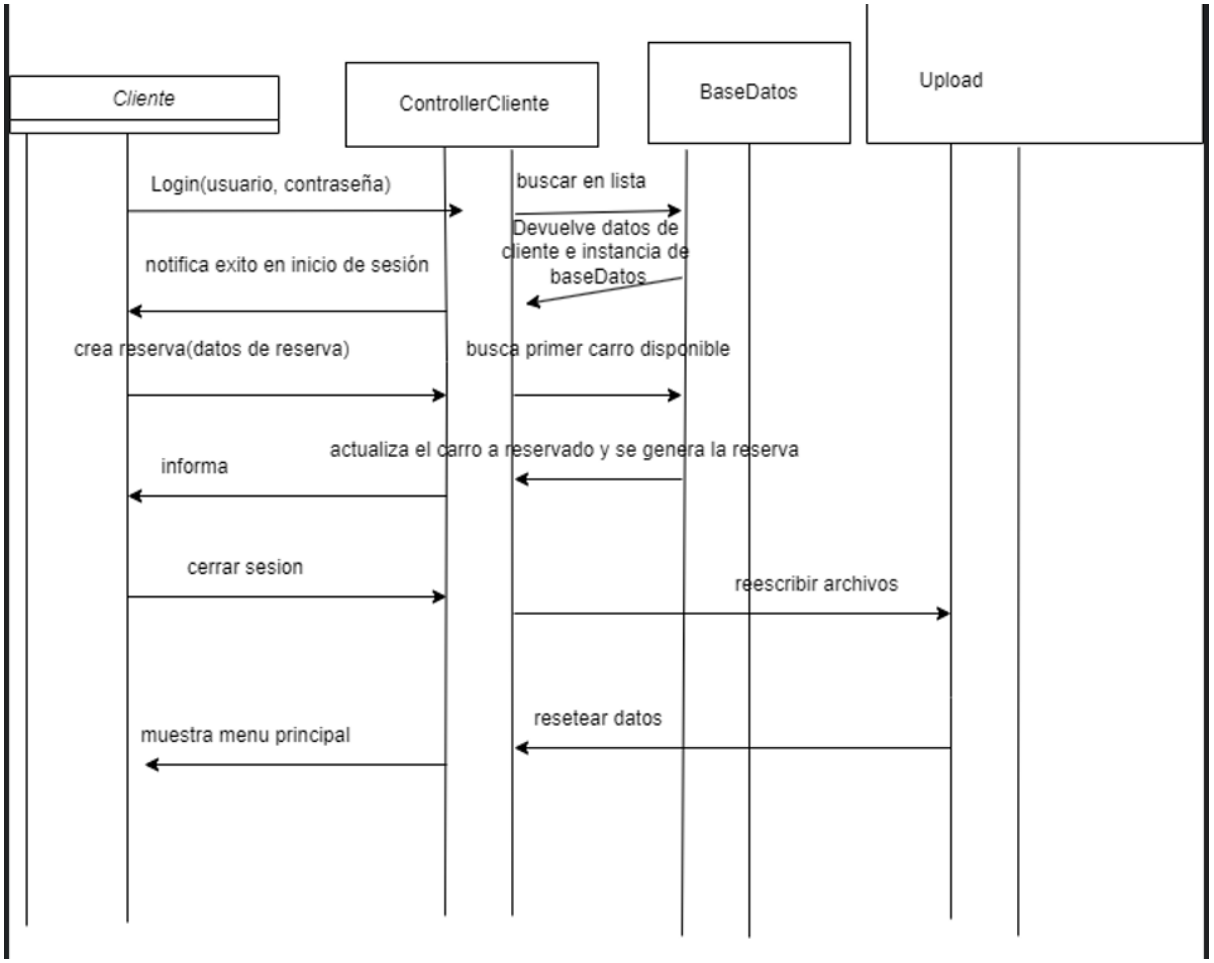
**Documento de diseño**

**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 cede). 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 estan 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.

