



MAKE THE  
FUTURE  
JAVA



September 22 – 26, 2013  
[Oracle.com/JavaOne](http://Oracle.com/JavaOne)

## Java EE 7 Hands-on Lab

Arun Gupta (Oracle)  
Java EE & Glassfish Guy, @arungupta

Antonio Goncalves (Independent)  
Java EE EG Member, Consultant, Java Champion, @agoncal

David Delabassee (Oracle)  
GlassFish Product Manager, @delabassee

Marian Muller (Serli)  
TBD, @mullermarian

Traducido al español por [CLOJUG](#)

ORACLE®

## Contenido

1.0 Introducción .....	3
1.1 Requerimientos de software .....	3
2.0 Enunciado del problema .....	4
2.1 Guía del laboratorio .....	6
2.2 Tiempo estimado .....	8
3.0 Recorrido por la Aplicación de ejemplo.....	8
4.0 Sala de chat (API de Java para WebSocket) .....	15
5.0 Venta de Tickets (Aplicaciones Batch para la plataforma Java) .....	24
6.0 Ver y borrar una película (API de Java para RESTful Web Services) .....	35
7.0 Adicionar Película (Java API para procesamiento JSON) .....	42
8.0 Puntos de Película (Servicio de Mensajería de Java - JMS) .....	52
9.0 Reservas (Java Server Faces) .....	60
10.0 Conclusión .....	71
11.0 Solución de problemas .....	72
12.0 Agradecimientos .....	73
13.0 Soluciones completas .....	73
13.1 TODO .....	73
13.2 Historial de revisión .....	74
Apéndice .....	74
Apéndice A: Configurar Glassfish 4 en NetBeans IDE .....	74

## 1.0 Introducción

La plataforma Java EE 7 continúa empujando el desarrollo fácil de aplicaciones por lo cual se ha caracterizado en previas versiones trayendo más simplificación al desarrollo empresarial. La plataforma adiciona nuevas e importantes APIs tales como la API Cliente para REST en JAX-RS 2.0 y la tan esperada API para los procesos en Batch. El servicio de mensajería de Java (JMS) 2.0 ha sufrido un cambio extremo para alinearse con las mejoras del lenguaje Java. Igualmente, hay una gran cantidad de mejoras a muchos otros componentes. Los nuevos estándar web como HTML 5, WebSocket, y JSON son acogidos para construir aplicaciones web modernas.

En este taller se construirá una aplicación típica de 3 capas de principio a fin usando las siguientes tecnologías de Java EE 7:

- API de Java para la persistencia - JPA 2.1 (JSR 338)
- API de Java para RESTful Web Services – JAX-RS 2.0 (JSR 339)
- Servicio de mensajería de Java – JMS 2.0 (JSR 343)
- Java Server Faces 2.2 (JSR 344)
- Contextos e inyección de dependencia – CDI 1.1 (JSR 346)
- Validaciones para Beans 1.1 (JSR 349)
- Aplicaciones Batch para la plataforma Java 1.0 (JSR 352)
- API de Java para procesamiento JSON 1.0 (JSR 353)
- API de Java para WebSocket 1.0 (JSR 356)
- API de Java para el manejo de transacciones - JTA 1.2 (JSR 907)

Juntas, estas APIs le permitirán ser más productivo ya que simplifican el desarrollo empresarial. La última versión de este documento puede ser descargada de <http://glassfish.org/hol/javaee7-hol.pdf>.

## 1.1 Requerimientos de software

El siguiente software debe ser descargado e instalado:

- JDK 7:  
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- NetBeans 7.4 o superior, versión “todo” o “Java EE”:  
<http://netbeans.org/downloads/>

Una vista previa de la página de descarga se muestra en la siguiente imagen y se resalta exactamente el botón “Download” donde se debe hacer clic.

NetBeans IDE Download Bundles					
Supported technologies *	Java SE	Java EE	C/C++	PHP	All
④ NetBeans Platform SDK	•	•			•
④ Java SE	•	•			•
④ Java FX	•	•			•
④ Java EE		•			•
④ Java ME					—
④ HTML5		•		•	•
④ Java Card™ 3 Connected					—
④ C/C++			•		•
④ Groovy					•
④ PHP				•	•
Bundled servers					
④ GlassFish Server Open Source Edition 3.1.2.2		•			•
④ Apache Tomcat 7.0.34					•
	Download	Download	Download	Download	Download
	Free, 75 MB	Free, 171 MB	Free, 49 MB	Free, 49 MB	Free, 184 MB

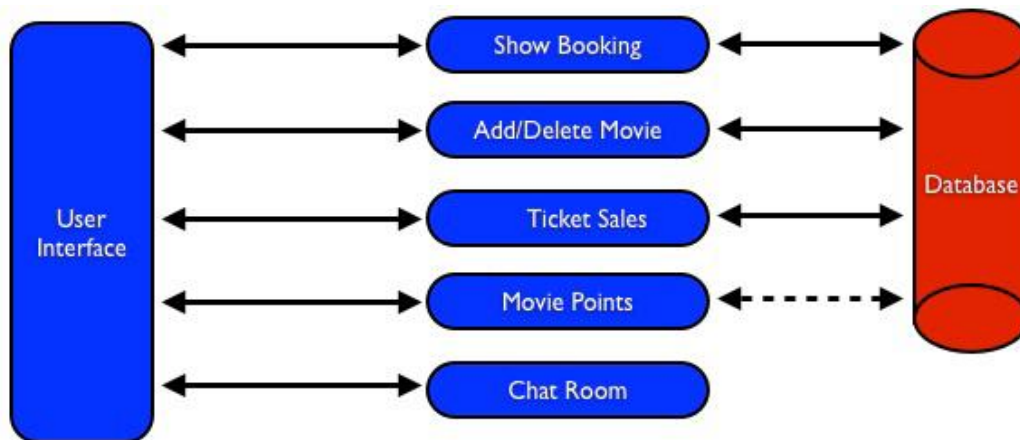
**Figura 1: NetBeans Download Bundles**

- GlassFish 4 viene pre-empaquetado junto con NetBeans 7.4+ y no es necesario descargarlo explícitamente. Sin embargo, si desea puede descargarlo desde [glassfish.org](http://glassfish.org)

Configure GlassFish 4 en NetBeans IDE siguiendo las instrucciones en el [Apéndice A](#).

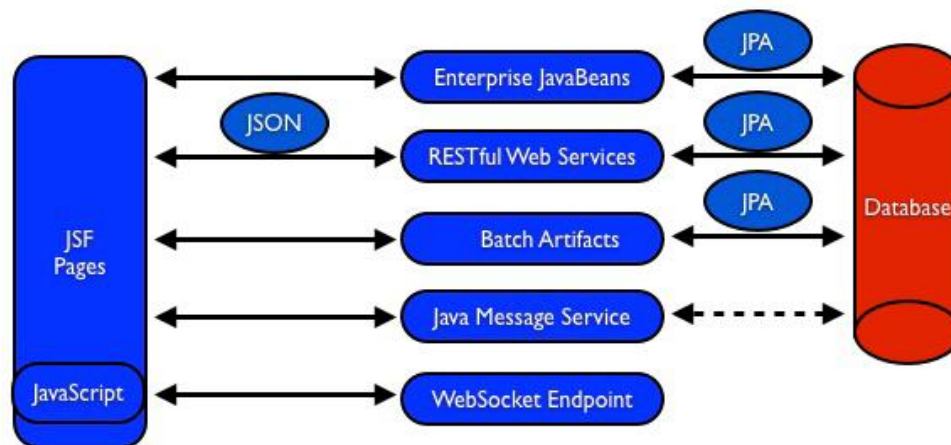
## 2.0 Enunciado del problema

Este taller construye una aplicación web Java EE 7 típica de 3 capas que permite a los usuarios ver los horarios de presentación y hacer reservaciones para una película en un multicine. Los usuarios pueden adicionar nuevas películas y borrar películas existentes. También pueden discutir acerca de la película en una sala de chat. Las ventas totales por cada presentación son calculadas al final del día. Los usuarios también pueden acumular puntos por ver películas.



La anterior imagen muestra los componentes clave de la aplicación. La interface de usuario inicia todos los flujos en la aplicación. Reservar película, adicionar/borrar película y Vender boletos interactúan con la base de datos; Puntos por película podría interactuar con la base de datos, sin embargo, esto está fuera del alcance de esta aplicación; por último, la sala de chat no interactúa con la base de datos.

Como se menciona más arriba, las diferentes funciones de la aplicación utilizan varias tecnologías Java y estándares web en su implementación. La siguiente imagen muestra como las diferentes tecnologías Java son usadas en los diferentes flujos de la aplicación



La siguiente tabla detalla los componentes de la aplicación y las tecnologías a ser usadas en su implementación.

Flujo	Descripción
Interfaz de usuario	Escrita enteramente en Java Server Faces (JSF).
Sala de Chat	Utiliza JavaScript del lado del cliente y JSON para comunicarse con un endpoint WebSocket.
Venta de tickets	Usa Aplicaciones en Batch para la plataforma Java con el fin de calcular el total de ventas y almacenar esta información en la base de datos.
Adicionar/Borrar Película	Implementado usando RESTful Web Services. JSON es usado como el formato de datos en que se comunican las partes.
Puntos por película	Usa el servicio de mensajería de Java (JMS) para actualizar y obtener puntos de premio por lealtad; una implementación opcional usando tecnología de base de datos puede ser llevada a cabo.
Reservar Película	Usa Enterprise JavaBeans ligeros para comunicarse con la base de datos usando el API de JPA

Este no es un tutorial comprensivo de Java EE. Se espera que los asistentes conozcan los conceptos básicos de Java tales como EJB, JPA, JAX-RS y CDI. El tutorial de Java EE 7 es un buen lugar para aprender todos estos conceptos. Sin embargo suficiente explicación es proveída en esta guía para que usted empiece con la aplicación.

**Advertencia:** Esta es una aplicación de ejemplo y el código puede no seguir las mejores prácticas para prevenir ataques por inyección de SQL, ataques de secuencias de comandos en sitios cruzados, escape de parámetros y otras características similares que son esperadas de una aplicación empresarial robusta. Esto es intencional de tal forma que puedan enfocarse en la explicación de la tecnología. Es altamente recomendable que el código copiado de esta aplicación ejemplo sea actualizado para cumplir con esos requerimientos.

## 2.1 Guía del laboratorio

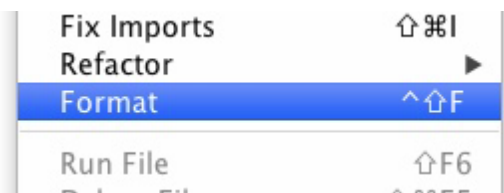
Los asistentes empezarán con una aplicación Maven existente. Al seguir las instrucciones y guías de este taller, ellos:

- Leerán el código fuente existente para ganar un entendimiento de la estructura de la aplicación y del uso de las tecnologías seleccionadas.
- Adicionarán código nuevo y actualizarán código existente con fragmentos proveídos, con el fin de demostrar el uso de diferentes tecnologías de la plataforma Java EE 7.

A continuación algunos consejos en cuanto a copiar/pegar código de este documento en NetBeans, que harán su experiencia más agradable:

- NetBeans provee la funcionalidad de formatear código fuente de una manera ordenada y siguiendo convenciones. Esto se puede hacer para cualquier tipo de código fuente ya sea XML, Java o cualquier otro. Es altamente recomendado hacer uso de esta funcionalidad una vez se copie/pegue código de este documento al editor de NetBeans para mantener el código legible.

Para ello, haga clic derecho sobre el editor de NetBeans y seleccione la opción “Format” del menú desplegable:



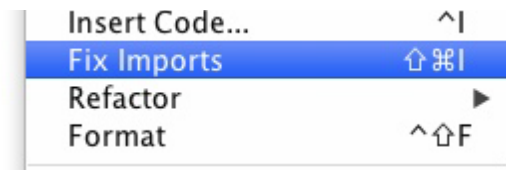
La misma funcionalidad se puede obtener por medio de la siguiente combinación de teclas:

Combinación	Sistema Operativo
Command + Shift + F	Mac
Ctrl + Shift + F	Windows
Alt + Shift + F	Linux

- Copiar/pegar el código desde este documento no hace la importación automática de paquetes/clases requeridos. Se requiere que este proceso se realice manualmente para que sus clases compilen. Esto se puede solucionar haciendo clic sobre el bombillo amarillo en la barra izquierda:



También puede solucionar todas las importaciones haciendo clic derecho sobre el editor de NetBeans y seleccionando la opción “Fix imports” del menú desplegable:



La misma funcionalidad se puede obtener por medio de la siguiente combinación de teclas:

Combinación	Sistema Operativo
Command + Shift + F	Mac
Ctrl + Shift + F	Windows
Alt + Shift + F	Linux

Las clases sugeridas por defecto son generalmente las requeridas. En caso de que una clase esté disponible en diferentes paquetes, se mostrarán todas las opciones. Si durante el desarrollo de este taller se tiene el caso de

múltiples opciones para importar, se especificará exactamente cual es la requerida.

## 2.2 Tiempo estimado

Seguir las instrucciones de esta guía puede tomar entre tres y cinco horas. La mayor parte de este tiempo se dedica al aprendizaje de las nuevas características, conocimiento de NetBeans, copiando/pegando código y depurando errores. El camino recomendado es seguir todas las secciones en orden. Alternamente, puede llevar a cabo las secciones 4.0 a 9.0 en el orden deseado, teniendo en cuenta sus preferencias por cada tecnología. Sin embargo, la sección 6.0 es prerrequisito para la sección 7.0.

La siguiente tabla muestra el tiempo estimado para cada sección:

Sección	Tiempo estimado
3.0 Recorrido por la aplicación de ejemplo	15 – 30 minutos
4.0 Sala de chat (API de Java para WebSocket)	30 – 45 minutos
5.0 Venta de tickets (Aplicaciones Batch para la plataforma Java)	30 – 45 minutos
6.0 Ver y borrar película (API de Java para RESTful Web Services)	30 – 45 minutos
7.0 Adicionar película (API de Java API para procesamiento JSON)	30 – 45 minutos
8.0 Puntos por película (Servicio de Mensajería de Java)	30 – 45 minutos
9.0 Reservas (Java Server Faces)	30 – 45 minutos

El tiempo mostrado en la anterior tabla, es solo estimado. Estas secciones han sido completadas en mucho menos tiempo, ¡tú también puedes hacerlo!

El tiempo mostrado también permite crear una versión personalizada de este taller que se ajuste a la audiencia y tiempo disponible.

## 3.0 Recorrido por la Aplicación de ejemplo

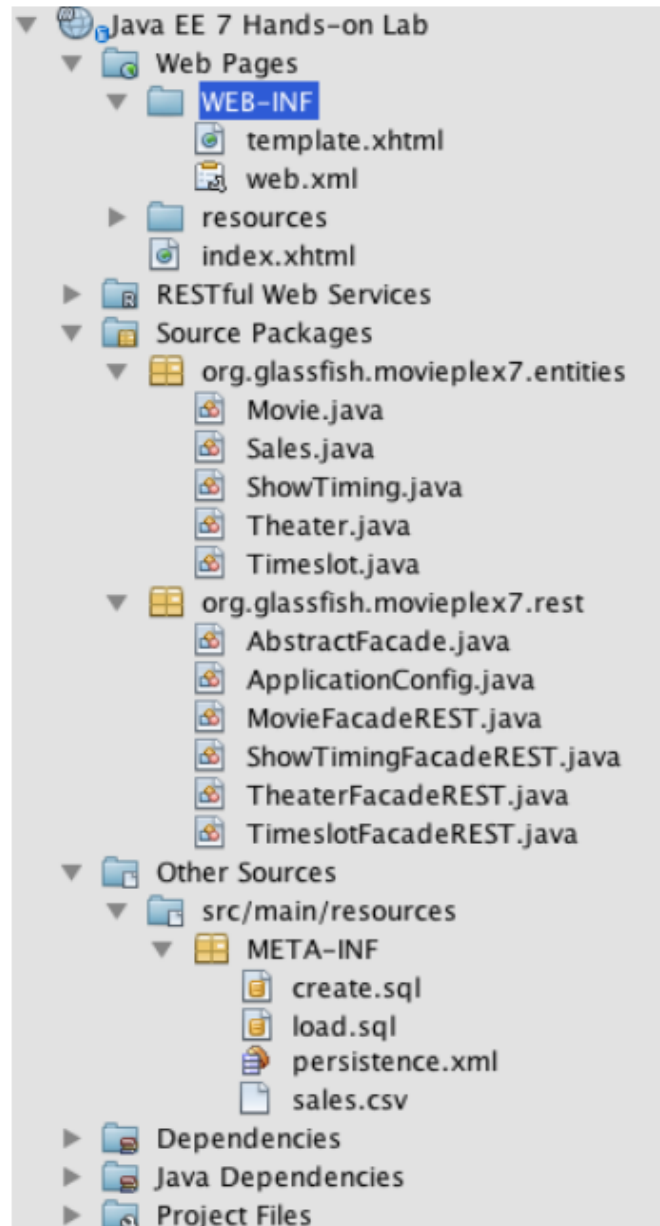
**Propósito:** En esta sección se descargará la aplicación de ejemplo a ser usada en este taller. Un recorrido por la aplicación será llevado a cabo para proveer un entendimiento de la arquitectura de la aplicación.

