

Informe de Laboratorio 3

José Miguel García¹

Soledad García¹

Myriam Delgado¹

1. INTRODUCCIÓN

El desarrollo de aplicaciones web requiere que pueda implementarse de manera más ágil para satisfacer las necesidades de los clientes. El uso de los frameworks de desarrollo provee herramientas automatizadas que ayudan a crear software a la medida o rápidas demostraciones para los clientes y el desarrollador, en este informe logramos evidenciar cómo podemos lograr esas características en un desarrollo mediante el framework Java Server Faces.

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.

¹ Estudiante de Maestría en Ingeniería, Grupo Investigación ITOS. Universidad de Antioquia

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

2. OBJETIVOS

Objetivo General

Desarrollar aplicaciones CRUD teniendo en cuenta los frameworks de automatización, los conceptos y modelos dados en clase.

Objetivos específicos

- Implementar los idioms de la aplicación mediante la generación automática del código.
- Usar la aplicación JPA Modeler para generar el modelo del dominio de la aplicación
- Crear las paginas JSP desde las Entity Classes mediante la generación automática del código.
- Agregar nuevos componentes JSP a las vistas creadas teniendo en cuenta el código generado.
- Desplegar y probar la aplicación desde el cliente web.

3. HERRAMIENTAS EMPLEADAS

Las herramientas utilizadas en este sistema son las siguientes:

- **PrimeFaces**: Librería de Java Server Faces (JSF) que permite dar forma a la vista, en este caso se utilizó para el calendario del administrador de autos.
- **Derby**: Base de datos para guardar la información de la pasarela del concesionario. Se utilizó la versión 10.12.1.1.
- **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 el componente de pago el cual maneja las transacciones entre el estudiante y la universidad.
- **Web Browser:** Permite visualizar la interfaz web gráfica de la aplicación.
- **LucidChart:** Herramienta web para realizar diagramas UML.
- **JPA Modeler:** Herramienta empleada para generar el modelo del dominio de la aplicación.
- **Facelets:** Lenguaje declarativo para construir vistas en Java Server Faces.

4. ARQUITECTURA

La arquitectura de la aplicación se ha basado en el modelo de dominio ilustrado en la Figura 1, en donde se tienen 3 entidades: Auto, Venta y Cliente.

En este modelo la entidad Auto tiene asociada una única venta, dado que no es coherente que el mismo carro se venda 2 veces a diferentes dueños. Y, la entidad Cliente puede tener asociada una o varias ventas, puesto que una personas puede comprar uno o varios carros.

