Universidad Politécnica Salesiana

Carrera de Computación

Materia:

Sistemas distribuidos

Practica:

Practica EJB

Docente:

Ing. Cristian Timbi

Integrantes:

Juan Francisco Quizhpi Fajardo

EJB

EJB es una tecnología de componentes que por lo general se usa para el desarrollo de aplicaciones empresariales. En esencia los EJB son componentes reutilizables que se ejecutan en un servidor de aplicaciones y pueden ser consumidos por clientes. Si hablamos de benéficos, gracias a que el servidor de aplicaciones se encarga de la seguridad, transacciones y concurrencia el desarrollador ya no requiere de la implementación de estos servicios manualmente.

Cliente EJB

En general cliente se considera a cualquier componente o aplicación que se ejecuta en el servidor de aplicaciones, servidor remoto o incluso en un dispositivo cliente. El cliente consume los servicios que proporciona el EJB a través interfaces específicas que proporcionan acceso a esos servicios.

Desarrollo del proyecto

Para esta práctica del uso del EJB implementaremos un registro de automóviles. Para esto en primer lugar crearemos la creación del modelo Carro que contara con sus atributos, getter, setter y su método toString ().

```
import java.io.Serializable;

@Entity
public class Carro implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 1L;


    @Id
    @Column(name="car_placa")
    private String placa;
    @Column(name="car_modelo")
    private String modelo;
    @Column(name="car_marca")
    private String marca;
```

```java
public String getPlaca() {
    return placa;
}
public void setPlaca(String placa) {
    this.placa = placa;
}
public String getModelo() {
    return modelo;
}
public void setModelo(String modelo) {
    this.modelo = modelo;
}
public String getMarca() {
    return marca;
}
public void setMarca(String marca) {
    this.marca = marca;
}
@Override
public String toString() {
    return "Carro [placa=" + placa + ", modelo=" + modelo + ", marca=" + marca + "]";
}
```

Como segundo creamos el DAO para la clase Carro en el que se diseñaran cada uno de los métodos del CRUD, y método para buscar un automóvil por placa.

```java
package ec.edu.ups.pweb.dao;


import java.util.List;

@Stateless
public class CarroDAO{


    @PersistenceContext
    private EntityManager em;

    public void insert(Carro carro) {
        em.persist(carro);
    }

    public void update(Carro carro){
        em.merge(carro);
    }
    public Carro read(String placa) {
        Carro p = em.find(Carro.class, placa);
        return p;
    }

    public void delete(String placa) {
        Carro p = em.find(Carro.class, placa);
        em.remove(p);
    }
```

```
    public List<Carro> getList(){
        String jsql = "SELECT p FROM Carro p";
        Query query = em.createQuery(jsql,Carro.class);
        List<Carro> lista = query.getResultList();
        return lista;
    }

    public Carro getCarroPorPlaca(String placa){
        String jpql = "SELECT c FROM Carro c WHERE c.placa = :placa";
        Query q = em.createQuery(jpql, Carro.class);
        q.setParameter("placa", placa);
        List<Carro> carros = q.getResultList();
        if(carros.size()>0)
            return carros.get(0);
        return null;
    }
```

Ahora procedemos a implementar las interfaces Remoto y Local, estas contendrán cada uno de los métodos CRUD que serán implementados en la clase GestiónCarros.

```
package ec.edu.ups.pweb.business;

import java.util.List;

@Remote
public interface GestionCarrosRemoto {

    public void guardarCarros(Carro carro);
    public void actualizarCarro(Carro carro) throws Exception;
    public Carro getCarroPorPlaca(String placa) throws Exception;
    public void borrarCarro(String placa);
    public List<Carro> getCarros();
}
```

```
package ec.edu.ups.pweb.business;

import java.util.List;

@Local
public interface GestionCarrosLocal {

    public void guardarCarros(Carro carro);
    public void actualizarCarro(Carro carro) throws Exception;
    public Carro getCarroPorPlaca(String placa) throws Exception;
    public void borrarCarro(String placa);
    public List<Carro> getCarros();
}
```

Ahora en la clase GestiónCarros implementamos la clase gestión carros remoto y local esto nos llevara a implementar todos los métodos abstractos o CRUD para la clase carro. En esencia en esta parte se diseñará toda la lógica o el cómo se llevará a cabo el registro de carros para esta aplicación.

```java
package ec.edu.ups.pweb.business;

import java.util.List;

@Stateless
public class GestionCarros implements GestionCarrosLocal, GestionCarrosRemoto{

    @Inject
    private CarroDAO daoCarro;

    @Override
    public void guardarCarros(Carro carro) {
        // TODO Auto-generated method stub
        Carro car = daoCarro.read(carro.getPlaca());
        if (car != null){
            daoCarro.update(carro);
        }else {
            daoCarro.insert(carro);
        }

    }
```

```java
@Override
public void actualizarCarro(Carro carro) throws Exception {
    // TODO Auto-generated method stub
    Carro car = daoCarro.read(carro.getPlaca());
    if (car != null){
        daoCarro.update(carro);
    }else {
        throw new Exception("Carro no existe");
    }
}

@Override
public Carro getCarroPorPlaca(String placa) throws Exception {
    // TODO Auto-generated method stub
    Carro car = daoCarro.read(placa);
    if(car != null) {
        return daoCarro.getCarroPorPlaca(placa);
    }else {
        throw new Exception("Carro no existe");
    }
}
```

```java
@Override
public void borrarCarro(String placa) {
    // TODO Auto-generated method stub
    daoCarro.delete(placa);
}

@Override
public List<Carro> getCarros() {
    // TODO Auto-generated method stub
    return daoCarro.getList();
}
```

