Informe de Laboratorio 1

José Miguel García[1]
Soledad García[1]
Myriam Delgado[1]

1. INTRODUCCION

Actualmente, se tiene la necesidad de realizar un sistema para un concesionario el cual gestione las ventas de sus autos.
El sistema debe permitir gestionar clientes, autos y ventas en donde estas últimas serían una relación entre un cliente determinado y un auto (ambos ya existentes).

Para cada crear cada cliente es necesario almacenar los siguientes datos:

- Identificación: Esta será la cédula o número de identificación del cliente.
- Nombre: Nombres completos del cliente.
- Apellidos: Apellidos completos del cliente.
- Edad: Edad en años del cliente.
- Dirección: Lugar de residencia del cliente.
- Teléfono: Teléfono del cliente

Para crear un auto que se desea vender es necesario almacenar los siguientes datos:

- Matricula: Placas del carro.
- Modelo: Año de creación del carro.
- Fabricante: Empresa o marca que elaboró el carro.
- Fecha: Fecha de ingreso del carro.
- Cilindraje: Cilindraje del motor del carro.
- Potencia: Caballos de fuerza del carro.
- Precio: Precio al cual se desea vender el carro.
- Transmisión: Indica si el carro es manual o automático.

Para crear una venta que asocia a un carro y un cliente es necesario almacenar los siguientes datos:

- Número de la venta: Identificación de la venta.
- Cliente: Identificación de un cliente existente.
- Carro: Matrícula de un carro existente.

Este documento detalla el sistema creado para satisfacer la necesidad descrita anteriormente el cual corresponde al Laboratorio 1 del curso de Profundización en la Arquitectura de Software, teniendo la siguiente estructura: En la **sesión 2** se detallan los objetivos que se desean cumplir con el desarrollo de dicho sistema, en la **sesión 3** se describen las herramientas empleadas en este desarrollo, en la **sesión 4** se ilustra la arquitectura empleada para este desarrollo, en la **sesión 5** se da explicación del desarrollo, anotaciones utilizadas y se ilustran algunos pantallazos del sistema, en la **sesión 6** se busca dar a conocer algunas mejoras que se desearían implementar en un futuro, en la **sesión 7** se especifican las conclusiones y finalmente, en la **sesión 8** se incluye la bibliografía.

2. OBJETIVOS

Objetivo General

Crear una aplicación Web que permita gestionar la información para un concesionario de vehículos por medio de operaciones CRUD sobre la base de datos.

Objetivos específicos

- ✓ Comprender el uso de los recursos JDBC y pool de conexiones para el manejo de las conexiones y concurrencia en una base de datos de un concesionario.
- ✓ Implementar las anotaciones JPA para simplificar la persistencia y el mapeo de una base de datos objeto-relacional de un concesionario.
- ✓ Aplicar el uso de EJB-Session Beans: Stateful, Stateless y Singleton para la seguridad de una aplicación transaccional.
- ✓ Utilizar Servlets para la lógica del negocio y JSP para la vista del concesionario.
- ✓ Implementar los objetos request y response que nos proporcionan los Servlets.
- ✓ Incluir JSP en la creación de una concesionario web.Aplicar una arquitectura básica de 3 o N capas para el desarrollo de una aplicación web.
- ✓ Aplicar el uso de los patrones de diseño DTO y DAO.

3. HERRAMIENTAS EMPLEADAS

Las herramientas utilizadas en este sistema son las siguientes:

- **Bootstrap**: Framework de CSS, HTML y JavaScript que permite dar forma a un sitio web.
- **MariaDB**: Base de datos para guardar la información del concesionario (es la versión nuevo de MySql). Se utilizó la versión 10.1.11.
- **NetBeans**: IDE seleccionado que facilitó el desarrollo del software. Se utilizó la versión 8.1.
- **GlassFish**: Servidor de aplicaciones para desplegar el sitio web. Se utilizó la versión 4.1.1.
- **Sequel Pro**: Programa para la gestión de bases de datos MySql.
- **Java EE**: Plataforma seleccionado para el desarrollo del software.
- **Java Servlets**: Servlets utilizados para responder a las solicitudes de la vista.
- **Enterprise JavaBeans**: Tecnología utilizada para crear y reutilizar los componentes DAO de Ventas, Carros y Clientes.
- **Web Browser**: Permite visualizar la interfaz web gráfica de la aplicación.
- **LucidChart**: Herramienta web para realizar diagramas UML.