**Tiempo estimado:** 15 – 30 minutos.



**3.1** Descargue la aplicación de ejemplo desde <http://glassfish.org/hol/movieplex7-starting-template.zip> y descomprímala. Esto creará un directorio "movieplex7" y descomprimirá todo el contenido allí.

**3.2** En NetBeans IDE, Seleccione "File", "Open Project...", seleccione el directorio descomprimido y haga clic en "Open Project". La estructura del proyecto es mostrada a continuación:



**3.3 Coordinar Maven:** Expanda "Project Files" y haga doble clic en el archivo "pom.xml". En este archivo la API de Java EE 7 es especificada como una dependencia:

```
<dependency>:
  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaeeapi</artifactId>
      <version>7.0</version>
    </dependency>
  </dependencies>
```

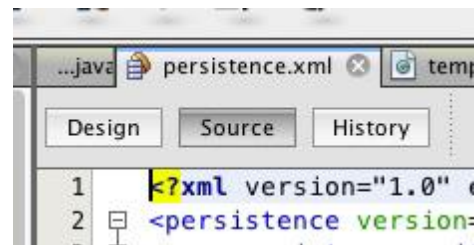
Esto asegura que las APIs de Java EE 7 son recuperadas del repositorio central de Maven.

La plataforma Java EE 6 introdujo el concepto de "perfiles". Un perfil es una configuración de la plataforma Java EE destinada para una clase específica de aplicaciones. Todos los perfiles de Java EE comparten un conjunto común de características, tales como nombrado e inyección de recursos, reglas de empaquetado, requerimientos de seguridad, etc. Un perfil puede contener un subconjunto individual o un súper conjunto de tecnologías contenidas en la plataforma.

El perfil Web de Java EE es un perfil de la plataforma Java EE específicamente orientado a aplicaciones web modernas. El conjunto completo de especificaciones definidas en el perfil Web está definido en la especificación del perfil Web de Java EE 7. GlassFish puede ser descargado en 2 diferentes sabores - plataforma completa o perfil Web.

Este taller requiere que se descargue la plataforma completa. Todas las tecnologías usadas en este laboratorio excepto el servicio de mensajería de Java (JMS) y aplicaciones Batch para la plataforma Java, pueden ser desplegados en el perfil Web.

**3.4 Fuente de datos por defecto:** Expanda "Other sources", "src/main/resources", "META-INF", y haga doble clic en "persistence.xml". Por defecto, NetBeans abre el archivo en la vista de diseño. Haga clic en la etiqueta fuente (source) para ver el código XML. Se verá algo como lo que sigue a continuación:



```

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
<persistence-unit name="movieplex7PU" transaction-type="JTA">
    <!--
    <jta-data-source>java:comp/DefaultDataSource</jta-data-source>
    -->
    <properties>
    <property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>
    <property name="javax.persistence.schema-generation.create-source" value="script"/>
    <property name="javax.persistence.schema-generation.drop-source" value="script"/>
    <property name="javax.persistence.schema-generation.drop-script-source" value="META-INF/drop.sql"/>
    <property name="javax.persistence.sql-load-script-source" value="META-INF/load.sql"/>
    <property name="eclipselink.deploy-on-startup" value="true"/>
    <property name="eclipselink.logging.exceptions" value="false"/>
    </properties>
</persistence-unit>
</persistence>

```

Note que la etiqueta `<jta-data-source>` esta entre comentarios, por lo tanto no se ha especificado ningún elemento de fuente de datos. Este elemento identifica el recurso JDBC que será usado para conectarse a la base de datos por el entorno de ejecución del servidor de aplicaciones subyacente.

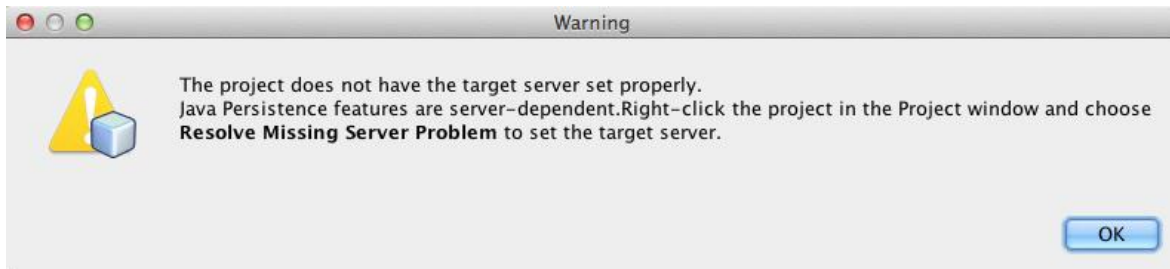
La plataforma Java EE 7 define una nueva fuente de datos por defecto que debe ser proveída en tiempo de ejecución. Esta fuente de datos pre-configurada se puede acceder bajo el nombre JNDI:

```
java:comp/DefaultDataSource
```

La especificación JPA 2.1 dice que si ni el elemento `jta-data-source` ni tampoco el elemento `non-jta-data-source` son especificados, entonces quién realice el despliegue de la aplicación debe especificar una fuente de datos JTA o la fuente de datos JTA por defecto será proveída por el contenedor.

Para GlassFish 4, la fuente de datos por defecto está asociada al recurso JDBC `jdbc/___default`.

Si se hace clic entre las etiquetas "Diseño" y "Fuente" puede que se muestre el error siguiente:



Esto se resolverá cuando corramos la aplicación. Haga clic en "OK" para cerrar el dialogo.

**3.5 Generación de Esquema:** JPA 2.1 define un nuevo conjunto de propiedades `javax.persistence.schema-generation.*` que pueden ser usadas para generar artefactos de bases de datos como tablas, índices, y restricciones. Esto ayuda en el prototipo de su aplicación cuando los artefactos requeridos son generados antes del despliegue de la aplicación o como parte de la creación del `EntityManagerFactory`. Esta característica le permitirá a su modelo de objetos de dominio JPA ser directamente generados en una base de datos. Es posible que el esquema generado deba ser afinado para un entorno de producción real.

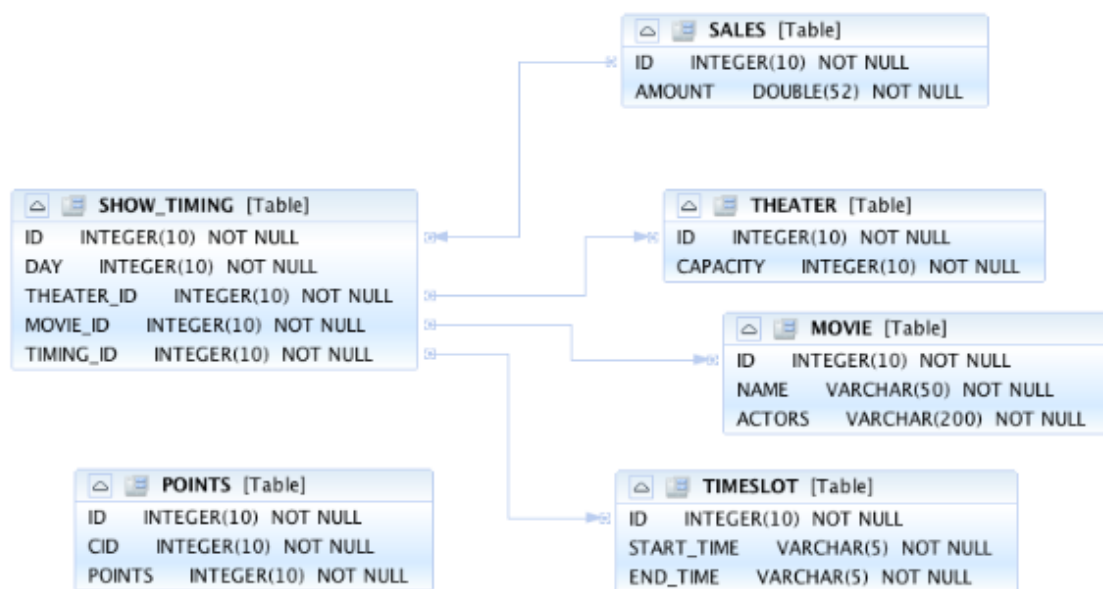
El archivo "persistence.xml" en la aplicación tiene las siguientes propiedades. Sus significados y posibles valores son explicados en la siguiente tabla:

Propiedad	Significado	Valor
<code>javax.persistence.schemageneration.database.action</code>	Especifica la acción a ser tomada por el proveedor de persistencia respecto a los artefactos de la base de datos.	"none" "create" "drop-and-create" "drop"
<code>javax.persistence.schemageneration.create-source/dropsource</code>	Especifica si la creación de los artefactos de la base de datos ocurre basado en el mapeo objeto/relacional, un script DDL, o la combinación de los 2.	"metadata" "script" "metadata-then-script" "script-then-metadata"
<code>javax.persistence.schemageneration.create-scriptsources/drop-script-source</code>	Especifica un <code>Java.IO.Reader</code> configurado para leer el script SQL o una cadena con la ruta donde está el archivo que contiene el	

	script SQL para la creación o borrado de artefactos de base de datos.	
javax.persistence.sql-loadscript-source	Especifica un <code>Java.IO.Reader</code> configurado para leer el script de carga SQL para inicializar la base de datos o una cadena que contiene la ruta donde se encuentra dicho script.	

Remítase a la especificación de [JPA 2.1](#) para un completo entendimiento de estas propiedades. En la aplicación los scripts son colocados en el directorio "META-INF" en el archivo WAR. Ya que la localización de estos scripts es especificada como una URL, los scripts pueden ser cargados también desde fuera del archivo WAR.

Siéntase libre de abrir "create.sql" y "load.sql" y mire los diferentes scripts SQL en ellos. El esquema de la base de datos a usar es mostrado a continuación:



Este folder también contiene el archivo "sales.csv" el cual contiene algunos datos separados por coma que serán usados más tarde en la aplicación.

**3.6 Entidades JPA, EJBs Stateless y endpoints REST:** Expanda "Source Packages". El paquete "org.glassfish.movieplex7.entities" contiene las entidades

JPA correspondientes a la definición de las tablas en la base de datos. Cada entidad JPA contiene muchos `@NamedQuery` definidos y usa restricciones de del API "Bean Validation" para reforzar la validación.

El paquete "org.glassfish.movieplex7.rest" contiene EJBs sin estado (stateless), correspondientes a diferentes entidades JPA.

Cada EJB tiene métodos que llevan a cabo operaciones CRUD sobre las entidades JPA y métodos de consulta convenientes. Cada EJB es también inyectable por EL (`@Named`) y publicado como un endpoint REST (`@Path`). La clase `ApplicationConfig` define la ruta base del endpoint REST. La ruta para el endpoint REST es el mismo nombre de la clase de la entidad JPA.

El mapeo entre las clases de entidades JPA, clases EJB y la URI de los correspondientes endpoint REST es mostrado en la siguiente tabla:

Clase entidad JPA	Clase EJB	Path RESTful
Movie	MovieFacadeREST	/webresources/movie
Sales	SalesFacadeREST	/webresources/sales
ShowTiming	ShowTimingFacadeREST	/webresources/showtiming
Theater	TheaterFacadeREST	/webresources/theater
Timeslot	TimeslotFacadeREST	/webresources/timeslot

Siéntase libre de explorar el código.

**3.7 Paginas JSF:** "WEB-INF/template.xhtml" define la plantilla a usar en las páginas Web y tiene un encabezado, barra de navegación izquierda y una sección de contenido principal. El archivo "index.xhtml" usa esta plantilla y los EJBs para desplegar el número de películas y teatros.

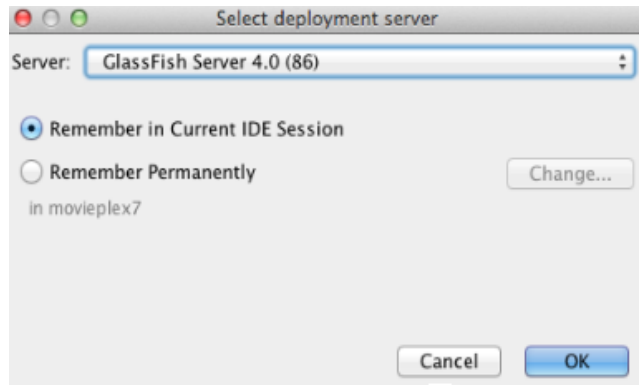
Java EE 7 habilita el descubrimiento de beans mediante CDI por defecto. No se requiere que el archivo "beans.xml" esté en "WEB-INF". Esto permite que todos los beans que cuenten con una anotación de bean, por ejemplo, un bean anotado explícitamente con alcance de CDI o un EJB, se encuentren disponibles para inyección.

Nota, "template.xhtml" está en la carpeta "WEB-INF" con el fin de que la plantilla pueda ser accedida únicamente por páginas empaquetadas dentro de la aplicación. Si fuera empaquetado con el resto de páginas sería accesible desde fuera de la aplicación permitiendo que páginas externas también puedan usarlo.

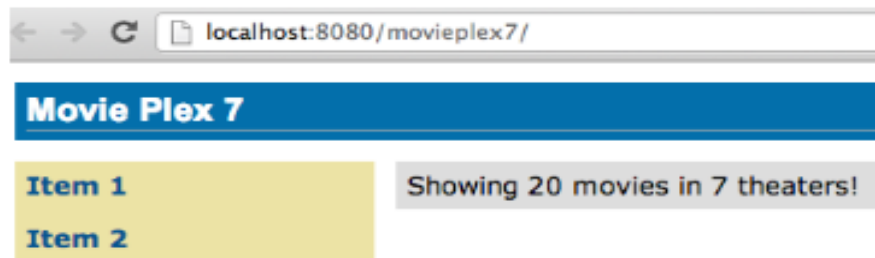
**3.8 Ejecute el ejemplo:** Haga clic derecho en el proyecto y seleccione "Run". Esto descargará todas las dependencias Maven en su máquina, construirá un archivo

WAR, lo desplegará en GlassFish 4 y apuntará a la URL `http://localhost:8080/movieplex7` en su navegador. Note que esto puede demorar un poco si hasta el momento no ha compilado una aplicación tipo Maven en su computadora. También, el proyecto mostrará líneas rojas en el código fuente indicando que las clases no se pueden encontrar. Esto es normal antes de que las dependencias sean descargadas. Todas las referencias se corregirán luego de que las dependencias sean descargadas y el proyecto sea compilado

Durante la primera ejecución, el IDE le pedirá que seleccione un servidor de despliegue. Escoja el servidor GlassFish 4.0 que ha configurado y haga clic en "OK".



La página que se despliega es mostrada a continuación:



## 4.0 Sala de chat (API de Java para WebSocket)

**Propósito:** Construir una sala de chat para los espectadores. Al hacer esto, las nuevas características de la API de Java para WebSocket 1.0 serán introducidas y demostradas haciendo uso de ellas en la aplicación.

