BIOS RENT

Ingeniería de Software

Desarrollo de Sistema de Administración

ÍNDICE

ETAPA DE DISEÑO

I. DEFINICIÓN DE LA ARQUITECTURA

Este apartado presenta la arquitectura de software del sistema, la cual refiere al nivel más alto que proporciona una visión global y general del mismo.

La arquitectura de software se define considerando los requerimientos no funcionales, con el objetivo de alcanzar un equilibrio entre distintos requerimientos de calidad: un correcto desempeño, asegurar niveles adecuados de seguridad del sistema y favorecer la mantenibilidad del software.

De esta manera se busca presentar un diseño que minimice las invocaciones y comunicaciones, utilice a la vez las capas internas con mecanismos adecuados de validación e incorpore la mayor cantidad de componentes autocontenidos, fácilmente intercambiables con el menor acoplamiento posible.

1.1 Software Architecture Document (SAD)

El SAD que se presenta se compone de las siguientes vistas:

- Vista de Modelo de Casos de Uso
- Vista de Subsistemas
- Vista del Modelo de Distribución (Deployment)
- Justificación de la arquitectura propuesta

1.1.1 Vista de Modelo de Caso de Uso

En esta sección se presenta una selección representativa de casos de uso cuyo propósito es justificar los componentes de la arquitectura que define la organización de nuestro software.
El sistema se organiza en base a diferentes componentes. Se agrupan y ordenan requerimientos funcionales (casos de uso), tecnológicos, etc., considerando sus características y los atributos de calidad (requerimientos no funcionales) planteados para el desarrollo del software.
A diferencia del diagrama de casos de uso donde se muestran todos los requerimientos funcionales del sistema, en este apartado se realiza una selección de los mismos, en busca de justificar cada componente definido en el sistema y abarcar todos los lìmites y responsabilidades de nuestra arquitectura: dominio, persistencia, servicios externos y front-end, etc.
El diagrama resultante presenta una primera etapa en el proceso de diseño que justifica cada componente propuesto como respuesta a las necesidades reales del negocio.

Caso de Uso	Registrar Cliente
Actor Principal	Empleado
Actor Secundario	Cliente
Descripción	El cliente solicita darse de alta en el sistema. El empleado ingresa el número de cédula de identidad del cliente, nombre completo, teléfono de contacto, país y se le crea un usuario con nombre y una clave de acceso.

Este caso de uso se utiliza como representativo y justifica la existencia del componente encargado de la **gestión de Usuarios** (empleado y cliente), en el cual se concentran las funcionalidades asociadas a su ciclo de vida (alta, baja, modificación y consultas).
Además de contemplar sus reglas de negocio correspondientes, estos casos de uso implican el tratamiento de datos privados y sensibles (como cédula de identidad y teléfono) sometidos a la Ley de Protección de Datos. En este sentido, se relaciona con el requerimiento no funcional RNF+02: Cumplimiento de Protección de Datos).
La agrupación de estas funcionalidades en un único componente permite simplificar la organización interna del sistema y facilita el mantenimiento de la información del trabajo general y en particular muchas tareas relacionadas con el sector de auditoría y control de datos.
Este caso de uso justifica la necesidad de la **Aplicación de Escritorio**, medianter la cual el empleado accede al sistema, de la **API Empleado** que actúa como intermediaria entre la interfaz gráfica y la lógica del sistema **y de las capas Lógica y**

Persistencia encargadas del procesamiento de las operaciones y del almacenamiento de los datos**.**

Caso de Uso	Registrar Sucursal
Actor Principal	Empleado
Actor Secundario	-
Descripción	El empleado ingresa los datos de la sucursal a dar de alta (nombre de la sucursal y la ciudad), se le asigna un número y queda registrada en el sistema

El caso de uso seleccionado justifica la existencia de un componente específico para la **gestión de sucursales** dentro de la capa lógica del sistema.
Este componente agrupa funcionalidades relacionadas al concepto Sucursal, algunas de las cuales (registrar, eliminar y modificar) son realizadas por el empleado a través de la aplicación de escritorio y otras (consultas) las ejecutará el cliente por su aplicación móvil.
El componente tiene reglas de negocio específicas que establecen vínculos con otros componentes especializados en vehículos y sucursales (Ej.: No se puede eliminar una sucursal si tiene vehículos o empleados).
Debido a que los datos de las sucursales también se almacenarán en el sistema en forma persistente, este caso de uso también justifica la existencia de la capa Persistencia en la arquitectura.

Caso de Uso	Registrar Vehículo
Actor Principal	Empleado
Actor Secundario	-
Descripción	El empleado da de alta a un nuevo vehículo ingresando la matrícula del nuevo vehículo y todos sus datos (tipo, características, foto, precio por día y se indica si se pondrá en servicio en ese momento y en caso afirmativo se lo vincula con la sucursal correspondiente) El sistema controlará que no exista un vehículo ya registrado con la matrícula que se ingresa en el alta.

Este caso de uso justifica la existencia de un componente específico para la **gestión de vehículos** dentro de la capa lógica del sistema.
Dado que los vehículos conforman el recurso principal de la empresa y su razón de ser, este componente agrupa las funcionalidades vinculadas a su gestión y consultas.

Las operaciones de registrar, eliminar, modificar y trasladar vehículo de sucursal son realizadas por los empleados a través de su aplicación de escritorio. Por su parte, los clientes podrán realizar consultas a través de su aplicación móvil.
Este componente cuenta con reglas de negocio propias, como las que se mencionan en la descripción del caso de uso que generan la necesidad de verificaciones, de relaciones del vehículo con las sucursales, con un alquiler, etc.
Finalmente, como los datos de los vehículos deben almacenarse de forma persistente, este caso de uso también justifica la existencia de la capa Persistencia en la arquitectura.

Caso de Uso	Registrar Alquiler de Vehículo
Actor Principal	Empleado
Actor Secundario	Cliente
Descripción	El empleado de una sucursal registra un nuevo alquiler de un vehículo a un cliente previamente registrado en el sistema. Selecciona un vehículo disponible en la sucursal, ingresa los datos del cliente, indica si contratará seguro y establece la fecha de devolución. El sistema calculará el costo total que incluirá el costo del alquiler del vehículo, el seguro (en caso de que sea contratado por el cliente) y el depósito en garantía si corresponde (podrá exonerarse si el cliente contrata seguro y tiene 3 alquileres previos con la empresa). Cuando se confirme el alquiler, se actualizará el estado del vehículo alquilado.

Este Caso de Uso comprende la acción fundamental del negocio. Justifica el componente Gestión de Alquileres y Devoluciones dentro de la capa lógica del sistema.
Se definió agrupar en este componente las funcionalidades de registro de alquileres, devolución del vehículo al finalizar el alquiler y la actualización de tarifas. Aunque las acciones de registrar alquiler y devolución de vehículos ocurren en distintos momentos del negocio, ambas se relacionan con el mismo concepto: Alquiler.
La actualización de tarifas también se incluye en este componente ya que se trata de valores únicos que no requieren ser instanciados múltiples veces, sino simplemente modificados de manera centralizada ya que cuentan con un valor único.
Para ejecutar sus funcionalidades este componente requiere la comunicación con los otros componentes del sistema, involucra Empleado, Cliente, Vehículo y Sucursal.

Las acciones de este componente son realizadas por los empleados a través de su aplicación de escritorio y todos los datos generados deben ser almacenados, por lo que este caso de uso también justifica la existencia de la capa Persistencia.