4. ARQUITECTURA

La arquitectura de la aplicación se ha diseñado teniendo en cuenta 3 capas: Lógica de la Aplicación, Lógica del Negocio y Modelo, en la Figura 1 se detalla dicha arquitectura de una manera general, dado que en la Figura 2, se muestra la arquitectura de la aplicación con más detalle.

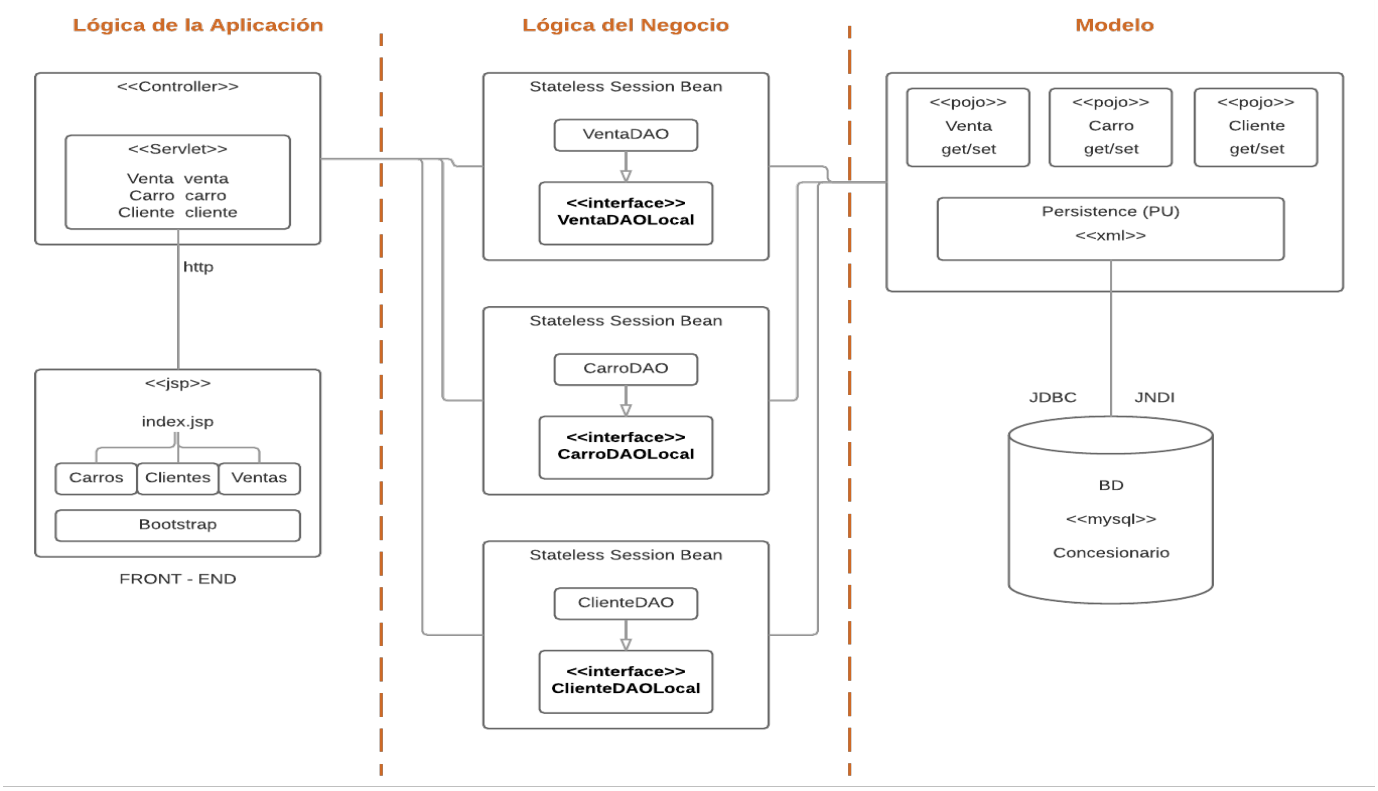

Figura 1 Arquitectura general de la aplicación

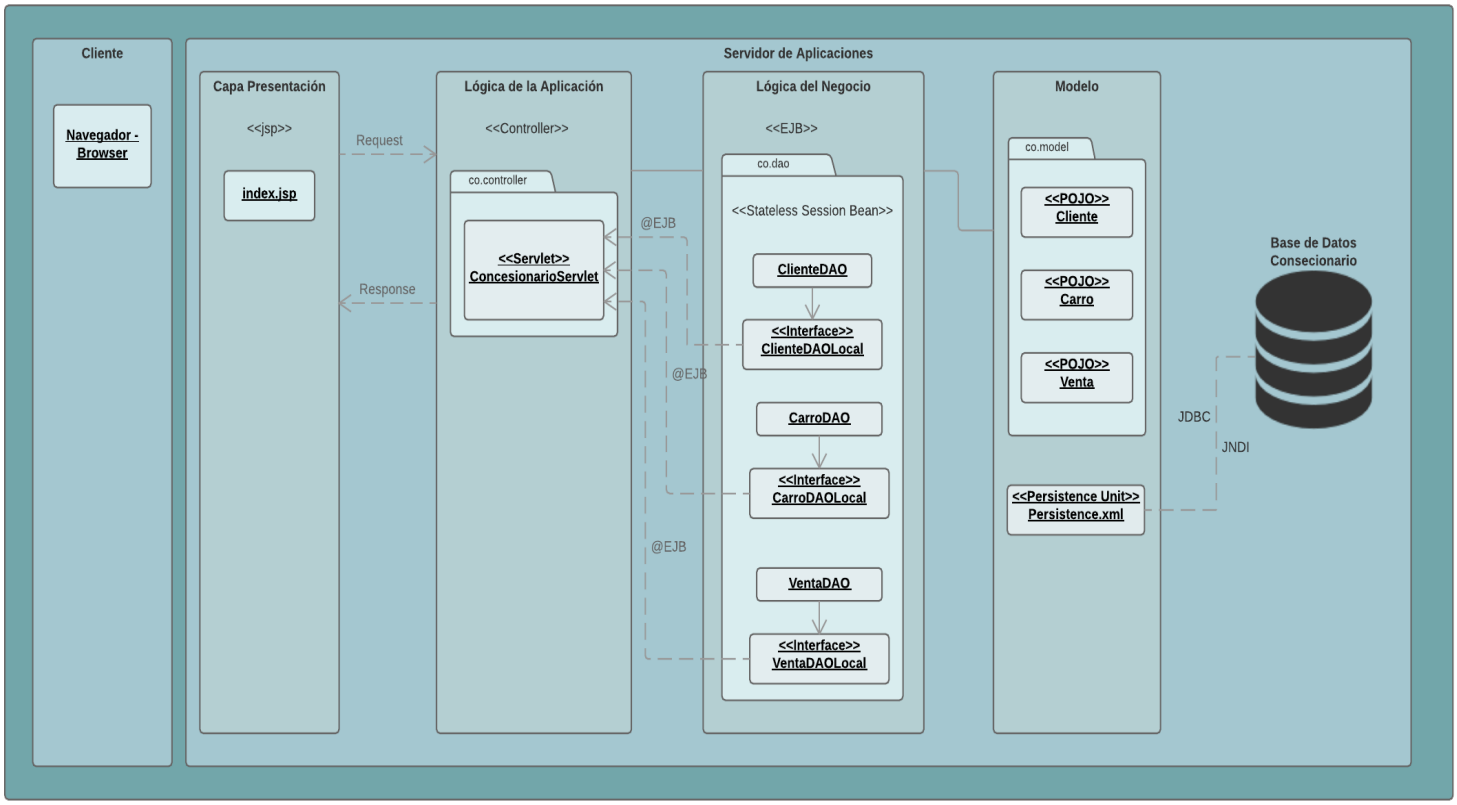

Figura 2 Arquitectura general de la aplicación

5. DESARROLLO

De acuerdo a las anteriores sesiones, creamos una simple aplicación tipo CRUD para gestionar las ventas de un concesionario; usamos el IDE NetBeans en la versión 8.1 con un servidor de base de datos 10.1.11-MariaDB Homebrew, la aplicación corre sobre un servidor de aplicaciones GlassFish Server Open Source Edition 4.1.1 (build 1).

Creamos la base de datos con la ayuda del IDE y nos aseguramos con SequelPro para gestionar las conexiones de forma más ágil y cómoda. Para el desarrollo del Modelo crearemos 3 POJOS (cliente, carro, venta) mapeados mediante una interfaz local de acceso a los datos, posteriormente se crean sus respectivos EJB. En el modelo se crea, también, la