

Documento de diseño del software Ramenautos

Por Samuel Peña, Daniela Echeverría y David Calderón

Introducción y Justificación del método de control:

En términos generales, los requerimientos funcionales son traducibles en manejar (crear, acceder, editar y mantener) tipos abstractos de datos. Cuando creo una reserva por ejemplo, únicamente debe verificar disponibilidad de vehículos, crear una instancia de una clase reserva y actualizar la lista de reservas del vehículo. Es por esto que, siguiendo esta aproximación, creemos pertinente utilizar un modelo híbrido de control centralizado y delegado, que nos permite dividir fácilmente el trabajo entre los miembros del grupo y simplificar el problema respetando encapsulamiento. En esa medida, proponemos una división primaria en tres partes. En primera instancia, un *modelo* compuesto solo de *data Holders* con únicamente getters y setters representando los tipos abstractos de datos que se deben manejar (ej carro y sede). En segunda instancia, un *controller* compuesto en principio de 7 clases, 4 de ellas representando cada una a un usuario (admin, cliente, empleado o admin general) con estereotipo de *controllers*, en el sentido de que deben definir cómo manejar los datos según lo que quiera hacer cada usuario. Además, en este nivel se incluye una clase *BaseDatos* compuesta de mapas de objetos *informationHolder* y delegadora, que debe cargar, descargar y mantener los datos. Esta última clase delega gran parte de sus funciones en las clases *Writer* y (que tiene métodos para convertir de objeto a string las instancias de los objetos) *Reader*, que hace lo opuesto. Cada miembro del equipo se encarga de un usuario (el de Admin empresa se encarga también de admin general) y desarrolla las clases que puedan apoyar a su controller o dejar toda la implementación en esta dicha clase únicamente, a criterio de cada uno. Por último, el programa cuenta con una consola general y 4 auxiliares, una por cada usuario, que regulan la interacción con los usuarios mediante login y menús de acciones.

Persistencia:

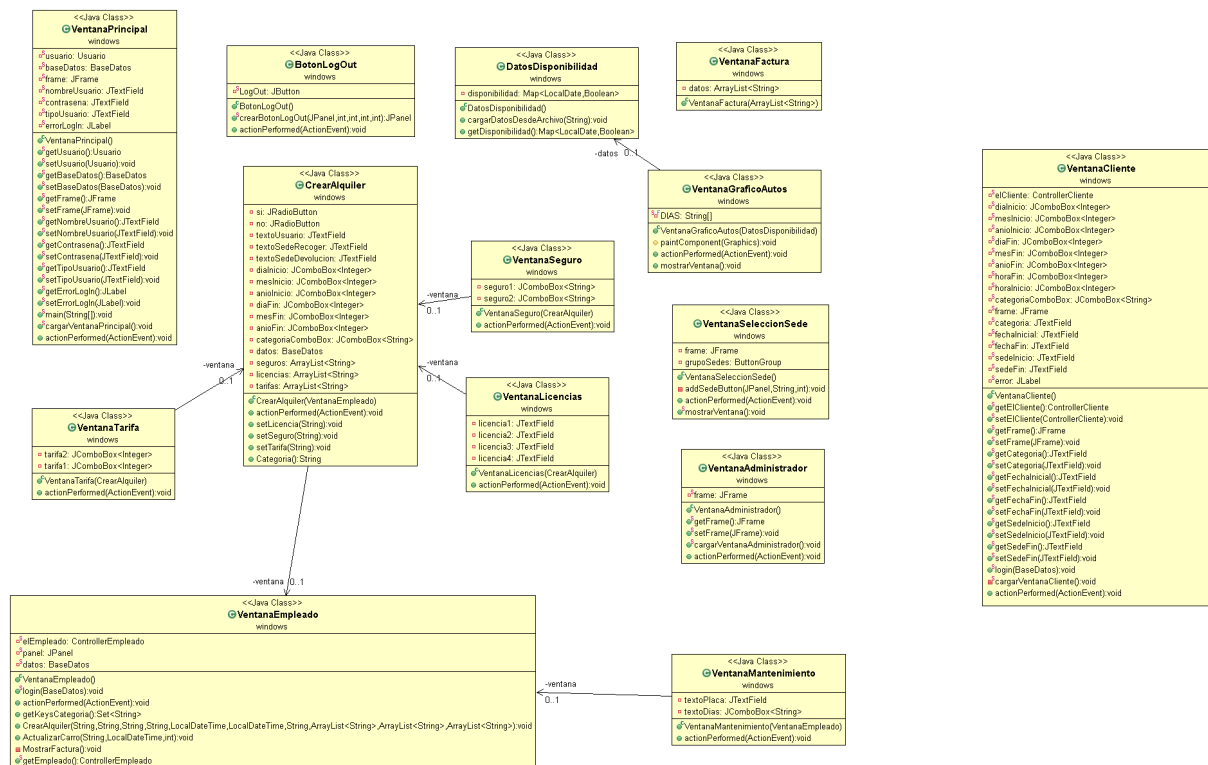
La persistencia se maneja conjuntamente entre las clases *BaseDatos*, *Reader* y *Writer*, y a través de archivos .txt. Hay un archivo txt por cada tipo abstracto de datos definido en el modelo, (ej carro, sede). Cada vez que un usuario cierra sesión, se reescriben los archivos que enlistan las instancias de los objetos a través de Strings que permitan su posterior descarga y conversión en objetos. El patrón para cada uno de estos objetos está descrito en la clase *Writer* (véase la entrega). Por ejemplo, para cargar un Cliente, se transforman todos sus atributos individuales en string y se juntan en un orden determinado (véase *writer*) separados por punto y coma. Para los atributos que son otros objetos no primitivos (tarjeta y licencia), insertamos el número que identifica cada una y lo añadimos al string de la forma descrita.

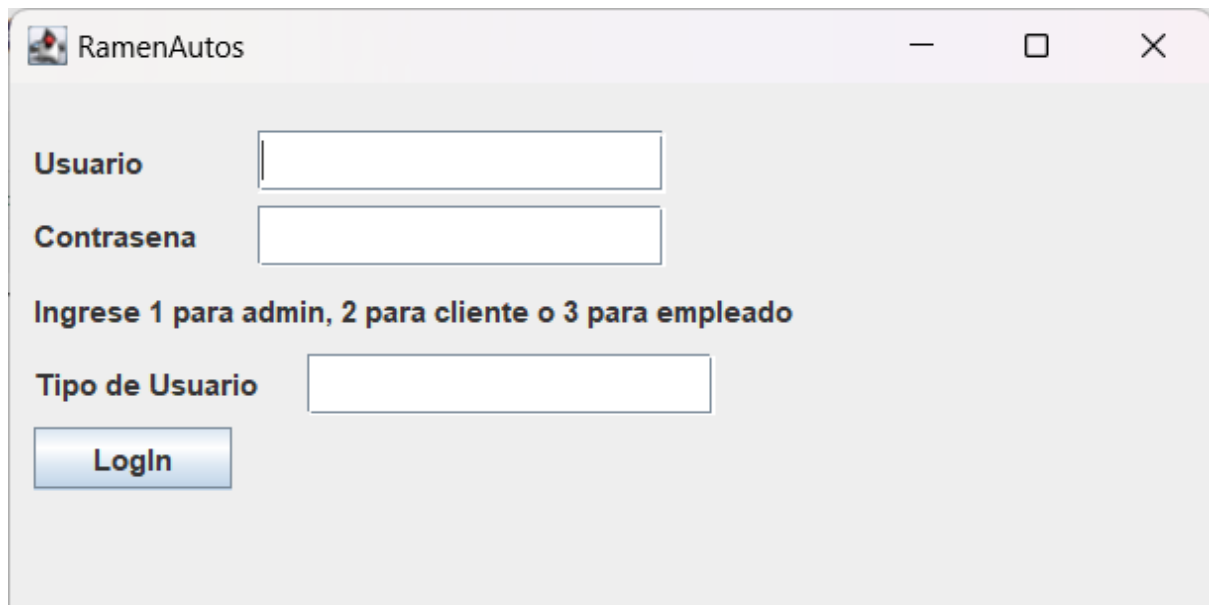
Cada vez que se inicializa la aplicación se descargan los datos, se crean los objetos y se añaden estos a los mapas. Se descargan todas las instancias de algún objeto y se almacenan en el map correspondiente antes de pasar al siguiente. Para esto se utiliza la clase *reader*, que a partir de las String devuelve instancias de los objetos. Para que el procedimiento funcione correctamente, se procura descargar los archivos en orden ascendente de la complejidad que tenga el objeto que representa. Esto es, se cargan primero los objetos que tengan los atributos más primitivos y la menor cantidad de atributos compuestos posibles. De esta forma, se

cargan primero los archivos correspondientes a tarjeta y licencia por ejemplo, que solo tienen atributos primitivos fácilmente traducibles entre String y su respectivo tipo, sin ninguna mediación anterior. Después se van cargando objetos más complejos y se procura que sus atributos compuestos ya hayan sido descargados y almacenados en un mapa. Por ejemplo, al descargar el cliente, primero se crea este a partir de sus atributos primitivos, con el número de licencia se busca la licencia en el mapaLicencias y se agrega, y lo mismo con el número de la tarjeta, luego de lo cual se agrega el dicho cliente al mapaClientes y se procede a descargar el siguiente. Si hay objetos que tengan asociación bidireccional con otros. Por ejemplo en este caso el alquiler tiene atributo Carro y carro tiene atributo Alquiler, entonces se descargan todos los carros sin alquiler y se almacenan asumiendo alquiler null y cuando se están descargando los alquileres, en cada iteración el alquiler se asocia a un carro y el carro al alquiler. Para que funcione correctamente es necesario que los constructores funcionen solo con datos primitivos y que los demás se puedan agregar con sets.

