

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:

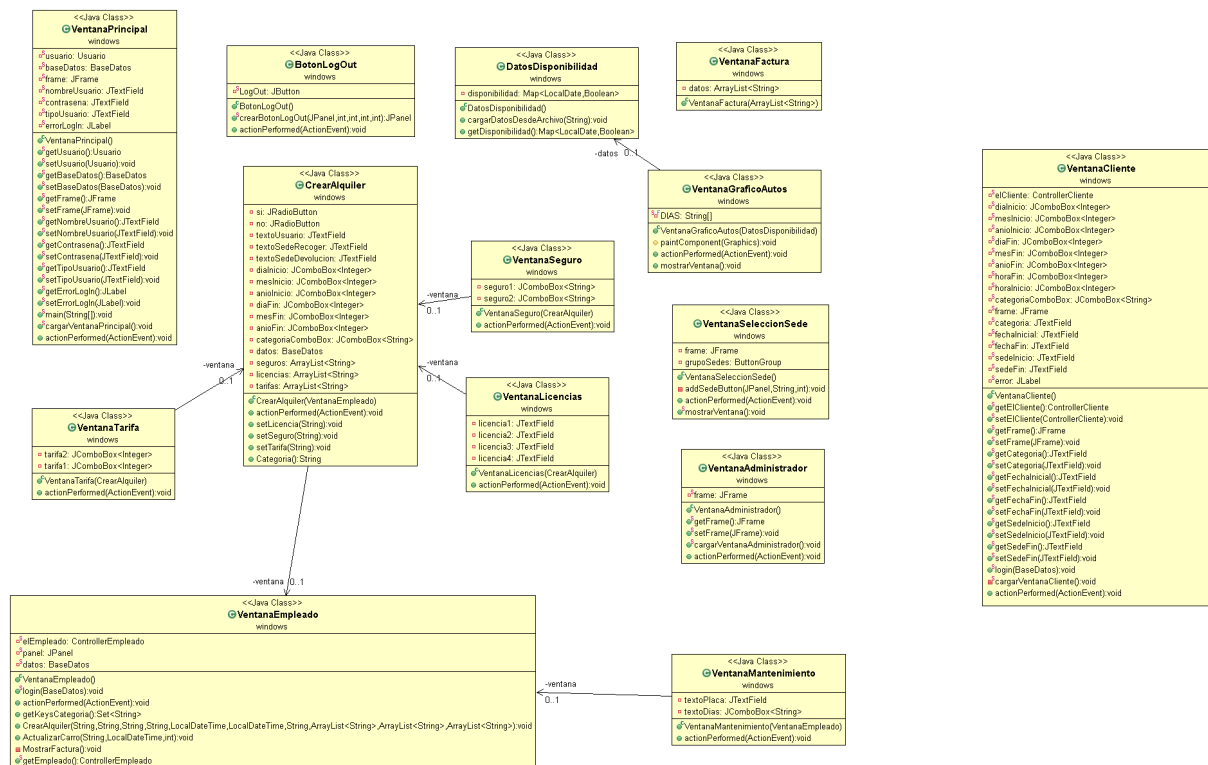
La persistencia se maneja conjuntamente entre las clases *BaseDatos*, *Reader* y *Writer*, y a través de archivos .txt. Hay un archivo txt por cada tipo abstracto de datos definido en el modelo, (ej carro, sede). Cada vez que un usuario cierra sesión, se reescriben los archivos que enlistan las instancias 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.