unidad de persistencia la cual persiste la base de datos conectándose a esta mediante JDBC (que maneja toda la conección de la base de datos) y JNDI (que maneja la ubicación de la base de datos).
Los pojos ya creados, con inyección de dependencias, se comunican a través de un solo controlador con una sola vista desarrollada con Java Server Pages (JSP) y componentes Bootstrap; dicha vista incluye 3 pestañas, una para cada módulo: Carros, Clientes y Ventas.
Adicional a lo anterior, se hizo uso de los patrones DAO, DTO y Fackade.

En el desarrollo de este sistema se utilizaron anotaciones para indicar el procesamiento de algunas clases, variables y métodos, a continuación se listan las anotaciones utilizadas:

@Entity
Significa que la clase será una entidad usada para realizar consultas en una base de datos

@NamedQueries(@NamedQuery(name="Nombre de la consulta ",query="consulta en SQL"))
Indica que existirá una consulta en la base de datos de nombre “name” y la consulta en lenguaje de persistencia de java estara contenido en “query”

@Table(name = "nombre de la tabla" ,schema = "dbo")
Con esta anotación podemos identificar la tabla en la base de datos a la cual hará referencia la clase, si nuestra tabla se llama diferente o tiene asignado un schema es necesario usar “name” para el nombre y “schema” para el Schema.

@Id
Con la anotación anterior indicamos que el campo definido será una clave primaria dentro de la base de datos.

@GeneratedValue(strategy = GenerationType.[AUTO|IDENTITY|SEQUENCE|TABLE])
Es usada en los proyectos que tienen claves primarias simples; define la forma de generación de la clave, si necesitamos descargar la responsabilidad en el proveedor de persistencia y la base de datos usamos AUTO, pero si necesitamos que el proveedor de persistencia tenga el control será IDENTITY si necesitamos consecutivos generados por la base de datos usamos SEQUENCE o si son valores que se aseguran por el proveedor de la base de datos usaremos TABLE

@Column(name="DESC", columnDefinition="CLOB NOT NULL", table="EMP_DETAIL")

La anotación anterior se usa para especificar la columna en la base de datos a la cual hara referencia el campo.

@Stateless
Define que el bean será de tipo Stateles sesión bean significa que no contendrá datos en mientras se realizan las diferentes peticiones, servirá solo por demanda.

