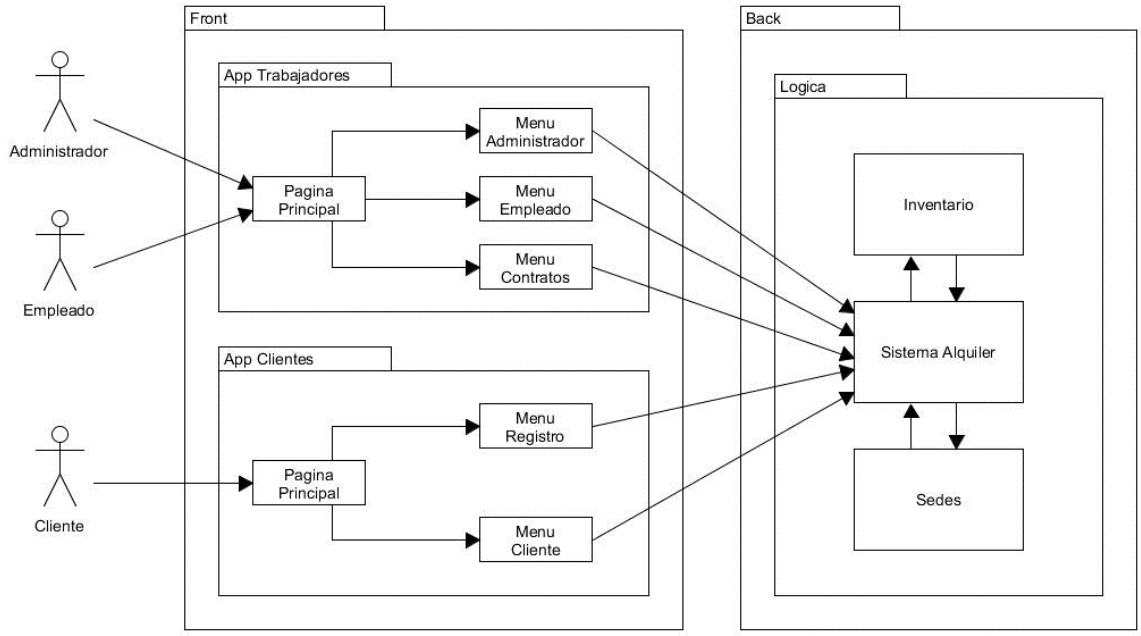
Documento de diseño

Entendimiento de los requerimientos funcionales:



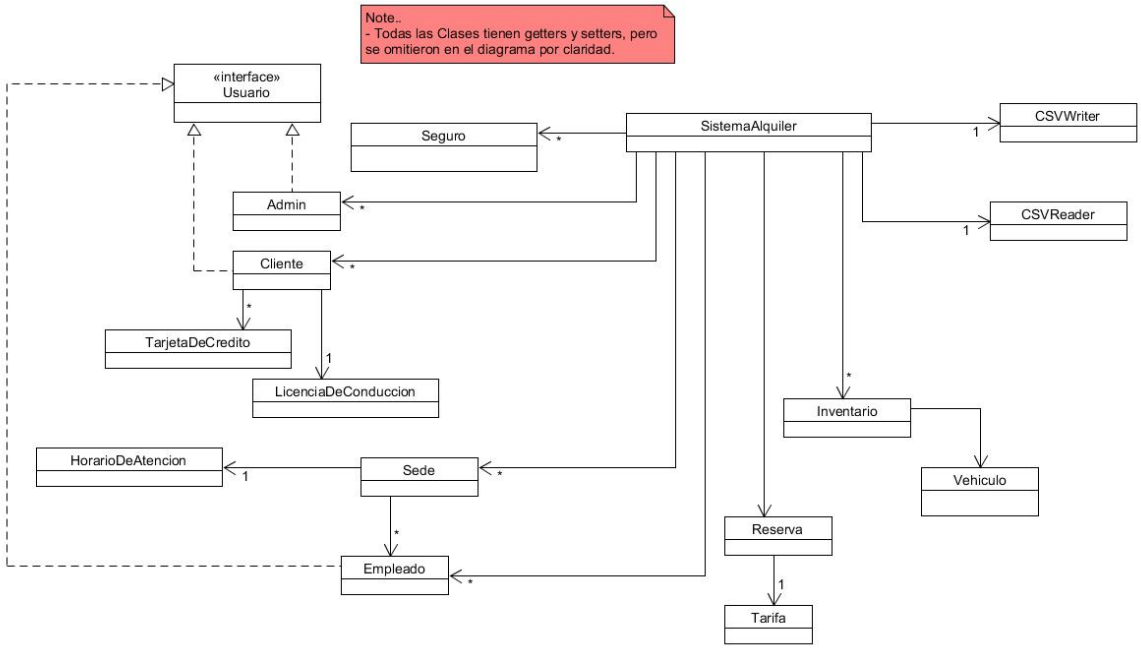
Antes de establecer un diagrama formal del diseño de la solución se definen funcionalidades de alto nivel que el programa debe satisfacer para los diferentes tipos de usuarios (Administradores, clientes, empleados). Este diagrama es útil para comprender los requerimientos funcionales independiente de cómo sea la implementación.

