomara esta instancia y abrían problemas de sincronización de acceso a la clase.

1. Revise las pruebas JUnit implementadas. En el proyecto Web, en el test del servicio de seguridad (*LoginBeanTest*), la inyección de la dependencia se hace por medio de la instanciación del *mock object* que la implementa. Por otro lado, en el proyecto EJB, la inyección de la dependencia en el test de *VendorServices* (*ServicioVendedoresMockTest*) se hace por medio de JNDI. ¿Cuál es la diferencia? Ejecute ambas pruebas dos veces, una con la aplicación desplegada en el servidor y otra sin dicho despliegue. ¿Qué puede concluir de dichas ejecuciones? Sea claro y concluyente.

|  |  |  |
| --- | --- | --- |
|  | **WEB - *LoginBeanTest*** | ***EJB - ServicioVendedoresMockTest*** |
| **Aplicación desplegada** | Paso | Paso |
| **Aplicacion no desplegada** | Paso | Fallo |

LoginTest la cual está en el proyecto web, al ejecutarse realiza la instanciación directa del bean LoginBean, con los cual tiene acceso a sus métodos, ejecutando así la prueba así este o no desplegada en otra instancia.

*ServicioVendedoresMockTest* la cual está en el proyecto EJB, busca la clase que tenga la implementación de la interfaz “IServicioVendedoresMockRemote”, esto lo hace buscándola dentro de la maquina “localhost” utilizando el protocolo RMI-IIOP por medio de CORBA examina el puerto 3700, encontrando en el contenedor EJB desplegado la instancia de la clase ServicioVendedoresMock la cual implementa la interfaz remota IServicioVendedoresMockRemote, toma esta clase y ejecuta efectivamente los test.