**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.

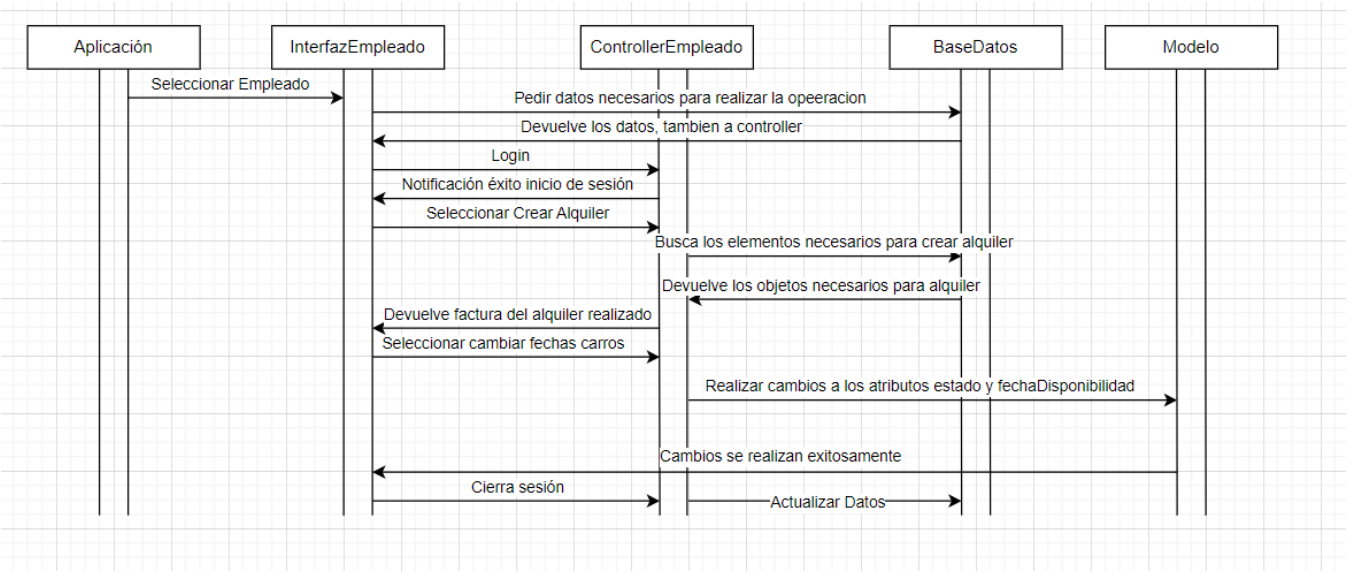
Al inicializar la aplicación, esta descargará todos los archivos cargando los datos y le pregunta al usuario qué tipo de usuario es, y dependiendo de lo que responda se utiliza la interfaz correspondiente. Si seleccionó cliente, le preguntará a través de menú si desea iniciar sesión, crear una reserva o salir del programa (nótese que si selecciona cualquiera de las dos últimas sin haber iniciado sesión, se le comunicará al usuario que esto no se puede y se le mostrará nuevamente el menú para que inicie sesión. Si selecciona iniciar sesión se le preguntará su usuario y contraseña, y con estos parámetros se llamará al controller que creará una instancia de la BaseDatos y buscará en ella si el login coincide con la contraseña. Si este usuario no existe o su contraseña no coincide, se le informa al cliente y se mostrará de nuevo el menú del cliente. Sí sí coincide, se almacenará el cliente en la instancia creada del controller, que contiene al cliente y la base de datos. En este punto, si el cliente selecciona crear reserva, se le preguntará por la categoría que desea, el intervalo de tiempo de la misma en formato LocalDateTime toString 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 cerrar sesión, se reescriben todos los archivos de persistencia y se cierra la interfazCliente, volviendo al menú principal.