Para esta iteración del proyecto se crearon dos Frontends conectados a un mismo Backend. Por ende, los administradores y empleados ingresan a una misma aplicación en la cual se da la opción de iniciar sesión y acceder a su menú respectivo. Los clientes ingresan a otra aplicación diferente donde pueden registrarse e iniciar sesión para acceder a su menú.

En la aplicación de clientes se permite el registro de un nuevo cliente con fin de crear un nuevo usuario, además se puede evaluar la disponibilidad de los diferentes vehículos en las sedes para las fechas que el cliente selecciona y por último se permite realizar la reserva del vehículo.

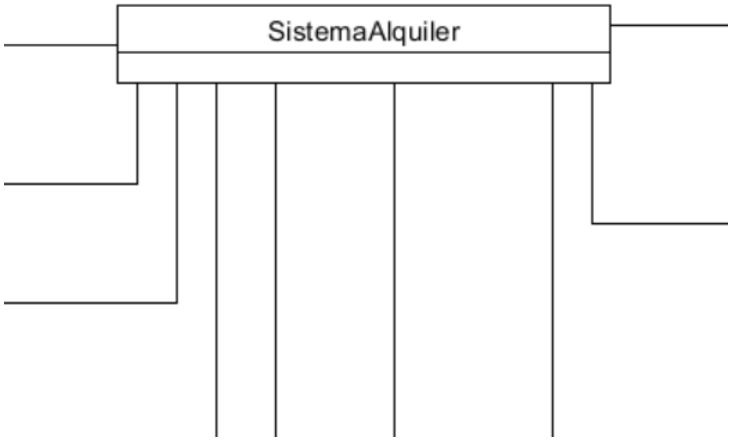
En la aplicación de administradores y empleados se puede iniciar sesión como administrador general, administrador local y empleado. Los administradores generales tienen la opción de añadir o eliminar vehículos del inventario; crear y gestionar los seguros; y gestionar las sedes. Por su parte, los administradores locales pueden registrar, añadir y eliminar empleados de su sede. Y en el caso de empleados se permite formalizar y terminar un alquiler y realizar un alquiler especial para trasladar un vehículo, además los empleados de mantenimiento deben poder actualizar el estado de un vehículo. De esta forma, tiene sentido que haya dos aplicaciones dependiendo del tipo de usuario, y que en cada una se muestre el menú acorde a las funcionalidades que el sistema les debe brindar.