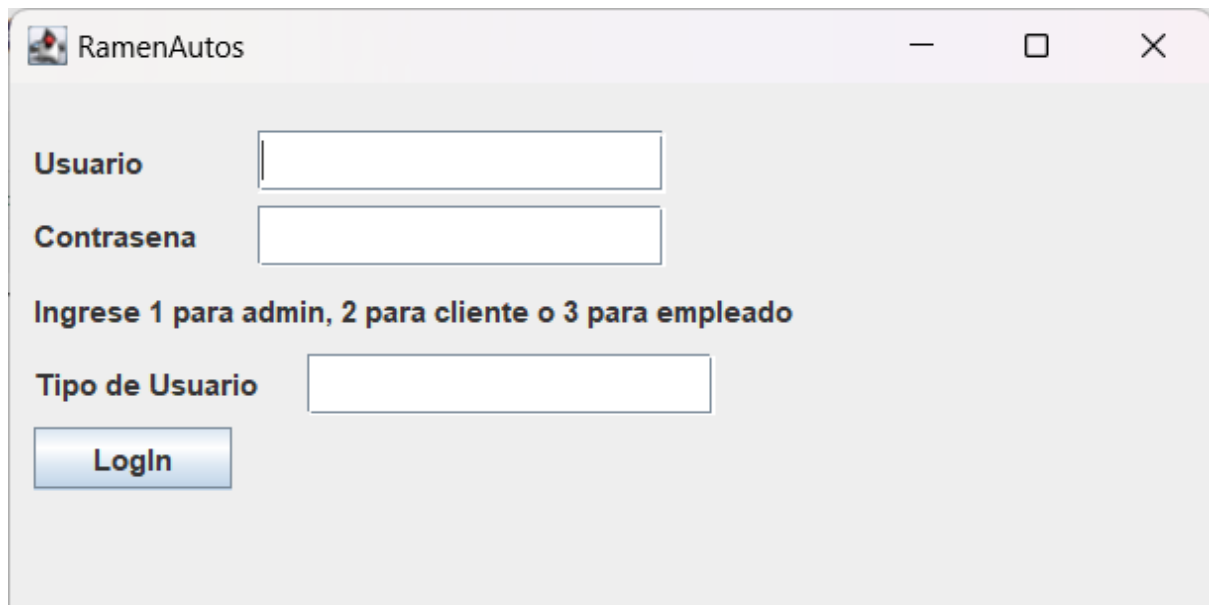
Cada vez que se inicializa la aplicación se descargan los datos, se crean los objetos y se añaden estos a los mapas. Se descargan todas las instancias de algún objeto y se almacenan en el map correspondiente antes de pasar al siguiente. Para esto se utiliza la clase *reader*, que a partir de las String 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:





RamenAutos

Usuario

Contraseña

Ingrese 1 para admin, 2 para cliente o 3 para empleado

Tipo de Usuario

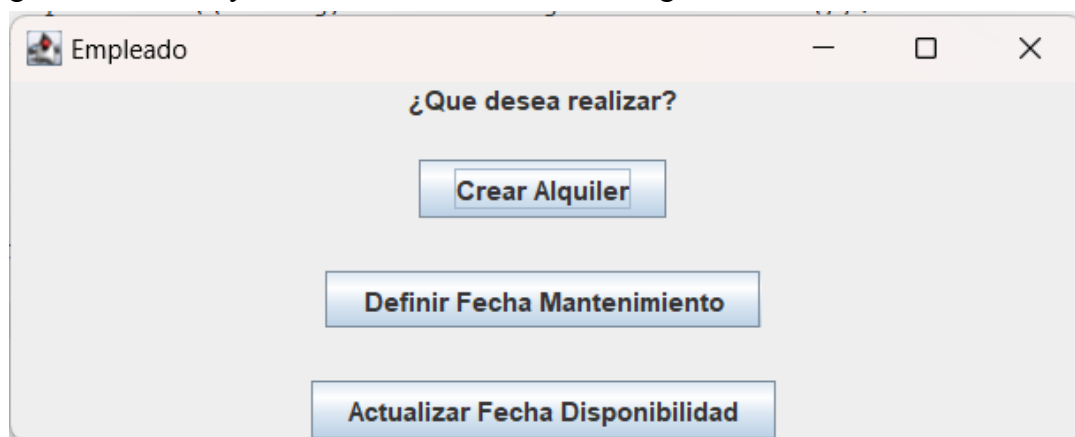
Login

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:



Empleado

¿Que desea realizar?