A los objetos que naturalmente no tienen identificador (como reserva), se les crea un número mediante un atributo static que va incrementando con cada instancia del objeto, según lo indica el constructor.

Interfaz Gráfica:





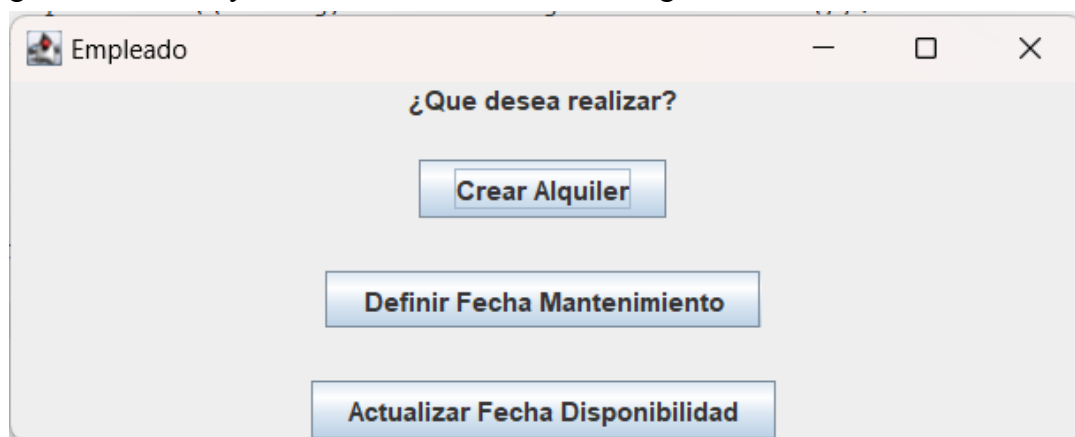
The screenshot shows a Java Swing window titled "RamenAutos". Inside the window, there are two text input fields labeled "Usuario" and "Contraseña". Below these fields is a text label that reads "Ingrese 1 para admin, 2 para cliente o 3 para empleado". Underneath this label is another text input field labeled "Tipo de Usuario". At the bottom left of the window is a button labeled "Login".

La interacción con el usuario se maneja a través de interfaces gráficas creadas con Java Swing. La implementación de estas interfaces se divide en 4 partes principales. Hay una ventana principal que se encarga de determinar el tipo de usuario y cargar los datos, y dependiendo de esto corre alguna de las 3 ventanas relacionadas con cada tipo de usuario. Cada una de estas ventanas principales se trata en una clase que hereda de `Action Listener` para la acción de algún botón. Si hay otros botones se crearon clases aparte para estos que heredan también de `Action Listener`. Los atributos de estas Ventanas ya sea información sobre la ventana como un `JLabel` o instancias del modelo son `Static`, porque solo deben crearse una vez y porque esto facilita que se puedan llamar estos atributos singulares desde otras clases.

Para ingresar se le pedirán al usuario los datos que aparecen y se buscará si coinciden, si no no dejara ingresar.

Interfaz Empleado:

Al ingresar el usuario y contraseña correcta se abre la siguiente ventana:



The screenshot shows a Java Swing window titled "Empleado". The window has a light gray background. At the top, below the title bar, is the text "¿Que desea realizar?". Below this text are three buttons arranged vertically. The first button is labeled "Crear Alquiler". The second button is labeled "Definir Fecha Mantenimiento". The third button is labeled "Actualizar Fecha Disponibilidad".

Esta cuenta con diferentes botones que permiten realizar los requerimientos funcionales de un empleado. Al presionar el botón "Crear Alquiler" se abre la siguiente ventana:

The 'Alquiler' window contains the following fields and controls:

- Reserva:** Radio buttons for 'Si' and 'No'. 'No' is selected.
- Cliente (usuario):** Text input field containing 'daniela'.
- Sede Recoger:** Text input field containing 'norte'.
- Sede Devolucion:** Text input field containing 'norte'.
- FechaInicio:** Date picker showing 2/1/2023.
- FechaFin:** Date picker showing 4/1/2023.
- Categoria:** Dropdown menu showing 'familiares'.
- Buttons:** 'Seguros', 'Licencias', 'Tarifas Excedentes', and 'Crear'.

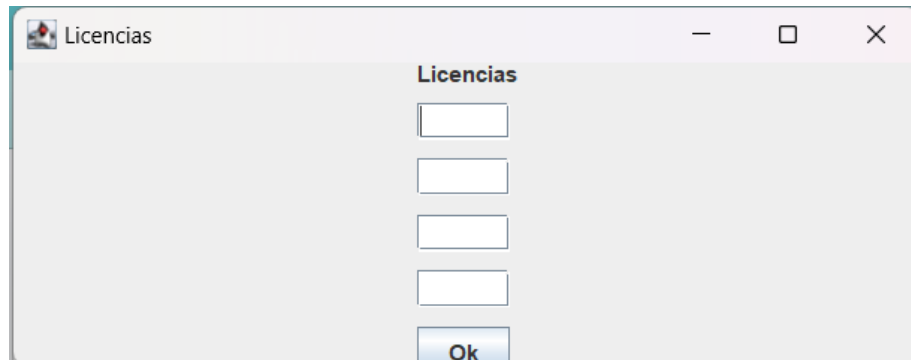
En esta ventana el empleado puede escoger entre si ya existe una reserva o no, debe digitar el usuario del cliente al cuál le pertenece el alquiler, las sedes a recoger y devolver y con ayuda de un comboBox puede escoger la fecha de inicio y fin del alquiler, teniendo el orden de dd/mm/aaaa, igualmente sucede con las categorías. Si es necesario añadir un seguro, licencias adicionales o alguna tarifa adicional, el empleado puede oprimir cualquiera de esos tres botones y saldrá una ventana para cada una de estas opciones. En todas estas ventanas al oprimir “Ok” se devuelve a la ventana que nos permite crear un nuevo alquiler guardando la información digitada en cada una de ellas. La vista para añadir un seguro es la siguiente:

The 'Seguros' window displays a dropdown menu for insurance types. The dropdown is open, showing the following options:

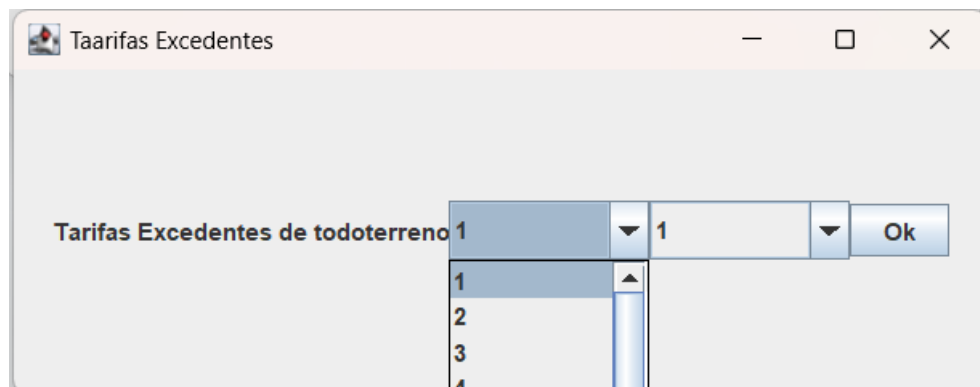
- Perdida
- Accidente

There is an 'Ok' button to the right of the dropdown.