Luego diseñarán los servicios que se podrán probar desde postman y que se podrán implementar en el FrontEnd de nuestra aplicación distribuida.

```java
package ec.edu.ups.pweb.services;



import java.util.List;

@Path("carros")
public class CarroServices {

    @Inject
    private GestionCarrosLocal gCarros;

    @POST
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    public Response crear(Carro carro) {
        try{
            gCarros.guardarCarros(carro);
            ErrorMessage error = new ErrorMessage(1, "OK");
            return Response.status(Response.Status.CREATED)
                    .entity(error)
                    .build();
        }catch (Exception e) {
            // TODO: handle exception
            ErrorMessage error = new ErrorMessage(99, e.getMessage());
            return Response.status(Response.Status.INTERNAL_SERVER_ERROR)
                    .entity(error)
                    .build();
        }
    }
```

```java
    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    public Response actualizar(Carro carro) {
        try{
            gCarros.actualizarCarro(carro);
            return Response.ok(carro).build();
        }catch (Exception e) {
            // TODO: handle exception
            ErrorMessage error = new ErrorMessage(99, e.getMessage());
            return Response.status(Response.Status.NOT_FOUND)
                    .entity(error)
                    .build();
        }
    }

    @DELETE
    @Produces(MediaType.APPLICATION_JSON)
    public String borrar(@QueryParam("placa") String placa) {
        try{
            gCarros.borrarCarro(placa);
            return "OK";
        }catch (Exception e) {
            // TODO: handle exception
            return "Error";
        }
    }
```

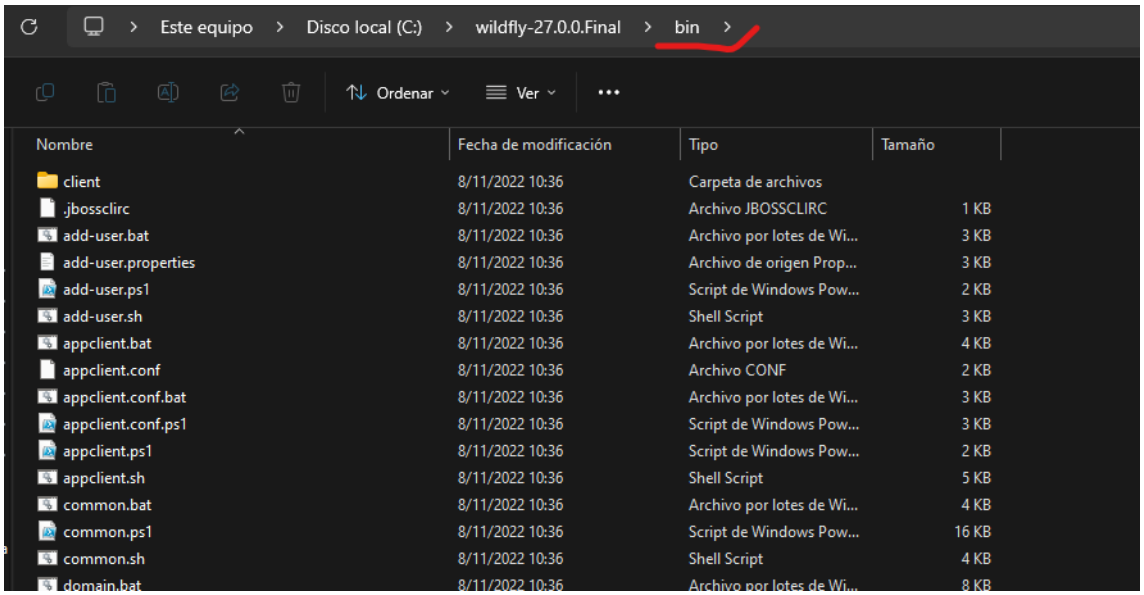
```java
@GET
@Produces(MediaType.APPLICATION_JSON)
public Response leer(@QueryParam("placa") String placa, @QueryParam("modelo") String modelo) {
    try{
        System.out.println("placa " +  placa + " modelo=" + modelo);
        Carro car = gCarros.getCarroPorPlaca(placa);
        return Response.ok(car).build();
    }catch (Exception e) {
        // TODO: handle exception
        ErrorMessage error = new ErrorMessage(4, "Carro no existe");
        return Response.status(Response.Status.NOT_FOUND)
                .entity(error)
                .build();
    }
}
```

```java
@GET
@Path("{placa}/{modelo}")
@Produces(MediaType.APPLICATION_JSON)
public Response leer2(@PathParam("placa") String placa, @PathParam("modelo") String modelo) {
    try{
        System.out.println("placa " +  placa + " modelo=" + modelo);
        Carro car = gCarros.getCarroPorPlaca(placa);
        return Response.ok(car).build();
    }catch (Exception e) {
        // TODO: handle exception
        ErrorMessage error = new ErrorMessage(4, "Carro no existe");
        return Response.status(Response.Status.NOT_FOUND)
                .entity(error)
                .build();
    }
}
```

```java
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("list")
public Response getCarros(){
    List<Carro> carros = gCarros.getCarros();
    if(carros.size()>0)
        return Response.ok(carros).build();

    ErrorMessage error = new ErrorMessage(6, "No se registran carros");
    return Response.status(Response.Status.NOT_FOUND)
            .entity(error)
            .build();

}
```

En general ya tenemos el servidor listo para que el cliente consuma sus servicios, antes de desarrollar al cliente primero deberemos crear un cliente para esto seguiremos los siguientes pasos

1. Ingresaremos a nuestra carpeta origen del nuestro servidor de aplicación Wildfly y ya dentro ingresaremos a la carpeta bin.