@PersistenceContext(unitName = "unidad de persistencia")
Expresa la unidad de persistencia de la cual depende

@Override
Significa que se sobrescribe un método que hace parte de una clase de orden superior

@MultipartConfig
Esta anotación nos permite el uso de formularios con el enctype="multipart/form-data" requerido para el manejo de uploads en el formulario

@EJB
Indica la dependencia con el bean que se expresa a continuación

6. POSIBLES MEJORAS

Durante el desarrollo de este software se analizaron posibles mejoras que podrían ser implementadas en un futuro. A continuación, se listan dichas mejoras:

- Validaciones en todos los campos de la interfaz gráfica para así guardar los datos correctamente y notificarle al usuario en caso tal ingrese algún dato erróneo.
- El servidor de aplicaciones expone públicamente aún la dirección del host, lo cual es un problema de seguridad, es necesario asegurar la aplicación y hacer las respectivas modificaciones para esto.

7. CONCLUSIONES

- La arquitectura implementada es una arquitectura permite un mantenimiento y soporte más sencillos gracias a su flexibilidad y alta escalabilidad; lo que se traduce en mayores posibilidades de distribución y paralelismo. también, se facilita la reutilización de código.

- Las tecnologías de desarrollo y programación de software han evolucionado muy rápidamente y java se ha posicionado como una herramienta de desarrollo a nivel mundial. Es un cambio de paradigma en la programación y junto con la orientación a al uso de anotaciones en las aplicaciones, no sólo webs; es otro reto que una vez asumido genera mucha utilidad y proporciona una herramienta de trabajo muy útil para el desarrollo de aplicaciones.

- Las tecnologías modernas, el avance en la programación orientada a objetos y aspectos, y el crecimiento en las prestaciones de servicios unido al creciente auge de Internet hace que las aplicaciones actuales y futuras se orienten cada vez más a la web

y al mundo de la internet, por varias razones, pero las más interesantes son, la facilidad de uso, y necesidad de pocas prestaciones de hardware y de software para manipular las diferentes aplicaciones.

- El correcto uso de los patrones y anotaciones, en este caso en el lenguaje java, permite crear aplicaciones web rápidamente, ya que anteriormente hacer un CRUD requería tareas tediosas como incluir las sentencias SQL en las clases Java; ahora, implementando estos dos componentes en el desarrollo del concesionario web se ahorraron muchas líneas de código puesto que no se incluyeron dichas sentencias y otras tareas que se evitaron con dicha inclusión. Adicional a lo anterior, el código quedó mucho más claro y escalable.

- Aún en la parte visual de la aplicación, es decir la vista, podemos apreciar una débil interacción de usuario independiente de la tecnología Bootstrap. El formulario no tiene validadores y se muestra de una manera que únicamente satisface el alcance de la práctica, a pesar de esto hay que rescatar el uso de hojas de estilo.

8. BIBLIOGRAFIA

ISOFT(2009). JDBC: acceso a bases de datos. Recuperado el 10 de Marzo de 2015 a partir de: http://www.vc.ehu.es/jiwotvim/ISOFT2009-2010/Teoria/BloqueIV/JDBC.pdf

Oracle. Annotations Basics**.** Recuperado el 10 de Marzo de 2015 a partir de: https://docs.oracle.com/javase/tutorial/java/annotations/basics.html

Oracle. The Java EE 6 Tutorial. Recuperado el 10 de Marzo de 2015 a partir de: http://docs.oracle.com/javaee/6/tutorial/doc/gipjg.html

Bootstrap community. About Bootstrap. Recuperado el 10 de Marzo de 2015 a partir de: http://getbootstrap.com/about/

SequelPro. Portal de SequelPro. Recuperado el 10 de Marzo de 2015 a partir de: http://www.sequelpro.com/

Botia, D(2016). WebSite Profundización en Arquitecturas de Software. Recuperado el 10 de Marzo de 2015 a partir de: https://sites.google.com/site/profarquitecurasdesw