**Tiempo estimado:** 30 – 45 minutos.

WebSocket provee un protocolo de comunicación full-duplex y bi-direccional sobre una única conexión TCP. WebSocket es una combinación de los protocolos [IETF RFC 6455](#) y [W3C JavaScript WebSocket API](#) (una recomendación candidata en el momento de escribir este documento).

El protocolo define un handshake abierto y transferencia de datos. El API habilita a las páginas web el uso del protocolo WebSocket como una comunicación de doble vía con el host remoto.

JSR 356 define una API estándar para crear aplicaciones WebSocket en la plataforma Java EE 7. El JSR provee soporte para:

- Crear endpoints de WebSocket usando anotaciones e interfaces.
- Iniciar e interceptar eventos WebSocket.
- Crear y consumir mensajes de texto y binarios de WebSocket.
- Configurar y administrar sesiones WebSocket.
- Integración con el modelo de seguridad de Java EE.

En esta sección se construirá una sala de chat para los espectadores del cine.

**4.1** Haga clic en "Source packages", seleccione "New", "Java Class...", nombre la clase como "ChatServer", nombre al paquete como "org.glassfish.movieplex7.chat", y haga clic en finalizar.

**4.2** Cambie la clase de tal manera que luzca así:

```
@ServerEndpoint("/websocket")
public class ChatServer {
    private static final Set<Session> peers =
        Collections.synchronizedSet(new HashSet<Session>());

    @OnOpen
    public void onOpen(Session peer) {
        peers.add(peer);
    }

    @OnClose
    public void onClose(Session peer) {
        peers.remove(peer);
    }

    @OnMessage
    public void message(String message, Session client) throws
        IOException, EncodeException {
        for (Session peer : peers) {
            peer.getBasicRemote().sendObject(message);
        }
    }
}
```

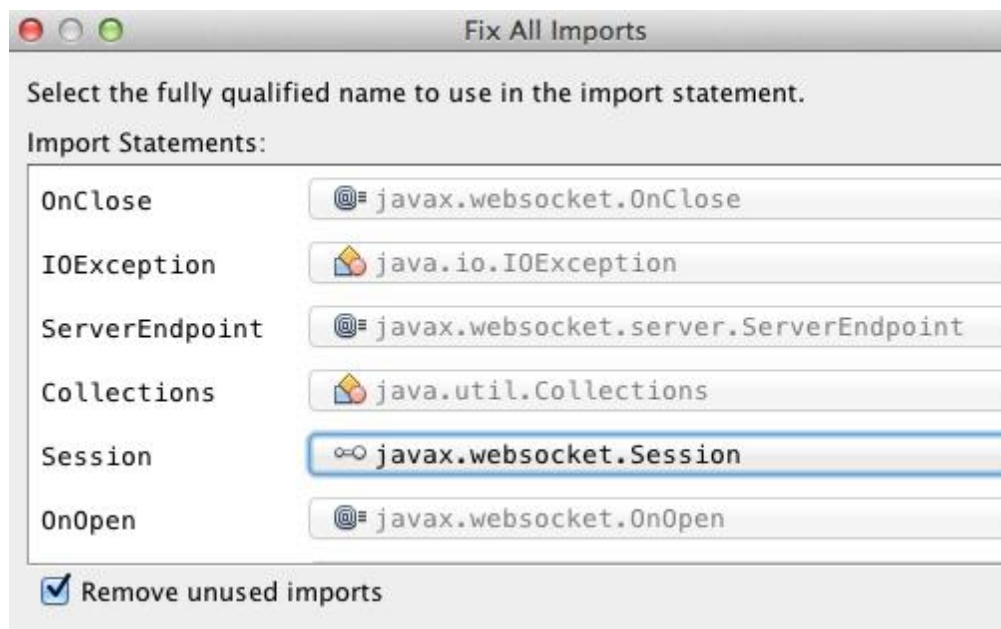


En este código:

- `@ServerEndPoint` decora la clase para que sea un endpoint de WebSocket tipo servidor. El valor define la URI donde este endpoint es publicado.
- `@OnOpen` y `@OnClose` decora los métodos que deben ser llamados cuando la sesión WebSocket es abierta o cerrada. El parámetro `peer` define la iniciación y terminación de la solicitud del cliente.
- `@OnMessage` decora el método que recibe el mensaje de Websocket entrante. El primer parámetro, `message`, es la parte relevante del mensaje (payload). El segundo parámetro, `client`, define el otro extremo de la conexión WebSocket. Esta implementación del método transmite el mensaje recibido a todos los clientes conectados a este servidor.

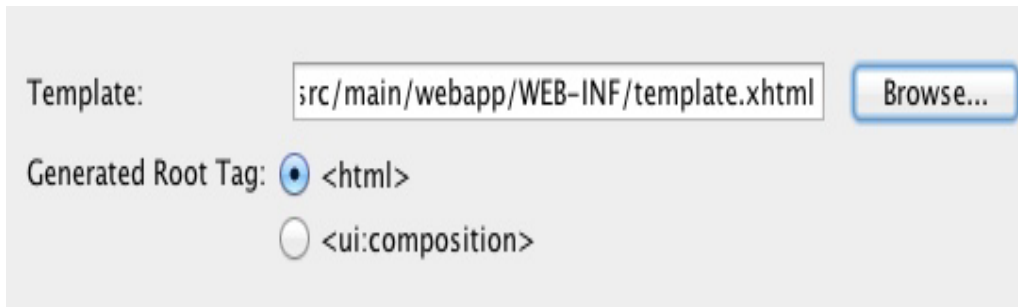
Corrija las importaciones haciendo clic derecho sobre el editor y seleccionando "Fix Imports" del menú desplegable o con la combinación de teclado (Command + Shift + I shortcut on Mac or Ctrl + Shift + I on Windows and Linux).

Asegúrese de referenciar la clase `java.websocket.Session` en lugar de la que muestra por defecto el IDE.



**4.3** En "Web Pages", seleccione "New", "Folder...", asigne el nombre como "chat" y haga clic en finalizar.

**4.4** Clic derecho sobre el directorio recién creado, seleccione la opción “New”, “Facelets Template Client”, asigne el nombre "chatroom". Clic en “Browse...” al lado de “Template:”. Expanda “Web Pages”, “WEB-INF”, seleccione “template.xhtml” y haga clic en “Select File”. Por último, clic en “Finish”.



En este nuevo archivo, se deben eliminar las secciones `<ui:define>` que tienen su atributo `name="top"` y `name="left"`, dado que éstos se heredan de la plantilla usada.

Reemplace la sección `<ui:define>` que tiene su atributo `name="content"` con el siguiente código:

```
<ui:define name="content">
    <form action="">
        <table>
            <tr>
                <td>
                    Chat Log<br/>
                    <textarea readonly="true" rows="6" cols="50" id="chatlog">
                    </textarea>
                </td>
                <td>
                    Users<br/>
                    <textarea readonly="true" rows="6" cols="20" id="users">
                    </textarea>
                </td>
            </tr>
        </table>
    </form>
</ui:define>
```

```

        <tr>
            <td colspan="2">
                <input id="textField" name="name" value="Duke" type="text"/>
                <input onclick="join();" value="Join" type="button"/>
                <input onclick="send_message();" value="Send" type="button"/><p/>
                <input onclick="disconnect();" value="Disconnect" type="button"/>
            </td>
        </tr>
    </table>
</form>
<div id="output"></div>
<script language="javascript" type="text/javascript"
    src="{facesContext.externalContext.requestContextPath}/chat/websocket.js">
</script>
</ui:define>

```

El anterior código construye un formulario HTML que tiene 2 áreas de texto: uno para desplegar el log del chat y el otro para desplegar la lista de usuarios actualmente conectados. Un único campo de texto es usado para capturar el nombre de usuario o el mensaje de chat. Haciendo clic en el botón "Join" se toma el valor como nombre de usuario y haciendo clic en "Send" toma el valor como mensaje de chat. Cuando se hace clic en estos botones se invocan métodos de JavaScript los cuales serán explicados en la próxima sección. Los mensajes de chat son enviados y recibidos como payloads WebSocket. Hay un botón explícito para desconectar el WebSocket. El div "output" es el contenedor para el mensaje de estado. La inicialización del WebSocket ocurre en "websocket.js" incluido en el parte inferior del fragmento de código mostrado.

**4.5** Haga clic derecho en "chat" en "Web Pages", seleccione "New", "Web", "JavaScript File".

Asigne el nombre "websocket" y haga clic en "Finish".

**4.6** Edite el contenido de "websocket.js" de tal manera que como se muestra a continuación:

```

var wsUri = 'ws://' + document.location.host
            + document.location.pathname.substr(0,
            document.location.pathname.indexOf("/faces"))
            + '/websocket';
console.log(wsUri);
var websocket = new WebSocket(wsUri);
var textField = document.getElementById("textField");
var users = document.getElementById("users");
var chatlog = document.getElementById("chatlog");
var username;
websocket.onopen = function(evt) { onOpen(evt); };
websocket.onmessage = function(evt) { onMessage(evt); };
websocket.onerror = function(evt) { onError(evt); };
websocket.onclose = function(evt) { onClose(evt); };
var output = document.getElementById("output");

function join() {
    username = textField.value;
    websocket.send(username + " joined");
}
function send_message() {
    websocket.send(username + ": " + textField.value);
}
function onOpen() { writeToScreen("CONNECTED");}
function onClose() {writeToScreen("DISCONNECTED");}
function onMessage(evt) {
    writeToScreen("RECEIVED: " + evt.data);
    if (evt.data.indexOf("joined") !== -1) {
        users.innerHTML += evt.data.substring(0, evt.data.indexOf("joined")) + "\n";
    } else {
        chatlog.innerHTML += evt.data + "\n";
    }
}
function onError(evt) {
    writeToScreen('<span style="color: red;">ERROR:</span> ' + evt.data);
}
function disconnect() {websocket.close();}
function writeToScreen(message) {
    var pre = document.createElement("p");
    pre.style.wordWrap = "break-word";
    pre.innerHTML = message;
    output.appendChild(pre);
}

```

La URI del endpoint del WebSocket es calculada usando variables estándar de JavaScript y adicionando la URI especificada en la clase ChatServer. El WebSocket es inicializado llamando a new WebSocket (...).

Los manejadores de eventos son registrados usando métodos que inician con onXXX. Los listeners registrados en este script son explicados a continuación:

Listeners	Llamado cuando
onOpen(evt)	La conexión WebSocket es iniciada.
onMessage(evt)	Un mensaje WebSocket es recibido.
onError(evt)	Un error ocurrió durante la comunicación.
onClose(evt)	Una conexión WebSocket es terminada.

Cualquier dato relevante se agrega también como parámetro de la función. Cada método imprime el estado en el browser usando el método utilitario writeToScreen. El método join envía un mensaje al endpoint al que un usuario en particular se ha unido. El endpoint luego difunde el mensaje a todos los clientes que lo escuchan.

El método send\_message concatena el nombre del usuario que se ha conectado y el valor del campo de texto y lo difunde a todos los clientes de manera similar. El método onMessage también actualiza la lista de usuarios conectados.

#### 4.7 Edite "WEB-INF/template.xhtml" y cambie:

```
<h: outputLink value ="item2.xhtml">Item2</h: outputLink>
```

Por:

```
<h:outputLink  
value="{facesContext.externalContext.requestContextPath}/faces/chat/chatroom.xhtml">  
Chat Room  
</h:outputLink>
```

La etiqueta outputLink se traduce en una etiqueta HTML <a> con un atributo href. La expresión {facesContext.externalContext.requestContextPath} provee la URI de la solicitud que identifica el contexto de la aplicación Web para esta solicitud. Esto permite que los enlaces en la barra de navegación de la izquierda sean URLs completas.

**4.8** Ejecute el proyecto haciendo clic derecho en él y seleccionando "Run". La barra de navegación del browser mostrara localhost:8080/movieplex7.

Haga clic en "Chat Room" para ver la salida como se muestra a continuación. El mensaje de estado "CONNECTED" es mostrado e indica que la conexión WebSocket con el endpoint ha sido establecida.

The screenshot shows the 'Movie Plex 7' web application. On the left, a yellow sidebar contains the links 'Book a movie' and 'Chat Room'. The 'Chat Room' link is selected. The main area is divided into two sections: 'Chat Log' and 'Users'. The 'Chat Log' section is empty. The 'Users' section is also empty. Below these sections, there is a text input field containing the name 'Duke', followed by 'Join' and 'Send' buttons. Below the input field is a 'Disconnect' button. At the bottom of the main area, the text 'CONNECTED' is displayed.

Por favor, asegúrese de que su navegador soporta WebSocket para que esta página se muestre correctamente. Chrome 14.0+, Firefox 11.0+, Safari 6.0+ e Internet Explorer 10.0+ lo soportan. Una lista completa de navegadores que soportan este protocolo se puede ver en [caniuse.com/websockets](http://caniuse.com/websockets).

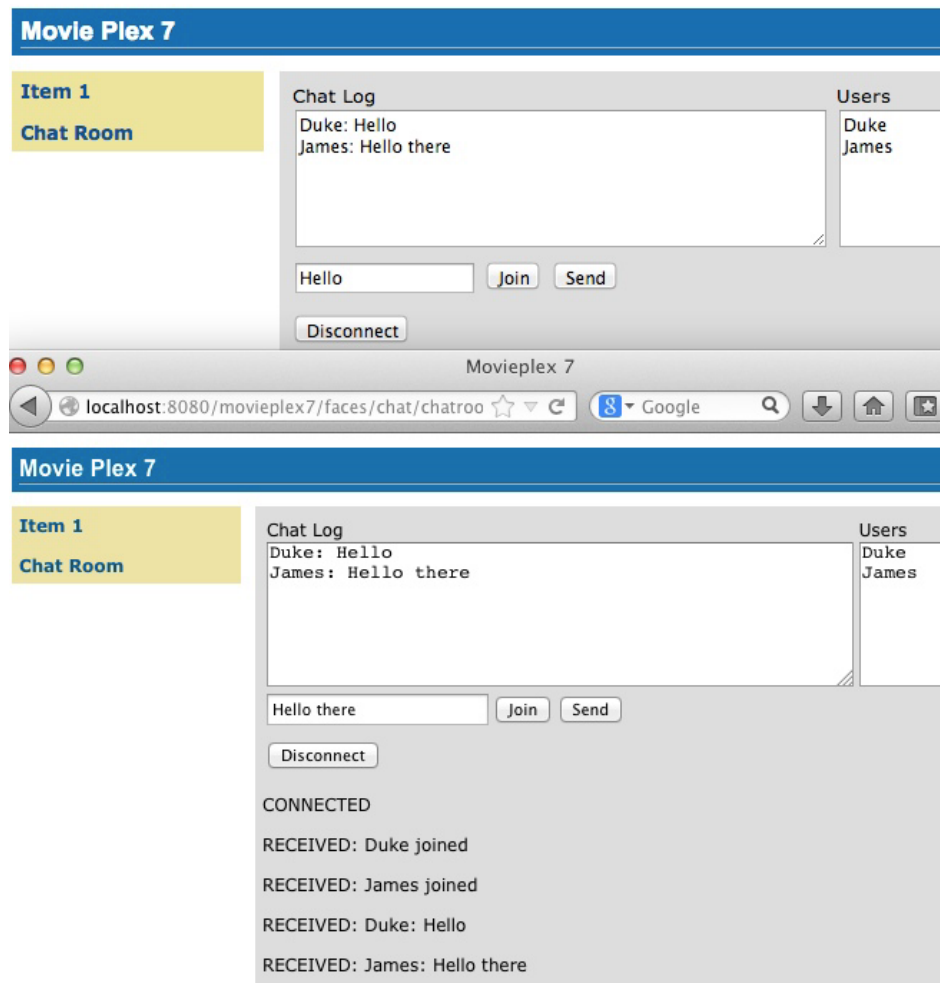
Abra la URI [localhost:8080/movieplex7](http://localhost:8080/movieplex7) en otra ventana de browser. Escriba "Duke" en la caja de texto en el primer browser y haga clic en "Join".