2. Luego abriremos un terminal y con el comando ls buscaremos que el archivo add-user.bat se encuentre en esta carpeta.

```
PS C:\wildfly-27.0.0.Final\bin> ls


    Directorio: C:\wildfly-27.0.0.Final\bin


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         8/11/2022     10:36                client
-a----         8/11/2022     10:36            477 .jbossclirc
-a----         8/11/2022     10:36           2459 add-user.bat
-a----         8/11/2022     10:36           2444 add-user.properties
-a----         8/11/2022     10:36           1095 add-user.ps1
-a----         8/11/2022     10:36           2392 add-user.sh
-a----         8/11/2022     10:36           4011 appclient.bat
-a----         8/11/2022     10:36           1841 appclient.conf
-a----         8/11/2022     10:36           2464 appclient.conf.bat
-a----         8/11/2022     10:36           2536 appclient.conf.ps1
-a----         8/11/2022     10:36           1175 appclient.ps1
-a----         8/11/2022     10:36           4621 appclient.sh
-a----         8/11/2022     10:36           3651 common.bat
-a----         8/11/2022     10:36          16357 common.ps1
-a----         8/11/2022     10:36           3577 common.sh
-a----         8/11/2022     10:36           7684 domain.bat
```

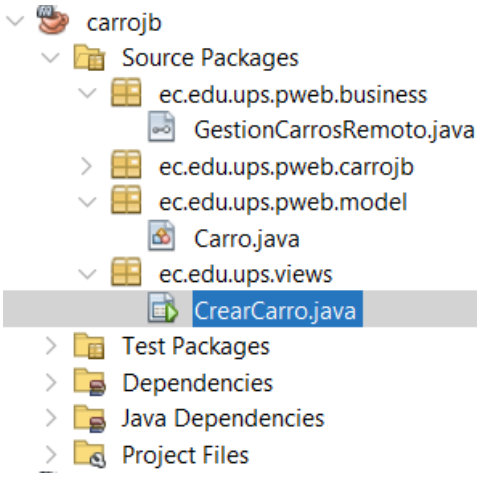
3. Luego con el siguiente comando “./add-user.bat” ingresaremos al proceso de creación del usuario nuevo como primer paso seleccionaremos la opción b.

```
PS C:\wildfly-27.0.0.Final\bin> ./add-user.bat
JAVA_HOME is not set. Unexpected results may occur.
Set JAVA_HOME to the directory of your local JDK to avoid this message.

What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): b
```

4. Luego ya ingresaremos las credencias de nuestro usuario nuevo en esta ocasión usaremos como usuario y clave “ejb63”.

```
PS C:\wildfly-27.0.0.Final\bin> ./add-user.bat
JAVA_HOME is not set. Unexpected results may occur.
Set JAVA_HOME to the directory of your local JDK to avoid this message.

What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): b

Enter the details of the new user to add.
Using realm 'ApplicationRealm' as discovered from the existing property files.
Username : ejb63
Password recommendations are listed below. To modify these restrictions edit th
 - The password should be different from the username
 - The password should not be one of the following restricted values {root, adm
 - The password should contain at least 8 characters, 1 alphabetic character(s)
)
Password :
```

Terminado el proceso de creación de usuarios procedemos a crear el proyecto cliente en NetBeans.



Como primera observación se destaca que los paquetes de business y modelo deben tener el miso nombre de nuestro servidor. Luego procedemos a la creación de nuestra interfaz Gestión Carros Remoto la cual tendrá todos los métodos CRUD que se van a consumir por el Cliente.

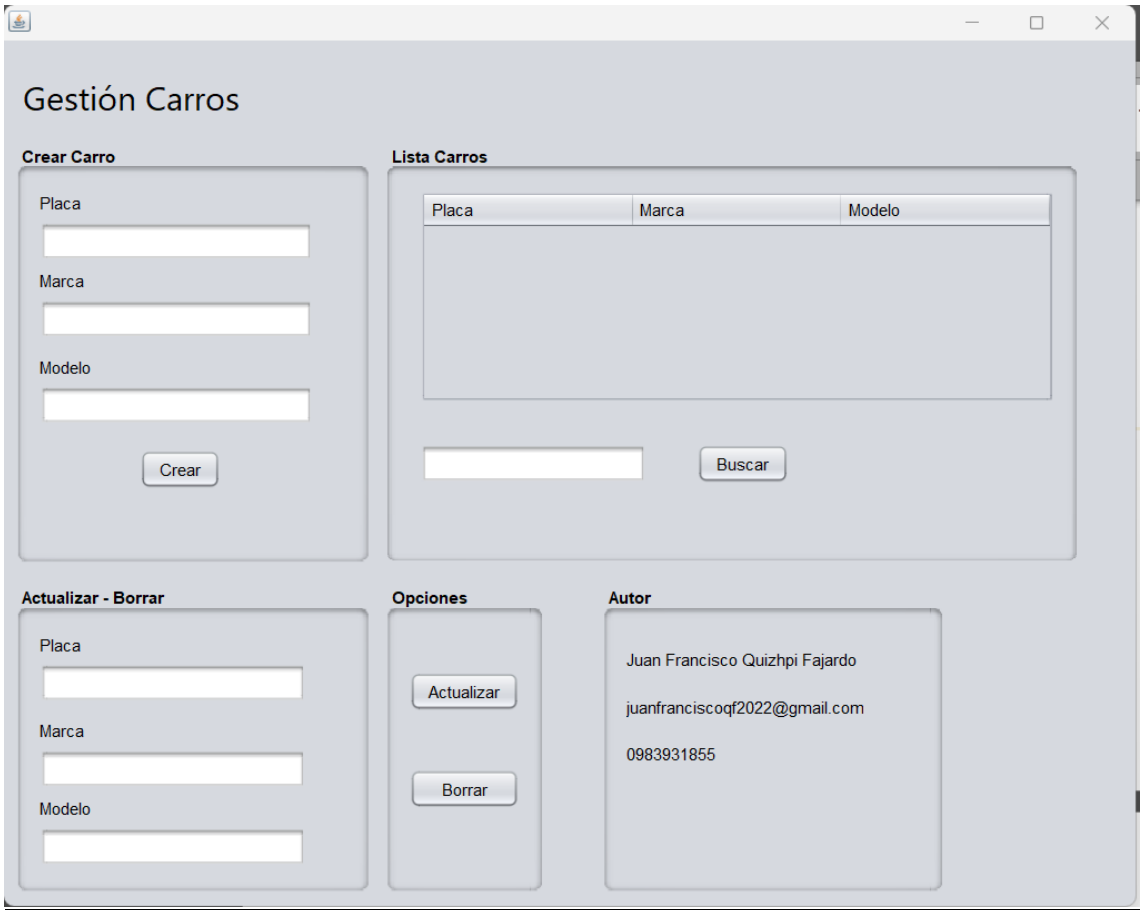
```java
package ec.edu.ups.pweb.business;

import ec.edu.ups.pweb.model.Carro;
import java.util.List;

/**
 *
 * @author juanf
 */
public interface GestionCarrosRemoto {
    public void guardarCarros(Carro carro);
        public void actualizarCarro(Carro carro) throws Exception;
        public Carro getCarroPorPlaca(String placa) throws Exception;
        public void borrarCarro(String placa);
        public List<Carro> getCarros();
}
```