Entendimiento de las relaciones entre clases y respectivos roles (Lógica del negocio)



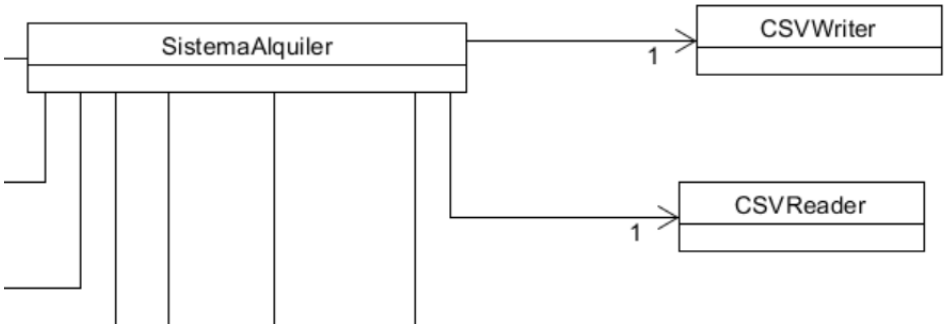
La figura anterior es un diagrama de alto nivel que propone la solución que se utilizó en este proyecto. En este se puede evidenciar que se optó por un sistema centralizado. Las siguientes partes del documento buscan explicar y fundamentar porque se eligió esta solución al problema planteado.

Bajo el modelo de responsability driven design se define lo siguiente:

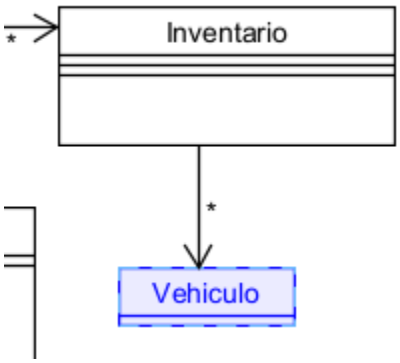


La clase SistemaAlquiler es el principal controlador del sistema. Por su parte, SistemaAlquiler le da sentido a los inputs que recibe de la interfaz gráfica (la cual se explica más adelante) y se los pasa a diferentes métodos de otras clases (que serán designadas más adelante) con el fin de que estos solucionen algún requerimiento funcional. Esto se basa en el patrón de diseño “Fachada” el cual proporciona una interfaz única y centralizada que dirige a métodos de clases más especializadas. al distribuir las tareas entre las clases correspondientes permite ocultar la complejidad interna del

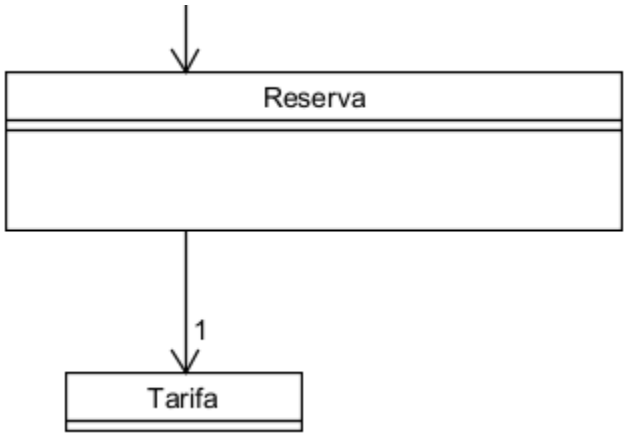
sistema y facilitar la coordinación y resolución de los diferentes requerimientos funcionales, lo cual mejora la claridad del código, hace más fácil el mantenimiento y fomenta un sistema más modular y desacoplado.



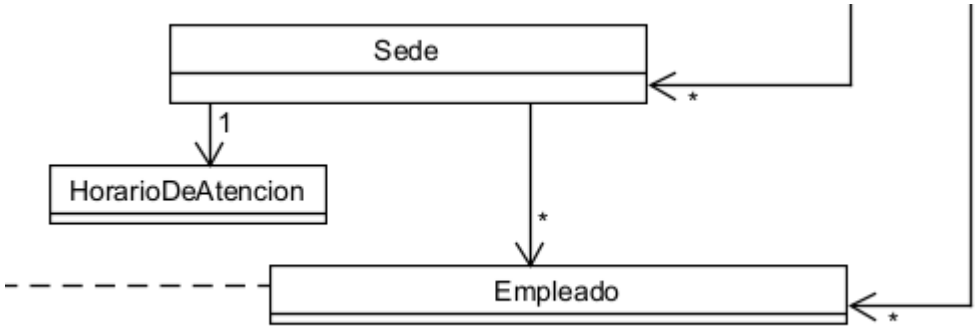
Las clases CSVWriter y CSVReader permiten hacer la persistencia de los datos del sistema. Como sus nombres indican, una de ellas se encarga de leer diferentes archivos CSV y la otra de escribir archivos CSV.