The screenshot shows the 'Movie Plex 7' web application. On the left, a yellow sidebar contains the links 'Item 1' and 'Chat Room'. The 'Chat Room' link is selected. The main area is divided into two sections: 'Chat Log' and 'Users'. The 'Chat Log' section is empty. The 'Users' section contains the name 'Duke'. Below these sections, there is a text input field containing the name 'Duke', followed by 'Join' and 'Send' buttons. Below the input field is a 'Disconnect' button. At the bottom of the main area, the text 'CONNECTED' is displayed, followed by the message 'RECEIVED: Duke joined'.

Note que la lista de usuarios y el mensaje de estado en ambos browser se actualizan. Escriba "James" en la caja de texto del segundo browser y haga clic en

"Join". Una vez más la lista de usuarios y el mensaje de estado en ambos browser se actualiza. Ahora usted puede teclear cualquier mensaje en cualquiera de los browser y hacer clic en "Send" para enviar el mensaje.

La salida de los 2 diferentes browser después del saludo inicial luce como se muestra a continuación.



Aquí se muestran desde Chrome en la parte de arriba y Firefox en la parte de abajo.

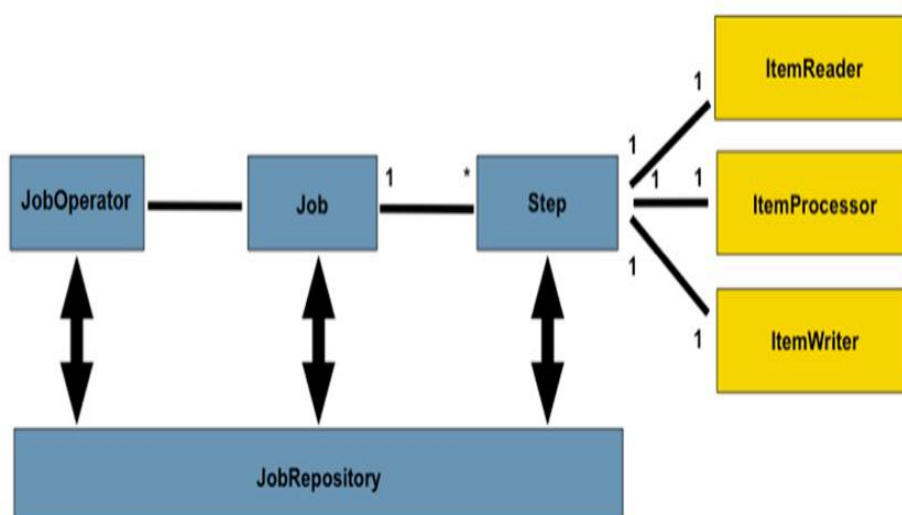
Puede usarse Chrome Developer Tools para monitorear el tráfico del WebSocket.

## 5.0 Venta de Tickets (Aplicaciones Batch para la plataforma Java)

**Propósito:** Leer la cantidad total en ventas de cada función y llenar la base de datos. Se demostrarán varias nuevas características del API de Java para el procesamiento de aplicaciones en Batch 1.0.

**Tiempo estimado:** 30 – 45 minutos

Procesamiento en Batch es la ejecución de una serie de trabajos y es adecuado usarlo cuando se tienen tareas no-interactivas, orientadas a trabajo con mucho volumen y son de larga duración. El API de Java para aplicaciones en Batch (JSR 352) define un modelo de programación y de ejecución para agendar y ejecutar trabajos.



Los conceptos principales del procesamiento en Batch son:

- Un **trabajo** es una instancia que encapsula un proceso en batch completo. Es definido usando algo denominado JSL (Job Specification Language) y está compuesto por múltiples pasos. JSL es implementado con XML y a veces es también llamado "Job XML".
- Un **paso** es un objeto que encapsula una fase de un trabajo y es independiente y secuencial. Contiene toda la información necesaria para definir y controlar el procesamiento actual.
- **Operador**, provee una interface para administrar todos los aspectos del procesamiento de un trabajo, incluyendo comandos operacionales, tales como: iniciar, re-iniciar y detener. También incluye comandos de repositorio tales como la recuperación de un trabajo y la ejecución de sus pasos.



- **Repositorio**, contiene la información a cerca de los trabajos, trabajos que están ejecutándose actualmente y trabajos que se han ejecutado anteriormente. Se puede acceder a este repositorio por medio del operador.
- **Lector-Procesador-Escritor**, patrón de diseño aplicado a este modelo también conocido como procesamiento orientado a partes. El lector lee un ítem a la vez, el procesador procesa el ítem basado en la lógica del negocio y lo pasa al escritor para que sea adicionado. Una vez que todas las partes han sido adicionadas, éstas son escritas y la transacción es ejecutada (commit).

En esta sección se leerán ventas acumuladas por cada función desde un archivo CSV y se escribirán a la base de datos.

**5.1** Clic derecho sobre “Source Packages”, seleccione “New”, “Java Package...”, especifique el nombre como “org.glassfish.movieplex7.batch” y haga clic en “Finish”.

**5.2** Clic derecho sobre el nuevo paquete, “New”, “Java Class...” especifique el nombre como “SalesReader”. Modifique la definición de esta clase para que herede de AbstractItemReader

```
extends AbstractItemReader
```

AbstractItemReader es una clase abstracta que implementa la interface ItemReader. Esta interface define métodos que leen ítems de un flujo para su procesamiento por partes. La implementación de este ItemReader retornará un String.

Adicione la etiqueta @Named a nivel de clase, lo que permitirá que el bean sea inyectado en el Job XML usando su nombre. También, adicione la etiqueta @Dependent a nivel de clase para marcar este bean como un bean definido por anotaciones y que esté disponible para ser inyectado.

Antes de continuar, asegúrese de resolver las importaciones.

**5.3** Adicione el siguiente campo

```
private BufferedReader reader;
```

Sobrescriba el método `open()` para inicializar el lector

```
public void open(Serializable checkpoint) throws Exception {
    reader = new BufferedReader(
        new InputStreamReader(
            Thread.currentThread()
                .getContextClassLoader()
                .getResourceAsStream("META-INF/sales.csv")));
}
```

El anterior método inicializa un `BufferedReader` desde el archivo “META-INF/sales.csv” que ha sido proveído junto con la aplicación.

Un ejemplo del contenido del archivo “sales.csv” se muestra a continuación:

```
1,500.00
2,660.00
3,80.00
4,470.00
5,1100.x0
```

Cada línea contiene el identificador de una función, separado por una coma del total de ventas para esa función. Nótese que la línea 5 contiene un error de tipografía. Este error junto con el de la línea 17, nos permitirán demostrar el manejo de errores.

#### 5.4 Sobrescriba el siguiente método:

```
@Override
public String readItem() {
    String string = null;
    try {
        string = reader.readLine();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    return string;
}
```

El método `readItem()` retorna el siguiente ítem del flujo. Cuando retorna `null` es porque ha llegado al final del flujo. Nótese que haber llegado al final del flujo

quiere decir que no se cuenta con más partes, por lo tanto las partes serán entregadas y este paso habrá terminado.

Antes de continuar asegúrese de resolver las importaciones.

**5.5** Clic derecho sobre el paquete “org.glassfish.movieplex7.batch” “New”, “Java Class...” especifique el nombre como “SalesProcessor”. Modifique la definición de esta clase para que implemente la interface ItemProcessor.

```
implements ItemProcessor
```

ItemProcessor es una interface que define un método que es usado para operar sobre un ítem de entrada y producir un ítem de salida. Este procesador acepta un String provisto por el lector, en nuestro caso SalesReader, y retorna una instancia de Sales al escritor. Sales ha sido precargada en la aplicación como una entidad JPA.

Adicione las etiquetas a nivel de clase @Named y @Dependent para que este bean pueda ser inyectado en el Job XML.

Antes de continuar, asegúrese de resolver las importaciones.

**5.6** Adicione implementación al método abstracto de la interfase:

```
@Override
public Sales processItem(Object s) {
    Sales sales = new Sales();
    StringTokenizer tokens = new StringTokenizer((String)s, ",");
    sales.setId(Integer.parseInt(tokens.nextToken()));
    sales.setAmount(Float.parseFloat(tokens.nextToken()));
    return sales;
}
```

Este método recibe un parámetro String que viene desde el SalesReader, analiza su valor y pobla una instancia de Sales para retornarla. Ésta, luego es agregada al escritor.

El método puede retornar null, indicando que el ítem procesado no debe ser agregado al escritor. Por ejemplo, si ocurren errores durante el procesamiento se puede retornar null para que el ítem no sea agregado. Sin embargo, el método es implementado de manera tal que puede lanzarse cualquier Excepción, pero el Job

XML se puede configurar para que omita estas excepciones y así mismo no agregar el ítem.

Antes de continuar, asegúrese de resolver las importaciones.

**5.7** Clic derecho sobre el paquete “org.glassfish.movieplex7.batch”, “New”, “Java Class...” especifique el nombre como “SalesWriter”. Modifique la definición de esta clase para que herede de AbstractItemWriter

```
extends AbstractItemWriter
```

AbstractItemWriter es una clase abstracta que implementa la interface ItemWriter. Esta interface define métodos que escriben ítems a un flujo para su procesamiento por partes. La implementación de este ItemWriter escribe una lista de ítems tipo Sales.

Adicione las etiquetas a nivel de clase @Named y @Dependent para que este bean pueda ser inyectado en el Job XML.

Antes de continuar, asegúrese de resolver las importaciones.

## **5.8** Inyecte el EntityManager

```
@PersistenceContext EntityManager em;
```

Sobrescriba el siguiente método de la clase abstracta:

```
@Override
@Transactional
public void writeItems(List list) {
    for (Sales s : (List<Sales>)list) {
        em.persist(s);
    }
}
```

El tiempo de ejecución Batch agrega el listado de Sales retornado por SalesProessor y lo hace disponible en este método como una lista, en dónde se itera sobre todos los ítems de la lista y se persisten en base de datos.

Este método hace uso de la etiqueta @Transactional. Esta etiqueta es nueva en JTA 1.2 y provee la habilidad de controlar, declarativamente, las transacciones en

beans inyectados con CDI o en clases definidas como managed beans por la especificación Java EE, por ejemplo, servlets, clases de recursos de JX-RS, servicios JAX-WS. Provee la semántica necesaria de los atributos transaccionales de un EJB sin depender de tecnologías como RMI. Este soporte es implementado por medio de un interceptor de CDI que lleva a cabo lo necesario: suspending, resuming, etc.

En este caso, una transacción es iniciada antes de que este método sea invocado, la transacción ejecuta un commit si no se lanzan excepciones y hace un rollback si hay excepciones en tiempo de ejecución. Este comportamiento puede ser cambiado usando los atributos rollbackOn y dontRollbackOn de la anotación.

Antes de continuar, asegúrese de resolver las importaciones.

**5.9** Cree un Job XML que define el trabajo, sus pasos y partes. En la etiqueta “Files”, expandir el proyecto ->”src”->”main”->”resources”, clic derecho sobre “META-INF”, “New”, “Folder...” especifique el nombre como “batch-jobs” y haga clic en “Finish”.

Clic derecho sobre el directorio creado, “New”, “Other...”, select “XML”, “XML Document”, clic en “Next >”, asigne el nombre “eod-sales”, click on “Next”, acepte los valores por defecto y haga clic en “Finish”.

Reemplace el contenido de este archivo con el siguiente:

```
<job id="endOfDaySales" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
  <step id="populateSales" >
    <chunk item-count="3" skip-limit="5">
      <reader ref="salesReader"/>
      <processor ref="salesProcessor"/>
      <writer ref="salesWriter"/>
      <skippable-exception-classes>
        <include class="java.lang.NumberFormatException"/>
      </skippable-exception-classes>
    </chunk>
  </step>
</job>
```

Este código muestra que se tiene un trabajo con un solo paso. Los elementos <reader>, <processor> y <writer> definen los nombres de las implementaciones de las interfaces ItemReader, ItemProcessor, and ItemWriter. El atributo item-count define que hay 3 items para leer/procesar/agregar y luego pasarlos al escritor. Todo el ciclo de leer/procesar/escribir se hace dentro de una transacción. El

elemento <skippable-exception-classes> especifica el conjunto de excepciones a ser omitidas durante el procesamiento de las partes.

El archivo CSV precargado con este taller contiene 2 errores de tipografía intencionales para que se lance una excepción de tipo `NumberFormatException`. Al especificar este elemento se puede omitir la excepción, ignorar ese elemento, y continuar con el procesamiento. Si este elemento no se encuentra en el Job XML, entonces todo el procesamiento en batch es cancelado. El atributo `skip-limit` especifica la cantidad de excepciones que un paso omitirá.

### 5.10 Invocar el trabajo batch.

Clic derecho sobre el paquete “org.glassfish.movieplex7.batch”, “New”, “Session Bean...” Ingrese el nombre como: “SalesBean” y haga clic en “Finish”.

Adicione el siguiente código al bean:

```
public void runJob() {
    try {
        JobOperator jo = BatchRuntime.getJobOperator();
        long jobId = jo.start("eod-sales", new Properties());
        System.out.println("Started job: with id: " + jobId);
    } catch (JobStartException ex) {
        ex.printStackTrace();
    }
}
```

Este método usa `BatchRuntime` para obtener una instancia de `JobOperator` que luego es usado para iniciar el trabajo. `JobOperator` es la interface para operar los trabajos batch. Puede ser usada para iniciar, detener y reiniciar trabajos. También puede revisar la historia de los trabajos para descubrir cuales trabajos están corriendo y cuales ya han sido ejecutados.

Adicione las etiquetas de clase `@Named` y `@RequestScoped` para que el bean pueda ser inyectado en EL.

Antes de continuar, asegúrese de resolver las importaciones.



## 5.11 Inyecte el EntityManager

@PersistenceContext EntityManager em;

Y adicione el siguiente método:

```
public List<Sales> getSalesData() {
    return em.createNamedQuery("Sales.findAll", Sales.class).getResultList();
}
```

Este método usa una consulta predefinida @NamedQuery para consultar a la base de datos y obtener todas las filas de la tabla.

Antes de continuar, asegúrese de resolver las importaciones.

## 5.12 Clic derecho en “Web Pages”, “New”, “Folder...”, especifique el nombre como “batch”, y clic en “Finish”.

Clic derecho sobre el directorio recién creado, seleccione la opción “New”, “Facelets Template Client”, asigne el nombre " sales". Clic en “Browse...” al lado de “Template:”. Expanda “Web Pages”, “WEB-INF”, seleccione “template.xhtml” y haga clic en “Select File”. Por último, clic en “Finish”.

En este nuevo archivo, se deben eliminar las secciones <ui:define> que tienen su atributo name="top" y name="left", dado que éstos se heredan de la plantilla usada.

Reemplace el contenido de la sección <ui:define> que tiene su atributo name="content" con el siguiente código:

```
<ui:define name="content">
    <h1>Movie Sales</h1>
    <h:form>
        <h:dataTable value="#{salesBean.salesData}" var="s" border="1">
            <h:column>
                <f:facet name="header">
                    <h:outputText value="Show ID" />
                </f:facet>
                #{s.id}
            </h:column>
            <h:column>
                <f:facet name="header">
                    <h:outputText value="Sales" />
                </f:facet>
                #{s.amount}
            </h:column>
        </h:dataTable>
        <h:commandButton value="Run Job" action="sales"
            actionListener="#{salesBean.runJob()}" />
        <h:commandButton value="Refresh" action="sales" />
    </h:form>
</ui:define>
```

Este código muestra el identificador de la función y las ventas de dicha función en una tabla por medio de SalesBean.getSalesData(). El primer botón invoca el trabajo que procesa el archivo CSV y llena la base de datos. El segundo botón refresca la página.