También crearemos la clase Carro que contendrá los atributos, getter, setter y toString (). Esta clase implementara de Serializable.

```java
package ec.edu.ups.pweb.model;

import java.io.Serializable;

/**
 *
 * @author juanf
 */
public class Carro implements Serializable {

    private static final long serialVersionUID = 1L;

    private String placa;
    private String modelo;
    private String marca;

    public String getPlaca() {
        return placa;
    }

    public void setPlaca(String placa) {
        this.placa = placa;
    }

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }
```

```java
public String getMarca() {
    return marca;
}

public void setMarca(String marca) {
    this.marca = marca;
}

@Override
public String toString() {
    return "Carro{" + "placa=" + placa + ", modelo=" + modelo + ", marca=" + marca + '}';
}
```

Ahora procedemos a crear la interfaz gráfica para consumir los servicios



En la parte de programación se reutilizará código en especial la siguiente parte la cual únicamente cambiará en función de parámetros como el cliente que creamos antes, en donde se ejecuta el servidor y la conexión con el servidor la cual la obtendremos al momento de la ejecución del servidor.

```
try {
    final Hashtable<String, String> jndiProperties = new Hashtable<>();
    jndiProperties.put(key:Context.INITIAL_CONTEXT_FACTORY, value: "org.wildfly.naming.client.WildFlyInitialContextFactory");
    jndiProperties.put(key:Context.PROVIDER_URL, value: "http-remoting://localhost:8080");
    jndiProperties.put(key:Context.SECURITY_PRINCIPAL, value:"ejb64"); // Reemplaza 'username' con tu nombre de usuario
    jndiProperties.put(key:Context.SECURITY_CREDENTIALS, value:"ejb64"); // Reemplaza 'password' con tu contraseña
    jndiProperties.put(key:"jboss.naming.client.ejb.context", value: "true");

    final Context context = new InitialContext(environment:jndiProperties);
    GestionCarrosRemoto gestionCarros = (GestionCarrosRemoto) context.lookup(name:"ejb:/demojq/GestionCarros!ec.edu.ups.pweb.business.GestionCarrosRemoto");

    // Uso del EJB
    Carro carro = new Carro();
    carro.setPlaca(placa: txtPlacaCarro.getText());
    carro.setModelo(modelo: txtModeloCarro.getText());
    carro.setMarca(marca: txtMarcaCarro.getText());
    gestionCarros.guardarCarros(carro);
    System.out.println(x: "Carro guardado!");
    JOptionPane.showMessageDialog(parentComponent:null, message:"Creación Exitosa");
    limpiarCrear();
} catch (Exception e) {
    e.printStackTrace();
}
```

```
java:global/demojq/GestionCarros!ec.edu.ups.pweb.business.GestionCarrosRemoto
java:app/demojq/GestionCarros!ec.edu.ups.pweb.business.GestionCarrosRemoto
java:module/GestionCarros!ec.edu.ups.pweb.business.GestionCarrosRemoto
java:jboss/exported/demojq/GestionCarros!ec.edu.ups.pweb.business.GestionCarrosRemoto
ejb:/demojq/GestionCarros!ec.edu.ups.pweb.business.GestionCarrosRemoto
java:global/demojq/GestionCarros!ec.edu.ups.pweb.business.GestionCarrosLocal
java:app/demojq/GestionCarros!ec.edu.ups.pweb.business.GestionCarrosLocal
java:module/GestionCarros!ec.edu.ups.pweb.business.GestionCarrosLocal
```

Ahora cada método del CRUD será lo mismo que vimos anteriormente con el cambio de que la creación, eliminación, actualización, búsqueda personalizada y listado dependerá del uso que le demos a nuestro variable “gestionCarros”.