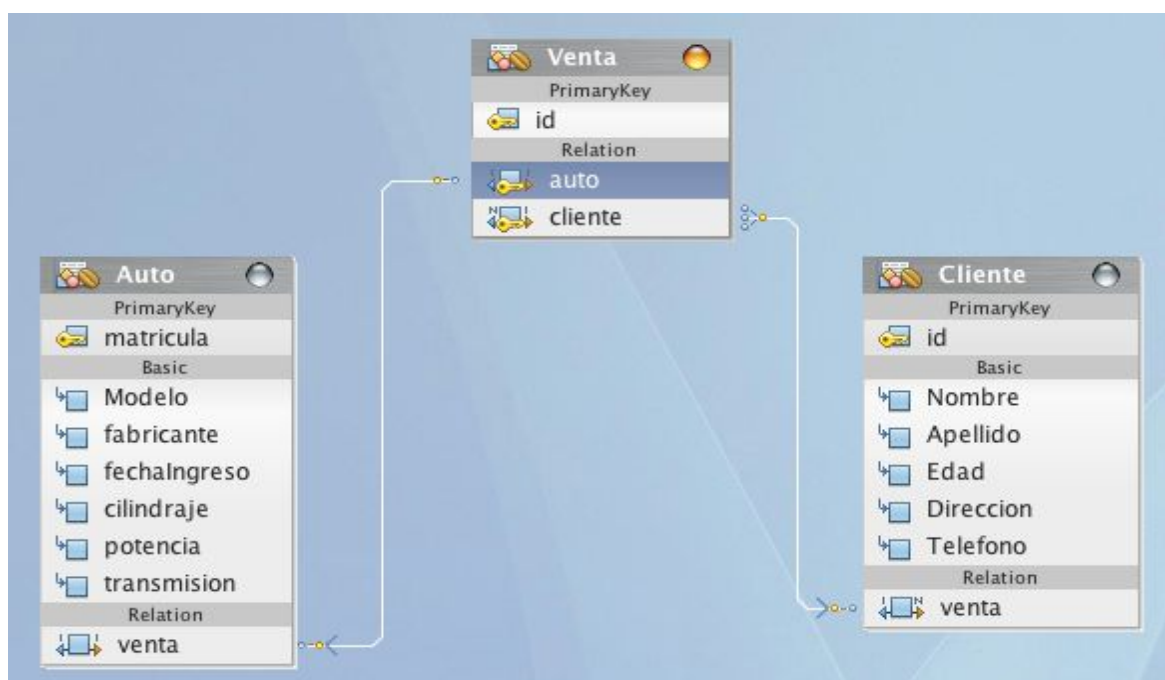


Figura 1. Modelo del dominio del sistema

En la Figura 2, se ilustra la arquitectura de la aplicación por paquetes de acuerdo a 3 capas: Lógica de la Aplicación, Lógica del Negocio y Modelo.

Esta arquitectura y su contenido fueron generados automáticamente de acuerdo al modelo de dominio ilustrado en la Figura 1.