Crear Alquiler

Definir Fecha Mantenimiento

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' (selected).
- Cliente (usuario):** Text input with 'daniela'.
- Sede Recoger:** Text input with 'norte'.
- Sede Devolucion:** Text input with 'norte'.
- FechaInicio:** Date picker showing 2/1/2023.
- FechaFin:** Date picker showing 4/1/2023.
- Categoria:** Dropdown menu with 'familiares' selected.
- 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.

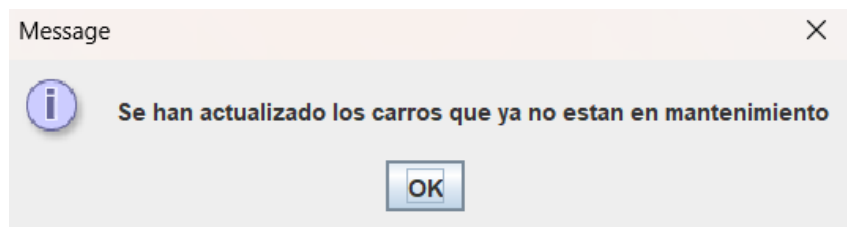
En esta ventana nos aparecen las opciones de seguros existentes se puede poner información en todas las opciones o dejar las dos en blanco o solo tener un espacio con información. La ventana de licencias tiene la siguiente visualización:

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:

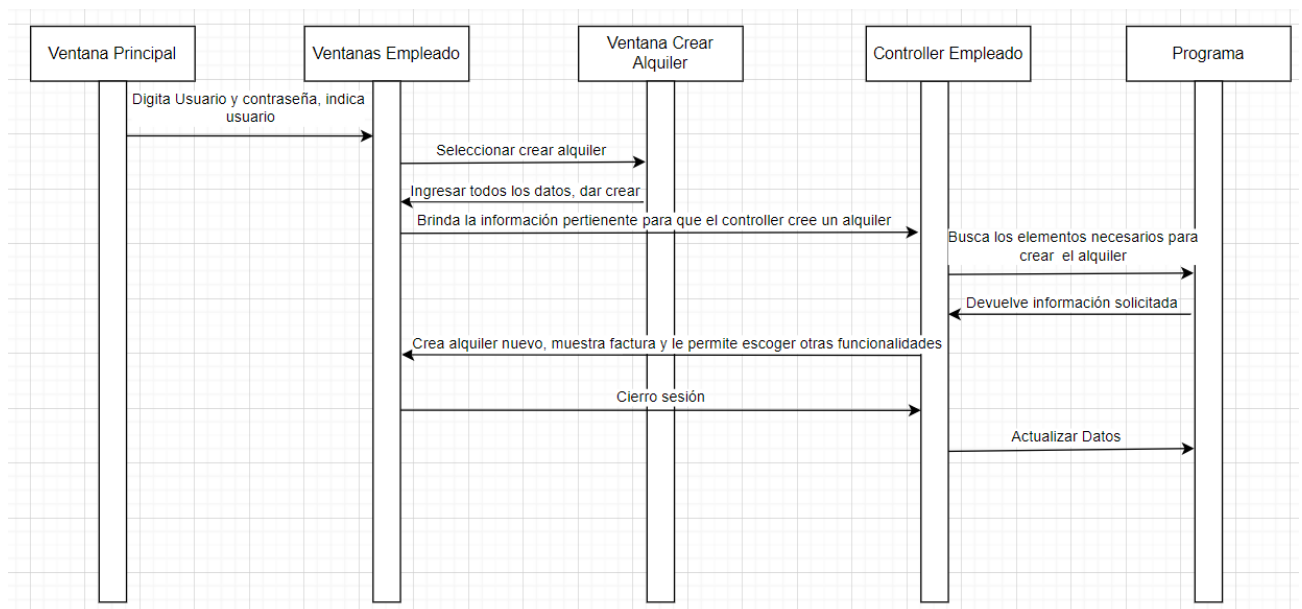
Esta ventana es muy similar a la de seguros, sin embargo aquí aparecerá el id de cada una de estas dependiendo de la categoría seleccionada en la anterior ventana de crear reserva. Después de haber añadido todo lo necesario para crear un alquiler se debe presionar el botón “Crear” y este te muestra una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