Crear carro

```
try {
    final Hashtable<String, String> jndiProperties = new Hashtable<>();
    jndiProperties.put(key:Context.INITIAL_CONTEXT_FACTORY, value: "org.wildfly.naming.client.WildFlyInitialContextFactory");
    jndiProperties.put(key:Context.PROVIDER_URL, value: "http-remoting://localhost:8080");
    jndiProperties.put(key:Context.SECURITY_PRINCIPAL, value: "ejb64"); // Reemplaza 'username' con tu nombre de usuario
    jndiProperties.put(key:Context.SECURITY_CREDENTIALS, value: "ejb64"); // Reemplaza 'password' con tu contraseña
    jndiProperties.put(key:"jboss.naming.client.ejb.context", value: "true");

    final Context context = new InitialContext(environment:jndiProperties);
    GestionCarrosRemoto gestionCarros = (GestionCarrosRemoto) context.lookup(name: "ejb:/demojq/GestionCarros!ec.edu.ups.pwe

    // Uso del EJB
    Carro carro = new Carro();
    carro.setPlaca(placa: txtPlacaCarro.getText());
    carro.setModelo(modelo: txtModeloCarro.getText());
    carro.setMarca(marca: txtMarcaCarro.getText());
    gestionCarros.guardarCarros(carro);
    System.out.println(x: "Carro guardado!");
    JOptionPane.showMessageDialog(parentComponent:null, message:"Creación Exitosa");
    limpiarCrear();
```

Actualizar Carro

```
if (txtPlacaModificaciones.getText().equalsIgnoreCase(anotherString: "")) {
    JOptionPane.showMessageDialog(parentComponent:null, message:"Nada que actualizar");
} else {
    if (txtMarcaActualizar.getText().equalsIgnoreCase(anotherString: "") || txtModeloActualizar.getText().equalsIgnoreCase(anot
        JOptionPane.showMessageDialog(parentComponent:null, message:"No pueden quedar campos vacios");
    } else {
        try {
            final Hashtable<String, String> jndiProperties = new Hashtable<>();
            jndiProperties.put(key:Context.INITIAL_CONTEXT_FACTORY, value: "org.wildfly.naming.client.WildFlyInitialContextFa
            jndiProperties.put(key:Context.PROVIDER_URL, value: "http-remoting://localhost:8080");
            jndiProperties.put(key:Context.SECURITY_PRINCIPAL, value: "ejb64"); // Reemplaza 'username' con tu nombre de usua
            jndiProperties.put(key:Context.SECURITY_CREDENTIALS, value: "ejb64"); // Reemplaza 'password' con tu contraseña
            jndiProperties.put(key:"jboss.naming.client.ejb.context", value: "true");

            final Context context = new InitialContext(environment:jndiProperties);
            GestionCarrosRemoto gestionCarros = (GestionCarrosRemoto) context.lookup(name: "ejb:/demojq/GestionCarros!ec.edu
            Carro nuCarro = new Carro();
            nuCarro.setPlaca(placa: txtPlacaModificaciones.getText());
            nuCarro.setMarca(marca: txtMarcaActualizar.getText());
            nuCarro.setModelo(modelo: txtModeloActualizar.getText());
            gestionCarros.actualizarCarro(carro: nuCarro);
            JOptionPane.showMessageDialog(parentComponent:null, message:"Carro actualizado");
            limpiarModificaciones();
            actualizarTabla();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

Borrar Carro

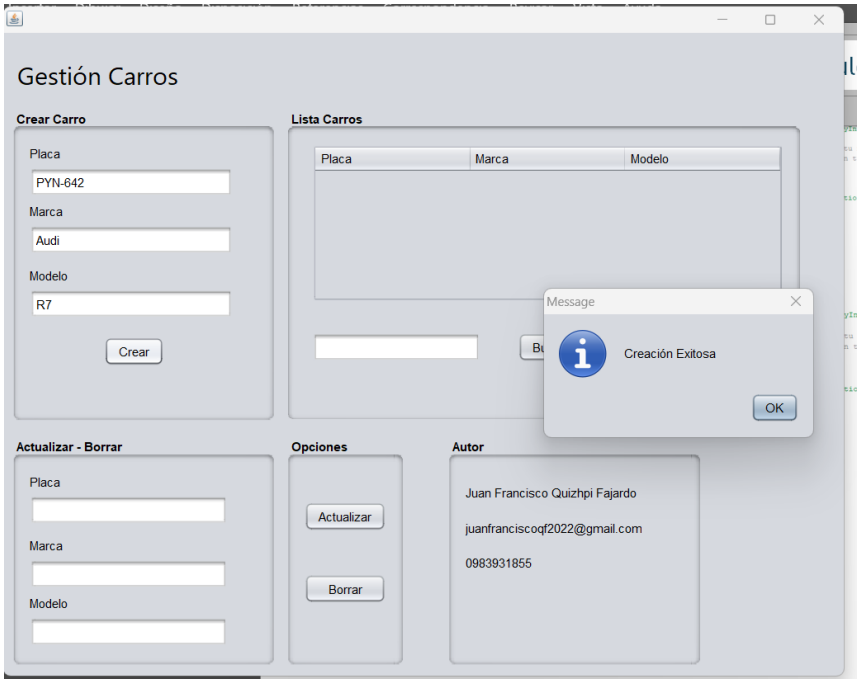
```
if (txtPlacaModificaciones.getText().equalsIgnoreCase(anotherString: "")) {
    JOptionPane.showMessageDialog(parentComponent:null, message:"Nada que borrar");
} else {
    int i = JOptionPane.showConfirmDialog(parentComponent:null, message:"Realmente deseas Eliminar los datos del Carro", title: "Eliminar Ca
    if (i == 0) {
        try {
            final Hashtable<String, String> jndiProperties = new Hashtable<>();
            jndiProperties.put(key:Context.INITIAL_CONTEXT_FACTORY, value: "org.wildfly.naming.client.WildFlyInitialContextFactory");
            jndiProperties.put(key:Context.PROVIDER_URL, value: "http-remoting://localhost:8080");
            jndiProperties.put(key:Context.SECURITY_PRINCIPAL, value: "ejb64"); // Reemplaza 'username' con tu nombre de usuario
            jndiProperties.put(key:Context.SECURITY_CREDENTIALS, value: "ejb64"); // Reemplaza 'password' con tu contraseña
            jndiProperties.put(key:"jboss.naming.client.ejb.context", value: "true");