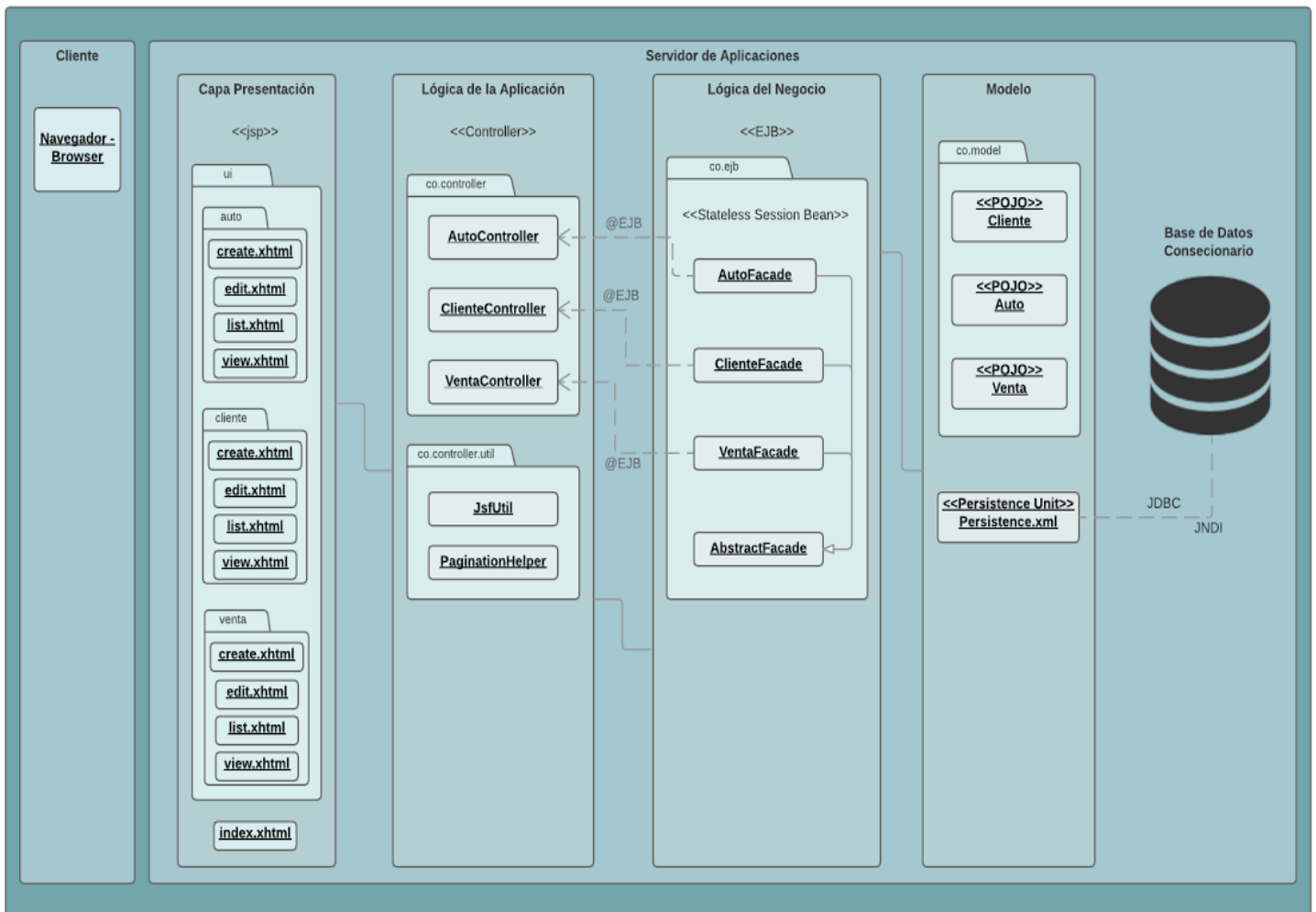


Figura 2. Arquitectura por capas del sistema

De acuerdo a la Figura 1, el **cliente o navegador** hace una petición http a la **Capa de Presentación**, en la cual se encuentran todos los archivos correspondientes a las vistas del sistema para Autos, Clientes y Ventas.

En esta capa, también, se encuentra el archivo faces-config el cual mapea por medio de beans los recursos que necesitan las páginas JSF para realizar las operaciones.

En la capa de la **Lógica de la Aplicación**, el paquete *co.controller* tiene las clases controlador para cada entidad, estas capturan los eventos de la capa de presentación. Esta capa también tiene el paquete *co.controller.util* el cual contiene los archivos *JsflUtil* y *PaginationHelper*, ambos utilizados por los controladores; el primer archivo tiene como fin la administración de los mensajes de error y la selección de los ítems de las listas de la vista, y el segundo archivo se encarga de la paginación de los elementos de la vista.

La capa de la **Lógica del Negocio** contiene el paquete *co.ejb* en donde los componentes de este paquete se encargan de la lógica del negocio, los elementos de este paquete están asociados con cada entidad (Auto, Cliente y Venta), y así suministrar esta información a la capa del **Modelo** para que esta a su vez la persista en la base de datos mediante una unidad de persistencia que se conecta a dicha base de datos utilizando JDBC y JNDI.

5. DESARROLLO

Este proyecto se caracteriza por su generación automática apoyada en los frameworks de NetBeans; como primer se hizo énfasis en la construcción del modelo del dominio usando JPA Modeler, es importante dedicar todo el tiempo que sea necesario a este paso para garantizar el éxito de los siguientes. Cuando se terminó el modelo de dominio creado, internamente se crea un descriptor que guarda en un archivo de propiedades lo que hemos modelado.

La creación de los POJOS parte de este diagrama, y se le agregan los IDIOMS toString(), equals() and hashCode(). Finalmente, creamos la vista y los *session bean* de forma automática, con esto se crearía rápidamente nuestro código CRUD.

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:

@Named("valor")

Permite acceder al bean mediante un nombre especificado en "Valor". En este caso el nombre del bean será "autocontroller" para el bean Auto, "clientecontroller" para el bean Cliente y "ventacontroller" para el bean Venta.

@EJB

Indica la dependencia con el bean que se expresa a continuación.

@FacesConverter(forClass = "clase".class)

En donde clase es el nombre de un Pojo (Auto, Cliente o Venta). Simplemente esta anotación es requerida siempre y cuando se necesite convertir de un objeto a String y de String a un objeto, muchas de estas conversiones son requeridas por Java Server Faces.

@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, en este caso unitName es igual tiene como valor "MatriculaEstudiantePU"

@Entity

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

@Column(name="nombre", nullable="verdadero o false")

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

@OneToMany(targetEntity = "Entidad1".class, mappedBy = "Entidad2")

Indica que hay una relación uno a muchos desde la Entidad1 a la Entidad2; en donde Entidad1 es la entidad que contiene la clave foránea de la Entidad2.

@Id

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

@Lob

Indica que el objeto anotado con esta propiedad puede persistir como Large Object en la base de datos; es decir se usa para mapear en la base de datos el objeto como tipo Lob, la base de datos debe permitir este tipo de datos.

@OneToOne(targetEntity = "Entidad1".class, mappedBy = "Entidad2")

Indica que hay una relación uno a uno desde la Entidad1 a la Entidad2; en donde Entidad1 es la entidad que contiene la clave foránea de la Entidad2.