Factura	
idFactura:	1
Pago Anticipado:	0.0
Precio Licencias:	0.0
Total:	700.0

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.

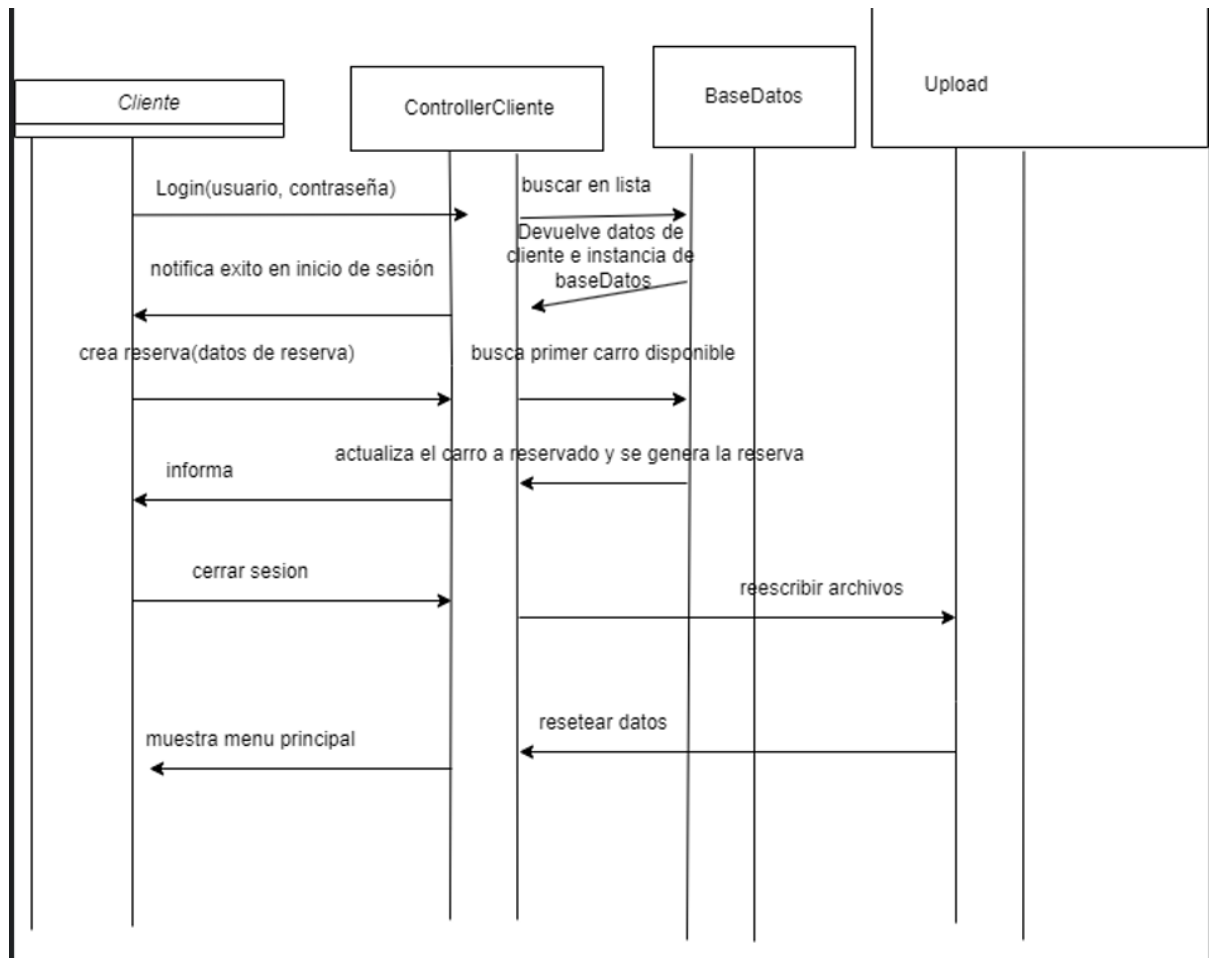


Para finalizar sesión solo es necesario que se oprima la x en la ventana del empleado.



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%.