Caso de Uso	Localizar Vehiculo
Actor Principal	Empleado
Actor Secundario	-
Descripción	El empleado ingresa la matrícula de un vehículo para obtener su ubicación. El sistema muestra la ubicación en el mapa, a partir de la información suministrada por el sistema de geolocalización de la Empresa de Geolocalización.

Este caso de uso tiene como elemento central la interacción con un servicio externo, una API Rest provista por una empresa que brinda el servicio de geolocalización.
Esta característica justifica la existencia del componente **Localizar Vehículo**. Debido a que se apoya en una lógica externa, se decidió encapsular esta funcionalidad en un componente independiente, encargado de abstraer los detalles relacionados con la conexión, autenticación y consumo del servicio externo.
La fuente de datos no es propiedad de la empresa BIOS Rent lo que implica una dependencia directa de un proveedor que podría cambiar o incluso dejar de estar disponible.
Este Caso de Uso depende de la respuesta del servicio externo para poder ejecutarse correctamente. Tiene sus reglas y familia de excepciones propias, específicas para este tipo de comunicación.
Debido a estas particularidades se decide encapsular la funcionalidad en un único componente con el objetivo de evitar que posibles problemas derivados del agente externo impacten con otras partes del sistema.

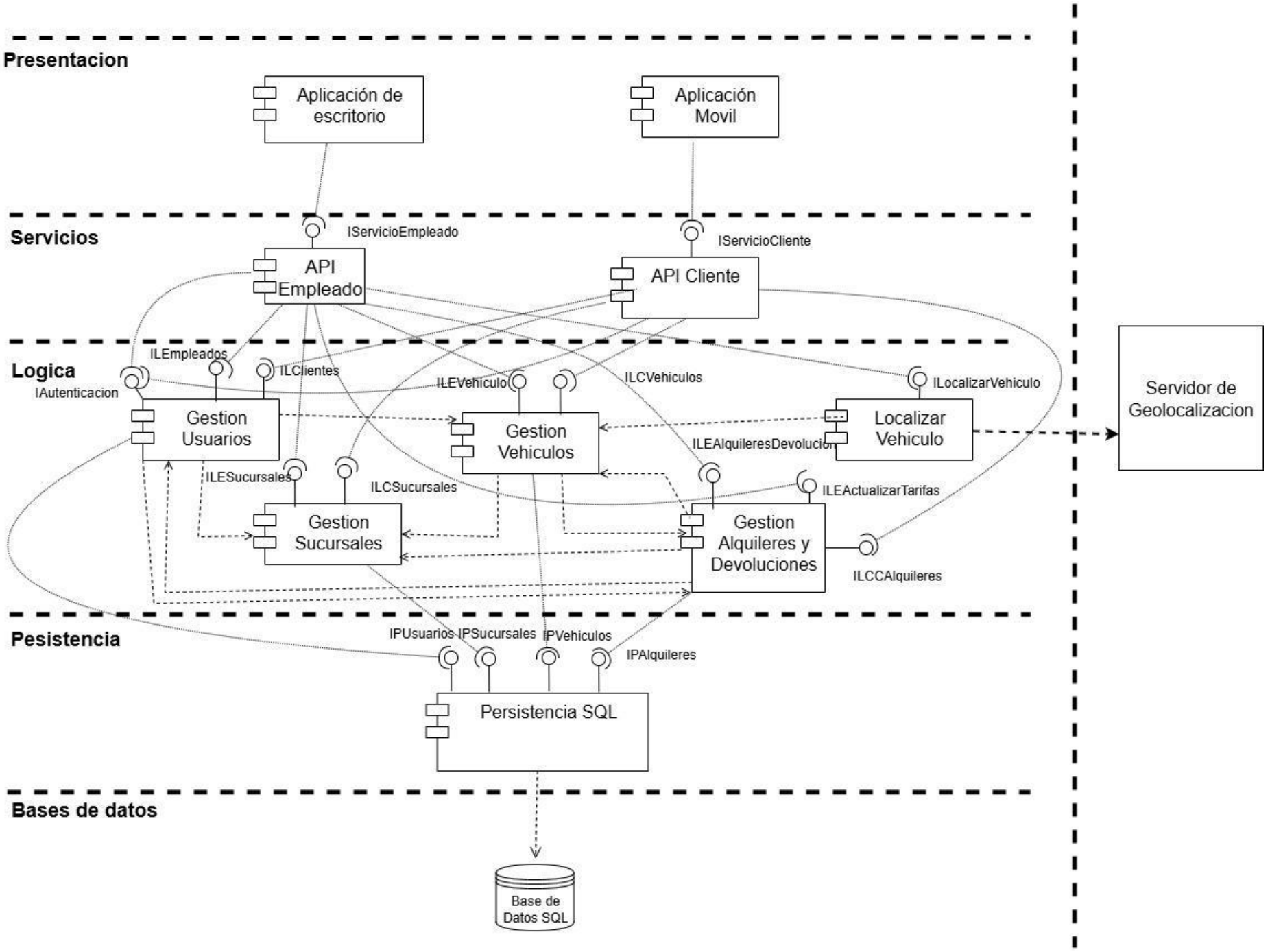
La organización planteada se relaciona con el requisito no funcional RNF-S02: Facilidad de Mantenimiento del Sistema para lo que se modularizan las funcionalidades del sistema (Ej. Incluir esta lógica dentro del componente Gestión de Vehículos implicaría sumarle responsabilidades adicionales, yendo en contra del principio de separación de responsabilidades.
La solución adoptada favorece un bajo acoplamiento entre clases. Tener un componente específico permite encapsular la lógica para interactuar con APIs externas, facilitando futuras integraciones con otros servicios. Esto incrementa tanto la escalabilidad como la mantenibilidad del sistema, al simplificar las posibles adaptaciones necesarias a lo largo del tiempo.

Caso de Uso	Consultar Vehículos Disponibles
Actor Principal	Cliente
Descripción	A través de la Aplicación Móvil el cliente accede a la sección donde se listan los vehículos disponibles para alquilar. El sistema despliega los vehículos con su información y la sucursal en la que se encuentra. Ofrece la posibilidad de filtrar los vehículos según una determinada sucursal. Desde la lista de vehículos desplegados también se da la posibilidad de marcar vehículos como favoritos, destacando éstos dentro de la lista

Este caso de uso justifica la creación de un componente dedicado a la **Aplicación Móvil** ya que a través de ella el cliente interactúa con el sistema. El usuario se autentica mediante nombre de usuario y contraseña y accede a funcionalidades generalmente relacionadas con consultas de información.
La comunicación con la Aplicación Móvil requiere una interfaz amigable y accesible desde dispositivos móviles. Las funcionalidades dirigidas al cliente deberán atender especialmente a distintos requerimientos no funcionales, entre ellos:
RNF- P01: Tiempo de respuesta de la App Móvil,
RNF-S01: Compatibilidad con App Móvil, asegurando soporte multiplataforma para sistemas Android 10.0 y IOS 16.
Además este caso de uso justifica la existencia de un componente llamado **API Cliente**, a través del cual la aplicación móvil consume las funcionalidades correspondientes expuestas por el sistema.