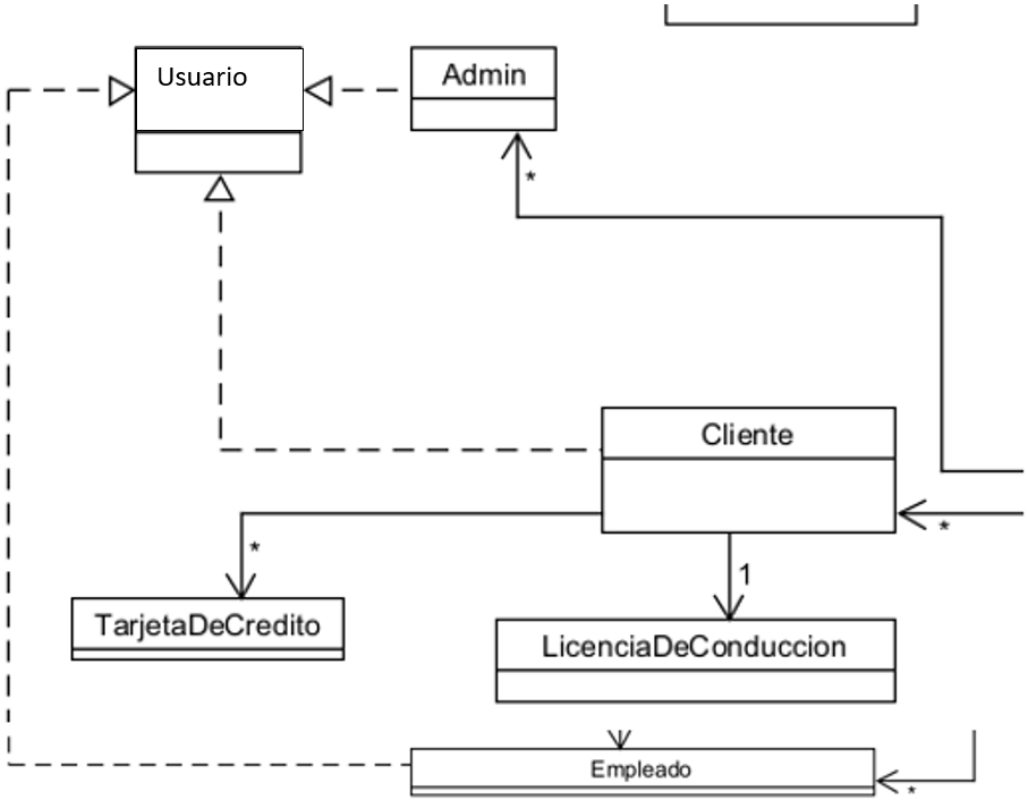
Inventario es una clase encargada de conocer y gestionar el estado de los vehículos del sistema, es aquí donde se pueden cumplir los requerimientos funcionales que permiten a un administrador general añadir y eliminar vehículos. Además, permite que se pueda actualizar el estado de algún vehículo, lo cual es necesario para otros requerimientos funcionales. De esta forma, decimos que inventario actúa principalmente como un service provider, aunque este tambien contiene la información de los vehículos de la empresa. Mientras que vehículo es un information holder que simplemente se encarga de conocer las características de un vehículo y le brinda a inventario la información necesaria para que este pueda completar los requerimientos mencionados anteriormente.



Por otro lado, la clase reserva es la encargada de que un usuario pueda crear una reserva, y que más adelante esta pueda ser formalizada en un alquiler. Esto quiere decir que una reserva pasa a ser un alquiler en el momento en que un empleado asigna un vehículo. Por lo tanto, esta se encarga de solucionar un requerimiento funcional y decimos que es un service provider. Mientras que tarifa es un information holder que se encarga de dar a reserva la información necesaria para que esta pueda calcular su costo.