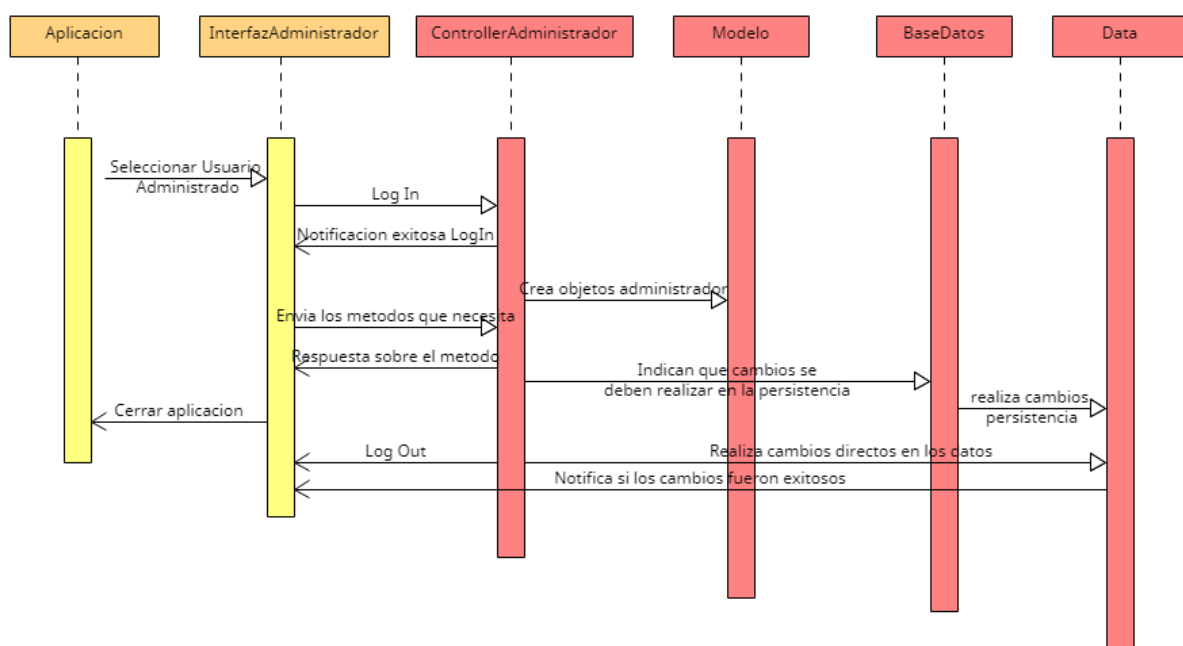
Nótese que el enunciado solo dice que el cliente no debe saber qué carro se le va a entregar, pero esto no implica que la reserva no pueda asociarse a un carro desde el momento en el que

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:

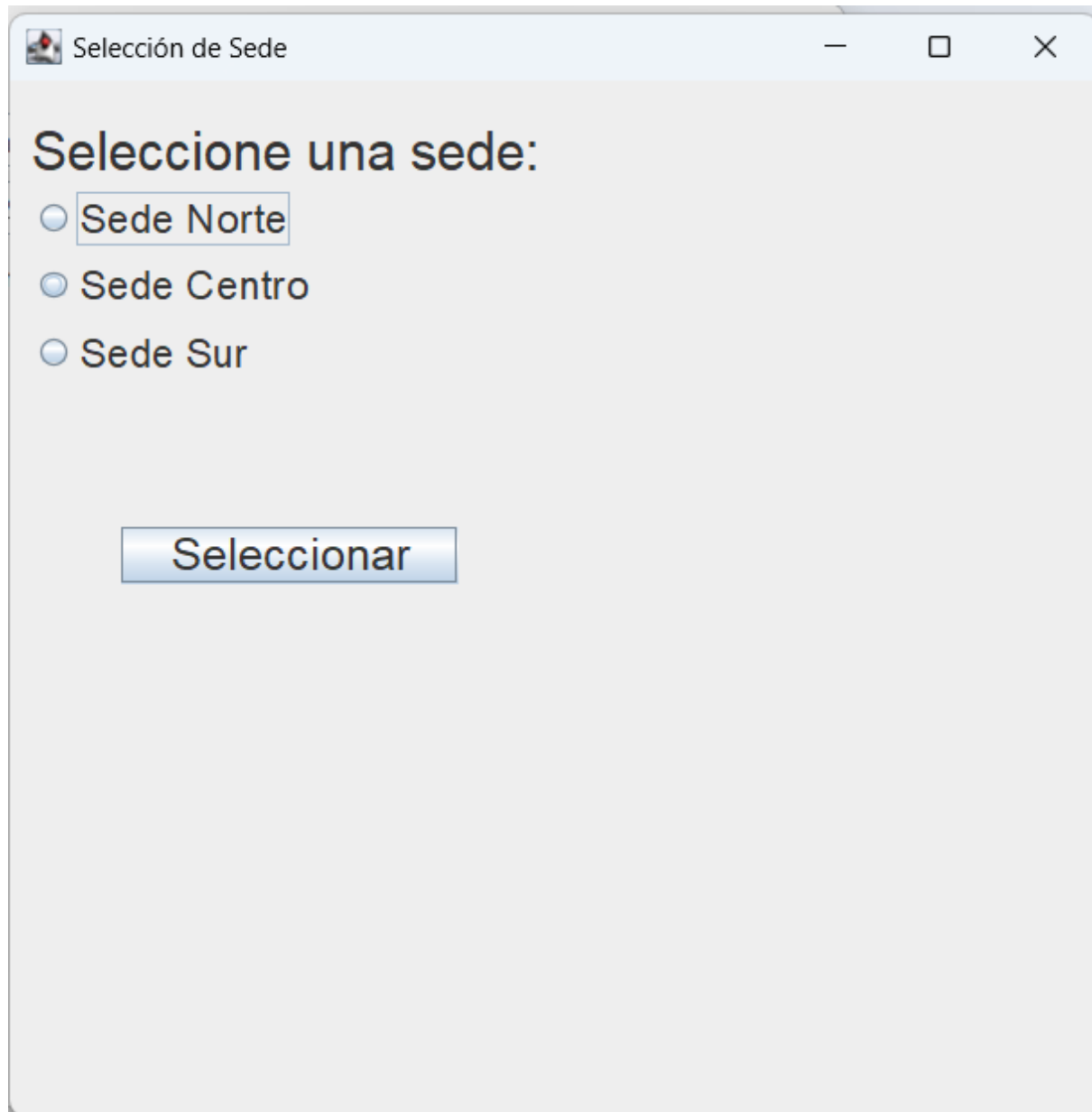


El administrador se encarga de gestionar los vehículos , en dismantelar los que ya no se necesitan así como en registrar los nuevos, hay que tener en cuenta que un administrador pertenece a solo una Sede por lo tanto y una Sede solo puede tener un administrador que también tiene la tarea de crear nuevos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