@Basic

Indica que un atributo debe ser persistido y un mapeo estándar debe ser utilizado

Como parámetro utilizamos *optional* que permite indicar si el campo será nulo. Por defecto true. En este caso el *id* no debe ser nulo, por lo que se pone *false*.

@Override

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

Java Server Faces soporta las anotaciones para los beans, el `@ManagedBean` es una anotación que registra automáticamente la clase como un recurso en la implementación de JSF, con esto podemos omitir la configuración del Scope en los recursos de configuración en la aplicación.

Cuando queremos omitir la configuración del Scope para darle un poco de orden a nuestro archivo o para reducir la carga o las tareas de configuración podemos usar las anotaciones que se encuentran en la sección 8.

Se debe tener cuidado al realizar estas referencias de los objetos, porque una mala referenciación podría terminar el hilo padre que provee JSF a la aplicación.

Se debe tener cuidado en la navegación entre vistas si solo estamos persistiendo los datos con `@ViewScoped`. Otra configuración interesante para los Managed Beans son los *Eager*, normalmente un componente es Lazily, es decir, está esperando la interacción del usuario para ser instanciado, si usamos `@ManagedBean(eager=true)` los componentes en nuestra aplicación serán instanciados tan pronto se despliegue la aplicación

Ahora, en la Figura 3, se ilustra la página principal de la aplicación, la cual contiene un mensaje de bienvenida y 3 enlaces, cada uno para administrar cada entidad de esta aplicación: Autos, Clientes y Ventas.



Figura 3. Página principal o Inicio de la aplicación

La Figura 4 ilustra el administrador de autos cuando el usuario da click en el link “Administrar Autos” (Ver Figura 3). Este administrador muestra una lista de todos los autos creados en la base de datos y para cada uno de estos registros hay opciones de editar, ver y eliminar. También, este administrador contiene la opción de crear un nuevo auto o volver al Inicio.

Administrar Autos

1..1/1

FechaIngreso	Potencia	Cilindraje	Venta	Modelo	Transmision	Matricula	Fabricante	
04/19/0016	23000	1800	1017191426	2016	Automatica, Manual cambio 1,2	IYT596	Mercedes Benz	Ver Editar Eliminar

[Crear nuevo auto](#)

[Inicio](#)

Figura 4. Administrador de autos

La Figura 5 ilustra el formulario de creación de un nuevo auto, en donde se le solicita al usuario ingresar toda la información necesario para crear un auto. Finalmente el usuario da click en “Crear” para almacenar dicho auto en la base de datos. También, aquí se puede observar el calendario utilizado con la librería de Primefaces.

Crear nuevo auto

FechaIngreso: 

Potencia:

Cilindraje:

Venta: 

Modelo:

Transmision:

Matricula:

Fabricante:

[Crear](#)

[Listar autos](#)

[Inicio](#)

Figura 5. Página para crear un auto

La Figura 6 ilustra la página de Visualizar un auto, aquí el usuario final puede observar toda la información de dicho auto.

Ver Auto

FechaIngreso: 04/19/0016

Potencia: 23000

Cilindraje: 1800

Venta: 1017191426

Modelo: 2016

Transmision: Automatica, Manual cambio 1,2

Matricula: IYT596

Fabricante: Mercedes Benz

[Eliminar](#)

[Editar](#)

[Crear Nuevo auto](#)


[Listar todos los autos](#)

[Inicio](#)

Figura 6. Página para ver un auto

La Figura 7 ilustra la página de editar un auto, aquí el usuario final puede editar toda la información de dicho auto.

Editar Auto

FechaIngreso: 

Potencia:

Cilindraje:

Venta:

Modelo:

Transmision:

Matricula:

Fabricante:

[Guardar](#)