1.1.2 Vista de Subsistemas

El diseño de la arquitectura del sistema Bios RENT se basa en una arquitectura en capas, combinado con una organización por dominios funcionales dentro de la capa de reglas de negocio. Con esto buscamos separar de forma clara las responsabilidades y dominios de los componentes y subsistemas. Se definirán los componentes correspondientes con sus respectivas interfaces. Finalmente se describe el cometido y el funcionamiento de cada subsistema.


Presentacion
Aplicación de escritorio
Aplicación Movil
Servicios
IServicioEmpleado
IServicioCliente
API Empleado
API Cliente
Logica
ILEmpleados
ILClientes
IAutenticacion
ILEVehiculo
ILCVehiculos
ILocalizarVehiculo
Gestion Usuarios
Gestion Vehiculos
Localizar Vehiculo
ILESucursales
ILCSucursales
ILEAlquileresDevolucion
ILEActualizarTarifas
Gestion Sucursales
Gestion Alquileres y Devoluciones
ILCCAlquileres
Pesistencia
IPUsuarios
IPSucursales
IPVehiculos
IPAlquileres
Persistencia SQL
Bases de datos
Base de Datos SQL
Servidor de Geolocalizacion

Cometido y Funcionamiento de cada Componente:

CAPA PRESENTACIÓN
La capa Presentación incluye dos tipos de aplicaciones, una aplicación de escritorio para ser utilizada por los empleados en cada sucursal de la empresa y una aplicación móvil para que los clientes.

Aplicación de Escritorio:
Este componente será utilizado por los empleados en las sucursales a las que se encuentran asignados. Se comunican con la capa lógica a través de un servicio API por el cual accederán a todas las funcionalidades administrativas del sistema que incluyen la gestión de usuarios, sucursales, vehículos y alquileres.

Aplicación móvil:
Por medio de este componente los clientes registrados en el sistema se comunican con la capa lógica del sistema a través de un Servicio API que les permitirá consultar vehículos disponibles, gestionar vehículos favoritos, acceder a información de sucursales, ver alquileres realizados por el cliente entre otras funcionalidades.

CAPA SERVICIOS
La Capa Servicios se compone por dos servicios API Rest que permiten la comunicación de Aplicaciones móviles y Aplicaciones de Escritorio con nuestro sistema.

APIEmpleado
Es el componente de la capa de servicio encargado de la comunicación entre la aplicación de escritorio con los componentes de la capa lógica, centralizando todas las funcionalidades del empleado. Si bien el sistema podría tener solo una api que conecte la aplicación de escritorio y la aplicación móvil con los componentes de la lógica se decidió hacerlo separado debido a varias razones.

Tener 2 apis una para la interfaz del cliente y otra para el empleado lo hace al sistema más escalable, seguro y mantenible. Escalable porque si el trafico de usuarios cliente sube, esto podría afectar el tiempo de respuesta en las comunicaciones con los usuarios empleados impactando en el negocio. Seguro porque manteniéndolos separados mejora el control de acceso por tipo de usuario y reduce el riesgo de que un usuario malintencionado desde la app móvil acceda a endpoints internos. Mantenible porque separa las responsabilidades .
Consume las interfaces

- IAutenticacion para las operaciones relacionadas a Inicio de sesion, cerrar sesion, y validaciones del usuario
- ILEmpleados, ILESucursales, ILEVehiculo para operaciones relacionadas a Alta, Baja y Modificar sobre las entidades respectivas.

- ILEGestionAlquileresDevolucion que agrupa las funcionalidades relacionadas a la entidad alquileres. Ofrece las operaciones para registrar alquileres de un vehículo y la devolución del mismo (cerrar el alquiler).
- ILActualizarTarifas ofrece la funcionalidad de gestionar las tarifas del sistema (seguro y garantia) a nivel global, a través de la cual se podrá actualizar datos de las tarifas.
- ILocalizarVehiculo brinda la posibilidad de obtener de la ubicación de un vehículo

APICliente

Es el componente de la capa Servicio encargado de la comunicación de la aplicación web a través del cual los clientes pueden conectarse con la capa lógica relacionados a las funcionalidades del cliente.
Consume las interfaces

- IAutenticacion del componente Gestion Usuarios desde donde podrá iniciar su sesión verificando nombre de usuario y contraseña.
- ILCSucursales de la entidad Gestión Sucursales, a partir de las cuales podrá realizar consultas de las sucursales de la empresa.
- ILCaAlquileres de la entidad Gestión Alquileres, a partir de la cual podrá consultar el alquiler vigente y los pasados.
- ILCVehiculos desde donde podrá realizar consultas de vehículos disponibles y gestionar vehículos favoritos.

CAPA LÓGICA

GestionUsuarios

Este componente tiene la responsabilidad de administrar los usuarios (clientes y empleados) del sistema y el control del acceso al mismo
Expone dos interfaces:
IAutenticacion que es consumida por la api de empleado y cliente para loguearse y cerrar sesión ya que empleado y cliente heredan de usuario, por lo que ambos podrán usar la misma interfaz para los casos de uso que tienen en común.
ILEmpleados aglomera las operaciones de los casos de uso referentes a las entidades de Empleado y Cliente. Esto significa que tiene las operaciones de gestion de usuario, alta, baja, modificacion de Empleado y cliente. a su vez es consumida por la api.
Este componente depende de Geestion de Alquileres y devolución porque para eliminar un usuario este no puede tener un alquiler vigente, también depende de Gestion sucursales porque al dar de alta a un empleado este tiene que trabajar en una sucursal

GestionSucursales

Este componente tiene la responsabilidad de la gestión y las consultas de las sucursales de Bios RENT. Expone dos interfaces
ILESucursales concentra operaciones de gestión (alta, baja, eliminar). Estas funcionalidades son para para los usuarios Empleado quien las consumirá por medio de la APIEmpleados. Tiene sus restricciones y reglas de negocio específicas y la necesidad de controles que impidan que se eliminen sucursales que cuenten con empleados o vehículos asociados.
ICSucursales presenta operaciones de consultas que son utilizadas por los clientes a través de la APIClientes.
Este componente consume la interfaz del componente de Persistencia SQL llamada IPSucursales.
A su vez depende de

GestionVehiculos
Este componente tiene la responsabilidad de gestionar vehículos, realizar consultas de los vehículos disponibles y gestionar los vehículos favoritos por los usuarios.
Expone dos interfaces:
ILEVehiculos corresponde a las funcionalidades relacionadas con la gestión de los vehículos de la empresa (registrar, eliminar, modificar, trasladar vehículo a sucursal). La gestión de los vehículos es responsabilidad de los empleados quienes la utilizan a través de la APIEmpleado.
ILCVehiculos concentra funcionalidades relacionadas al cliente como consulta de vehículos disponibles y gestionar vehículos favoritos. Las funcionalidades que se incluyen aquí son utilizadas por los clientes mediante la aplicación móvil a través del Servicio APIClientes.
Este componente consume la interface IPVehículos del componente Persistencia SQL
Depende de Gestión Sucursales ya que los vehiculos disponibles se encuentran en una sucursal y se permite listado parametrizado (vehiculos por sucursal). También depende de Gestion de Alquileres y Devoluciones ya que necesita hacer verificaciones en cuanto a la disponibilidad del vehículo

GestionAlquileresyDevoluciones
Este componente tiene la responsabilidad de gestionar el alquiler, devoluciones, realizar consultas y actualizar tarifas. Expone tres interfases:
ILEAlquileresDevolucion corresponde a los casos de uso relacionados con el registro de un alquiler y la devolución de vehículos finalizado el alquiler. Es utilizada por el Empleado a través de API Empleado.
ILCAlquileres corresponde a los casos de uso relacionados con consultas de alquileres pasados y vigentes. Es utilizada por los clientes a través de APICliente.
ILActualizarTarifas corresponde al caso de uso responsable de actualizar las tarifas de vehículos (seguro y garantía de alquiler). Es utilizada por los clientes a través de APICliente.

