|  |
| --- |
| **Nombre Integrante 1: Daniel Melgarejo**  **Nombre Integrante 2: Miguel Quintero**  **Nombre Integrante 3: Ian Sarasty Medina** |

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?

El managed bean hace una inyección del EJB IServicioVendedoresMockLocal por medio de la anotación @EJB. Al objeto que inyecta la implementación del EJB lo llama servicio y es por el cual invoca los servicios de negocio para poder administrar las operaciones de un vendedor.

El método agregarVendedor() invoca la implementación del servicio (que se hizo por medio de inyección de EJB) y utiliza la operación agregarVendedor para crear el vendedor al que se le han asignado atributos por medio de la vista xhtml. Después de agregar el vendedor por medio del servicio se inicializa una nueva instancia, tanto del vendedor como de la experiencia, que se están mostrando en la vista xhtml.

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?

En la definición del botón de eliminación dentro de la tabla en la plantilla xhtml, se utiliza el tag <f:param> del api de JSF: xmlns:f=<http://java.sun.com/jsf/core>. Por medio de este, utilizando la propiedad “value”, se aloja el valor del objeto seleccionado en la tabla (en este caso el id del vendedor seleccionado en la tabla) como parámetro dentro del contexto de JSF. Posteriormente en el método eliminarVendedor() (del managed Bean) se busca el valor del parámetro que se alojó en el contexto de JSF de la siguiente manera:

FacesContext context = FacesContext.getCurrentInstance();

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

Se busca el valor del parámetro con el identificador que se asignó en la plantilla xhtml con el tag <f:param>, por medio de la propiedad “name”:

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

En este caso, se asigna el valor del parámetro buscado a una variable especifica por medio de una operación de casteo.

De esta forma se obtiene el id del vendedor seleccionado en la vista y así poder buscarlo para eliminarlo del modelo.

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

*IServicioVendedoresMockLocal* es una interface local, es decir que el contenedor proveerá acceso a este únicamente desde el módulo de despliegue que lo contiene.

*IServicioVendedoresMockRemote* es una interface remota, es decir que el contenedor proveerá acceso no solo desde el modulo que lo contiene, sino que también desde otros módulos dentro del contenedor por JNDI y también un acceso desde otros contenedores por medio de RMI.

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

Agregar la etiqueta @Singleton ocasionaría que el vean solo tuviera una instancia que será accedida por todas las peticiones que se abrieran en el contenedor.

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.

Prueba sin despliegue:

*ServicioVendedoresMockTest:* Aparece un error de conexión de CORBA, no pasa ningún test 0%

*LoginBeanTest:* Aparece un error de Assertion, pasa solo un test 50%