            final Context context = new InitialContext(environment:jndiProperties);
            GestionCarrosRemoto gestionCarros = (GestionCarrosRemoto) context.lookup(name: "ejb:/demojq/GestionCarros!ec.edu.ups.pweb.
            gestionCarros.borrarCarro(placa: txtPlacaModificaciones.getText());
            JOptionPane.showMessageDialog(parentComponent:null, message:"Carro Borrado");
            actualizarTabla();
```

Listar Carro

```
//Conexion a servidor
try {
    final Hashtable<String, String> jndiProperties = new Hashtable<>();
    jndiProperties.put(key:Context.INITIAL_CONTEXT_FACTORY, value: "org.wildfly.naming.client.WildFlyInitialContextFactory");
    jndiProperties.put(key:Context.PROVIDER_URL, value: "http-remoting://localhost:8080");
    jndiProperties.put(key:Context.SECURITY_PRINCIPAL, value: "ejb64"); // Reemplaza 'username' con tu nombre de usuario
    jndiProperties.put(key:Context.SECURITY_CREDENTIALS, value: "ejb64"); // Reemplaza 'password' con tu contraseña
    jndiProperties.put(key:"jboss.naming.client.ejb.context", value: "true");

    final Context context = new InitialContext(environment:jndiProperties);
    GestionCarrosRemoto gestionCarros = (GestionCarrosRemoto) context.lookup(name: "ejb:/demojq/GestionCarros!ec.edu.ups.pwel
    //Cargamos a la tabla
    List<Carro> carros = gestionCarros.getCarros();
    for (Carro aux : carros) {
        datos[0] = aux.getPlaca();
        datos[1] = aux.getMarca();
        datos[2] = aux.getModelo();
        modelo.addRow(rowData:datos);
    }
```

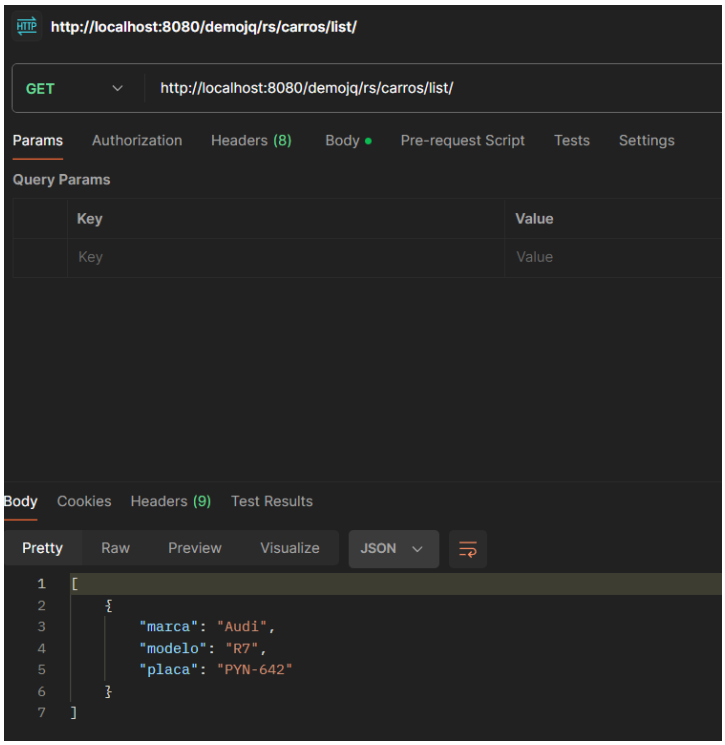
Prueba de aplicación

1. Creamos al carro ingresando cada uno de sus atributos verificamos su creación en el servidor y con uso de postman

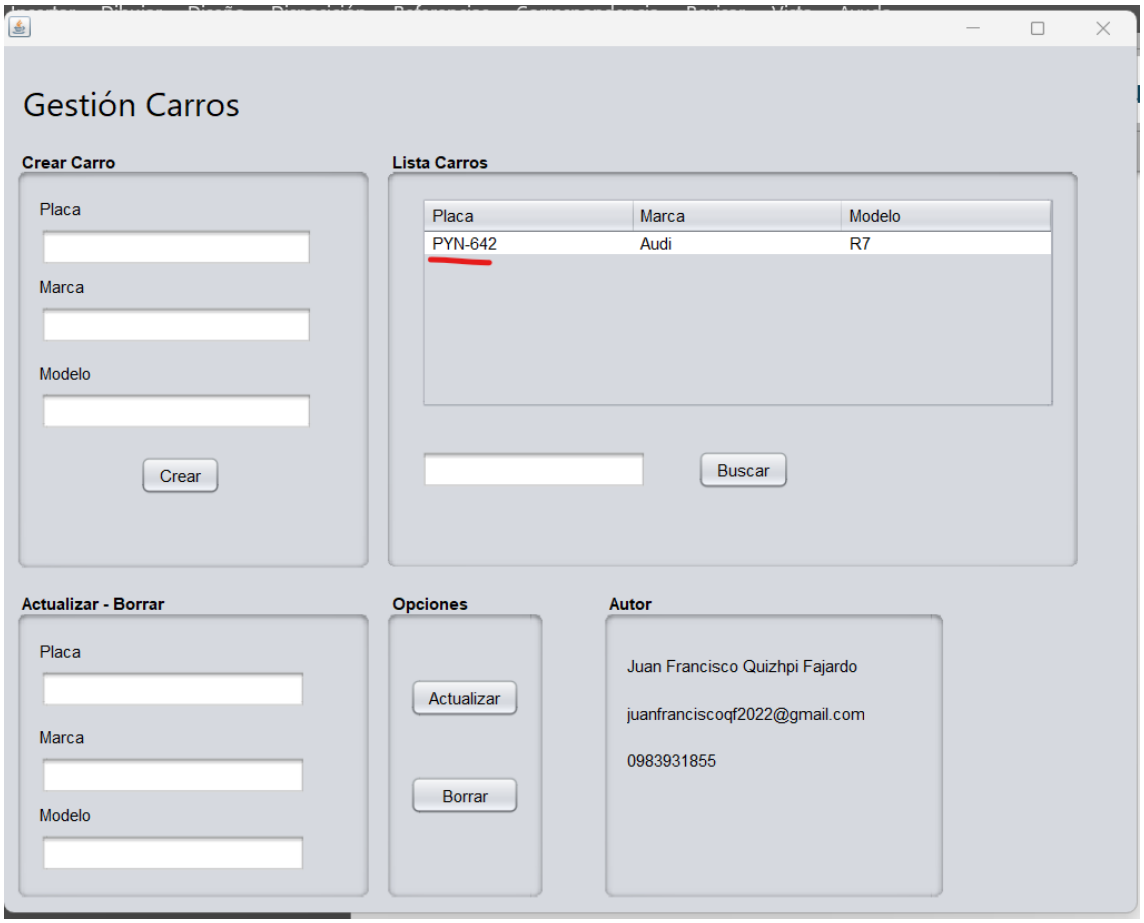