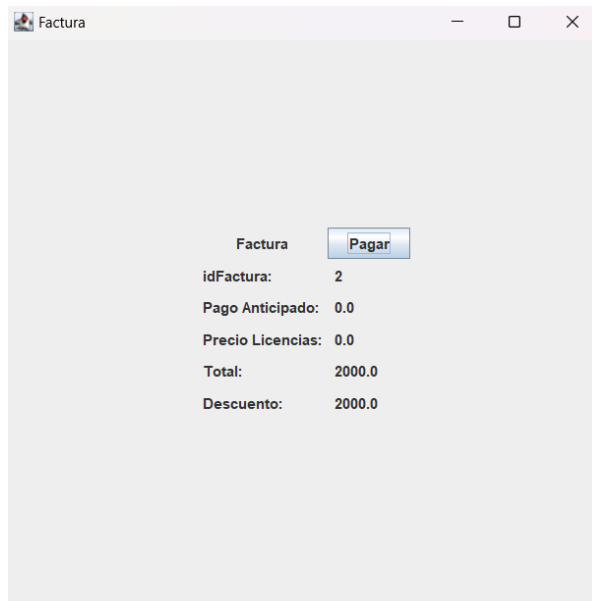
En esta ventana nos aparecen las opciones de seguros existentes se puede poner información en todas las opciones o dejar las dos en blanco o solo tener un espacio con información. La ventana de licencias tiene la siguiente visualización:



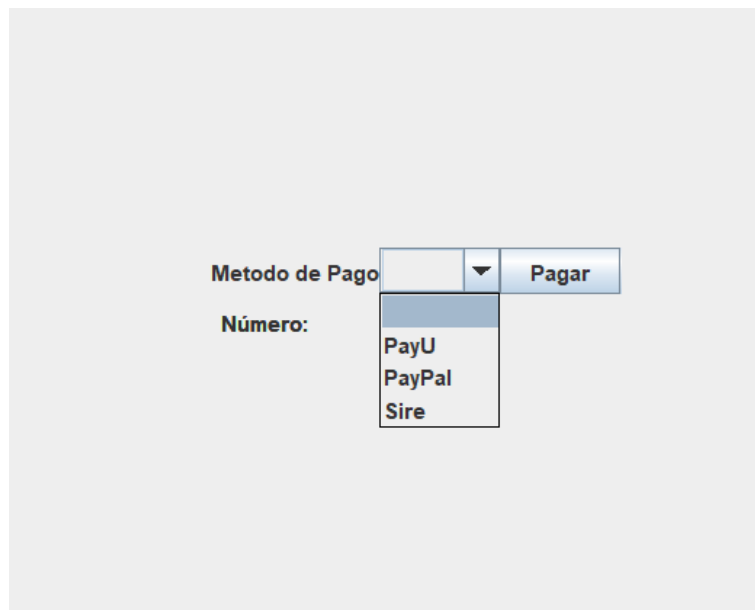
En este caso se tienen campos de texto en donde se digita el número de cada una de las licencias que se desean utilizar, estas licencias ya deben existir en la base de datos y como máximo se pueden tener cuatro licencias adicionales. La ventana de tarifas adicionales se ve de la siguiente forma:



Esta ventana es muy similar a la de seguros, sin embargo aquí aparecerá el id de cada una de estas dependiendo de la categoría seleccionada en la anterior ventana de crear reserva. Después de haber añadido todo lo necesario para crear un alquiler se debe presionar el botón “Crear” y este te muestra una ventana con la factura del alquiler.



Al presionar el botón pagar sale una nueva ventana en donde muestra todas las plataformas posibles de pago y también es necesario digitar el número de la tarjeta. Cuando el empleado oprima pagar se genera un pdf con la factura y la firma del administrador y es guardado en las carpeta de facturas. Posteriormente, se devuelve al menú principal del empleado.

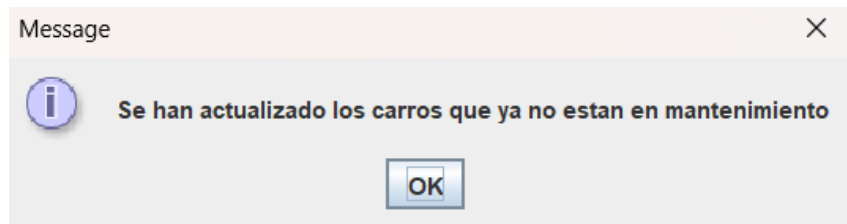


Las otras funcionalidades que tiene la ventana del empleado es poder definir la fecha de mantenimiento de un automóvil, dependiendo de su placa, como máximo un carro puede estar en mantenimiento dos días. También puede oprimir “Actualizar Fecha Disponibilidad”, se muestra un mensaje cuando se actualizan todos los carros de en mantenimiento a disponible.

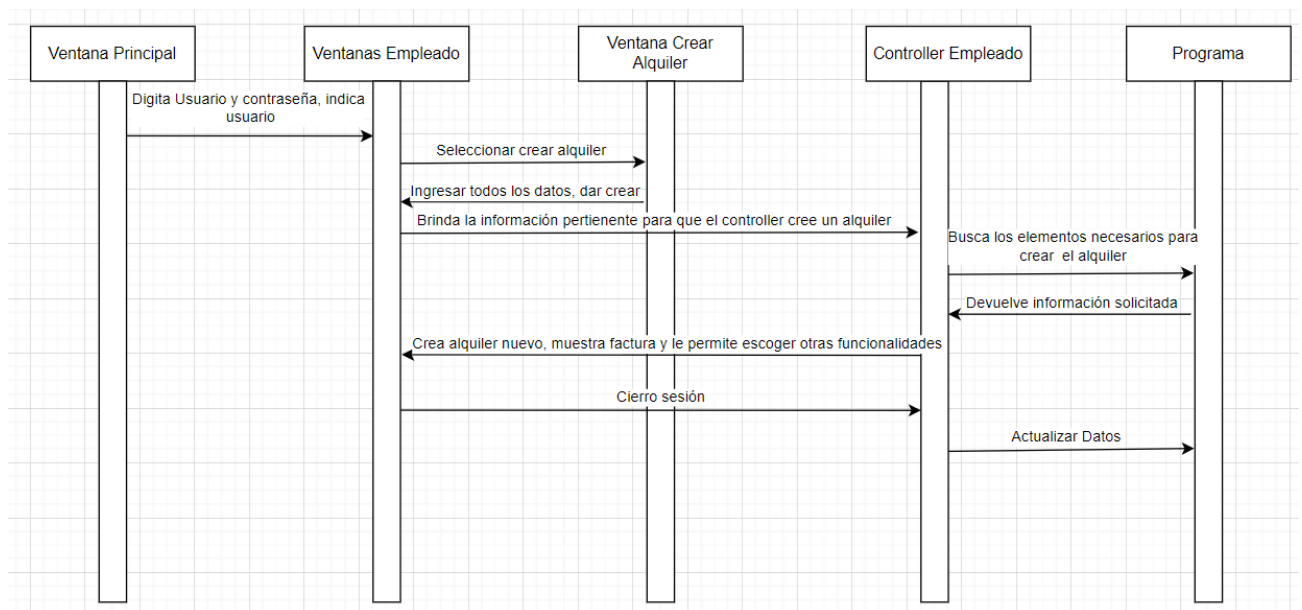
Placa:

Dias:

Ok

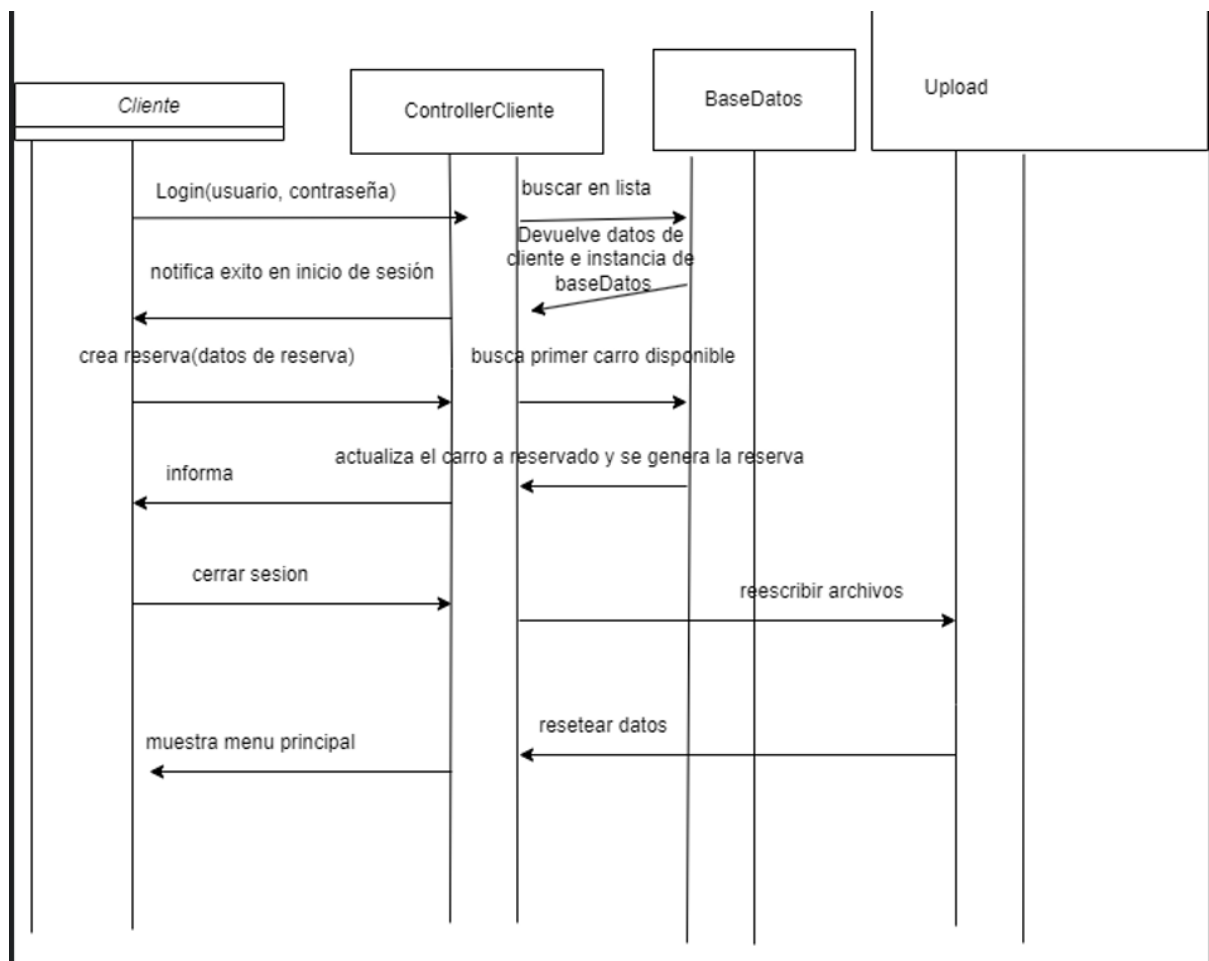


Para finalizar sesión solo es necesario que se oprima la x en la ventana del empleado.



Con este diagrama se puede observar la secuencia que realiza la aplicación para poder cumplir con los requerimientos funcionales del empleado.

Historia de Usuario Cliente:



Resumen simplificado de una creación exitosa de reserva por parte del cliente.

Como indica el enunciado del proyecto, el cliente solo puede acceder al sistema si tiene login, y su única funcionalidad es crear y gestionar reservas. Gestionar puede entenderse como crear, llenar o desarrollar, según la RAE, luego no es necesario asumir que las reservas puedan cambiarse o eliminarse, tampoco dice eso en ninguna parte del enunciado por lo que se supone que no se puede cancelar o modificar una reserva creada desde el login del cliente. En ningún lado dice que el cliente debe ser capaz de crear un usuario, por lo que se asume que los usuarios ya están dados. En ese sentido, su única funcionalidad es la de crear una reserva.

Al inicializar la aplicación, esta descargará todos los archivos cargando los datos y le pregunta al usuario qué tipo de usuario es, y dependiendo de lo que responda se utiliza la interfaz correspondiente.

Crear Reserva

Categoria

FechaInicio: Día 1 Mes 1 Año 2023 Hora 0

FechaFin: 1 1 2023 0

Sede Inicial

Sede Final

Crear Reserva LogOut

Esta ventana, consiste de 2 JButtons para crear reserva y salir de la sesión , 3 JLabels y 3 JTexts para ingresar la categoría y las sedes, y 8 JComboBoxs para facilitar la digitación de las fechas y otras 4 JLabels para indicar donde poner cada parte de la fecha.

En este punto, como muestra la imagen, se le preguntará por la categoría que desea, el intervalo de tiempo estimado de la reserva y los nombres de las sedes de entrega y alquiler. Con estos datos se procede a validar si en el intervalo seleccionado hay algún carro de la categoría libre. Los datos de las sedes se obvian en este paso porque el enunciado dice que el admin puede cambiar carros de sede para cumplir con las reservas, luego para reservar no es impedimento la sede de alquiler o entrega. Para cumplir con esto, se iteran todos los carros del inventario y se valida si el intervalo se interseca con algún alquiler, mantenimiento, limpieza o reserva y se van descartando los carros si se detecta intersección con alguno de estos items. Si se completa la iteración sin encontrar ningún carro se le dice al cliente que no se pudo crear reserva. Si se encuentra el carro, este procede a reservarse. Para esto, se crea una reserva y se guarda en la base de datos, se añade la dicha reserva al carro y se vuelve a guardar este en la base de datos y se actualiza la información de la tarjeta del cliente para indicar que está bloqueada. Por último, se calcula la tarifa a partir de la temporada, la categoría, la longitud del intervalo y si se entrega en la misma sede o no y se le informa al cliente del valor del cobro correspondiente al 30%.

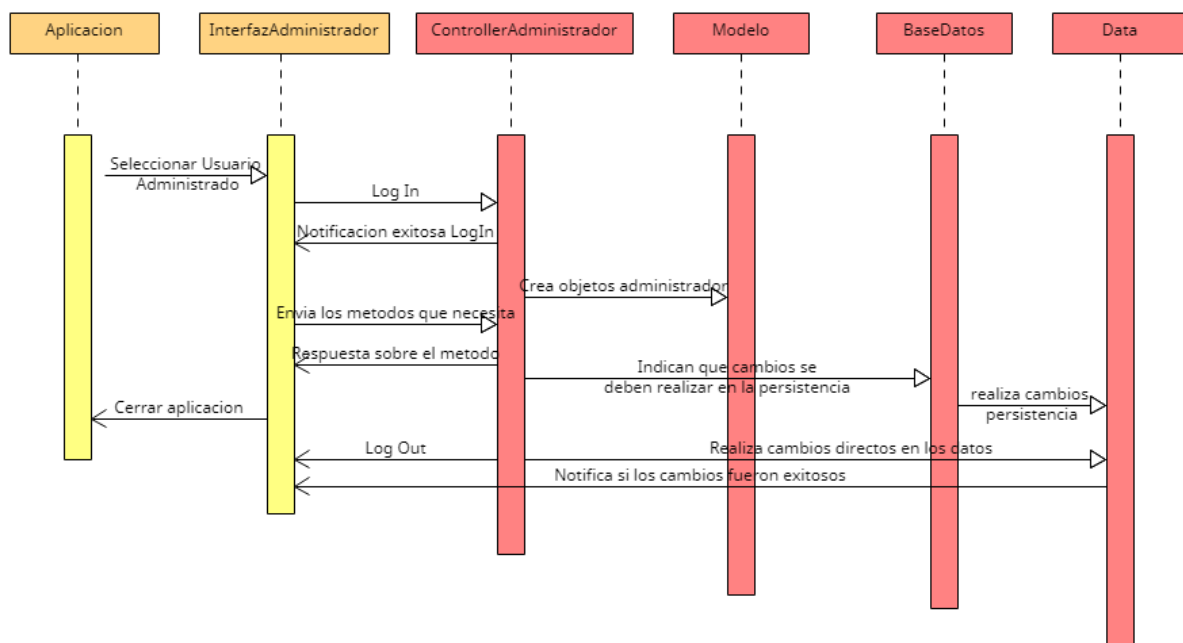
Nótese que el enunciado solo dice que el cliente no debe saber qué carro se le va a entregar, pero esto no implica que la reserva no pueda asociarse a un carro desde el momento en el que

se hace. Para cumplir con el requerimiento es suficiente con no informar al cliente que carro recibirá. Adicionalmente, tampoco dice en ninguna parte que deba generarse alguna factura persistente y que alguno de los usuarios deba tener acceso a ella, la contabilidad monetaria en ningún punto está presente en los requerimientos expresados.

Nótese también que asumimos que los intervalos de las reservas y alquileres preexistentes duran 2 días más de lo que realmente duran, asumiendo que el plazo máximo para que un vehículo esté en mantenimiento o limpieza es de 2 días (esto es arbitrario), garantizando así que siempre se cumpla con las reservas. También, asumimos que el excedente por no entregar el carro en la sede es de 5 (nótese que esto tampoco está dado, y no es responsabilidad de ningún usuario fijarlo).

Por último, al seleccionar LogOut, se reescriben todos los archivos de persistencia y se cierra la interfazCliente, volviendo al menú principal.