**Historia de Usuario Empleado:**



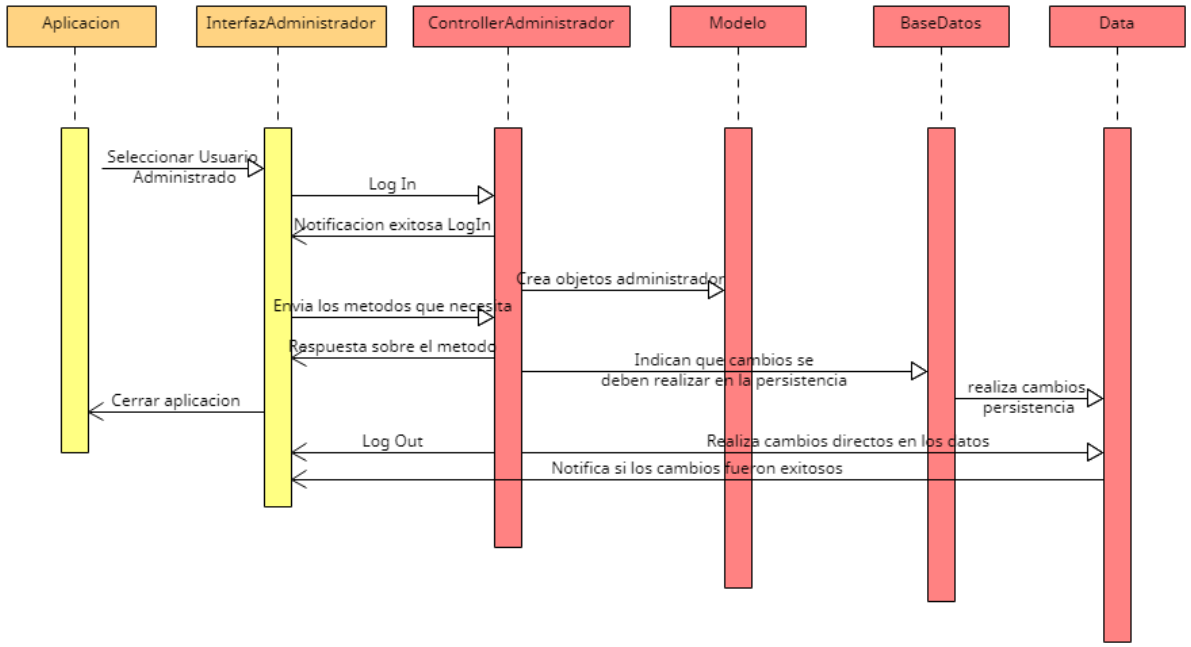

El empleado necesita poder realizar dos cosas: crear un alquiler y poder determinar la fecha en la que un carro este disponible después de realizar limpieza o mantenimiento, si es necesario. Adicionalmente, es importante tener en cuenta que cada uno de ellos necesita un login y contraseña para poder acceder al sistema.

Al comienzo de la aplicación es necesario que se selecciones la primera opción para poder cargar todos los datos. Posteriormente, se debe seleccionar el número 3 para poder acceder a la interfaz del empleado. El usuario debe ingresar su login y contraseña, si estos son correctos podrá utilizar las demás funcionalidades, de lo contrario deberá ingresar de nuevo estos atributos. La opción dos es para crear un alquiler en este preguntan los datos como login del usuario, sede a recoger, devolver, fecha para recoger y devolver el vehículo , en el formato de (LocalDateTime) y si existe una reserva, si esta existe se utiliza una función diferente para crear el alquiler en el controllerEmpleado, en esta solo se busca la reserva del cliente y se añade los valores necesarios al alquiler que se está creando; si esta no existe es necesario buscar en los carros de la categoria deseada la disponibilidad y después de encontrar el carro que se ajusta a las necesidades del cliente el alquiler se crea, sino es posible crear un alquiler esto se le informa al empleado. Luego si hay un alquiler se genera un texto de la factura de nuevo alquiler que se mostrará en la consola.

Para actualizar las fechas de disponibilidad de un carro se tienen dos opciones. La primera es que al empleado le esten entregando un carro y según su criterio digita la cantidad de días

(máximo dos días) necesarios para que el carro se encuentre en buen estado. El encargado de hacer estos cambios a la fecha del carro es el Controller. La segunda es cuando se quiere mirar si los carros ya están disponibles para ser reservados y se cambia su fecha de disponibilidad a null, cuando su fecha de disponibilidad es antes de la fecha de hoy. Después de realizar las funciones que deseaba el usuario, este puede escoger salir del programa, cuando se realice esto los datos se actualizarán en los archivos dentro del proyecto.

**Historia de Usuario Administrador:**




El administrador se encarga de gestionar los vehículos , en desmantelar 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,