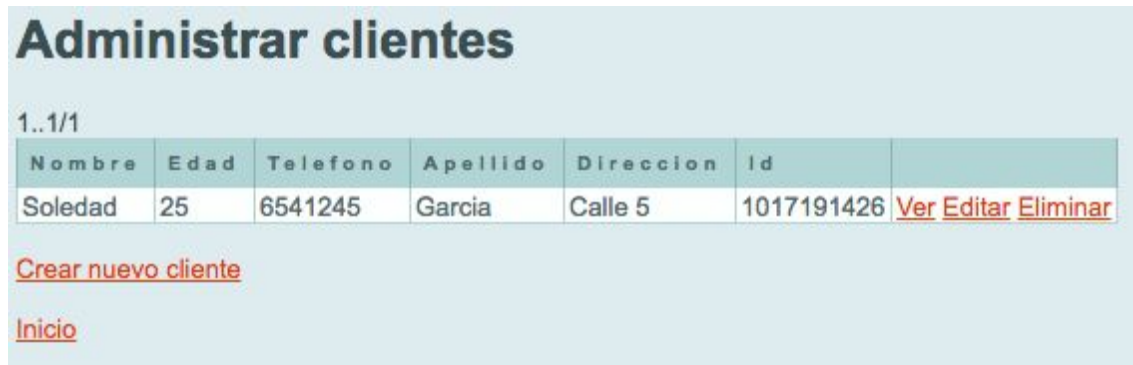
[Ver](#)

[Listar todos los autos](#)

[Inicio](#)

Figura 7. Página para editar un auto

La Figura 8 ilustra el administrador de autos cuando el usuario da click en el link “Administrar Clientes” (Ver Figura 3). Este administrador muestra una lista de todos los clientes creados en la base de datos y para cada uno de estos registros hay opciones de editar, ver y eliminar. También, este administrador contiene la opción de crear un nuevo cliente o volver al Inicio.



Nombre	Edad	Telefono	Apellido	Direccion	Id	
Soledad	25	6541245	Garcia	Calle 5	1017191426	Ver Editar Eliminar

[Crear nuevo cliente](#)

[Inicio](#)

Figura 8. Administrador de clientes

La Figura 9 ilustra el formulario de creación de un nuevo cliente, en donde se le solicita al usuario ingresar toda la información necesario para crear un cliente. Finalmente el usuario da click en “Guardar” para almacenar dicho cliente en la base de datos.



Crear Cliente

Nombre:

Edad:

Telefono:

Apellido:

Direccion:

Id:

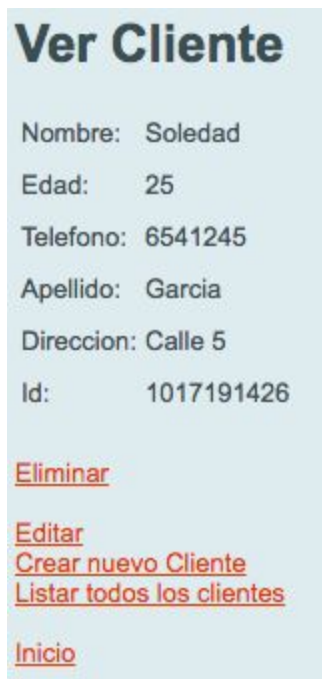
[Guardar](#)

[Listar todos los clientes](#)

[Inicio](#)

Figura 9. Página para crear un cliente

La Figura 10 ilustra la página de Visualizar un cliente, aquí el usuario final puede observar toda la información de dicho cliente.



Ver Cliente

Nombre: Soledad
Edad: 25
Telefono: 6541245
Apellido: Garcia
Direccion: Calle 5
Id: 1017191426

[Eliminar](#)
[Editar](#)
[Crear nuevo Cliente](#)
[Listar todos los clientes](#)
[Inicio](#)

Figura 10. Página para ver un cliente

La Figura 11 ilustra la página de editar un cliente, aquí el usuario final puede editar toda la información de dicho cliente.



Editar Cliente

Nombre:
Edad:
Telefono:
Apellido:
Direccion:
Id:

[Guardar](#)
[Ver](#)
[Listar todos los clientes](#)
[Inicio](#)

Figura 11. Página para editar un cliente

La Figura 12 ilustra el administrador de ventas cuando el usuario da click en el link “Administrar Ventas” (Ver Figura 3). Este administrador muestra una lista de todos las ventas creados en la base de datos y para cada uno de estos registros hay opciones de editar, ver y eliminar. También, este administrador contiene la opción de crear un nuevo cliente o volver al Inicio.



Figura 12. Administrador de ventas

La Figura 13 ilustra el formulario de creación de una nueva venta, en donde se le solicita al usuario ingresar toda la información necesario para crear una venta. Finalmente el usuario da click en “Guardar” para almacenar dicha venta en la base de datos.



Figura 13. Página para crear una venta

La Figura 14 ilustra la página de Visualizar una venta, aquí el usuario final puede observar toda la información de dicha venta.