Historia de Usuario Administrador:

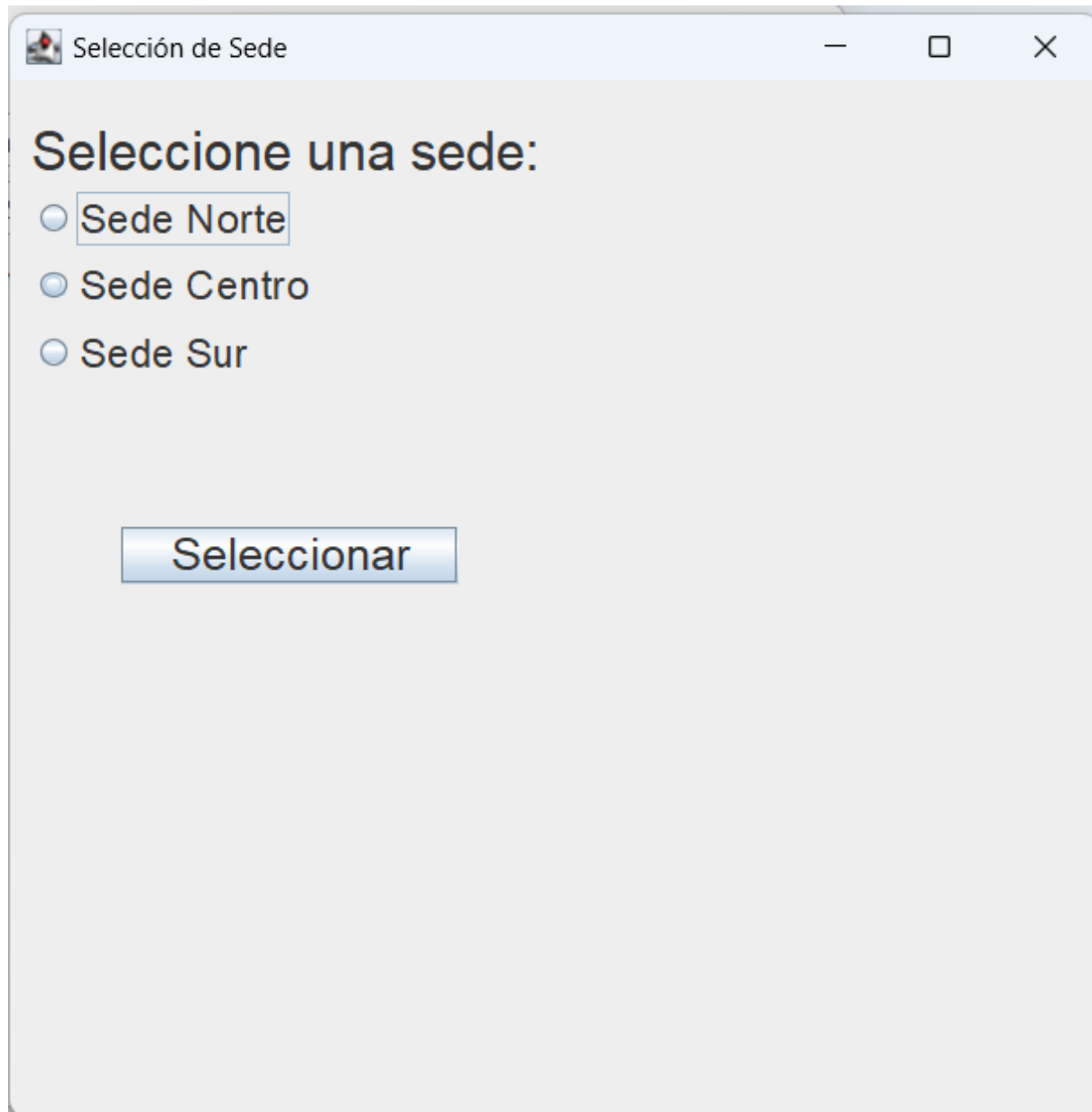


El administrador se encarga de gestionar los vehículos , en dismantelar los que ya no se necesitan así como en registrar los nuevos, hay que tener en cuenta que un administrador pertenece a solo una Sede por lo tanto y una Sede solo puede tener un administrador que también tiene la tarea de crear nuevos empleados .

Por lo tanto cuando inicie la aplicación y el login sea exitoso, el administrado tendrá la capacidad de ingresar nuevos vehiculos asi como de eliminar los que ya no se consideren en uso, para eso solo necesita consultar la placa del vehículo y desde el controller la instrucción

de eliminar sera directa a los archivos persistentes en la carpeta data , contrario al login que, desde el controller accede a la clase BaseDatos para cargar y actualizar archivos , aunque por supuesto esto datos también se actualizan. Lo mismo ocurre cuando el administrador quiere agregar o eliminar los empleados de la sede, la instrucción va directa desde el controller a los datos de persistencia para hacer las operaciones de crear o eliminar.

Interfaz Grafica Administrador



Al ingresar exitosamente las credenciales del administrador principal se muestra una ventana aparte con un menu de seleccion para la sede , al seleccionar cualquiera se muestra el siguiente menú:



Bienvenido Admin!

Selecciona alguna de las siguientes opciones:

Agregar/Eliminar Administrador Local

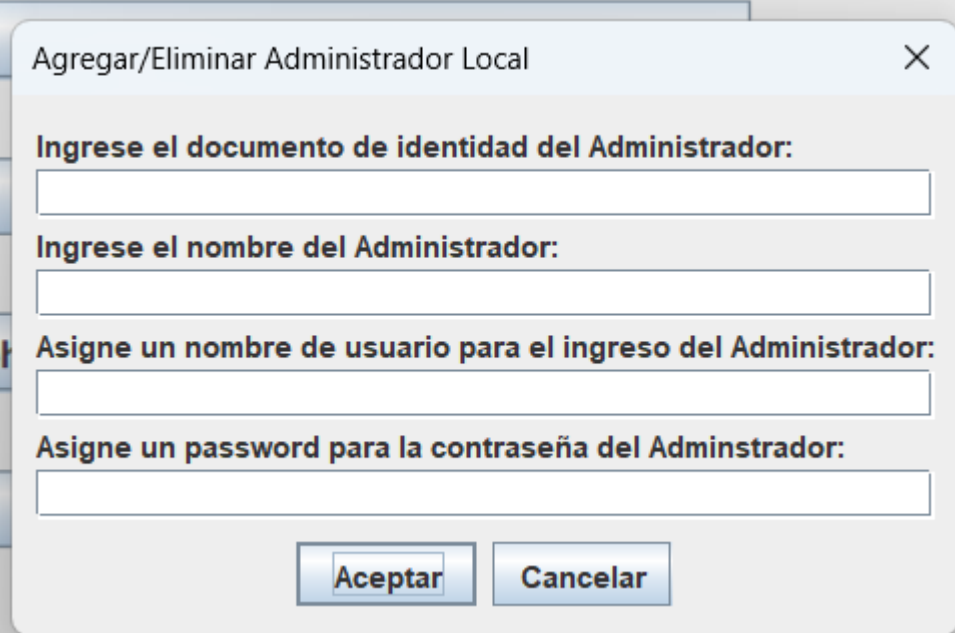
Agregar/Eliminar Empleado

Actualizar/Eliminar Auto

Vehículos Disponibles al año(Gráfica)

LogOut

cada uno tiene la opción para agregar a la persistencia los datos 👍



Agregar/Eliminar Administrador Local

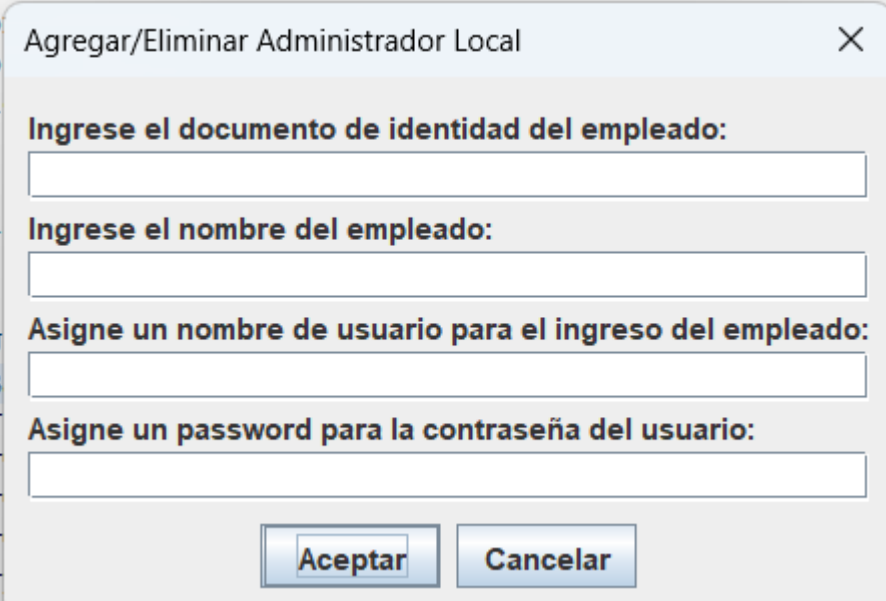
Ingrese el documento de identidad del Administrador:

Ingrese el nombre del Administrador:

Asigne un nombre de usuario para el ingreso del Administrador:

Asigne un password para la contraseña del Adminstrador:

Aceptar **Cancelar**



Agregar/Eliminar Administrador Local

Ingrese el documento de identidad del empleado:

Ingrese el nombre del empleado:

Asigne un nombre de usuario para el ingreso del empleado:

Asigne un password para la contraseña del usuario:

Aceptar **Cancelar**

Actualizar/Eliminar Auto

Ingrese la placa del vehiculo:

Ingrese la marca del vehiculo:

Ingrese el modelo del vehiculo:

Escriba el tipo de transmision del vehiculo:

Aceptar
Cancelar

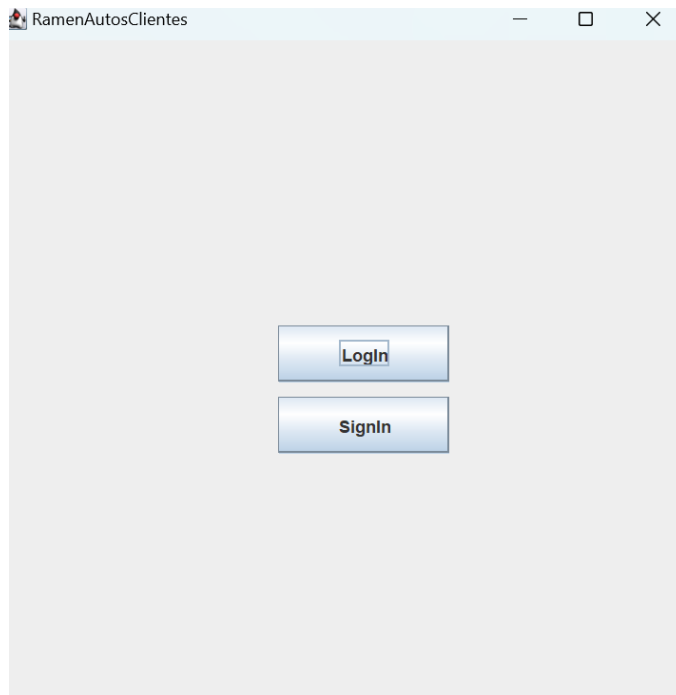
y la opción de la gráfica de autos disponibles :



App personalizada para uso del cliente:

Esta app para uso exclusivo del cliente cumple a cabalidad con todo lo solicitado. Permite 1. crear un usuario de cliente, 2. Revisar los carros disponibles en una fecha, y reservar con 10% de descuento.

Al inicializar la aplicación se le mostrará al usuario la siguiente ventana que solo permite crear un nuevo usuario (Sign In) o iniciar sesión (LogIn):



Al presionar el botón SignIn, se le solicita al usuario unos datos personales, de licencia y de tarjeta, así como un usuario y contraseña para la app.

Si el usuario elige un nombre de usuario en uso o no llena todas las casillas necesarias, se le informará de esto para que así lo haga como muestran las siguientes 2 imágenes:

Left Screenshot (Empty Fields):

LOGIN INFO
 Usuario: Contraseña:

DATOS PERSONALES
 Nombre: eMail:
 Nacionalidad: Ruta de imagen de ID:

DATOS LICENCIA
 Número: País:
 Fecha de vencimiento (MM/AA): Ruta imagen de licencia:

DATOS TARJETA
 Número: Código:

Buttons:

Red text: **Debe llenar todos los campos**

Right Screenshot (Filled Fields):

LOGIN INFO
 Usuario: Contraseña:

DATOS PERSONALES
 Nombre: eMail:
 Nacionalidad: Ruta de imagen de ID:

DATOS LICENCIA
 Número: País:
 Fecha de vencimiento (MM/AA): Ruta imagen de licencia:

DATOS TARJETA
 Número: Código:

Buttons:

Red text: **Usuario en uso, pruebe con otro**

Si por el contrario se suministraron los datos solicitados, se le informará al usuario del éxito para que proceda a hacer login y se guardarán sus datos, como muestra la siguiente imagen.

The screenshot shows a web application window titled "SignIn". It contains several sections for user registration and login:

- LOGIN INFO**: Fields for "Usuario" (MarioBros) and "Contraseña" (Dpoo21).
- DATOS PERSONALES**: Fields for "Nombre" (Mario Sanchez), "eMail" (@uniandes), "Nacionalidad" (maymar), and "Ruta de imagen de ID" (http.mario).
- DATOS LICENCIA**: Fields for "Número" (6767), "Pais" (Italia), "Fecha de vencimiento (MM/AA)" (05/26), and "Ruta imagen de licencia" (http.mariolic).
- DATOS TARJETA**: Fields for "Número" (8000) and "Código" (555).

At the bottom right, there are two buttons: "SignIn" and "Volver". Below these buttons, a green message states: "SignIn realizado correctamente, presione volver para hacer Login".

NOTA: se asume que los datos que ingresa el usuario son correctos y que la imagen de la identificación y de la licencia viene dada por un link privado tipo googleDocs, lo que en nada incumple los requerimientos funcionales del sistema.

Así, al hacer login con un usuario existente (como el acabado de crear), se le presentará al usuario la opción o de crear reserva o de consultar disponibilidad de carros en fecha y sede dada, como muestra la siguiente imagen.

The screenshot shows a web application window titled "Cliente". It displays a welcome message and two buttons:

- Bienvenido MarioBros**: A blue text message.
- Ver Disponibilidad**: A button to check availability.
- Crear Reserva**: A button to create a reservation.

Cuando el usuario selecciona Ver Disponibilidad, se le preguntará por la fecha y la sede, y podrá consultar la disponibilidad como muestra la siguiente imagen.

The screenshot shows a window titled "Disponibilidad". It contains a date selection section with "Fecha" and a "Sede" dropdown. The date is set to 1/1/2024 and the location is "norte". Below these are "Consultar" and "Volver" buttons. At the bottom, a list of vehicles is displayed, each with a unique ID and a set of characteristics separated by semicolons.

Fecha	Día	Mes	Año	Sede
1/1/2024	1	1	2024	norte

Consultar Volver

NEK123;chevrolet;10;plateado;b;;todoterreno;norte;Disponible>null;au...
NEK123;chevrolet;10;plateado;b;;todoterreno;norte;Disponible>null;auto;5
DEF123;mazda;10;plateado;b;;familiares;norte;Disponible>null;auto;2

Note que para que aparezca lo solicitado debe presionar Consultar primero. Cuando el usuario presione volver regresará a la ventana anterior.. Se itera la lista de los carros y el carro se añade a la lista objetivo si está disponible en esa sede en esa fecha. Para determinar los carros disponibles en una sede en un momento dado solo se toma en cuenta el alquiler. Si el carro no está alquilado y su sede es la solicitada, o si está alquilado pero el alquiler termina en la sede solicitada y antes de la fecha solicitada, se mete el carro en la lista objetivo, de lo contrario se continúa la iteración. Para más detalles, refiérase a las nuevas funciones del Controller cliente `vehículosDisponiblesEnSedeEnFechaDada` y `sedeEnLaQueEstaraUnCarrodadaFecha`.

Desde esta ventana anterior, cuando el usuario presiona Crear Reserva, lo llevará a una ventana donde le pedirá los datos para la creación de esta. El proceso de creación de reserva con su validación respectiva funciona igual que el de la aplicación general (para más detalles remítase a la sección Historia de usuario cliente del presente documento. En ese sentido, si resulta que no hay carros disponibles de las características solicitadas en las fechas solicitadas o si la tarjeta del cliente está bloqueada, se le informará al cliente que hubo un error al crear su reserva y esta no se creará, como muestra la siguiente imagen.