```
--- exec:3.1.0:exec (default-cli) @ carrojb ---
may 07, 2024 10:46:23 P. M. org.wildfly.naming.client.Version <clinit>
INFO: WildFly Naming version 1.0.14.Final
may 07, 2024 10:46:23 P. M. org.wildfly.security.Version <clinit>
INFO: ELY00001: WildFly Elytron version 1.11.2.Final
may 07, 2024 10:46:23 P. M. org.xnio.Xnio <clinit>
INFO: XNIO version 3.8.2.Final
may 07, 2024 10:46:23 P. M. org.xnio.nio.NioXnio <clinit>
INFO: XNIO NIO Implementation Version 3.8.2.Final
may 07, 2024 10:46:23 P. M. org.jboss.threads.Version <clinit>
INFO: JBoss Threads version 2.3.3.Final
may 07, 2024 10:46:23 P. M. org.jboss.remoting3.EndpointImpl <clinit>
INFO: JBoss Remoting version 5.0.17.Final
may 07, 2024 10:46:23 P. M. org.jboss.ejb.client.EJBClient <clinit>
INFO: JBoss EJB Client version 4.0.44.Final
Carro guardado!
```

```
22:46:24,132 INFO  [stdout] (default task-1)     insert
22:46:24,132 INFO  [stdout] (default task-1)     into
22:46:24,132 INFO  [stdout] (default task-1)         Carro
22:46:24,132 INFO  [stdout] (default task-1)         (car_marca, car_modelo, car_placa)
22:46:24,132 INFO  [stdout] (default task-1)     values
22:46:24,132 INFO  [stdout] (default task-1)         (?, ?, ?)
```

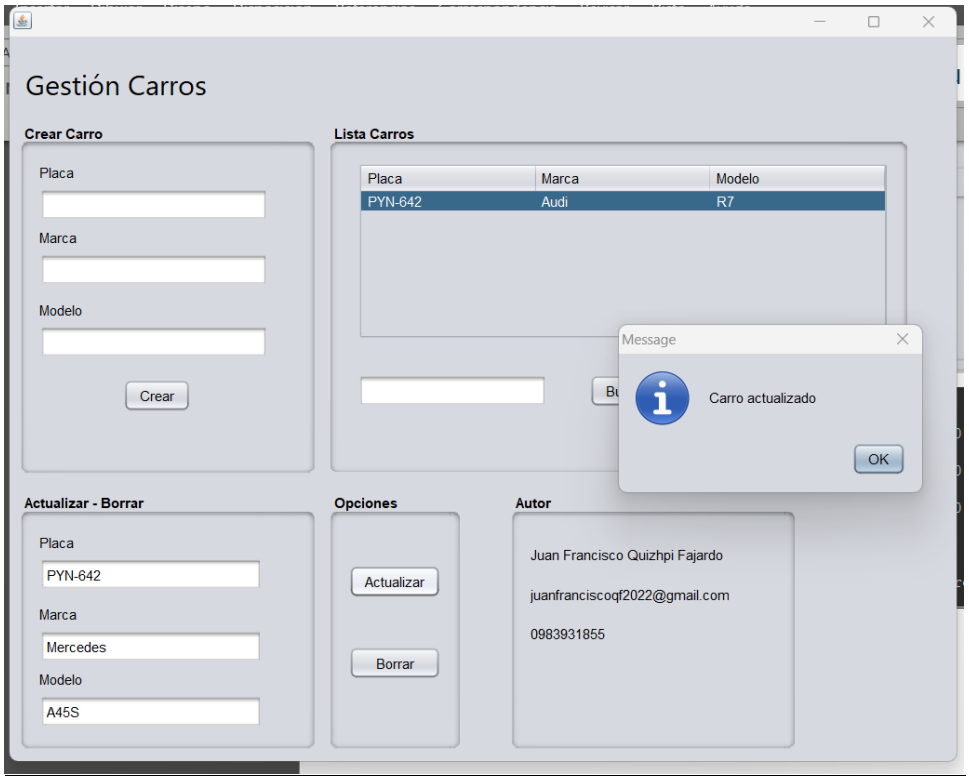


2. Buscamos los carros creados

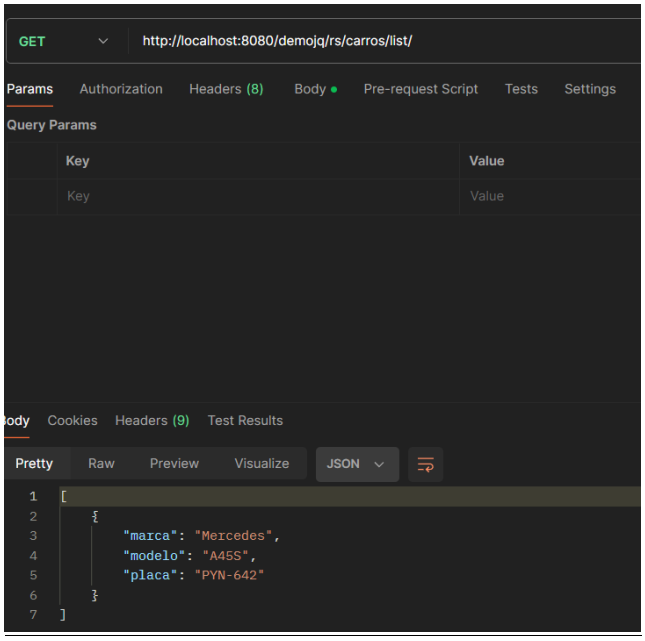