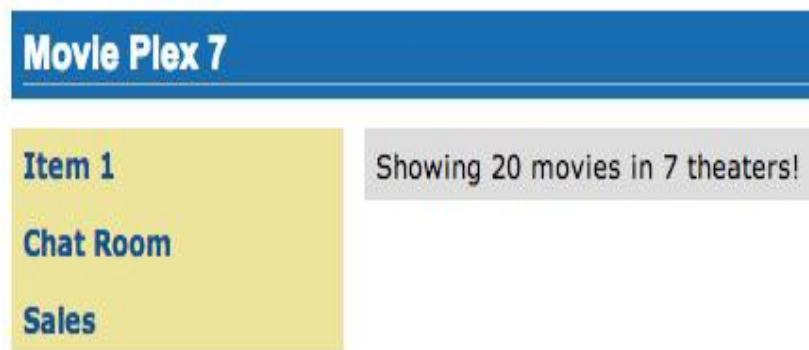
Clic derecho sobre le bombillo amarillo para solucionar espacios de nombres para los prefijos h y f.

**5.13** Adicione lo siguiente a “template.xhtml” junto con los otros <outputLink>s:

```
<p/><h:outputLink
value="{facesContext.externalContext.requestContextPath}/faces/batch/sales.xhtml">
    Sales
</h:outputLink>
```



**5.14** Ejecute el proyecto y podrá ver lo siguiente



La barra de navegación izquierda ahora cuenta con una nueva opción “Sales”.

**5.15** Clic en la opción “Sales” para ver la siguiente página:



La tabla se muestra vacía dado que no hay datos en la base de datos.

**5.16** Presione el botón “Run job” para iniciar el procesamiento del archivo CSV. Busque el mensaje “Waiting for localhost” en la barra de estado del navegador y espere unos cuantos segundos mientras se termina el proceso, luego presione el botón “Refresh” para ver que la información fue almacenada en la base de datos

## Movie Plex 7

Item 1

Chat Room

Sales

### Movie Sales

Show ID	Sales
2	660.0
1	500.0
3	80.0
4	470.0
6	240.0
8	2300.0
7	1000.0
9	230.0
11	800.0
10	600.0
12	1400.0
13	780.0
14	890.0
15	490.0
16	670.0
18	1230.0
20	900.0
19	700.0

Run Job

Refresh

La tabla ha sido poblada con la información de ventas. Note que el registro 5 no se encuentra en la tabla dado que tenía un valor incorrecto y el Job XML explícitamente definió omitir este tipo de errores.

## 6.0 Ver y borrar una película (API de Java para RESTful Web Services)

**Propósito:** Ver y borrar una película. Al hacer esto varias de las nuevas características de JAX-RS 2 serán introducidas y demostradas usándolas en la aplicación.

**Tiempo Estimado:** 30 – 45 minutos

JAX-RS 2 define una API estándar para crear, publicar e invocar un endpoint REST. JAX-RS 2 adiciona muchas características nuevas:

- La API Cliente que puede ser usada para acceder recursos Web y provee integración con proveedores de JAX-RS. Sin esta API, los usuarios necesitan usar un `URLConnection` de bajo nivel para acceder a un endpoint REST.
- Capacidades para ejecutar procesos de forma asíncrona en el cliente y servidor lo cual posibilita aplicaciones más escalables.
- Filtros de mensajes e interceptores de entidad así como también puntos de extensión bien definidos para extender las capacidades de una implementación.
- Restricciones de validación sobre parámetros y valores de retorno. Integración con Bean Validation.

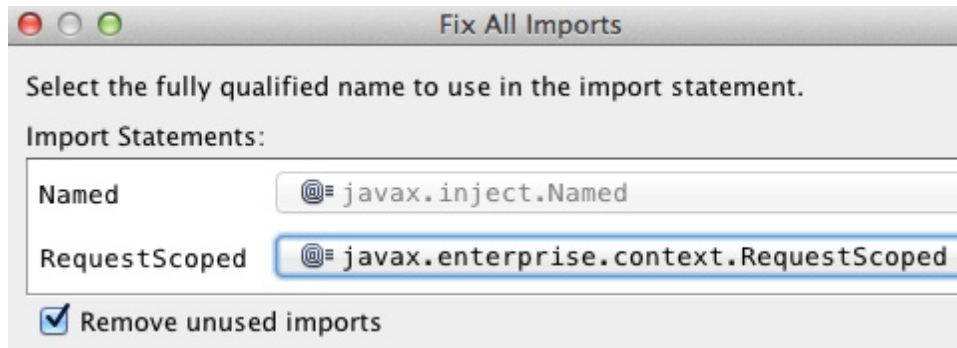
Esta sección proveerá la habilidad para ver todas las películas, detalles de la película seleccionada y borrar una película existente usando la API cliente de JAX-RS.

**6.1** Haga clic derecho en "Source Packages", seleccione "New", "Java Class...". Asigne el nombre de "MovieClientBean", colóquela en el paquete "org.glassfish.movieplex7.client" y haga clic en "Finish"

Este bean será usado para invocar el endpoint REST.

**6.2** Adicione a la clase las anotaciones `@Named` y `@RequestScoped`. Esto le permitirá a la clase ser inyectada en una expresión EL y también hace que el bean sea automáticamente activado y pasivado con la sesión.

Resuelva las importaciones y asegúrese de importar `javax.enterprise.context.RequestScoped`.



### 6.3 Adicione el siguiente código a la clase:

```
Client client;
WebTarget target;
@PostConstruct
public void init() {
    client = ClientBuilder.newClient();
    target = client.target("http://localhost:8080/movieplex7/webresources/movie/");
}
@PreDestroy
public void destroy() {
    client.close();
}
```

`ClientBuilder` es el punto de entrada principal a la API Cliente. Este usa un constructor fluido para invocar endpoints REST. Una nueva instancia de `Client` es creada usando la implementación del constructor de cliente por defecto que provee la implementación JAX-RS del proveedor. Es altamente recomendado crear únicamente el número de instancias requeridas de `Client` y cerrarlas apropiadamente.

En este caso, la instancia `Client` es creada y destruida en ciclo de vida de los métodos callback. La URI del endpoint es iniciada en esta instancia al llamar el método `target`.

### 6.4 Adicione el siguiente método a la clase:

```

public Movie[] getMovies() {
    return target
        .request()
        .get(Movie[].class);
}

```

Una solicitud es preparada al llamar el método request. El método GET de HTTP es invocado al llamar el método get. El tipo de respuesta es especificado en la llamada al último método así que el tipo del valor que retorna es Movie[].

**6.5** En NetBeans IDE, haga clic derecho en "Web Pages", seleccione "New", "Folder...", especifique el nombre del folder colocándole el nombre "client" y haga clic en "Finish".

Clic derecho sobre el directorio recién creado, seleccione la opción "New", "Facelets Template Client", asigne el nombre "movies". Clic en "Browse..." al lado de "Template:". Expanda "Web Pages", "WEB-INF", seleccione "template.xhtml" y haga clic en "Select File". Por último, clic en "Finish".

En este nuevo archivo, se deben eliminar las secciones <ui:define> que tienen su atributo name="top" y name="left", dado que éstos se heredan de la plantilla usada.

**6.6** Remplace el contenido dentro de <ui:define> que tiene su atributo name="content" con el fragmento de código mostrado abajo:

```

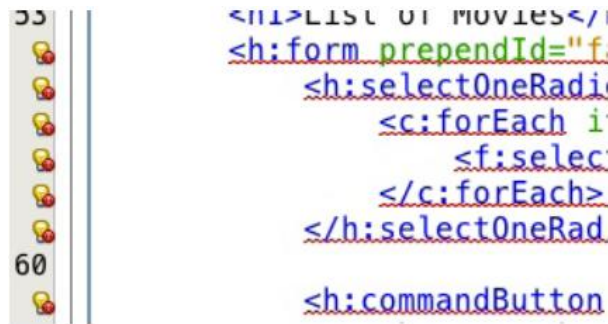
<h:form prependId="false">
<h:selectOneRadio value="#{movieBackingBean.movieId}" layout="pageDirection">
<c:forEach items="#{movieClientBean.movies}" var="m">
    <f:selectItem itemValue="#{m.id}" itemLabel="#{m.name}"/>
</c:forEach>
</h:selectOneRadio>
<h:commandButton value="Details" action="movie" />
</h:form>

```

Este fragmento de código invoca el método getMovies de MovieClientBean, itera sobre la respuesta en un bucle for y despliega el nombre de cada película con un botón de radio. El valor del botón de radio seleccionado está atado a la expresión EL #{movieBackingBean.movieId}.

El código también tiene un botón con la etiqueta "Details" y busca una página "moviex.xhtml" en el mismo directorio. Crearemos este archivo más tarde.

Haga clic en el bombillo amarillo en la barra de navegación izquierdo para resolver el espacio de nombres para la resolución de los prefijos. Esto necesita ser repetido 3 veces para los prefijo h: c: y f:



**6.7** Clic derecho sobre el paquete “org.glassfish.movieplex7.client”, seleccione la opción “New”, “Java Class...” especifique el valor como “MovieBackingBean” y haga clic en “Finish”.

Adicione el siguiente campo:

```
int movieId;
```

Adicione los métodos get/set haciendo clic derecho sobre el panel de edición y seleccionando la opción “Insert Code...” (Ctrl + I en Mac), “Getter and Setter...” y seleccione el campo creado recientemente y haga clic en “Generate”.

Adicione las etiquetas de clase @Named and @SessionScoped e implemente la interface Serializable.

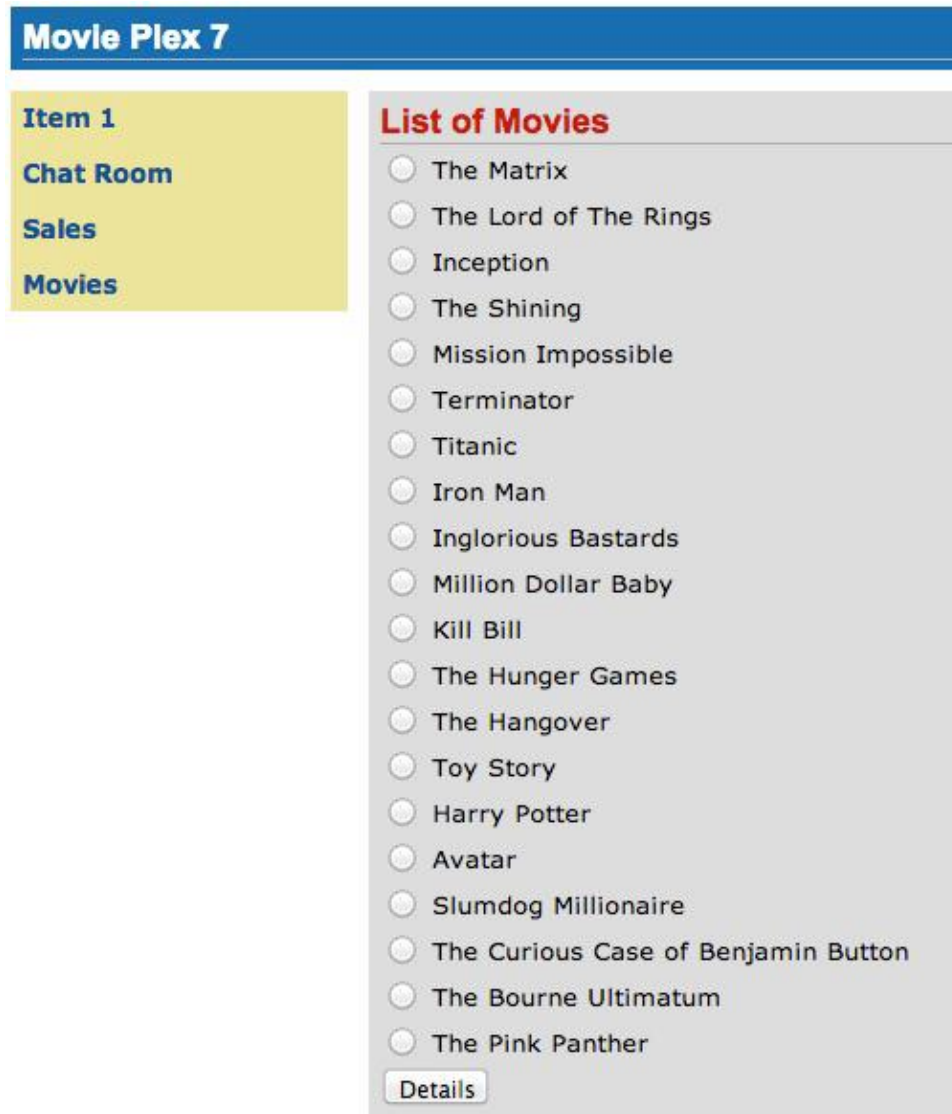
Asegúrese de tener importado lo siguiente:

javax.enterprise.context.SessionScoped y no javax.faces.bean.SessionScoped.

**6.8** En el archivo “template.xhtml” adicione el siguiente código dentro de <ui:insert> con el atributo name=”left”, junto con las otras etiquetas <h:outputlink>.

```
<p/><h:outputLink
value="{facesContext.externalContext.requestContextPath}/faces/client/movies.xhtml">
Movies
</h:outputLink>
```

Ejecutando el proyecto (Fn + F6 en Mac) y haciendo clic en “Movies” en la barra de navegación izquierda, podremos observar lo siguiente:



Se puede apreciar el listado de todas las películas con un botón de radio al lado izquierdo.

**6.9** En la clase “MovieClientBean”, se debe inyectar “MovieBackingBean” para leer el valor de la película seleccionada en la página. Para ello, adicione el siguiente código dentro de la clase:

```
@Inject  
MovieBackingBean bean;
```

**6.10** En la clase “MovieClientBean” adicione el siguiente método:

```
public Movie getMovie() {
    Movie m = target
        .path("{movie}")
        .resolveTemplate("movie", bean.getMovieId())
        .request()
        .get(Movie.class);
    return m;
}
```

El anterior código hace reuso de las instancias de Client y WebTarget creadas en el @PostConstruct. Además, adiciona una variable al final del URI del servicio REST, definida como {movie}, y le asigna un valor en concreto por medio del método resolveTemplate. El tipo de retorno es indicado como un parámetro del método get.

**6.11** Sobre la carpeta “client” en “Web Pages“, seleccione la opción “New” y luego “Facelets Template Client”.

Asigne el nombre “movie” al archivo. Luego, haga clic en “Browse...” al lado de “Template:”, expanda el nodo “Web Pages”, “WEB-INF”, seleccione “template.xhtml” y haga clic en “Select File”. Por último, clic en “Finish”.

En este nuevo archivo, se deben eliminar las secciones <ui:define> que tienen su atributo name=”top” y name=”left”, dado que éstos se heredan de la plantilla usada. Reemplace el contenido de la sección <ui:define> que tiene su atributo name=”content” con el siguiente código:

```
<h1>Movie Details</h1>
<h:form>
    <table cellpadding="5" cellspacing="5">
        <tr>
            <th align="left">Movie Id:</th>
            <td>#{movieClientBean.movie.id}</td>
        </tr>
        <tr>
            <th align="left">Movie Name:</th>
            <td>#{movieClientBean.movie.name}</td>
        </tr>
        <tr>
            <th align="left">Movie Actors:</th>
            <td>#{movieClientBean.movie.actors}</td>
        </tr>
    </table>
    <h:commandButton value="Back" action="movies" />
</h:form>
```



Clic en el bombillo amarillo para resolver el espacio de nombres para el prefijo h:. Los valores de salida son desplegados invocando al método getMovie y usando los valores de las propiedades id, name y actors.

**6.12** Ejecute el proyecto, seleccione la opción “Movies” en la barra de navegación izquierda, luego seleccione alguno de los botones de radio que se encuentran al lado izquierdo de cada película y por último, haga clic en “Details” para observar la siguiente pantalla:

Movie Plex 7	
<b>Item 1</b>	<b>Movie Details</b>
<b>Chat Room</b>	<b>Movie Id:</b> 10
<b>Sales</b>	<b>Movie Name:</b> Million Dollar Baby
<b>Movies</b>	<b>Movie Actors:</b> Hillary Swank, Clint Eastwood
	<input type="button" value="Back"/>

Presione el botón “Back” para seleccionar otra película.