Consume componentes de vehículos, sucursales y usuarios. Persiste sus datos a través de Persistencia de Datos SQL Server (capa persistencia).
Este componente es el que tiene más dependencias. depende de gestión de usuarios porque al dar de alta un alquiler, con el empleado logueado obtiene la sucursal. también depende de Gestion sucursales, todo alquiler tiene una sucursal de inicio y una de fin, y por supuesto depende también de gestion vehículos ya que es un vehículo lo que se alquila

Localizar Vehiculo
Este componente tiene la responsabilida de comunicarse con el Sistema de geolocalización externo a nuestro sistema.
Obtiene los datos de ubicación de los vehículos (latitud y longitud) y mapea ubicaciones de los vehículos.
Expone una interfaz ILocalizarVehiculo en la que agrupa las operaciones relacionadas a este caso de uso. A su vez depende del componente Gestión de Vehículos ya que primero hay que buscar y verificar la matrícula del vehículo para localizarlo.

CAPA PERSISTENCIA
La Capa Persistencia tiene un único componente donde se persistirán los datos del sistema.

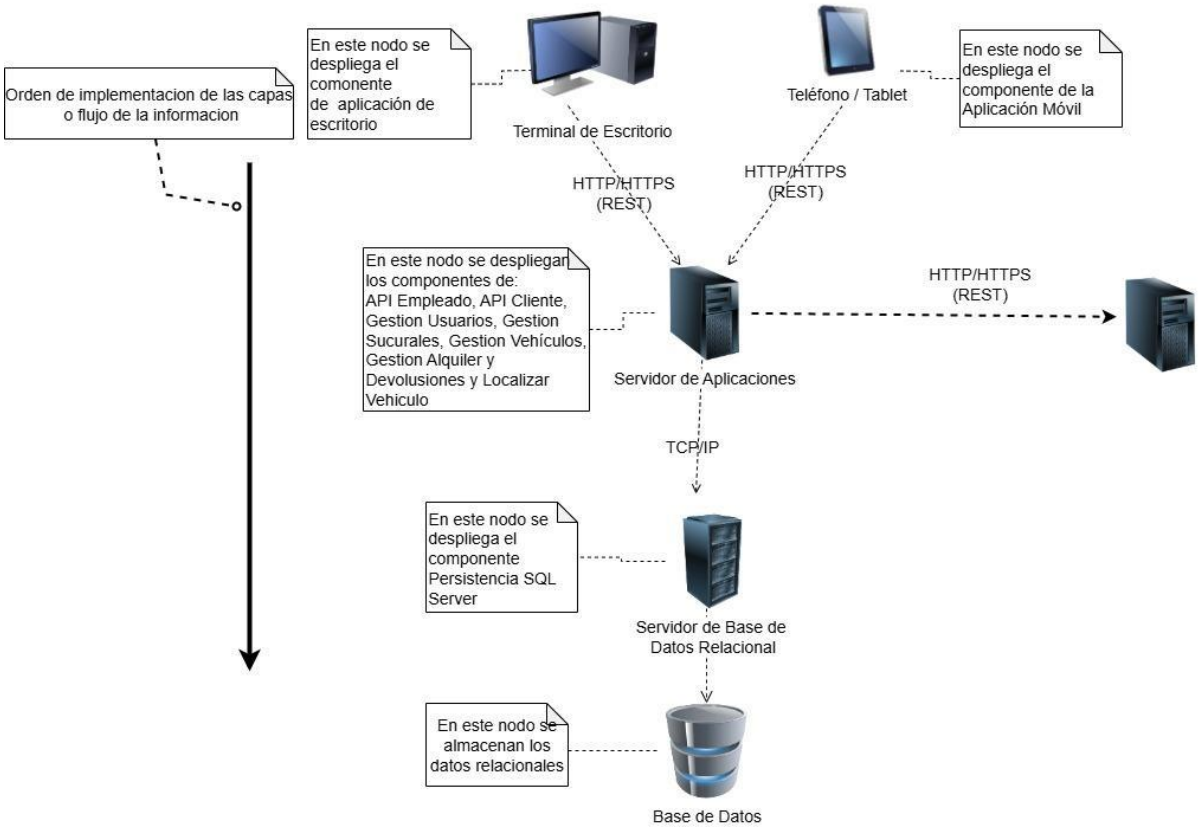
PersistenciaSQL
Este componente se encarga de persistir la información en la base de datos relacional por lo que tiene clases relacionadas a la conexión a la base de datos, ejecución de procedimientos y transacciones.
Expone las Interfaces IPUsuarios, IPSucursales, IPVehiculos, e IPAlquileres para persistir datos ingresados, modificados o eliminados por las operaciones relacionadas a alta, baja, Modificar y consultar de cada concepto.
Es consumido por los componentes de la capa lógica Gestión Usuarios, Gestión Sucursales, Gestión Vehículos y Gestión Alquileres y Devoluciones.

1.1.3 Vista de Deployment

El presente apartado muestra la vista de despliegue (deploy) o distribución, donde se representan en nodos de ejecución los componentes físicos del sistema.
Se trata de la infraestructura, dispositivos físicos que dan soporte a nuestro sistema BIOS Rent y cómo se distribuyen los componentes de software en servidores, dispositivos móviles, estaciones de trabajo (escritorio).

Esta vista describe el despliegue del software en el entorno real, indicando cómo se comunican los distintos nodos entre sí y qué recursos físicos se requieren. Permite comprender visualmente la infraestructura tecnológica que soporta el sistema,

identificar la ubicación de los componentes del sistema, detectar cómo se comunican entre ellos y considerar las dependencias tecnológicas que puedan surgir y puedan representar riesgos en términos de disponibilidad, escalabilidad, rendimiento y seguridad.



Terminal Escritorio

La Terminal de Escritorio es una estación de trabajo ubicada en una sucursal de BIOS Rent la cual presenta una interfaz gráfica para que el empleado pueda gestionar el sistema en formato de Aplicación de Escritorio.
La aplicación se comunica con un con la API Empleado utilizando el protocolo HTTP/HTTPS
En aspectos relacionados al hardware necesario, existe una amplia gama de dispositivos PC de escritorio o laptop empresarial que se podrían utilizar. Se sugieren los siguientes requisitos mínimos para garantizar un correcto funcionamiento:

- Procesador: Intel Core I5 O AMD Ryzen 5
- Memoria RAM: 8 GB
- Almacenamiento: SSD de 225 GB
- Sistema Operativo: Windows 10
- Conectividad: Acceso estable a internet

Dispositivo Cliente

El dispositivo cliente comprende a la aplicación móvil ejecutable para dispositivos móviles que cuenten con Android / IOS.
Se conecta con el servidor por medio de una API REST utilizando protocolos HTTP/HTTPS (https ofrece mayor seguridad).
En aspectos relacionados al hardware, existe una amplia gama para podría soportar los requerimientos de la aplicación móvil. Se sugieren los siguientes requisitos mínimos para garantizar un correcto funcionamiento:

- Procesador: ARM 64-bit
- Memoria RAM: 3 GB
- Sistema operativo: Android 10 / 16 IOS
- Conectividad: WIFI o datos móviles

Servidor de Aplicaciones

En el Servidor de Aplicaciones es el nodo principal donde estará contenido y desplegado el backend de la aplicación. Comprende la lógica de nuestro sistema.