Así mismo, la clase sede es la encargada de gestionar la informacion asociada a cada sede (la cual es gestionada por un administrador general) y de sus empleados(la cual es manejada por un administrador local). De manera análoga a la reserva esta clase satisface los requerimientos funcionales asociados al manejo de disponibilidad de vehículos en cada sede, por lo cual es un service provider. Mientras que HorarioDeAtencion es análoga a tarifa, por lo que es un information holder.



La anterior figura fue ligeramente modificada respecto a como se muestra en el diagrama completo con el fin de evidenciar a los tipos de usuario que existen en el sistema. En concreto las clases Admin, Cliente y Empleado heredan sus atributos de la superclase Usuario, la cual se asegura que todos puedan iniciar sesión en el sistema y que todos sean considerados Usuario y se pueda acceder a estos como tal. Estas 3 clases son information holders principalmente que indican a que menú accede el usuario dependiendo de su tipo, y por lo tanto limitan las capacidades del usuario según su papel dentro del sistema. Además, las clases TarjetaDeCredito y Licencia de Conducción son information holders que guardan información relevante para el cliente y que es necesaria al momento de crear una reserva.

Es importante discutir los tipos de usuario que actúan en el sistema, pues aunque solo se construyeron 3 clases realmente existen 5 tipos de usuario en total. Por un lado, los administradores se dividen en el administrador general y los administradores locales; la diferenciación de estos en la clase Admin se hace a partir del atributo sede (Si sede == null => admin General, o si sede != null => admin local), así pues en el menú de administrador se muestran 2 menus diferentes dependiendo de si el administrador actual se trata de un administrador general o local. De igual forma ocurre con los empleados, pues estos pueden ser encargados del manejo de alquileres (empleados estándar) o de actualizar el estado de vehículos (empleados de mantenimiento), igual que con los administradores la diferenciación entre estos se logra con el atributo rol, y dependiendo de esto el menú de empleado será diferente.

Entendimiento de las interfaces y su relación con el mundo (Interfaces y GUI)

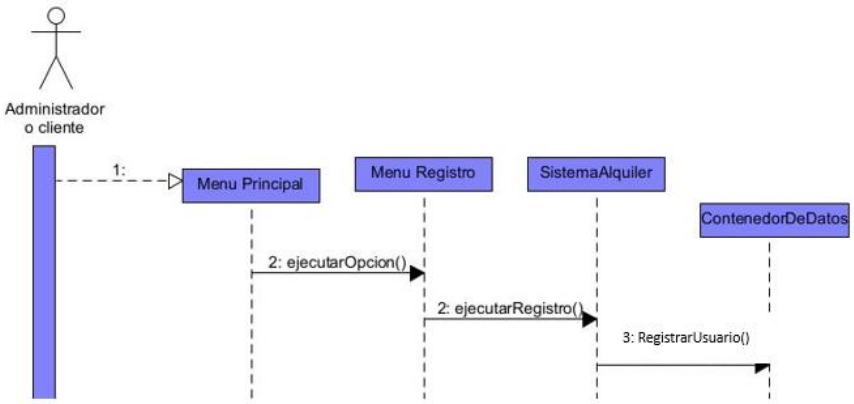
Inicialmente se planeó hacer tres interfaces que interactuaran con el usuario y se encargaran de mostrar las funciones asociadas a cada actor. A lo largo de la implementación se fueron añadiendo paquetes como interfaz, interfaz componentes, interfaz registro e interfaz pagos, el objetivo detrás de esta decisión se centra en reducir las responsabilidades de las clases y reutilizar el código ya hecho para las diferentes interfaces. De este modo Navegador y Pagina Principal funcionan como controllers que se encargan de presentar y redirigir al usuario a diferentes frames de acuerdo a su rol y necesidades. Además, se implementaron clases cuya responsabilidad se centra en el registro e ingreso de usuarios al sistema en el paquete Interfaz registros siendo LandingPage el encargado de controlar a que frame se redirige el usuario de acuerdo a su rol. Igualmente, interfaz pagos tiene como responsabilidad manejar las pasarelas de pagos de diferentes medios de pago. Interfaz componentes tiene como único objetivo definir objetos, widgets, que se usaron en múltiples ocasiones durante la implementación con tal de no repetir el código ya hecho. Finalmente, cada una de las interfaces de usuarios contiene frames que se encargan de suplir las opciones del menú asociado a un actor y sus clases auxiliares, avisos, advertencias, entre otros. Entre estas clases se encuentra un controler llamado menú usuario que conoce al navegador y es finalmente por esta clase donde se muestran las posibles acciones a realizar.