**6.13** Adicionaremos la opción para borrar películas. En el archivo “movies.xhtml”, adicione el siguiente código junto con el código de los otros botones:

```
<h:commandButton value="Delete" action="movies"
actionListener="#{movieClientBean.deleteMovie()}" />
```

Este botón muestra el texto “Delete”, invoca al método deleteMovie del objeto movieClientBean y despliega la página “movie.xhtml”.

**6.14** Adicione el siguiente código a la clase “MovieClientBean”:

```

public void deleteMovie() {
    target
        .path("{movieId}")
        .resolveTemplate("movieId", bean.getMovieId())
        .request()
        .delete();
}

```

El código anterior, nuevamente hace reuso de las instancias de Client y WebTarget creadas en @PostConstruct. También adiciona una variable a la parte final de la URI del recurso REST, definido como {movieId} y le asocia un valor en concreto usando el método resolveTemplate. La URI de la película es preparada y por último el método delete es llamado para eliminar el recurso.

Antes de continuar, asegúrese de resolver la importación de paquetes y clases.

Al ejecutar el proyecto, se muestra lo siguiente:



Seleccione una película y presione el botón “Delete”. Con esto, la película seleccionada será borrada de la base de datos y refrescará el listado que se está mostrando en la página. Nótese que un redespliegue del proyecto borrará todas las películas y las adicionará nuevamente automáticamente.

## 7.0 Adicionar Película (Java API para procesamiento JSON)

**Propósito:** Adicionar una nueva película. Durante este desarrollo, se introducirán y demostrarán varias de las nuevas características que ofrece el API de Java para procesamiento JSON.

**Tiempo estimado:** 30 – 45 minutos.

El API de Java para procesamiento JSON provee un API estándar para analizar y generar JSON de manera tal que las aplicaciones pueden confiar en un API portable. Este nuevo API permite:

- Producir/Consumir JSON a manera de flujo (Similar a como lo hace el API StAX con XML)
- Construir un modelo de objetos Java para JSON (Similar a como lo hace el API DOM en XML)

En esta sección se definirán proveedores de entidades JAX-RS que nos permitirán leer y escribir JSON para la clase Movie. El API cliente de JAX-RS solicitará esta representación JSON de la clase mencionada.

Los proveedores de entidad de JAX-RS proveen servicios para asociar las representaciones JSON y sus tipos de dato en Java. Varios tipos de datos estándar tales como: String, byte[], javax.xml.bind.JAXBElement, java.io.InputStream, java.io.File, y otros, tienen asociaciones predefinidas y esto es requerido por el estándar. Sin embargo, las aplicaciones pueden proveer sus propias asociaciones a tipos de datos personalizados usando las interfaces MessageBodyReader y MessageBodyWriter.

Esta sección proveerá la habilidad para crear una nueva película en la aplicación. Generalmente, esta funcionalidad estará disponible una vez se cuente con las credenciales para autenticación/autorización.

**7.1** Clic derecho sobre “Source Packages”, seleccione la opción “New...”, “Java Package...”, use el siguiente valor: “org.glassfish.movieplex7.json” y presione “Finish”.

**7.2** Clic derecho sobre el paquete recién creado, seleccione la opción “New...”, “Java Class...”, use el nombre “MovieReader” y presione “Finish”. Adicione las siguientes anotaciones a nivel de clase:

```
@Provider
@Consumes(MediaType.APPLICATION_JSON)
```

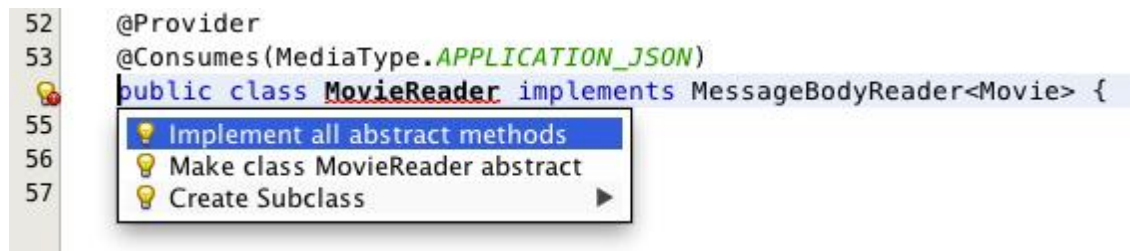
@Provider Permite que esta implementación sea descubierta por JAX-RS en tiempo de ejecución, durante la fase de escaneo de proveedores.

@Consumes indica que esta implementación consumirá una representación en JSON del recurso.

Antes de continuar, asegúrese de resolver los imports necesarios tal como se muestra:



**7.3** Hacer que la clase creada implemente la interface `MessageBodyReader<Movie>`



Clic sobre el bombillo amarillo en la definición de la clase y seleccionar la opción “Implement all abstract methods”.

**7.4** Cambie la implementación del método `isReadable` como sigue:

```
return Movie.class.isAssignableFrom(type);
```

Este método comprueba si el `MessageBodyReader` puede producir una instancia de un tipo (`type`) particular.

**7.5** Reemplace el método `readFrom` con el siguiente:

```

@Override
public Movie readFrom(Class<Movie> type, Type type1, Annotation[] antns,
                      MediaType mt, MultivaluedMap<String, String> mm, InputStream in)
                      throws IOException, WebApplicationException {
    Movie movie = new Movie();
    JsonParser parser = Json.createParser(in);
    while (parser.hasNext()) {
        switch (parser.next()) {
            case KEY_NAME:
                String key = parser.getString();
                parser.next();
                switch (key) {
                    case "id":
                        movie.setId(parser.getInt());
                        break;
                    case "name":
                        movie.setName(parser.getString());
                        break;
                    case "actors":
                        movie.setActors(parser.getString());
                        break;
                    default:
                        break;
                }
                break;
            default:
                break;
        }
    }
    return movie;
}

```

Este código lee un tipo de dato desde un flujo de entrada. Se crea una instancia de `JsonParser`, que es un analizador JSON a manera de flujo. Se leen las parejas llave/valor y se pobla la instancia de la clase `Movie`.

Antes de continuar, asegúrese de resolver los imports necesarios.

**7.6** Clic derecho sobre el paquete recién creado, seleccione la opción “New...”, “Java Class...”, use el nombre “`MovieWriter`” y presione “Finish”. Adicione las siguientes anotaciones a nivel de clase

```
@Provider
@Produces(MediaType.APPLICATION_JSON)
```

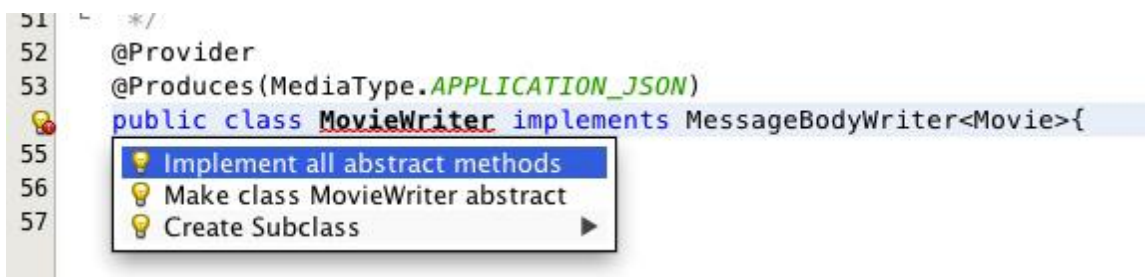
@Provider Permite que esta implementación sea descubierta por JAX-RS en tiempo de ejecución, durante la fase de escaneo de proveedores.

@Produces indica que esta implementación producirá una representación en JSON del recurso.

Antes de continuar, asegúrese de resolver las importaciones de paquetes necesarios tal como se muestra:



## 7.7 Hacer que la clase creada implemente la interface `MessageBodyWriter<Movie>`



Clic sobre el bombillo amarillo en la definición de la clase y seleccionar la opción “Implement all abstract methods”.

### 7.8 Cambie la implementación del método `isWritable` como sigue:

```
return Movie.class.isAssignableFrom(type);
```

Este método comprueba si el `MessageBodyWriter` puede producir una instancia de un tipo (`type`) particular.

### 7.9 Adicione la siguiente implementación al método `getSize`:

```
return -1;
```

Originalmente, este método era invocado para comprobar el tamaño en bytes. En JAX-RS 2.0, este método ha sido deprecado y valor retornado por éste es ignorado por JAX-RS en tiempo de ejecución. Se sugiere que todas las implementaciones de `MessageBodyWriter` retornen el valor `-1`.

### 7.10 Cambie la implementación del método `writeTo` como sigue:

```
public void writeTo(Movie t, Class<?> type, Type type1, Annotation[] antns,
    MediaType mt, MultivaluedMap<String, Object> mm, OutputStream out)
    throws IOException, WebApplicationException {
    JsonGenerator gen = Json.createGenerator(out);
    gen.writeStartObject()
        .write("id", t.getId())
        .write("name", t.getName())
        .write("actors", t.getActors())
        .writeEnd();
    gen.flush();
}
```

Este método escribe una instancia de `Type` a un mensaje HTTP. Un objeto de tipo `JsonGenerator` escribe texto JSON a manera de flujo. El encadenamiento del método `write` es usado para escribir diferentes tipos de datos al flujo de salida.

Antes de continuar, asegúrese de resolver las importaciones de paquetes necesarios.

### 7.11 En “Web Pages”, clic derecho sobre el directorio “client”, seleccione la opción “New”, “Facelets Template Client”

Asigne el nombre “addmovie” al archivo. Luego, haga clic en “Browse...” al lado de “Template:”, expanda el nodo “Web Pages”, “WEB-INF”, seleccione “template.xhtml” y haga clic en “Select File”. Por último, clic en “Finish”.

En este nuevo archivo, se deben eliminar las secciones <ui:define> que tienen su atributo name=”top” y name=”left”, dado que éstos se heredan de la plantilla usada.

Reemplace el contenido de la sección <ui:define> que tiene su atributo name=”content” con el siguiente código:

```
<h1>Add a New Movie</h1>
  <h:form>
    <table cellpadding="5" cellspacing="5">
      <tr>
        <th align="left">Movie Id:</th>
        <td><h:inputText value="#{movieBackingBean.movieId}"/></td>
      </tr>
      <tr>
        <th align="left">Movie Name:</th>
        <td><h:inputText value="#{movieBackingBean.movieName}"/></td>
      </tr>
      <tr>
        <th align="left">Movie Actors:</th>
        <td><h:inputText value="#{movieBackingBean.actors}"/></td>
      </tr>
    </table>
    <h:commandButton value="Add" action="movies"
      actionListener="#{movieClientBean.addMovie()}" />
  </h:form>
```

Este código crea un formulario que acepta el ingreso de los valores para id, name y actors de una película. Estos valores son asociados a los campos de “MovieBackingBean”. La acción del botón invoca al método addMovie de “MovieClientBean” y por último muestra la página “movies.xhtml”.

Clic en el bombillo amarillo para resolver problemas de espacios de nombres como se muestra a continuación:





### 7.12 Adicionar los campos movieName y actors a la clase “MovieBackingBean”:

```
String movieName;
String actors;
```

Genere los métodos get/set para los anteriores campos. Clic derecho sobre el editor de código y luego seleccione la opción “Insert Code...”.

### 7.13 Adicionar el siguiente código en “movies.xhtml” junto con los demás botones:

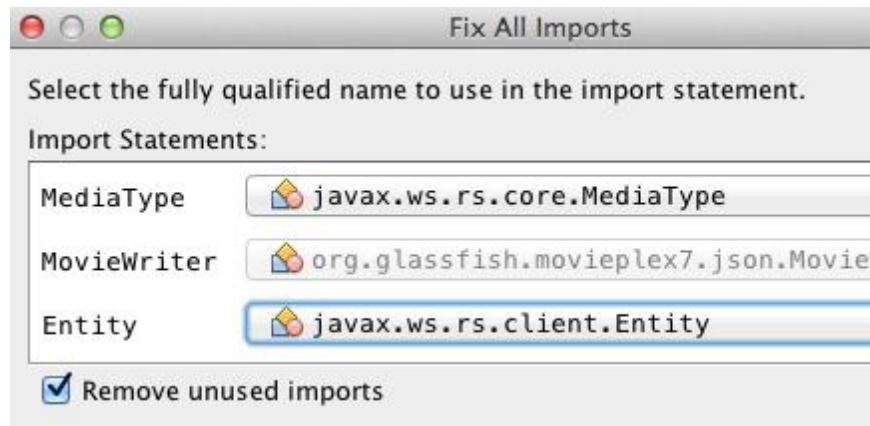
```
<h:commandButton value="New Movie" action="addmovie" />
```

### 7.14 Adicionar el siguiente método a la clase “MovieClientBean”:

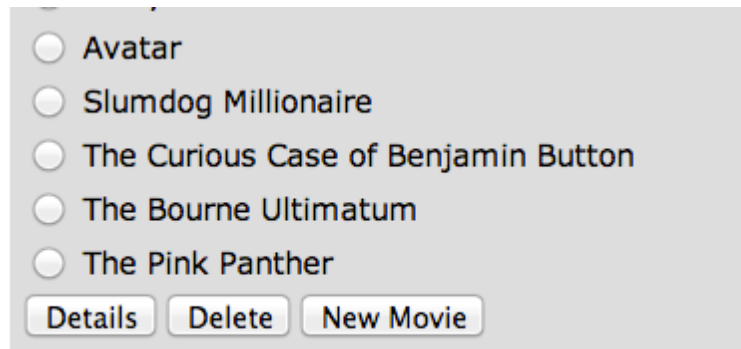
```
public void addMovie() {
    Movie m = new Movie();
    m.setId(bean.getMovieId());
    m.setName(bean.getMovieName());
    m.setActors(bean.getActors());
    target
        .register(MovieWriter.class)
        .request()
        .post(Entity.entity(m, MediaType.APPLICATION_JSON));
}
```

Este método crea una nueva instancia de una película, la pobla con los valores obtenidos por medio del bean “MovieBackingBean” y ejecuta un POST contra el servicio REST. El método register registra la clase MovieWriter, quién provee la conversión entre POJO (nuestra clase Movie) y JSON. El Media Type se especifica por medio de MediaType.APPLICATION\_JSON.

Antes de continuar, se deben resolver las importaciones de paquetes como se muestra a continuación:



**7.15** Ejecute el proyecto para ver la página principal actualizada:



Ahora es posible adicionar una nueva película haciendo clic en el botón “New Movie”.

**7.16** Ingrese los datos como se muestra a continuación:

**Movie Plex 7**

**Item 1**

- Chat Room
- Sales
- Movies**

**Add a New Movie**

**Movie Id:**

**Movie Name:**

**Movie Actors:**

Clic sobre el botón “Add”. El campo “Movie Id:” debe tener un valor mayor a 20 o de lo contrario se presentarán errores de violación sobre la llave primaria. La definición de la tabla en base de datos puede ser actualizada para obtener este campo a través de una secuencia, sin embargo, esto no lo realizaremos en este taller.

La página se actualiza con la nueva información:

- ☐ Harry Potter
- ☐ Avatar
- ☐ Slumdog Millionaire
- ☐ The Curious Case of Benjamin Button
- ☐ The Bourne Ultimatum
- ☐ The Pink Panther
- ☒ Skyfall

Nótese que la nueva película se muestra en el listado.

## 8.0 Puntos de Película (Servicio de Mensajería de Java - JMS)

**Propósito:** Los clientes acumulan puntos por ver películas.

**Tiempo estimado:** 30-45 minutos

Java Message Service 2.0 permite enviar y recibir mensajes entre sistemas distribuidos. JMS 2.0 incluye mejoras sobre la versión anterior:

- Nueva interface JMSContext.
- AutoCloseable en JMSContext, Connection y Session.
- Usa excepciones de tiempo de ejecución.
- JMSProducer cuenta con encadenamiento de métodos.
- Simplificación a la hora de enviar un mensaje.

En esta sección se creará una página para simular el envío de puntos de película acumulados por un cliente. Estos puntos son enviados a una cola JMS que es luego leída sincrónicamente por otro bean. Se usa una cola JMS para posterior procesamiento, posiblemente se puede almacenar en base de datos usando JPA.