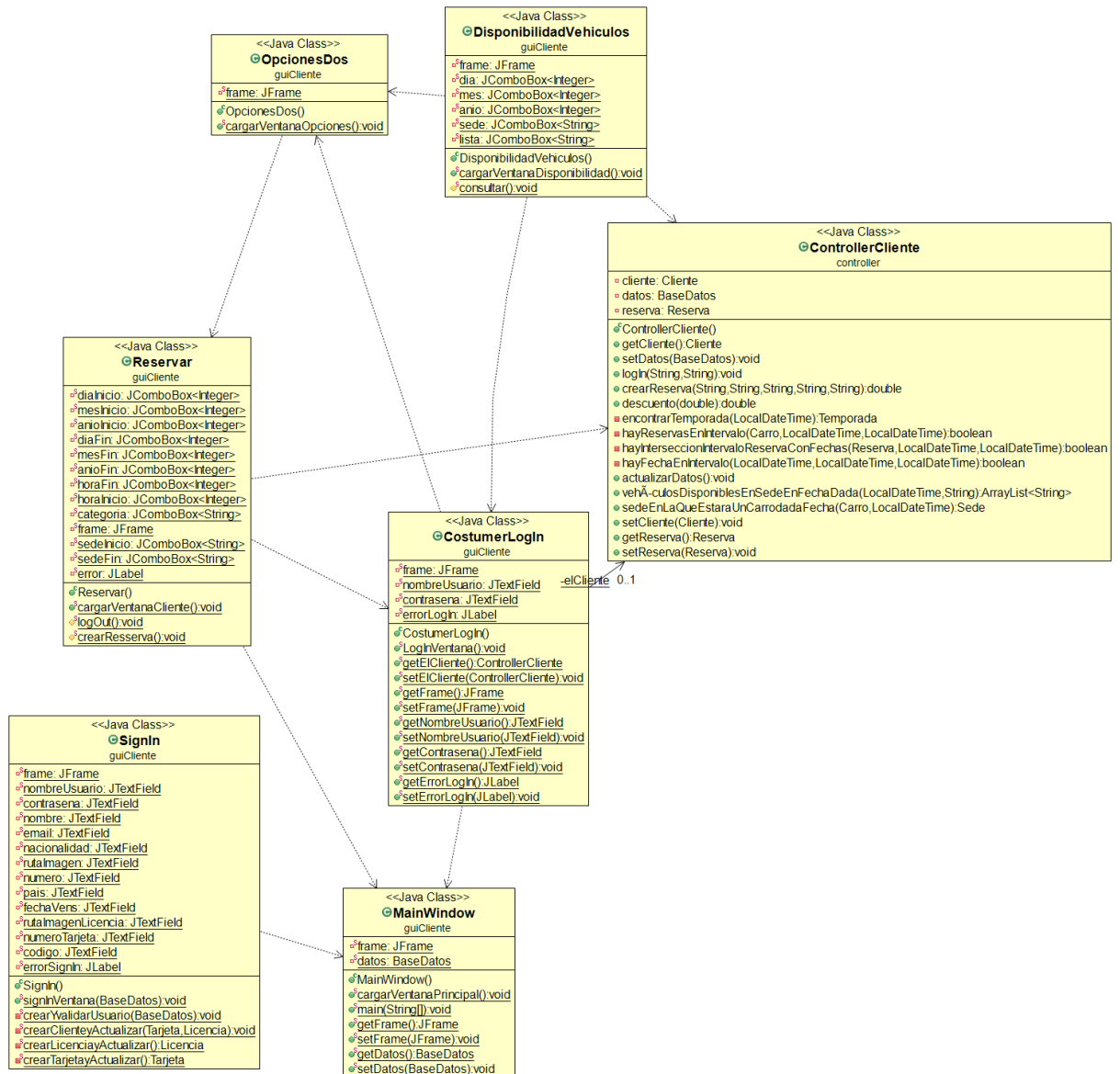
The screenshot shows a web application window titled "Crear Reserva". It contains several form fields: "Categoria" (dropdown menu with "todoterreno" selected), "FechaInicio:" (with sub-fields for "Día", "Mes", "Año", and "Hora"), "FechaFin:" (with sub-fields for "Día", "Mes", "Año", and "Hora"), "Sede Inicial" (dropdown menu with "norte" selected), and "Sede Final" (dropdown menu with "norte" selected). Below these fields, a red error message reads "Error creando reserva". At the bottom right, there are two buttons: "Reservar" and "LogOut".

Si por el contrario no se presentaron inconvenientes (existían carros disponibles en las fechas y la tarjeta no estaba bloqueada), la reserva se creará exitosamente y se le informará al cliente el monto que se le cobró, como muestra la siguiente imagen.

The screenshot shows the same "Crear Reserva" window, but now with a green success message: "Su reserva está lista, se le cobró el 30% con descuento correspondiente a 89559.0". The form fields are the same as in the previous image, but the "FechaInicio:" "Mes" field is now "3" and the "FechaFin:" "Mes" field is now "4". The "Reservar" and "LogOut" buttons are still present at the bottom right.

Al presionar LogOut, se generará persistencia y se cerrará la sesión devolviendo al usuario a la primera imagen.

Diagrama de clases general y de alto nivel:




```
Carro.java Reader.java x
Center
111     return s;
112 }
113 // SEPTIMO OBJETO: CARRO
114 public Carro descomprimirCarro(String linea, HashMap<String, Sede> mapaSedes,
115     HashMap<String, Categoria> mapaCategorias) {
116     String[] partes = linea.split(";");
117     String placa = partes[0];
118     String marca = partes[1];
119     String modelo = partes[2];
120     String color=partes[3];
121     String trans=partes[4];
122     String nombrCateg=partes[5];
123     String nombreCede=partes[6];
124     String estado=partes[7];
125     String dispon=partes[8];
126     String tipo=partes[9];
127     String porc=partes[10];
128     Carro car =new Carro(placa,marca, modelo,  color,trans,tipo,porc);
129     if(dispon.equals("null")==false) {
130         car.setFechaDisponibleCons (LocalDateTime.parse(dispon));
131     }
132
133     Categoria categoria = mapaCategorias.get(nombrCateg);
134     car.setEstado(estado);
135     car.setCategoria(mapaCategorias.get(nombrCateg));
136     car.setCede(mapaSedes.get(nombreCede));
137     Sede sede= mapaSedes.get(nombreCede);
138     sede.setCarro(car);
139     categoria.setCarro(car);
140     return car;
141 }
142 }
```

```
Carro.java Reader.java Writer.java x
2
3 import java.util.HashMap;
19
20 // Esta clase traduce los objetos a string para la persistencia
21 /**
22  * @author Daniela
23  *
24  */
25 public class Writer {
26     /// SOLO METODOS
27     // PRIMER OBJETO: CARRO
28
29     public String comprimirCarro(Carro carro) {
30         String placa = carro.getPlaca();
31         String marca = carro.getmarca();
32         String modelo = carro.getModelo();
33         String color = carro.getColor();
34         String tipoTransmision = carro.getTipoTransmision();
35
36         String nombreCategoria = carro.getCategoria().getNombre();
37         String nombreSede = carro.getSede().getNombre();
38         String estado = carro.getEstado();
39         String fechaDisp = String.valueOf(carro.getFechaDispCons());
40         String str = placa + ";" + marca + ";" + modelo + ";" + color + ";" + tipoTr
41             + nombreCategoria + ";" + nombreSede + ";" + estado + ";" + fechaDis
42             + carro.getTipo()+";"+carro.getPorcentajeRiesgoPrima();
43         return str;
44     }
45 }
46 }
```

Como se puede ver en las anteriores imágenes, se incorporaron las 2 nuevas características de los carros al sistema, como atributos de la clase Carro y en consecuencia de sus instancias. Para esto se modificaron los archivos de prueba existentes y las clases writer y reader encargadas de la persistencia. Ahora el sistema maneja también estos datos por cada carro, y estos nuevos atributos son accesibles donde se requiera, como indica el nuevo requerimiento. De hecho, como se ve en la imagen de la ventana para consultar disponibilidad presentada anteriormente, estos 2 atributos aparecen como parte de la descripción de los carros disponibles. (Note que el requerimiento no pedía explícitamente nada más allá de eso, luego se está cumpliendo a cabalidad).

Pruebas de integración de la creación de reservas:

```
Carro.java Reader.java Writer.java ReservaTestIntegracion.java ControllerCliente.java x
53 }
54 public double crearReserva(String nombreCategoria, String sedeRec,
55     String timeReco, String sedeFin, String timeFin) {
56     HashMap<String,Carro> mapaCarros=datos.getMapaCarros();
57     //Vamos a iterar el inventario hasta encontrar el primer
58     //carro que cumple las características y lo vamos a reservar
59     // si se hace la reserva retornamos true, si se itera toda la lista
60     //sin éxito retornamos false
61     LocalDateTime fechaPed1=LocalDateTime.parse(timeReco);
62     LocalDateTime fechaPed2=LocalDateTime.parse(timeFin);
63
64     for(Carro carro:mapaCarros.values()) {
65         LocalDateTime fechadisp=carro.getFechaDispCons();
66         // Descartamos por no ser de la categoría
67         if (carro.getCategoria().getNombre().equals(nombreCategoria)==false) {
68             continue;
69         }
70         if(fechadisp!=null && fechadisp.plusDays(2).isAfter(fechaPed1)) {
71             continue; //descartamos el carro por fecha disponibilidad
72         }
73         if (carro.getUsoActual()!=null) {
74             LocalDateTime entregaAlquiler=carro.getUsoActual().getFechaDeb();
75
76             if(entregaAlquiler.isAfter(fechaPed1)) {
77
78                 continue; //descartamos el carro por estar alquilado
79             }
80         }
81         if(hayReservasEnIntervalo(carro, fechaPed1, fechaPed2)==true) {
82             continue;
83         }
84         if(cliente.getTarjeta().getBloqueo()==false) {
```

```

Carro.java Reader.java Writer.java ReservaTestIntegracion.java ControllerCliente.java ×
81 All 2 branches covered. reservasEnIntervalo(carro, fechaPed1, fechaPed2) == true) {
82     continue;
83 }
84 if(cliente.getTarjeta().getBloqueo() == false) {
85     Categoria categoria = datos.getMapaCateg().get(nombreCategoria);
86     Sede sede1 = datos.getMapaSedes().get(sedeRec);
87     Sede sede2 = datos.getMapaSedes().get(sedeFin);
88     System.out.println("Reservas " + Reserva.numeroReservas);
89     reserva = new Reserva(cliente, fechaPed1, fechaPed2,
90         categoria, carro, sede1, sede2, "0");
91     //Pongo reserva en mapa reservas
92     System.out.println("Reservas " + Reserva.numeroReservas);
93     String idReserva = String.valueOf(reserva.getNumReserva());
94     datos.getMapaReservas().put(idReserva, reserva);
95     //poner reserva en carro
96     carro.agregarReserva(reserva);
97     //guardar carro actualizado
98     datos.getMapaCarros().replace(carro.getPlaca(), carro);
99     //generar bloqueo de tarjeta
100    cliente.getTarjeta().bloquear();
101    //calcular tarifa, a partir de la fija en el día previsto para el alquiler
102    double tarifaCateg = categoria.tarifaCat();
103    //calcular tarifa a partir de temporada
104    Temporada temp = encontrarTemporada(fechaPed1);
105    double tarifaTemp = temp.getTarifaTemporada();
106    //calcular diferencia en días
107    long diffDays = ChronoUnit.DAYS.between(fechaPed1, fechaPed2);
108    double difDias = (double) diffDays;
109    return difDias * (tarifaTemp + tarifaCateg) * 0.3;
110 }
111 return 0;
112 }