Figura 14. Página para ver una venta

La Figura 15 ilustra la página de editar una venta, aquí el usuario final puede editar toda la información de dicha venta.



Figura 15. Página para editar una venta

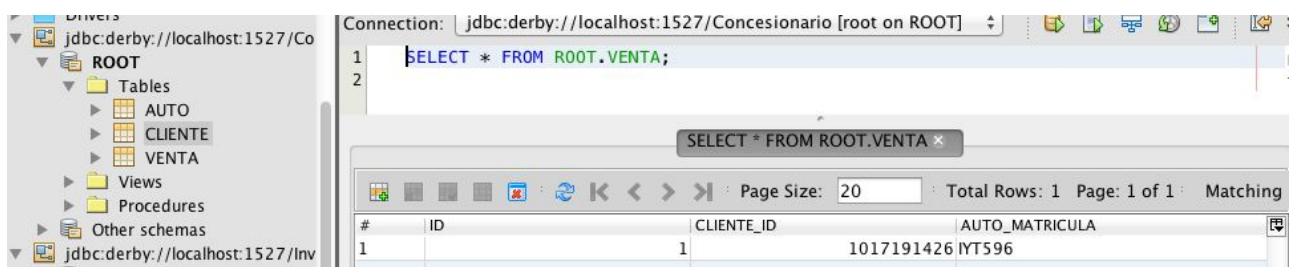
En la Figura 16, se verifica que la tabla Cliente de la base de datos Concesionario tiene sus respectivos datos creados desde el administrador de clientes en la vista.



#	ID	APELLIDO	DIRECCION	EDAD	NOMBRE	TELEFONO
1	1017191426	Garcia	Calle 5	25	Soledad	6541245

Figura 16. Datos de la tabla Cliente

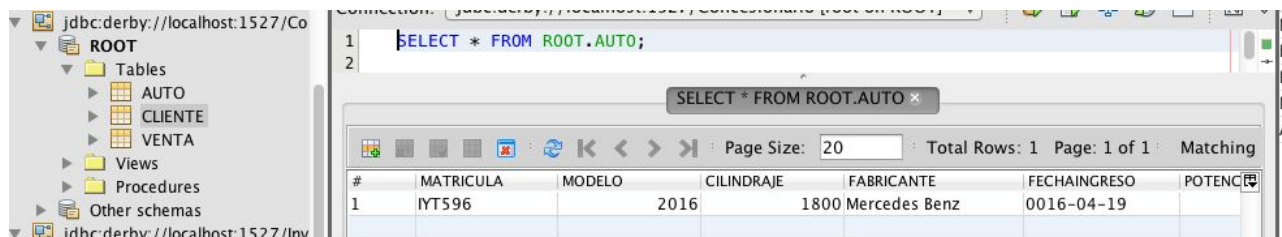
En la Figura 17, se verifica que la tabla Venta de la base de datos Concesionario tiene sus respectivos datos creados desde el administrador de ventas en la vista.



#	ID	CLIENTE_ID	AUTO_MATRICULA
1		1	1017191426 IYT596

Figura 17. Datos de la tabla Venta

En la Figura 18, se verifica que la tabla Auto de la base de datos Concesionario tiene sus respectivos datos creados desde el administrador de autos en la vista.



The screenshot shows a database interface with a tree view on the left containing 'ROOT', 'Tables' (with sub-items 'AUTO', 'CLIENTE', 'VENTA'), 'Views', 'Procedures', and 'Other schemas'. The main window displays a SQL query: 'SELECT * FROM ROOT.AUTO;'. Below the query, a table titled 'SELECT * FROM ROOT.AUTO' is shown with the following data:

#	MATRICULA	MODELO	CILINDRAJE	FABRICANTE	FECHAINGRESO	POTENCIA
1	IYT596	2016	1800	Mercedes Benz	0016-04-19	

Figura 18. Datos de la tabla Venta

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:

- Exponer esta pasarela a futuras implementaciones, esto podría ser mediante un servicio web o la posibilidad de agregarse como widget o iFrame y así otras aplicaciones que en un futuro se desarrollen y se necesite realizar un pago, podría utilizarse este desarrollo. Por supuesto, para lograr esto se debe estandarizar más esta aplicación para que se pueda integrar en diferentes contextos.
- Personalizar la pasarela de pagos y adaptarla a todos los entornos y dispositivos garantizando la usabilidad que mejora la experiencia del usuario.
- Mostrar diferentes idiomas.
- Ofrecer pagos alternativos para aumentar la probabilidad de compra.