```
22:48:55,519 INFO  [stdout] (default task-1)     select
22:48:55,519 INFO  [stdout] (default task-1)         c1_0.car_placa,
22:48:55,519 INFO  [stdout] (default task-1)         c1_0.car_marca,
22:48:55,519 INFO  [stdout] (default task-1)         c1_0.car_modelo
22:48:55,519 INFO  [stdout] (default task-1)     from
22:48:55,519 INFO  [stdout] (default task-1)         Carro c1_0
```

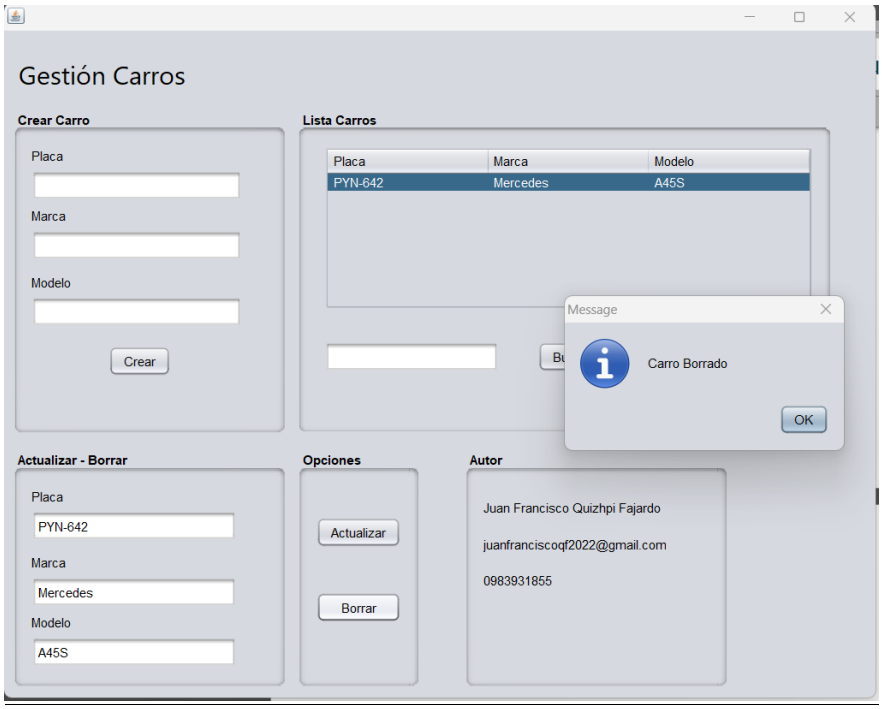
3. Actualización de Carro




```
22:50:12,787 INFO  [stdout] (default task-1)     update
22:50:12,787 INFO  [stdout] (default task-1)         Carro
22:50:12,788 INFO  [stdout] (default task-1)     set
22:50:12,788 INFO  [stdout] (default task-1)         car_marca=?,
22:50:12,788 INFO  [stdout] (default task-1)         car_modelo=?
22:50:12,788 INFO  [stdout] (default task-1)     where
22:50:12,788 INFO  [stdout] (default task-1)         car_placa=?
```



4. Borrado de Carro




```
22:51:52,826 INFO  [stdout] (default task-1)     delete
22:51:52,826 INFO  [stdout] (default task-1)     from
22:51:52,826 INFO  [stdout] (default task-1)         Carro
22:51:52,826 INFO  [stdout] (default task-1)     where
22:51:52,826 INFO  [stdout] (default task-1)         car_placa=?
```