```

Como se puede ver en las imágenes, al correr las pruebas de integración (en la clase ReservaTestIntegracion) se cubre toda la función de crearReserva, y se incluyen todos los casos de error o excepción que podrían haber para comprobar que estén bien manejados. Las pruebas consisten en 10 validaciones que prueban todos y cada uno de los casos posibles que se presentan al intentar crear una reserva (ej. que la tarjeta no pase, que el carro esté alquilado, que todos los carros de la categoría estén reservados o que no haya peros y la reserva se pueda realizar exitosamente).

```

62 double cobro1 = elCliente.crearReserva("familiares", "sur", "2024-01-01T00:00:00",
63     "sur", "2024-02-01T00:00:00");
64 assertEquals(0, cobro1);
65 assertEquals(true, elCliente.getCliente().getTarjeta().getBloqueo());
66 assertEquals(1, datos.getMapaReservas().size());
67
68 // prueba de que no dejará reservar si desbloqueamos la
69 // tarjeta otro carro lujoso en las mismas fechas
70 elCliente.getCliente().getTarjeta().desbloquear();
71 double cobro2 = elCliente.crearReserva("lujoso", "sur", "2024-01-01T00:00:00",
72     "sur", "2024-02-01T00:00:00");
73
74 assertEquals(0, cobro2);
75 assertEquals(1, datos.getMapaReservas().size());
76
77 // Sin embargo si dejará crear reserva de otro carro
78 // como de familiares, que hay muchos
79 double cobro3 = elCliente.crearReserva("familiares", "sur", "2024-01-01T00:00:00",
80     "sur", "2024-02-01T00:00:00");
81
82 assertEquals(95790, cobro3);
83 assertEquals(2, datos.getMapaReservas().size());
84
85 // prueba de que no dejará reservar si reserva interseca intervalo por :
86 elCliente.getCliente().getTarjeta().desbloquear();
87 double cobro4 = elCliente.crearReserva("lujoso", "sur", "2024-01-05T00:00:00",
88     "sur", "2024-02-08T00:00:00");
89
90 assertEquals(0, cobro4);
91 assertEquals(2, datos.getMapaReservas().size());

```

Como se ve en la imagen, cada una de las validaciones consiste en 3 pasos, primero se filtra y se deja listo el escenario con lo que se quiere probar, luego se llama la función de crear la reserva y luego se verifica su resultado con el resultado esperado. Nótese que se cubren todos los casos reutilizando al máximo los escenarios para disminuir la cantidad de pruebas, incluyendo los casos de excepción donde no se debe permitir crear la reserva pero tampoco se debe caer el programa. Se cumple a cabalidad con el requerimiento.

Pruebas unitarias para cargar los archivos:

NOTA: El enunciado no dice probar el manejo completo de la persistencia, únicamente la carga, que se entiende por cargar los datos del sistema a los archivos de persistencia (lo contrario sería descargar).

En estas pruebas, se sigue un esquema muy similar al de las de integración para la creación de reservas, pero antes de cada prueba creamos un backup de los datos que habían en el archivo para que al final de la prueba el archivo original vuelva a reescribirse a partir del dicho backup y termine inalterado (para esto usamos un `before all` y un `after all`). Para los test primero cuadramos el escenario, luego llamamos el método correspondiente a reescribir el archivo que estamos probando para luego confirmar con `assert` que el texto que se escribió si era el esperado, todo después de `before all` y antes del `after all`.

A continuación se muestra un ejemplo de las licencias:


```
Carro.java Reader.java Writer.java ReservaTestIn... ControllerCli... clienteTest.j... *licenciaTest... >
1 package pruebasCargaDatos;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4
21
22 public class licenciaTest {
23     private static final String TEST_FILE_PATH = "data/licencias.txt";
24     private static final String BACKUP_FILE_PATH = "backup/back.txt";
25
26     @BeforeAll
27     public static void setup() throws IOException {
28         // Hacer una copia del archivo original antes de la prueba
29         copyFile(TEST_FILE_PATH, BACKUP_FILE_PATH);
30     }
31
32     @AfterAll
33     public static void cleanup() throws IOException {
34         // Restaurar el archivo original después de la prueba
35         copyFile(BACKUP_FILE_PATH, TEST_FILE_PATH);
36         // Eliminar el archivo de respaldo
37         new File(BACKUP_FILE_PATH).delete();
38     }
39
40     @Test
41     public void testLicenciaToFile() throws IOException {
42         BaseDatos datos = new BaseDatos();
43         Licencia lic1 = new Licencia("1", "USA", "10/26", "www.mariosanchez");
44         Licencia lic2 = new Licencia("2", "Peru", "9/26", "www.josue");
45         datos.getMapaLicencias().put("1", lic1);
46         datos.getMapaLicencias().put("2", lic2);
47
48         datos.actualizarArchivoLicencias();
49     }
}
```

```
Carro.java Reader.java Writer.java ReservaTestIn... ControllerCli... clienteTest.ja... *licenciaTest.j... ×
Maximize
50     datos.actualizarArchivoLicencias();
51
52     //Assert
53     String expectedText = "10/26;1;USA;www.mariosanchez;-\\n"
54     + "9/26;2;Peru;www.josue;-\\n";
55     String fileContent = readFile(TEST_FILE_PATH);
56     assertEquals(expectedText, fileContent);
57 }
58
59 private static void copyFile(String sourcePath, String destinationPath) throws IOE
60 {
61     // Copiar el archivo
62     try (InputStream in = new FileInputStream(source);
63          OutputStream out = new FileOutputStream(destination)) {
64         byte[] buffer = new byte[1024];
65         int length;
66         while ((length = in.read(buffer)) > 0) {
67             out.write(buffer, 0, length);
68         }
69     }
70 }
71
72 private String readFile(String filePath) throws IOException {
73     StringBuilder content = new StringBuilder();
74     try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
75         String line;
76         while ((line = reader.readLine()) != null) {
77             content.append(line).append("\\n");
78         }
79     }
80     return content.toString();
}
```

La prueba cubre acertadamente la totalidad de los métodos encargados de la sobreescritura del archivo, como se muestra a continuación.



```
licenciaTes... tarjetaTest... TemporadaTes... TestEmpleado... TestSeguro.java Writer.java × »10
100 String codigo = tarjeta.getCodigo();
101 String bloqueada = "0";
102 if (tarjeta.getBloqueo() == true) {
103     bloqueada = "1";
104 }
105 return numero + ";" + codigo + ";" + bloqueada;
106 }
107
108 // SEPTIMO OBJETO LICENCIA
109 public String comprimirLicencia(Licencia licencia) {
110     String fechaVens = licencia.getFechaVens();
111     String numero = licencia.getNumero();
112     String pais = licencia.getPais();
113     String rutaImagen = licencia.getRutaImagen();
114     String idAlquiler = licencia.getIdAlquiler();
115
116     return fechaVens + ";" + numero + ";" + pais + ";" + rutaImagen + ";" + idAlquiler;
117 }
118
119
licenciaTes... TestEmpleado... TestSeguro.java Writer.java BaseDatos.java × »12
203 Licencia licencia=reader.descomprimirLicencia(linea);
204 mapaLicencias.put(numero, licencia);
205 linea = br.readLine();
206 }
207 br.close();
208 }
209 //Write: Actualizar archivo, reescribirlo.
210
211 private String generarTextoLicencias() {
212     String texto="";
213     for(Licencia licencia:mapaLicencias.values()) {
214         texto+=writer.comprimirLicencia(licencia);
215         texto+="\n";
216     }
217     return texto;
218 }
219 public void actualizarArchivoLicencias() throws IOException {
220     String texto=generarTextoLicencias();
221     FileWriter fichero = new FileWriter("data/licencias.txt");
222     fichero.write(texto);
223     fichero.close();
224 }
225
```

Esto es así para el resto de las pruebas (puede comprobar esto corriendolas) luego la cobertura es del 100%, acorde a lo pedido.