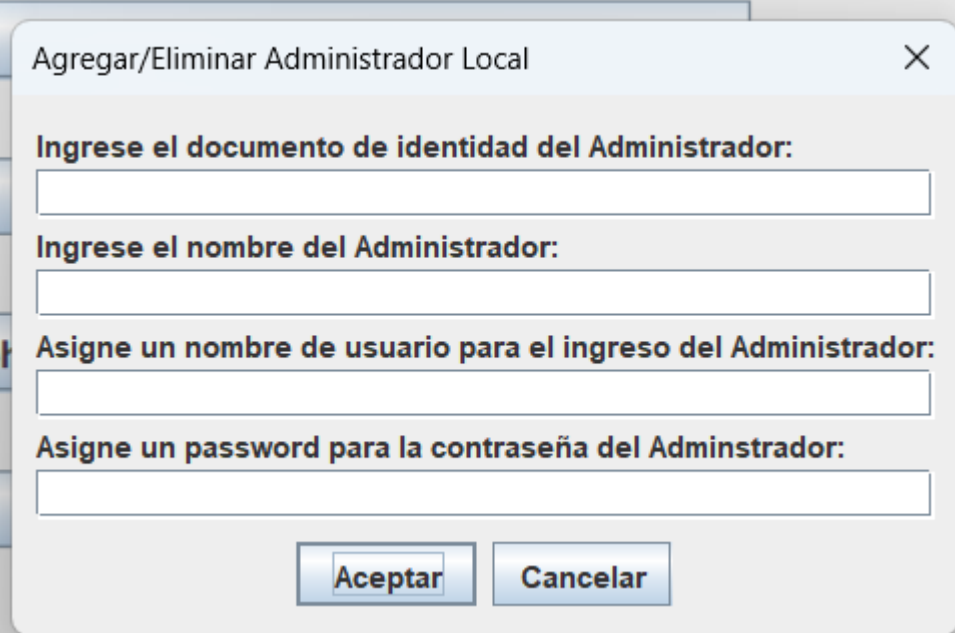
Agregar/Eliminar Empleado

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

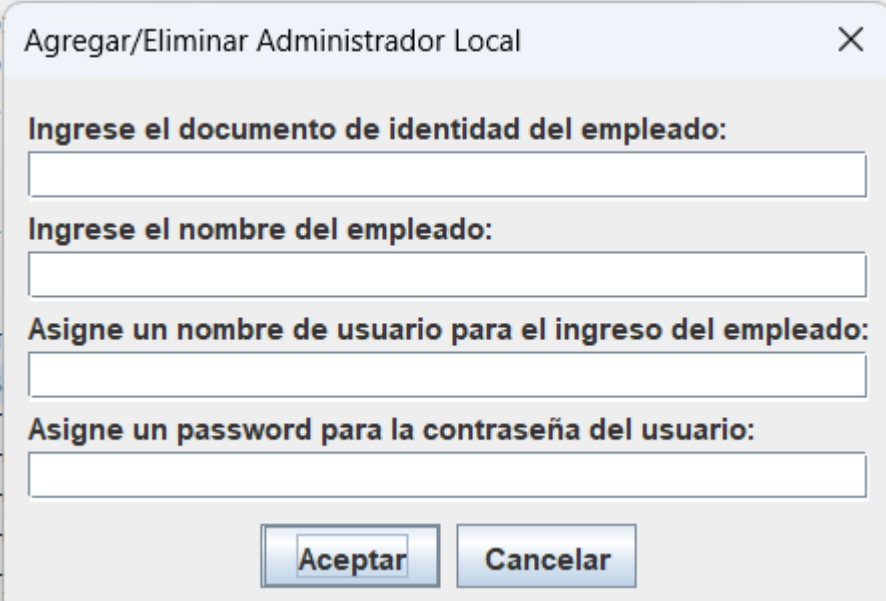
Ingrese el documento de identidad del Administrador:

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**

y la opción de la gráfica de autos disponibles :