Se comunica con las terminales de escritorio, con los dispositivos de los clientes, con el Servidor de la base de datos y con el Servidor externo.
La comunicación de las terminales de escritorio y las terminales del cliente (teléfono / Tablet) se despliegan por API Rest mediante el protocolo HTTP/REST.
Se comunicará con el servidor externo (API Rest externa) por medio del protocolo HTTP/REST.
Los servicios del servidor donde se aloja el componente de la capa persistencia y la base de datos relacional utilizan el protocolo TCP/IP para la transmisión de datos.

En lo relacionado al hardware se recomienda analizar la posibilidad de implementar un servicio de Clouding (Azure, AWS, Google cloud) ya que puede traer beneficios en cuanto a los gastos de capital (costo que incurre la empresa para poner en marcha su negocio, normalmente refiere a la infraestructura requerida), escalabilidad y mantenibilidad.
Se puede optar por un método de “Pay as you use” para los recursos (por ejemplo app services para las API), y una reserva “long time” para la infraestructura a emplear (servicio de IaaS). De esta manera podría significar un ahorro considerable para la empresa ya que los componentes de infraestructura son a largo plazo.
Es posible también emplear un servidor físico tipo datacenter empresarial con las siguientes características:

- Tipo: Datacenter empresarial
- Procesador: Intel Xeon / AMD EPYC
- Memoria RAM: 24 GB
- SSD NVMe
- Sistema Operativo: Windows Server

- API REST Y .NET core Runtime

Servidor de bases de datos
El componente que se aloja en este servidor es el componente de Persistencia SQL y su respectiva base de datos relacional.
El componente persistencia y Base de datos se alojan en el mismo servidor para reducir la latencia y optimizar costos.
El propósito de este nodo es solicitar, persistir, almacenar y consultar información.
Se comunica con el Servidor de Aplicaciones utilizando protocolo de transmisión de datos TCP/IP.
Si se sigue la recomendación de implementar un servicio Clouding se podría utilizar una Azure Database, Cloud SQL o Amazon RDS.
Sino existe la posibilidad de implementar un servidor físico para el que se definen requisitos mínimos:

- Intel Xeon E5
- Memoria RAM: 32GB de RAM
- Almacenamiento: 500 GB SDD
- Motor de SQL Server

Servidor Externo:
Es el nodo que refiere al servidor externo que le brinda el servicio de geolocalización al sistema.
Cuenta con una API por la cual el componente de la capa de la lógica Localizar Vehiculo consume para obtener la ubicación del vehículo.
Se puede implementar directamente desde el proveedor de Clouding si se optó por esta opción (servicio PaaS desde el proveedor) o por un servicio independiente.
Algunas opciones son: Google Maps API, HERE Maps.
Debe comunicarse mediante protocolo HTTP/HTTPS

1.1.4 Justificación de la Arquitectura

En este apartado se describen los estilos arquitectónicos que se utilizarán.

La arquitectura general del sistema se desarrolla siguiendo el estilo **Cliente - Servidor.**
El servidor es quien brinda los servicios. Procesa y devuelve respuestas a las solicitudes del cliente. El cliente (empleados y usuarios) consume los servicios que brinda un servidor. Mediante la interfaz gráfica se comunica con el servidor, hace solicitudes y recibe respuestas del sistema.
Se da una interacción entre cliente y servidor, pero no entre los clientes.

La arquitectura contendrá algunos aspectos generalmente relacionados a validaciones de información desde la aplicación cliente, la lógica de negocio estará contenida en su mayor proporción en el servidor.

El sistema se organiza en **Capas.** Definimos cuatro capas ordenadas, cada una de las cuales cumple una responsabilidad determinada y le proveerá servicios a capas superiores: La presentación que contendrá las aplicaciones del usuario (empleados y clientes), la capa servicios donde se desarrollará todo lo que necesite para su desarrollo la comunicación con servicios externos, la capa lógica incluye la lógica general del negocio que se desarrolla en el servidor de aplicaciones y la persistencia encargada de persistir la información en servidores de bases de datos.
Esta organización es adecuada en aspectos de comprensión del sistema, mantenimiento, reutilización y portabilidad del sistema.

El sistema utiliza el estilo **Descomposición Orientada a Objetos**. Mediante este estilo el sistema se traduce a objetos que se comunican entre sí. Estos objetos cuentan con un estado y comportamiento determinado y brindan posibilidades de herencia, polimorfismo y sobrecarga de operaciones.
Nuestro sistema incluye la necesidad de obtener información con un Servicio Externo que nos provee de información de localización de un vehículo para poder desarrollar nuestro la funcionalidad de obtener la ubicación de los vehículos alquilados.

Utilizaremos el estilo **REST.** Desde la aplicación externa se definen servicios independientes que se exponen a través de interfaces estandarizadas. Desde nuestro software se consumen estos servicios para desarrollar nuestra funcionalidad de ubicación de vehículos alquilados.
Los estándares de comunicación se basarán en Json.
La arquitectura diseñada permitirá que pueda modificarse de manera sencilla y con un mínimo impacto la empresa que suministra los servicios de geolocalización

II. DEFINICIÓN DE INTERFACES DEL SISTEMA Y CONTROLADORES

Organización del diseño – Criterios GRASP

La organización de nuestro diseño toma como referencia los principios GRASP (General Responsability Assignment Software Partners).
Uno de los principios de mayor importancia en nuestro diseño es el GRASP Controller, el cual guiará la creación de los Controladores, clases responsables en la ejecución de las operaciones del sistema.

Interacción entre la Capa Presentación y la Capa Lógica

Debido a que nuestro sistema se organiza en capas, los usuarios interactúan con la lógica del sistema a través de la Capa Presentación. Esta capa accede a la lógica interna a través de interfaces.

Interfaces
En la Capa Lógica se resuelven los diferentes Casos de Uso, cada uno compuesto por un conjunto de operaciones.
Estas operaciones se expondrán a través de interfaces públicas -cada una de las interfaces agrupa un conjunto de operaciones-, que serán los puntos de acceso desde la capa Presentación.

Controladores
Siguiendo los criterios establecidos en el GRASP Controller, los controladores son las clases encargadas de implementar las interfaces y de que se ejecuten las operaciones correspondientes a uno o varios Casos de Uso.
Los Controladores no serán accesibles por la Capa Presentación, sino que serán instanciados por una fábrica y retornados como interfaces.
Cada controlador agrupa las operaciones de uno o varios Casos de Uso relacionados. Esto permite que la misma instancia del controlador atienda operaciones altamente relacionadas entre sí, favoreciendo una alta cohesión.
Para cumplir con los Contratos de Software, los controladores delegan parte del proceso a otras clases internas de la Capa Lógica. De esta manera estas clases contendrán la lógica de negocio más compleja, mientras que los controladores tendrán métodos más livianos, actuando como articuladores de otras clases que realizarían el trabajo de mayor complejidad.