Para las clases que se encargan del manejo de datos, su creación y modificación se estableció una relación entre estas con el Sistema Alquiler.

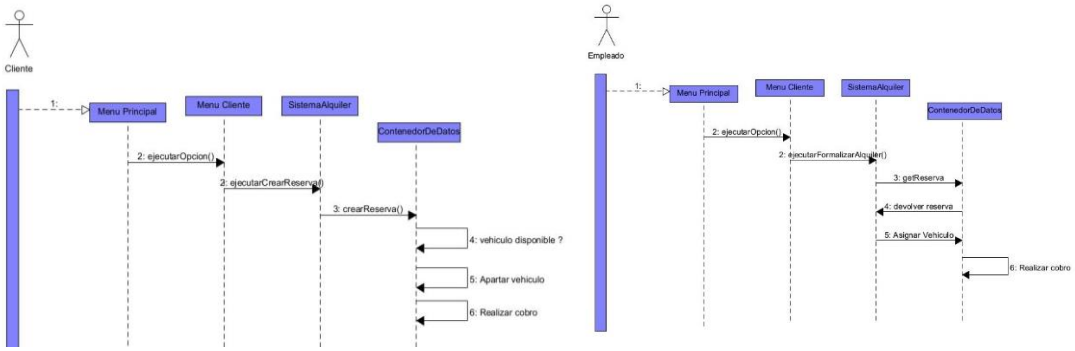
Para la división del programa entre la aplicación empleados y clientes se realizó un singleton, GestorDatos que tiene como atributo el sistema alquiler e implementa métodos del mismo que tienen que ver con las funcionalidades de la app. Por cuestiones de tiempo y facilidad, se decidió utilizar la misma lógica ya implementada sin embargo las clases usadas no heredan directamente de las clases hechas anteriormente, aunque en un principio la idea era que lo hicieran. El objetivo detrás del Singleton es evitar que los datos se dupliquen y garantizar el encapsulamiento.

Diagramas de secuencias de funcionalidades especificas

Para entender mejor la funcionalidad del programa, los diagramas de secuencia son la solución a la hora de representar la interacción entre componentes según el funcionamiento de requerimientos que consideramos críticos en la construcción del proyecto**.**