7. CONCLUSIONES

- En muchas aplicaciones web se crean continuamente acciones y plantillas que realizan las operaciones CRUD para una determinada tabla de datos. El modelo de base de datos contiene la información necesaria para poder generar de forma automática el código de las operaciones CRUD, de forma que se simplifica el desarrollo inicial de la aplicación y las interfaces que deben ser creadas para la gestión de las aplicaciones.
- El desarrollo de metodologías que no dependen de la infraestructura tecnológica para construir modelos basados en el dominio, permiten garantizar la calidad y al mismo tiempo aumentar la productividad de las organizaciones desarrolladoras de software. De esta manera, los analistas y diseñadores pueden concentrarse en la lógica del negocio o modelo del dominio y utilizar un lenguaje universal, posiblemente basado en UML para desarrollar sus modelos.
- Los modelos independientes de la plataforma de desarrollo permiten desarrollar herramientas y/o procesos que agilizan la construcción de sistemas de software de

calidad. Entonces comprender los detalles esenciales del dominio cobra más relevancia que conocer las especificaciones del sistema.

- En la construcción de los sistemas también deben modelarse los aspectos tecnológicos separadamente del modelo del dominio para tener mayor flexibilidad cuando se produzcan cambios tecnológicos; no es necesario analizar nuevamente el dominio del problema, ahorrando tiempo valioso en la mantenibilidad de los sistemas.

8. CONSULTA: SCOPE MANAGED BEAN

Se pueden utilizar anotaciones para definir el ámbito en el que se almacenará el bean. Puede especificarse uno de los siguientes ámbitos para una clase de bean

@ApplicationScoped: El ámbito de la aplicación persiste a través de las interacciones de todos los usuarios con una aplicación web.

@SessionScoped: La sesión persiste a través de múltiples peticiones HTTP en una aplicación web.

@ViewScoped: La vista persiste durante la interacción del usuario con una sola página de una aplicación web.

@RequestScoped: La solicitud persiste durante una sola solicitud HTTP en una aplicación web.

@NoneScoped: indica un ámbito que no está definido para usarlo

@Depend: Significa que un objeto existe para servir exactamente a un cliente (bean) y tiene el mismo ciclo de vida como el cliente (bean).

@ConversationScoped: La interacción de un usuario con una aplicación Java Server Faces, dentro de los límites explícitos del desarrollo amplían el alcance a través de múltiples invocaciones del ciclo de vida de Java Server Faces. En la ejecución las conversaciones están limitadas a una sesión HTTP y no pueden cruzar los límites de la sesión.

(@CustomScoped): Un ámbito definido por el usuario, no estándar. Su valor debe configurarse como un `java.util.Map`. Ámbitos personalizados que se utilizan con poca frecuencia.

Se puede usar **@NoneScoped** cuando un *managed bean* hace referencia a otro managed bean. El segundo bean no debe estar en un ámbito (**@NoneScoped**) si se supone que se crea sólo cuando se hace una referencia. Si se define un bean como **@NoneScoped** se crea una instancia nuevamente cada vez que se hace una referencia.

9. BIBLIOGRAFIA

Botia, D(2016). WebSite Profundización en Arquitecturas de Software. Recuperado el 19 de Marzo de 2015 a partir de: <https://sites.google.com/site/profarquitecturasdesw>

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 19 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 19 de Marzo de 2015 a partir de:
<http://docs.oracle.com/javaee/6/tutorial/doc/gipjg.html>

Bootstrap community. About Bootstrap. Recuperado el 19 de Marzo de 2015 a partir de:
<http://getbootstrap.com/about/>

SequelPro. Portal de SequelPro. Recuperado el 19 de Marzo de 2015 a partir de:
<http://www.sequelpro.com/>

Prime Faces. Why PrimeFaces. Recuperado el 19 de Marzo de 2015 a partir de:
<http://www.primefaces.org/whyprimefaces>

The Java EE 6 Tutorial. Recuperado el 22 de abril de 2016 a partir de:
<https://docs.oracle.com/javaee/6/tutorial/doc/gjbbk.html>