Patrón Fábrica
Para romper con la dependencia y el acoplamiento entre la Capa Presentación y los Controladores se utilizará el patrón Fábrica.
Este patrón se encargará de crear instancias de controladores y retornarlas como interface

2.1.Estructuración de Controladores, Interfaces, Fábricas en el Sistema
En este sistema se optó por la definición de controladores que se gestionan varios casos de uso relacionados, excepto el ControladorLocVehiculo, que gestiona exclusivamente el Caso de Uso “Localizar Vehículo”, debido a que este requiere la interacción con un servicio externo. Su aislamiento permite encapsular las particularidades técnicas y de comunicación asociadas a este servicio, lo que reduce posibles impactos con el resto del sistema.
También el ControladorLEActualizarTarifas que solamente gestiona las operaciones del caso de uso “Actualizar Tarifas”, es un concepto aparte de el alquiler pero el alquiler depende del valor de este para generar el precio.

Fábrica

Se define la Fábrica que brindará acceso a los controladores. Contendrá las siguientes operaciones estáticas:

- getControladorLEmpleado
- getIControladorAutenticacion
- getControladorLESucursal
- getControladorLCSucursal
- getControladorLEActualizarTarifa
- getControladorLEAlquileresDevolucion
- getControladorLCAlquiler
- getControladorLocVehiculo
- getControladorLEVehiculo
- getControladorLCVehiculo

Controladores

De acuerdo a los criterios del GRASP Controller descritos previamente se definen los controladores del sistema y los Casos de Uso que gestiona cada uno.

- ControladorLEmpleado
    - Registrar Empleado
    - Eliminar Empleado
    - Modificar empleado
    - Registrar Cliente
    - Eliminar Cliente
    - Modificar Cliente

- ControladorLocVehiculo
    - Localizar Vehículo

- ControladorLEAlquileresDevolucion
    - Registrar Alquiler
    - Registrar Devolución Vehículo
- ControladorLEVehiculos:
    - Registrar Vehículo
    - Eliminar Vehículo
    - Modificar Vehículo
    - Trasladar Vehículo a Sucursal

- ControladorLESucursal
    - Registrar Sucursal
    - Eliminar Sucursal
    - Modificar Sucursal

- ControladorLEActualizarTarifas:
    - Actualizar Tarifas

- ControladorAutenticacion:
    - Iniciar Sesión
    - Finalizar Sesión

- ControladorLCSucursales:
    - Consultar Sucursales

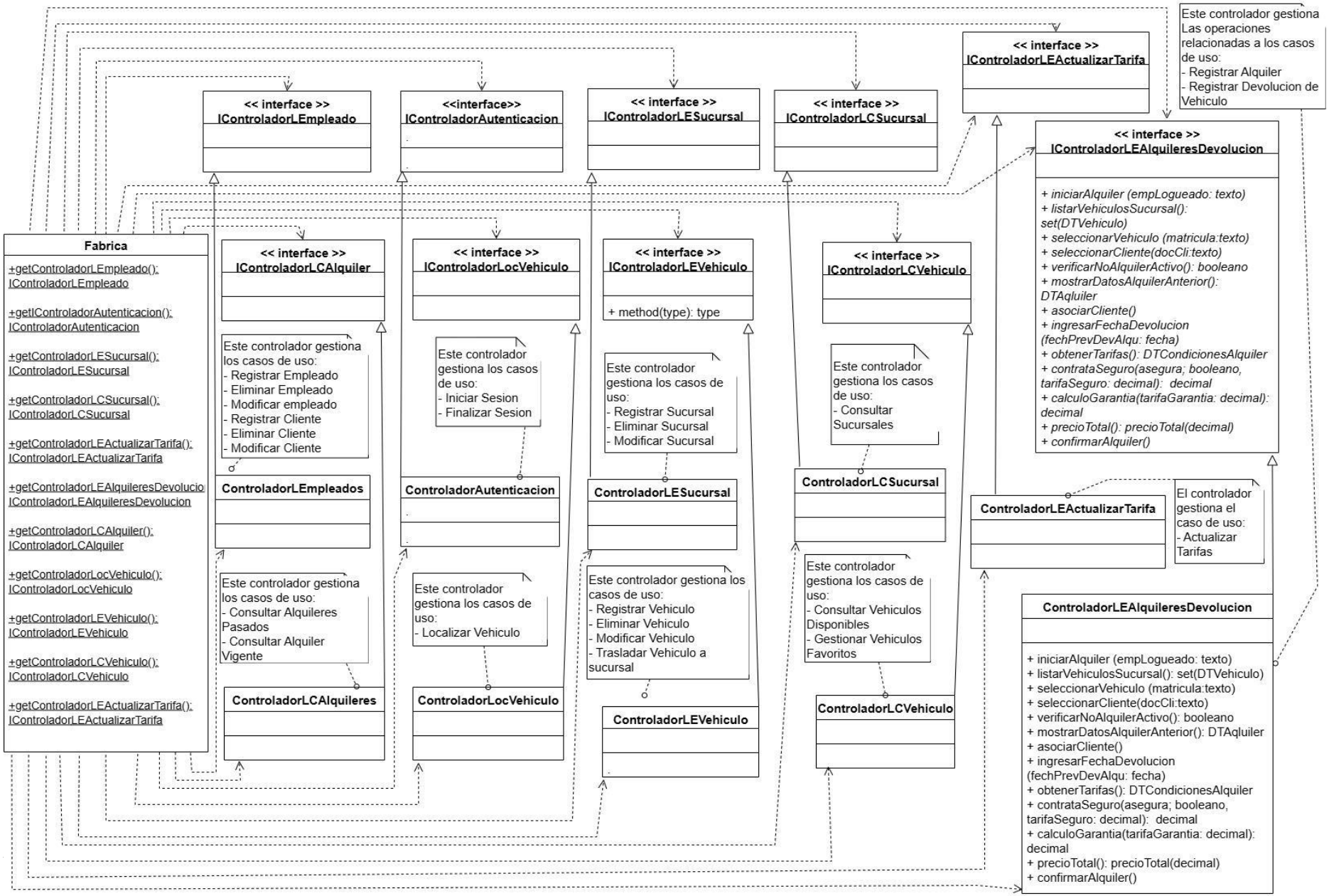
- ControladorLCVehiculos:
    - Consultar Vehículos Disponibles
    - Gestionar Vehículos Favoritos

- ControladorLCAlquileres
    - Consultar Alquiler Vigente
    - Consultar Alquileres Pasados

Interfaces

Cada Controlador tendrá su respectiva interfase por lo que se definen:

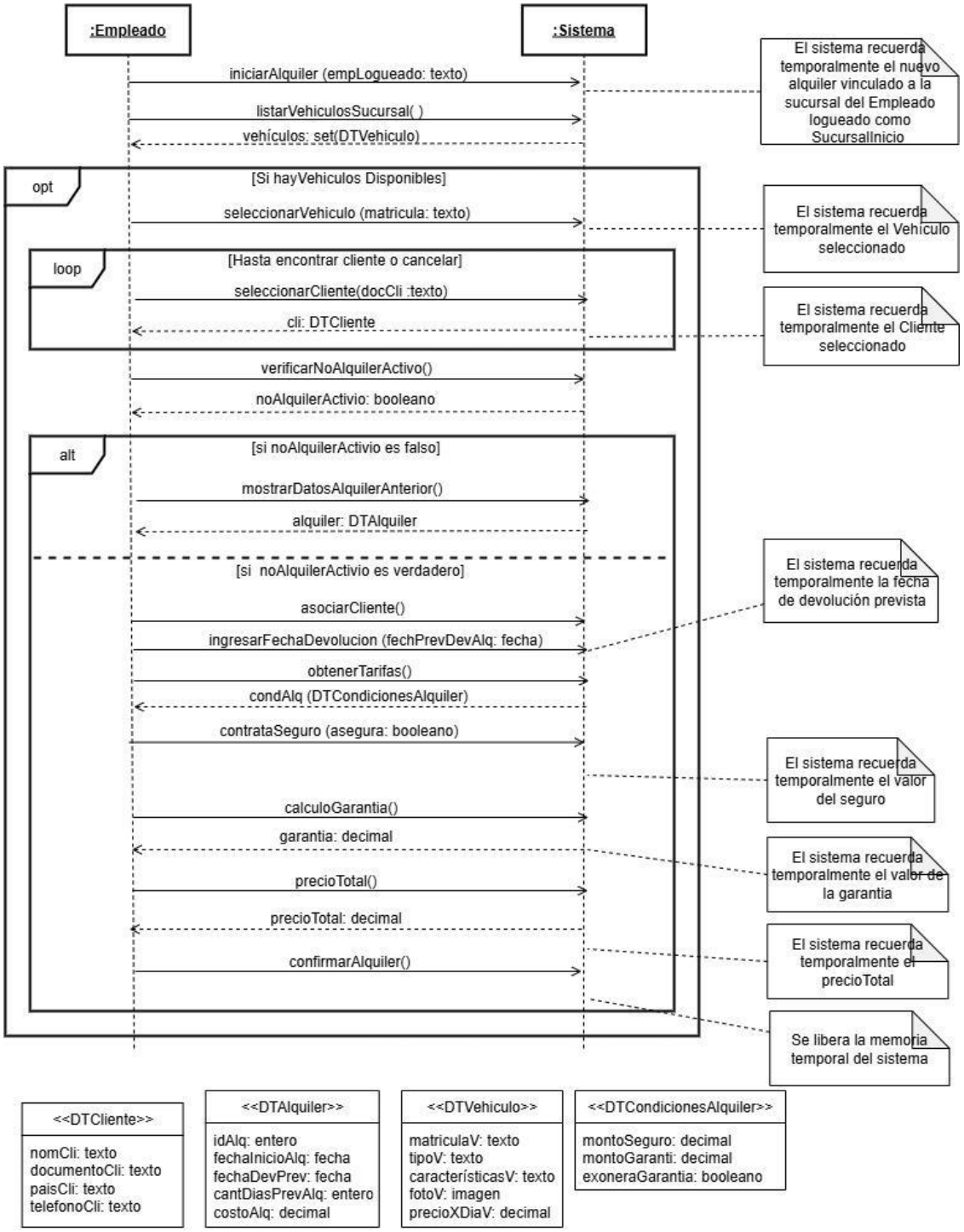
- IControladorLEmpleado
- IControladorLocVehiculo
- IControladorLEAlquileresDevolusion
- IControladorLEVehiculos
- IControladorLESucursal
- IControladorLEActualizarTarifas
- IControladorAutenticacion
- IControladorLCSucursales
- IControladorLCVehiculos
- IControladorLCAlquileres


Este controlador gestiona Las operaciones relacionadas a los casos de uso:
- Registrar Alquiler
- Registrar Devolucion de Vehiculo
<< interface >>
IControladorLEActualizarTarifa
<< interface >>
IControladorLEAlquileresDevolucion
+ iniciarAlquiler (empLogueado: texto)
+ listarVehiculosSucursal(): set(DTVehiculo)
+ seleccionarVehiculo (matricula:texto)
+ seleccionarCliente(docCli:texto)
+ verificarNoAlquilerActivo(): booleano
+ mostrarDatosAlquilerAnterior(): DTAqluiler
+ asociarCliente()
+ ingresarFechaDevolucion (fechPrevDevAlqu: fecha)
+ obtenerTarifas(): DTCondicionesAlquiler
+ contrataSeguro(asegura; booleano, tarifaSeguro: decimal): decimal
+ calculoGarantia(tarifaGarantia: decimal): decimal
+ precioTotal(): precioTotal(decimal)
+ confirmarAlquiler()
<< interface >>
IControladorLEEmpleado
<<interface>>
IControladorAutenticacion
<< interface >>
IControladorLESucursal
<< interface >>
IControladorLCSucursal
Fabrica
+getControladorLEmpleado(): IControladorLEmpleado
+getIControladorAutenticacion(): IControladorAutenticacion
+getControladorLESucursal(): IControladorLESucursal
+getControladorLCSucursal(): IControladorLCSucursal
+getControladorLEActualizarTarifa(): IControladorLEActualizarTarifa
+getControladorLEAlquileresDevolucion(): IControladorLEAlquileresDevolucion
+getControladorLCAlquiler(): IControladorLCAlquiler
+getControladorLocVehiculo(): IControladorLocVehiculo
+getControladorLEVehiculo(): IControladorLEVehiculo
+getControladorLCVehiculo(): IControladorLCVehiculo
+getControladorLEActualizarTarifa(): IControladorLEActualizarTarifa
<< interface >>
IControladorLCAlquiler
<< interface >>
IControladorLocVehiculo
<< interface >>
IControladorLEVehiculo
+ method(type): type
<< interface >>
IControladorLCVehiculo
Este controlador gestiona los casos de uso:
- Registrar Empleado
- Eliminar Empleado
- Modificar empleado
- Registrar Cliente
- Eliminar Cliente
- Modificar Cliente
Este controlador gestiona los casos de uso:
- Iniciar Sesion
- Finalizar Sesion
Este controlador gestiona los casos de uso:
- Registrar Sucursal
- Eliminar Sucursal
- Modificar Sucursal
Este controlador gestiona los casos de uso:
- Consultar Sucursales
ControladorLEmpleados
ControladorAutenticacion
ControladorLESucursal
ControladorLCSucursal
ControladorLEActualizarTarifa
El controlador gestiona el caso de uso:
- Actualizar Tarifas
Este controlador gestiona los casos de uso:
- Consultar Alquileres Pasados
- Consultar Alquiler Vigente
Este controlador gestiona los casos de uso:
- Localizar Vehiculo
Este controlador gestiona los casos de uso:
- Registrar Vehiculo
- Eliminar Vehiculo
- Modificar Vehiculo
- Trasladar Vehiculo a sucursal
Este controlador gestiona los casos de uso:
- Consultar Vehiculos Disponibles
- Gestionar Vehiculos Favoritos
ControladorLCAlquileres
ControladorLocVehiculo
ControladorLEVehiculo
ControladorLCVehiculo
ControladorLEAlquileresDevolucion
+ iniciarAlquiler (empLogueado: texto)
+ listarVehiculosSucursal(): set(DTVehiculo)
+ seleccionarVehiculo (matricula:texto)
+ seleccionarCliente(docCli:texto)
+ verificarNoAlquilerActivo(): booleano
+ mostrarDatosAlquilerAnterior(): DTAqluiler
+ asociarCliente()
+ ingresarFechaDevolucion (fechPrevDevAlqu: fecha)
+ obtenerTarifas(): DTCondicionesAlquiler
+ contrataSeguro(asegura; booleano, tarifaSeguro: decimal): decimal
+ calculoGarantia(tarifaGarantia: decimal): decimal
+ precioTotal(): precioTotal(decimal)
+ confirmarAlquiler()

III. DISEÑO DE INTERACCIONES

Se agrega el DSS y contratos de software ya que sufrieron cambios durante la etapa de diseño.