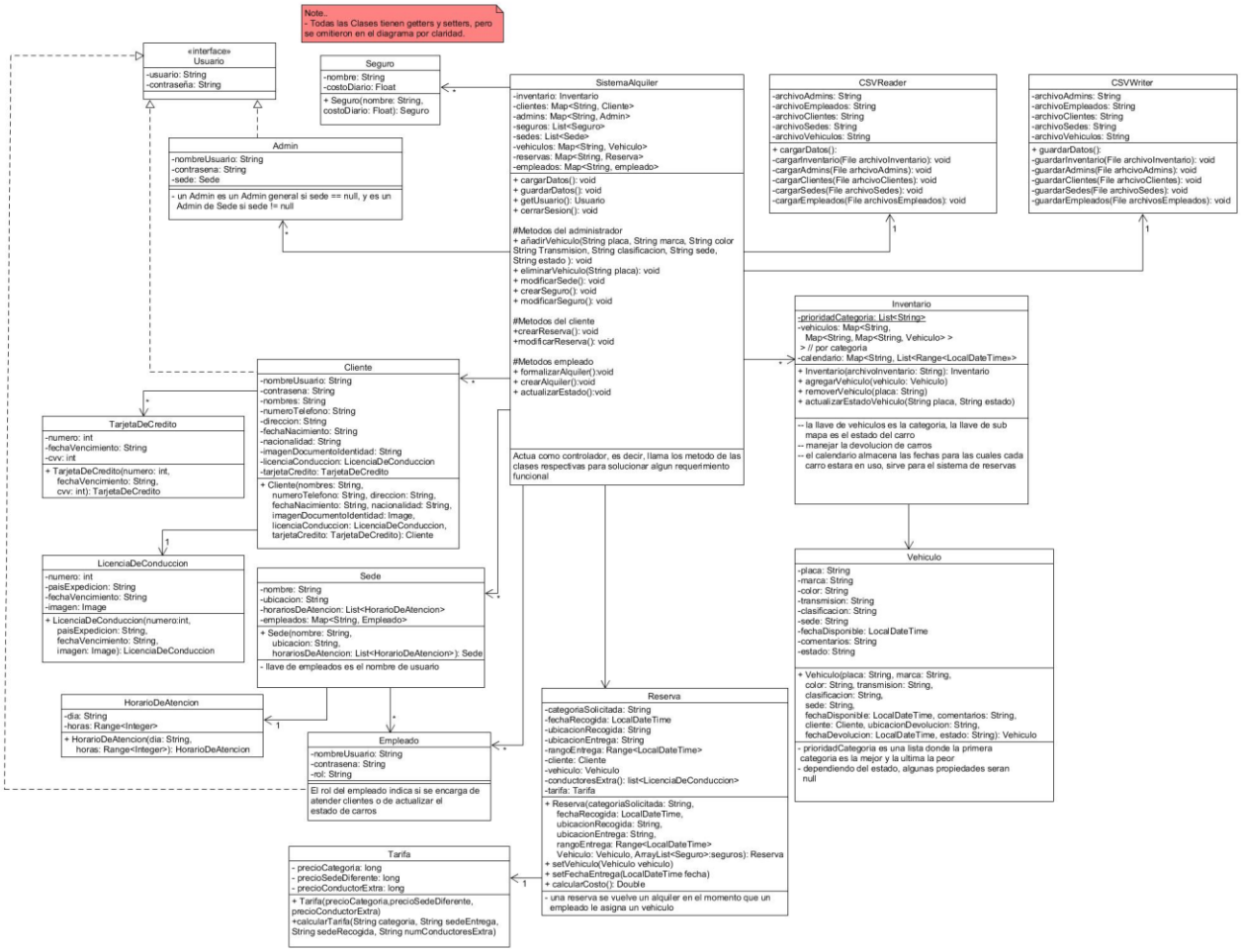
El primer requerimiento critico es el que permite registrar clientes y administradores. En el diagrama se muestra que en el caso de registrar clientes y administradores primero se accede al menú de registro y allí con los parámetros solicitados se llama al sistema alquiler y posteriormente al Contenedor de Datos para almacenar los usuarios recientemente creados. Por otro lado, en el caso de registro de empleados (no se muestra en el diagrama por simplicidad) el proceso es el mismo pero únicamente es capaz de realizarlo un administrador local a través de su menú de administrador.



Otro requerimiento crucial es la realización de reservas-alquileres. En el caso de una reserva el cliente accede a través de su menú y a parta un cupo en una categoría de interés y se le realiza un primer cobro. Luego, un empleado accede a la reserva a través de su sistema y le asigna un vehículo, formalizando la reserva en un alquiler. En el caso en el que se realiza una reserva sin previo alquiler se debe verificar la disponibilidad de vehículos en la categoría, en caso de no haber no se podrá llevar a cabo el proceso. En cambio, si existe una reserva previa y, por cualquier motivo, no hay vehículos de la categoría deseada el empleado asignara un vehículo de la siguiente mejor categoría.

Diagrama final

Finalmente se presenta el diagrama completo



Errores

Únicamente se utilizó un tipo de error, la clase base de java llamada Exception(), colocando un mensaje especifico que detalla el error que ocurrió en cada caso. En casi todos los casos se toma el mensaje retornado y se le muestra al usuario.