**8.1** Clic derecho sobre “Source Packages”, seleccione la opción “New...”, “Java Package...”, use el siguiente valor: “org.glassfish.movieplex7.points” y presione “Finish”.

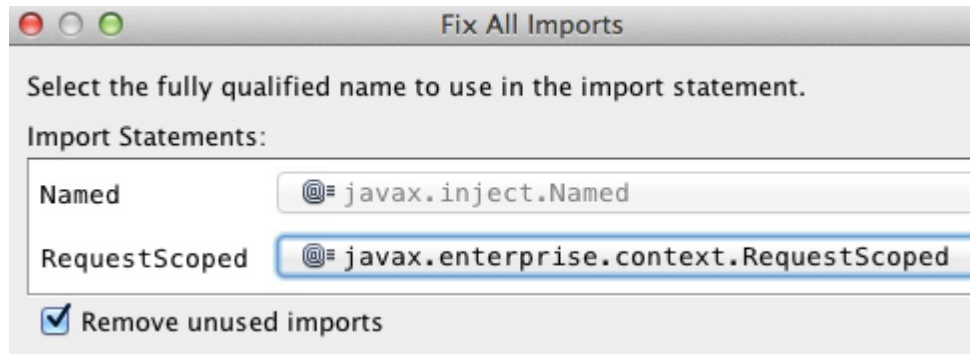
**8.2** Clic derecho sobre el paquete recién creado, seleccione la opción “New...”, “Java Class...”, use el nombre “SendPointsBean” y presione “Finish”.

Añada las siguientes anotaciones a nivel de clase:

```
@Named  
@SessionScoped
```

Con esto logramos que el bean pueda ser inyectado en EL y que automáticamente sea activado y pasivado junto con la sesión.

Antes de continuar, asegúrese de haber importado:



**8.3** Un mensaje es adicionado a la cola JMS una vez el cliente ha comprado sus boletos. Otro bean obtendrá los mensajes de la cola y actualizará los puntos para dicho cliente. Con esto se logra que ambos sistemas: el que genera datos a cerca de los boletos comprados y el que acredita la cuenta del cliente con los nuevos puntos, estén completamente desacoplados.

En este taller se simulará el envío y consumo de un mensaje por medio de la invocación directa del bean desde la página JSF.

Adicione el siguiente campo a la clase:

```
@NotNull
@Pattern(regexp = "^\\d{2},\\d{2}", message = "Message format must be 2 digits, comma, 2
digits, e.g. 12,12")
private String message;
```

Este campo contiene el mensaje enviado a la cola. El valor de este campo será asociado a un campo de texto en una página JSF (mas adelante). Se han asignado restricciones para permitir validación en formularios. Especifica que el dato se compone de 2 dígitos numéricos, seguidos por una coma y luego por otros 2 dígitos numéricos. Si el dato no cumple con el criterio de validación, entonces se mostrará el mensaje de error especificado en el atributo “message”.

De este modo podemos transmitir el identificador del cliente junto con los puntos ganados.

Adicione los métodos get/set haciendo clic derecho sobre el panel de edición y seleccionando la opción “Insert Code...” (Ctrl + I en Mac), “Getter and Setter...”. Seleccione el campo creado recientemente y haga clic en “Generate”.

**8.4** Adicione el siguiente código a la clase:

```

@Inject
JMSContext context;
@Resource(mappedName = "java:global/jms/pointsQueue")
Queue pointsQueue;
public void sendMessage() {
    System.out.println("Sending message: " + message);
    context.createProducer().send(pointsQueue, message);
}

```

Java EE requiere que se predefina una fábrica de conexiones JMS bajo el nombre JNDI `java:comp/DefaultJMSConnectionFactory`. Si no se especifica una fábrica de conexiones entonces se usará la predefinida. En un ambiente Java EE, donde CDI está habilitado por defecto, se puede inyectar un `JMSContext` administrado por contenedor así:

```

@Inject
JMSContext context;

```

Este código usa la fábrica de conexiones JMS predefinida para inyectar el `JMSContext` administrado por contenedor.

`JMSContext` es una nueva interface en JMS 2.0. Aquí se combina en un solo objeto la funcionalidad de dos objetos de JMS 1.1: `Connection` y `Session`.

Cuando una aplicación necesita enviar mensajes, usa el método `createProducer` para crear un `JMSProducer` quién provee métodos de configuración y para enviar mensajes. Los mensajes pueden ser enviados de manera síncrona o asíncrona.

Cuando una aplicación necesita recibir mensajes, usa alguno de los métodos `createConsumer` o `createDurableConsumer` para crear un `JMSConsumer`, quién provee métodos para recibir mensajes ya sean síncronos o asíncronos.

Todos los mensajes son enviados a una cola (creada posteriormente) que se identifica por el nombre JNDI `java:global/jms/pointsQueue`. El mensaje a enviar se obtiene del valor introducido en la página JSF que luego es asociado al campo `"message"`.

Asegúrese de resolver las importaciones de paquetes. La clase `Queue` debe ser importada de `javax.jms.Queue` y no de `java.util.Queue`.

**8.5** Clic derecho sobre el paquete “org.glassfish.movieplex7.points”, seleccione la opción “New”, “Java Class...”, especifique el nombre como “ReceivePointsBean”.

Adicione las siguientes anotaciones a nivel de clase:

```
@JMSDestinationDefinition(name = "java:global/jms/pointsQueue",  
                           interfaceName = "javax.jms.Queue")  
  
@Named  
@SessionScoped
```

Con esto, permitimos que el bean sea referido desde una expresión de EL y que automáticamente sea activado y pasivado junto con la sesión.

JMSDestinationDefinition es una nueva anotación en JMS 2.0. Es usada por la aplicación para proveer los recursos requeridos y permitir que ésta sea desplegada en un ambiente Java EE con configuración mínima. El código anterior crea una cola bajo el nombre JNDI java:global/jms/pointsQueue.

**8.6** Adicione el siguiente código a la clase:

```
@Inject  
JMSContext context;  
@Resource(mappedName="java:global/jms/pointsQueue")  
Queue pointsQueue;  
public String receiveMessage() {  
    String message =  
        context.createConsumer(pointsQueue).receiveBody(String.class);  
    System.out.println("Received message: " + message);  
    return message;  
}
```

El anterior código es muy parecido al del método SendPointsBean.createConsumer, crea un JMSConsumer el cual es usado para recibir el mensaje sincrónicamente.

**8.7** Adicione el siguiente método a la clase:

```

public int getQueueSize() {
    int count = 0;
    try {
        QueueBrowser browser = context.createBrowser(pointsQueue);
        Enumeration elems = browser.getEnumeration();
        while (elems.hasMoreElements()) {
            elems.nextElement();
            count++;
        }
    } catch (JMSEException ex) {
        ex.printStackTrace();
    }
    return count;
}

```

Este código crea un QueueBrowser, que es usado para obtener mensajes de una cola sin borrarlos de ésta. Calcula y retorna la cantidad de mensajes que existen en la cola.

Asegúrese de importar javax.jms.Queue, las demás importaciones pueden ir por defecto.

**8.8** Clic derecho sobre “Web Pages”, seleccione la opción “New”, “Folder...” especifique el nombre como “points” y haga clic en “Finish”.

En “Web Pages”, clic derecho sobre el nuevo directorio, seleccione la opción “New”, “Facelets Template Client”.

Asigne el nombre “points” al archivo. Luego, haga clic en “Browse...” al lado de “Template:”, expanda el nodo “Web Pages”, “WEB-INF”, seleccione “template.xhtml” y haga clic en “Select File”. Por último, clic en “Finish”.

En este nuevo archivo, se deben eliminar las secciones <ui:define> que tienen su atributo name=”top” y name=”left”, dado que éstos se heredan de la plantilla usada.

Reemplace el contenido de la sección <ui:define> que tiene su atributo name=”content” con el siguiente código:



```

<h1>Points</h1>
  <h:form>
    Queue size:
    <h:outputText value="#{receivePointsBean.queueSize}"/><p/>
    <h:inputText value="#{sendPointsBean.message}"/>
    <h:commandButton value="Send Message" action="points"
      actionListener="#{sendPointsBean.sendMessage()}" />
  </h:form>
  <h:form>
    <h:commandButton value="Receive Message" action="points"
      actionListener="#{receivePointsBean.receiveMessage()}" />
  </h:form>

```

Clic sobre le bombillo amarillo para resolver espacios de nombres del prefijo h:.

Esta página muestra la cantidad de mensajes que existen en la cola. Provee un campo de texto para ingresar el mensaje que será enviado a la cola. El primer botón invoca al método `sendMessage` del bean `SendPointsBean` y refresca la página actual. La cola se actualiza, en 1 para este caso, y es mostrado su nuevo valor. El segundo botón invoca al método `receiveMessage` del bean `ReceivePointsBean` y refresca la página. La cola es actualizada nuevamente, en este caso descontando 1 mensaje.

Si el mensaje ingresado no cumple con las reglas de validación definidas, entonces el mensaje de error es mostrado.

**8.9** Adicione el siguiente código al archivo “`template.xhtml`” junto a los demás `<outputLink>`s:

```

<p/><h:outputLink
  value="{facesContext.externalContext.requestContextPath}/faces/points/points.xhtml">
  Points</h:outputLink>

```

**8.10** Ejecute el proyecto, la página actualizada debe lucir como se muestra a continuación:

## Movie Plex 7

Item 1

Chat Room

Sales

Movies

Points

Showing 20 movies in 7 theaters!

Seleccione la opción "Points" para ver lo siguiente:

## Movie Plex 7

Item 1

Chat Room

Sales

Movies

Points

### Points

Queue size: 0

Send Message

Receive Message

Como se puede apreciar, la cola cuenta con cero mensajes. Ingrese el siguiente mensaje en el campo de texto "1212" y haga clic sobre el botón "Send Message" para obtener un mensaje de error, dado que el mensaje no cumple con las restricciones de validación:

## Movie Plex 7

Item 1

Chat Room

Sales

Movies

Points

### Points

Queue size: 0

Send Message

Receive Message

- Message format must be 2 digits, comma, 2 digits, e.g. 12,12

Ingrese el mensaje “12,12” en el campo de texto y presione el botón “Send Message” para obtener la siguiente pantalla:

The screenshot shows a web application titled "Movie Plex 7". On the left, there is a yellow sidebar with a list of items: "Item 1", "Chat Room", "Sales", "Movies", and "Points". The "Points" item is currently selected. On the right, there is a grey panel titled "Points" in red. Below the title, it says "Queue size: 1". There is a text input field containing "12,12" and a "Send Message" button to its right. Below the input field is a "Receive Message" button.

Como se puede apreciar, la cola fue actualizada y ahora vemos que hay un mensaje en ella. Presione el botón “Receive Message” y obtendrá la siguiente pantalla:

The screenshot shows the same "Movie Plex 7" interface. The "Points" item is still selected in the sidebar. In the "Points" panel, the "Queue size" is now "0". The text input field is empty, and the "Receive Message" button is still present below it. The "Send Message" button is still to the right of the input field.

Se puede apreciar que la cola ha sido actualizada, el mensaje ha sido retirado de ésta y ahora tiene cero mensajes.

Presione el botón “Send Message” 4 veces para obtener lo siguiente:

**Movie Plex 7**

Item 1  
Chat Room  
Sales  
Movies  
Points

**Points**  
Queue size: 4  
12,12 Send Message  
Receive Message

La cola es actualizada y ahora vemos que hay 4 mensajes en ella. Presione el botón “Receive Message” 2 veces para obtener lo siguiente:

**Movie Plex 7**

Item 1  
Chat Room  
Sales  
Movies  
Points

**Points**  
Queue size: 2  
12,12 Send Message  
Receive Message

La cola es actualizada nuevamente mostrando que se consumieron 2 mensajes y aún quedan otros 2 en ella.

## 9.0 Reservas (Java Server Faces)

**Propósito:** Crear las páginas que permitan a un usuario reservar funciones de películas. Durante esta sección se demostrará una de las nuevas características de JSF 2.2.

**Tiempo estimado:** 30-45 minutos

En la nueva versión de Java Server Faces, 2.2, existe una nueva característica llamada “Faces Flow” que provee una encapsulación de páginas que están relacionadas y que cuentan con una entrada y salida definida en la aplicación. Esta nueva característica se basa en algunos conceptos encontrados en ADF Task Flow, Spring Web Flow y Apache MyFaces CODI.

Junto con esta nueva característica, existe una nueva anotación CDI denominada `@FlowScoped` para definir un nuevo alcance, local al flujo. También se introduce `@FlowDefinition` para definir el flujo usando métodos productores de CDI. Deben existir puntos de entrada/salida y parámetros bien definidos. Con esto, se puede empaquetar el flujo como un archivo .jar o .zip y ser reusado en otras partes de la aplicación. De este modo, la aplicación se convierte en una colección de flujos y de algunas páginas sueltas. Generalmente, los objetos dentro de un flujo son diseñados para permitir que un usuario realice una tarea que requiere de la interacción con diferentes vistas.

En este taller se construirá un flujo que le permitirá a un usuario el realizar una reserva para una película. El flujo contendrá las siguientes cuatro páginas:

1. Mostrar el listado de películas.
2. Mostrar el listado de horarios disponibles.
3. Confirmación de la selección.
4. Realizar la reserva y mostrar el boleto de entrada.

**9.1** Los objetos dentro de un flujo están relacionados entre ellos, por tal razón se deben almacenar juntos en un mismo directorio. En NetBeans, clic derecho sobre “Web Pages”, seleccione la opción “New”, “Folder...”, especifique el nombre como “booking” y haga clic en “Finish”.

**9.2** En “Web Pages”, clic derecho sobre el nuevo directorio, seleccione la opción “New”, “Facelets Template Client”, asigne el nombre “booking” al archivo. Luego, haga clic en “Browse...” al lado de “Template:”, expanda el nodo “Web Pages”, “WEB-INF”, seleccione “template.xhtml” y haga clic en “Select File”. Por último, clic en “Finish”.

En este nuevo archivo, se deben eliminar las secciones `<ui:define>` que tienen su atributo `name="top"` y `name="left"`, dado que éstos se heredan de la plantilla usada.

**9.3** “booking.xhtml” es el punto de entrada a nuestro flujo (explicaremos esto más adelante). Reemplace el contenido de la sección <ui:define> que tiene su atributo name=”content” con el siguiente código:

```
<ui:define name="content">
<h2>Pick a movie</h2>
  <h:form prependId="false">
    <h:selectOneRadio value="#{booking.movieId}"
      layout="pageDirection" required="true">
      <f:selectItems value="#{movieFacadeREST.all}" var="m"
        itemValue="#{m.id}" itemLabel="#{m.name}"/>
    </h:selectOneRadio>
    <h:commandButton id="shows" value="Pick a time"
      action="showtimes" />
  </h:form>
</ui:define>
```

El código anterior muestra un listado de películas con botones de selección. La película escogida es asociada a #{booking.movieId}, el cual será definido con un bean con alcance de flujo (@FlowScoped). El valor del atributo “action” del botón hace referencia a la siguiente vista en el flujo, por ejemplo, “showtimes.xhtml” que, para nuestro caso, se encuentra en el mismo directorio (la crearemos más adelante).

Clic en el bombillo amarillo y acepte las sugerencias para resolver espacios de nombres.



**9.4** Clic derecho sobre “Source Packages”, seleccione la opción “New...”, “Java Class...”, especifique el nombre de la clase como “Booking” y el nombre del paquete: “org.glassfish.movieplex7.points” y presione “Finish”.

Adicione la etiqueta `@Named` a nivel de clase para permitir que la clase sea inyectada por EL. Adicione la etiqueta `@FlowScoped("booking")` para definir que el bean se alojará con alcance del flujo. El bean será activado/pasivado automáticamente cuando se entre o salga del flujo.

Adicione el siguiente campo a la clase:

```
int movieId;
```

Adicione los métodos `get/set` por medio del menú “Source”, “Insert Code...”, “Getter and Setter...” y seleccionando el campo.

Inyecte `EntityManager` a esta clase por medio del siguiente código dentro de la clase:

```
@PersistenceContext
EntityManager em;
```

Adicione el siguiente método:

```
public String getMovieName() {
    try {
        return em.createNamedQuery("Movie.findById",
            Movie.class).setParameter("id", movieId).getSingleResult().getName();
    } catch (NoResultException e) {
        return "";
    }
}
```

Este método retornará el nombre de una película basado en la selección actual. Alternativamente, los valores de `movieId` y `name` podrían ser pasados desde la selección actual al backing bean, con esto se reduciría la cantidad de viajes necesarios a la base de datos.

**9.5** Cree el archivo “showtimes.xhtml” dentro del directorio “bookings”. Siga los mismos pasos de la sección 9.2. Reemplace el contenido de la sección `<ui:define>` que tiene su atributo `name="content"` con el siguiente código:

```

<ui:define name="content">
    <h2>Show Timings for <font color="red">#{booking.movieName}</font></h2>
    <h:form>
        <h:selectOneRadio value="#{booking.startTime}"
            layout="pageDirection" required="true">
            <c:forEach items="#{timeslotFacadeREST.all}" var="s">
                <f:selectItem itemValue="#{s.id},#{s.startTime}"
                    itemLabel="#{s.startTime}"/>
            </c:forEach>
        </h:selectOneRadio>
        <h:commandButton value="Confirm" action="confirm" />
        <h:commandButton id="back" value="Back" action="booking" />
    </h:form>
</ui:define>

```

El código anterior crea un formulario HTML que muestra la película seleccionada y todas sus funciones. El código `#{timeslotFacadeREST.all}` retorna una lista de todas las funciones de la película y se itera sobre ella usando `c:forEach`. El id y la fecha de inicio de la función son asociados a `#{booking.startTime}`. Existe un botón (vale="Back") que permite volver a la página anterior y otro botón (value="Confirm") permite ir a la siguiente vista en el flujo, en este caso a "confirm.xhtml".

Normalmente, un usuario espera que se desplieguen funciones solo de la película seleccionada, sin embargo, las funciones de todas las películas son mostradas en este taller. Con esto podremos ir y venir dentro del flujo si se escoge una función incorrecta para la película. Se requeriría una consulta a la base de datos diferente para permitir que se muestren únicamente las funciones de la película seleccionada, sin embargo, no haremos eso en este taller.

## 9.6 Adicione los siguientes campos a la clase "Booking":

```

String startTime;
int startTimeId;

```

También adicione los siguientes métodos:

```

public String getStartTime() {
    return startTime;
}

```



```

public void setStartTime(String startTime) {
    StringTokenizer tokens = new StringTokenizer(startTime, ",");
    startTimeld = Integer.parseInt(tokens.nextToken());
    this.startTime = tokens.nextToken();
}

```

```

public int getStartTimeld() {
    return startTimeld;
}

```

Los anteriores métodos asociarán los valores recibidos del formulario. También se debe adicionar lo siguiente:

```

public String getTheater() {
    // for a movie and show
    try {
        // Always return the first theater
        List<ShowTiming> list =
            em.createNamedQuery("ShowTiming.findByMovieAndTimingId",
                               ShowTiming.class)
                .setParameter("movieId", movieId)
                .setParameter("timingId", startTimeld)
                .getResultList();

        if (list.isEmpty())
            return "none";

        return list
            .get(0)
            .getTheaterId()
            .getId().toString();

    } catch (NoResultException e) {
        return "none";
    }
}

```

El anterior método buscará el teatro más próximo para la película seleccionada y mostrará el horario.

Se podría mostrar una lista de teatros que ofrecen la película en una vista adicional, pero no lo haremos en este taller.

**9.7** Cree la página “confirm.xhtml” en el directorio “booking” y siga los pasos definidos en la sección 9.2. Reemplace el contenido de la sección <ui:define> que tiene su atributo name=”content” con el siguiente código:

```
<ui:define name="content">
  <c:choose>
    <c:when test="#{booking.theater == 'none'}">
      <h2>No theater found, choose a different time</h2>
      <h:form>
        Movie name: #{booking.movieName}<p/>
        Starts at: #{booking.startTime}<p/>
        <h:commandButton id="back" value="Back"
          action="showtimes"/>
      </h:form>
    </c:when>
    <c:otherwise>
      <h2>Confirm ?</h2>
      <h:form>
        Movie name: #{booking.movieName}<p/>
        Starts at: #{booking.startTime}<p/>
        Theater: #{booking.theater}<p/>
        <p/><h:commandButton id="next" value="Book"
          action="print"/>
        <h:commandButton id="back" value="Back"
          action="showtimes"/>
      </h:form>
    </c:otherwise>
  </c:choose>
</ui:define>
```

El código muestra la película seleccionada, el horario y el teatro si fue posible encontrarlo. La reserva continuará si todos los 3 valores se encuentran disponibles. “print.xhtml” es la última vista del flujo y es accedida por medio del botón con value=”Book”.

El atributo actionListener puede ser adicionado al botón para invocar la lógica del negocio para hacer la reserva. Igualmente, vistas adicionales pueden crearse para obtener los datos de tarjeta de crédito y correo electrónico.

**9.8** Cree la página “print.xhtml” en el directorio “booking” y siga los pasos definidos en la sección 9.2. Reemplace el contenido de la sección `<ui:define>` que tiene su atributo `name="content"` con el siguiente código:

```
<ui:define name="content">
    <h2>Reservation Confirmed</h2>
    <h:form>
        Movie name: #{booking.movieName}<p/>
        Starts at: #{booking.startTime}<p/>
        Theater: #{booking.theater}<p/>
        <p><h:commandButton id="home" value="home" action="goHome" /></p>
    </h:form>
</ui:define>
```

Esta página simplemente muestra el nombre de la película, horario y teatro seleccionado.

El botón inicia la salida del flujo. Su atributo `action` define una regla de navegación que veremos en el siguiente paso.

**9.9** Las páginas “booking.xhtml”, “showtimes.xhtml”, “confirm.xhtml” y “print.xhtml” se encuentran todas dentro del mismo directorio. Ahora debemos indicar que las páginas en este directorio deben ser tratadas como nodos vista de un flujo y no como páginas normales. Esto se logra a través de la adición del archivo “booking/booking-flow.xml” o teniendo una clase con las etiquetas `@Produces` y `@FlowDefinition`.

Clic derecho sobre el directorio “Web Pages/booking”, seleccione la opción “New”, “Other”, “XML”, “XML Document”, asigne el nombre “booking-flow”, clic en “Next>”, acepte los valores por defecto para “Well-formed Document”, y clic en “Finish”. Edite el archivo de forma tal que luzca así:

```
<faces-config version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
    <flow-definition id="booking">
        <flow-return id="goHome">
            <from-outcome>/index</from-outcome>
        </flow-return>
    </flow-definition>
</faces-config>
```

Este archivo define el grafo de nuestro flujo. Utiliza el mismo encabezado de “faces-config.xml”, pero define un <flow-definition>.

La etiqueta <flow-return> define el nodo de retorno en el grafo del flujo. La etiqueta <from-outcome> contiene el nodo o una expresión EL que define el nodo al que se retorna. En nuestro caso, el nodo al que se retorna es la página de inicio.

**9.10** Finalmente, invoque el flujo editando el archivo “WEB-INF/template.xhtml” así:

El código:

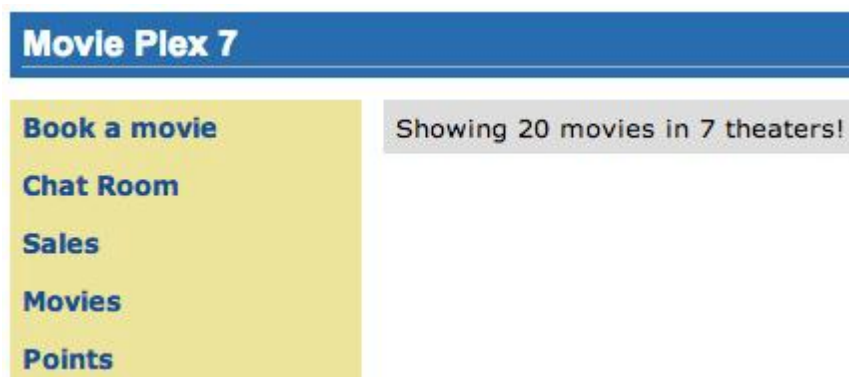
```
<h:commandLink action="item1">Item 1</h:commandLink>
```

Debe cambiar a:

```
<h:commandLink action="booking">Book a movie</h:commandLink>
```

La etiqueta commandLink se convierte en una etiqueta HTML <a> que se comporta como un botón de tipo submit en un formulario. El atributo action indica el directorio en el cual se encuentran todas las vistas del flujo. Este directorio contiene el archivo “booking-flow.xml” el cual define al flujo.

**9.11** Ejecute el proyecto haciendo clic derecho sobre él y seleccionando la opción “Run”. A continuación la salida que se debe ver en el navegador:



Clic sobre la opción “Book a movie” para ver lo siguiente:

## Movie Plex 7

**Book a movie**

**Chat Room**

**Sales**

**Movies**

**Points**

### Pick a movie

- ☐ The Matrix
- ☐ The Lord of The Rings
- ☐ Inception
- ☐ The Shining
- ☐ Mission Impossible
- ☐ Terminator
- ☐ Titanic
- ☐ Iron Man
- ☐ Inglorious Bastards
- ☐ Million Dollar Baby
- ☐ Kill Bill
- ☐ The Hunger Games
- ☐ The Hangover
- ☐ Toy Story
- ☐ Harry Potter
- ☐ Avatar
- ☐ Slumdog Millionaire
- ☐ The Curious Case of Benjamin Button
- ☐ The Bourne Ultimatum
- ☐ The Pink Panther

Pick a time

Seleccione una película, por ejemplo, “The Shining” and click on “Pick a time”. Se debe mostrar lo siguiente:

**Movie Plex 7**

**Book a movie**  
**Chat Room**  
**Sales**  
**Movies**  
**Points**

### Show Timings for **The Shining**

☐ 10:00  
☐ 12:00  
☐ 02:00  
☐ 04:00  
☐ 06:00

Seleccione un horario, por ejemplo, “04:00” y presione el botón “Confirm” para ver la siguiente página:

**Movie Plex 7**

**Book a movie**  
**Chat Room**  
**Sales**  
**Movies**  
**Points**

### Confirm ?

Movie name: The Shining

Starts at: 04:00

Theater: 1

Clic sobre el botón “Book” y obtendrá lo siguiente:



Intenta ingresar otras combinaciones, vuelva a iniciar el flujo y note como los valores del bean son consistentes durante todo el flujo.

Clic sobre el botón “home” para retornar a la página inicial.

## 10.0 Conclusión

Durante este taller se construyó una aplicación Web de 3 capas sencilla usando Java EE 7 y demostrando las siguientes nuevas funcionalidades de la plataforma:

- Java EE 7 Platform (JSR 342)
  - Maven coordinates
  - DataSource por defecto
  - JMSConnectionFactory por defecto
- Java Persistence API 2.1 (JSR 338)
  - Propiedades para la generación de Schemas
- Java API para RESTful Web Services 2.0 (JSR 339)
  - Client API
  - Entity Providers personalizados
- Java Message Service 2.0 (JSR 343)
  - ConnectionFactory por defecto
  - Inyección del JMSContext
  - Envío y recibo de mensajes sincrónicamente
- JavaServer Faces 2.2 (JSR 344)
  - Faces Flow
- Contexts and Dependency Injection 1.1 (JSR 346)
  - Descubrimiento automático de beans.

- Inyección de beans
- Bean Validation 1.1 (JSR 349)
  - Integración con Java Server Faces
- Batch Applications for the Java Platform 1.0 (JSR 352)
  - Procesamiento Chunk-style
  - Manejo de excepciones
- Java API for JSON Processing 1.0 (JSR 353)
  - Generación de JSON usando el API a modo de flujo
  - Consumo de JSON usando el API a modo de flujo
- Java API for WebSocket 1.0 (JSR 356)
  - Anotación server endpoint
  - Cliente JavaScript
- Java Transaction API 1.2 (JSR 907)
  - @Transactional

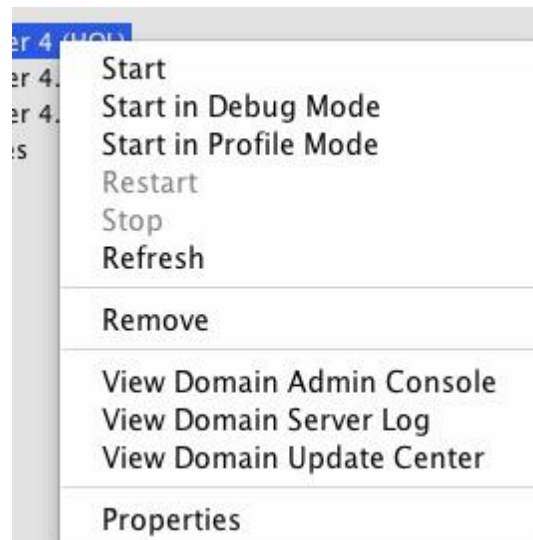
Con suerte, este taller ha elevado su interés para que continúe usando Java EE 7 y Glassfish 4.

Envíenos su retroalimentación a [users@glassfish.java.net](mailto:users@glassfish.java.net)

## 11.0 Solución de problemas

### 11.1 Como puedo iniciar/detener/reiniciar Glassfish desde el IDE?

En el tab “Services”, clic derecho sobre “Glassfish Server 4”. Todos los comandos para iniciar, detener o reiniciar se pueden apreciar en el menú emergente. El log del servidor puede ser visto accediendo a la opción “View Server Log” y la consola de administración web puede ser accedida por medio de la opción “View Admin Console”.





**11.2** Cerré por accidente la ventana del log del servidor, ¿Cómo la puedo restaurar?

En el tab “Services” de NetBeans, expanda “Servers”, seleccione el nodo Glassfish y “View Domain Server Log”.

## **12.0 Agradecimientos**

Los siguientes miembros de la comunidad Glassfish contribuyeron en la revisión de este taller:

- Antonio Goncalves (@agoncal)
- Markus Eisele (@myfear)
- Craig Sharpe (@dapugs)
- Marcus Vinicius Margarites (@mvfm)
- David Delabasse (@delabasse)
- John Clingan (@jclingan)
- Reza Rahman (@reza\_rahman)

Muchas gracias por proveer su valiosa retroalimentación.

## **13.0 Soluciones completas**

La solución completa se puede descargar de:

<http://glassfish.org/hol/movieplex7-solution.zip>

### **13.1 TODO**

1. Adicionar los siguientes casos de uso:
  - a. Utilidades de Concurrencia para Java EE
  - b. Cliente Java para WebSocket
  - c. Filtro para Logging en JAX-RS
2. Deshabilitar errores en persistence.xml
3. ¿Cómo sobrescribir .m2/repository en NetBeans?
4. Adicionar iconos: “Fix Imports”, “Format”, “Fix namespaces”, “Run the Project”.
5. Cambiar la forma de llevar el registro para que use java.util.Logging

## 13.2 Historial de revisión

1.
  - a. Paso de las fuentes del documento desde Microsoft Word a Páginas.
  - b. Adición de tiempos estimados para cada sección.
  - c. Actualizado para usar NetBeans 7.4 nightly.
  - d. Reorganizado para asegurarse que las nuevas características de Java EE 7 son mostradas primero.
2. Actualización del código desde UberConf.
3. Incorporación de solución a errores de escritura, falta de cuadros de dialogo, y optimización al código, recibidos durante DevovxUK.
4. Actualización al código luego de que se corrigieran bugs.
5. Uso de GlassFish 4 final, build (b89).

## Apéndice

### Apéndice A: Configurar Glassfish 4 en NetBeans IDE

A.1 En NetBeans, clic sobre el tab “Services”.

A.2 Clic derecho sobre “Servers”, seleccione “Add Server...” en el menú emergente.

A.3 Seleccione “Glassfish Server” en el wizard de adición de servidores, asigne el nombre “Glassfish 4.0” y clic en “Next >”.

A.4 Clic en “Browse...” y busque el directorio donde descomprimió o instaló Glassfish y seleccione el directorio “glassfish4”. Seleccione “Finish”.