DSS:



Contratos de Software:

Operación	iniciarAlquiler(empLogueado: texto)
Responsabilidades	Iniciar un nuevo alquiler contratado por un cliente
Salida	No Aplica
Pre-condiciones	- Existe un usuario Empleado E con el nombre de usuario (empLogueado) igual al ingresado por parámetro. - El empleado con nombre de usuario empLogueado se encuentra asociado a una sucursal S. - No existe en el sistema la instancia alquiler A
Post-condiciones	- Se creó el alquiler A - Se vinculó el alquiler A con la sucursal S. - Se seteó el atributo fechaInicio con la fecha actual - El sistema recordó temporalmente el alquiler A vinculado a la sucursal S

Operación	listarVehiculosSucursal(): Set(DTVehiculo)
Responsabilidades	Obtener la colección datos de los vehículos disponibles para alquilar en la sucursal
Salida	Los datos de los vehículos
Pre-condiciones	- El sistema recuerda temporalmente el alquiler A y la sucursal S vinculada al mismo. - Sucursal S tiene una lista de vehículos con el atributo activo = falso
Post-condiciones	- Se devolvió la colección de datos de los vehículos disponibles en la sucursal S

Operación	seleccionarVehiculo(matricula: texto)
Responsabilidades	Seleccionar un vehículo y asignarlo al alquiler que se está

	registrando.
Salida	No aplica
Pre-condiciones	- El sistema recuerda temporalmente el Alquiler A vinculado a la sucursal S. - Existe en el sistema un vehículo V con una matrícula igual que la pasada por parámetro. - El vehículo V está en la sucursal S - El alquiler A no está relacionado con el Vehículo V
Post-condiciones	- Se vinculó el vehículo V con el alquiler A

Operación	seleccionarCliente (DocCli: texto): DTCliente
Responsabilidades	Seleccionar un cliente para el alquiler y mostrar sus datos
Salida	Los datos del cliente con el documento ingresado
Pre-condiciones	- Los datos del documento pasados por parámetro son válidos.
Post-condiciones	Si existe Cliente C con su documento igual al pasado por parámetro DocCli - El sistema devolvió los datos del cliente C. - El sistema recuerda temporalmente el cliente C Sino - El sistema devolvió un valor nulo.

Operación	verificarNoAlquilerActivo(): booleano
Responsabilidades	Verificar si el cliente no tiene un alquiler activo que le impida registrar un nuevo alquiler
Salida	Informar si el cliente está en condiciones de alquilar (no tiene un alquiler activo) o no (tiene un alquiler activo)
Pre-condiciones	- El sistema recuerda temporalmente al cliente C
Post-condiciones	Si no existe alquiler Aa con el atributo activoAl = verdadero, que se encuentre vinculado al cliente C: - Devolvió verdadero. Si existe alquiler Aa con el atributo activoAl = verdadero, que se encuentre vinculado al cliente C: - Devolvió falso.

Operación	MostrarDatosAlquilerAnterior(): DTAlquiler

Responsabilidades	Mostrar datos de un alquiler activo asociado a un cliente
Salida	Los datos de un alquiler asociado a un cliente.
Pre-condiciones	- El sistema recuerda un cliente C - Existe un alquiler Aa con activoAlq verdadero vinculado al cliente C
Post-condiciones	- El sistema devuelve los datos del Alquiler Aa

Operación	asociarCliente()
Responsabilidades	Asociar el cliente seleccionado al nuevo alquiler en curso
Salida	Los datos de un alquiler asociado a un cliente.
Pre-condiciones	- El sistema recuerda un cliente C - El sistema recuerda un alquiler A - El cliente C no está vinculado al alquiler A
Post-condiciones	Si cliente C y alquiler A no son nulos - Se vinculó el cliente C al alquiler A.

Operación	IngresarFechaDevolucion(fecDevPrev: fecha)
Responsabilidades	Ingresar fecha de devolución prevista del alquiler que se está registrando.
Salida	No aplica
Pre-condiciones	- El sistema recuerda temporalmente el Alquiler A - La fecha ingresada por parámetro debe ser válida - La fecha fecDevPrev deberá ser posterior a fechaInicioAlq
Post-condiciones	- Se seteó el atributo fechaDevPrevistaAlq con el dato ingresado por parámetro.

Operación	obtenerTarifas(): DTCondicionesAlquiler
Responsabilidades	Informar tarifas actualizadas de alquiler. monto de seguro,

	monto de la garantía e informar si es posible que el cliente pueda exonerar la garantía.
Salida	Datos de monto de tarifas y posibilidad de exonerar la garantía
Pre-condiciones	- El sistema recuerda temporalmente el cliente C - Existe en el sistema el valor asignado a la tarifa de seguro VS. - Existe en el sistema el valor asignado al monto del depósito de Garantía VG.
Post-condiciones	El dato montoSeguro del DTCondicionesAlquiler devolvió el valor VS. El dato montoGarantía del DTCondicionesAlquiler devolvió el valor VG. En caso de que existan al menos 3 alquileres previos vinculados con el cliente C - El valor de exoneraGarantia del DTCondicionesAlquiler devolvió el valor Verdadero. En caso de que no existan al menos 3 alquileres previos vinculados con el cliente C : - El valor de exoneraGarantia del DTCondicionesAlquiler devolvió el valor Falso. El sistema recordó temporalmente el valor booleano de ExoneraGarantía

Operación	contrataSeguro (asegura: booleano)
Responsabilidades	Asignarle un valor al monto del seguro del automóvil (en caso de que no se quiera contratar seguro, devuelve valor 0)
Salida	El valor del monto a abonar por concepto de seguro.
Pre-condiciones	- El sistema recuerda temporalmente el Alquiler A - Existe un valor asignado a la tarifa de seguro - El valor ingresado por parámetro es válido
Post-condiciones	- Se seteó el atributo seguro en Alquiler A de acuerdo al dato ingresado por parámetro. Si asegura corresponde el monto del seguro, si no asegura el monto del seguro será 0

Operación	calculoGarantia ()
Responsabilidades	Asignarle un valor al monto del depósito de garantía (en caso de que se exonere la garantía asigna valor 0)

Salida	El monto total a depositar por concepto de garantía
Pre-condiciones	- El sistema recuerda temporalmente el Alquiler A con un cliente C asociado. - El Alquiler A tiene el atributo seguro cargado. - El sistema recuerda temporalmente el parámetro exonera que indica si el cliente puede alquilar. - Existe un valor asignado a la tarifa de garantía
Post-condiciones	En caso de que exonera sea verdadero y VS sea mayor que 0 - Se seteó el atributo garantía del Alquiler A con el valor 0 En caso contrario - Se seteó el atributo garantía del Alquiler A con el valor VG. El sistema devolvió el valor del atributo garantía

Operación	precioTotal(): decimal
Responsabilidades	Informar el precio total que debe abonar el cliente de acuerdo a las condiciones del alquiler.
Salida	El monto total a pagar
Pre-condiciones	- El sistema recuerda temporalmente el Alquiler A vinculado a un vehículo V y a un cliente C. - Los montos de seguro, garantía, fecha de inicio y fecha de devolución se encuentran seteados.
Post-condiciones	- Se seteó el atributo precio en alquiler A - El sistema devolvió el precio total compuesto por la suma de alquiler (precioxDia x Cantidad de Dias Previstos del Alquiler) + montoSeguro + montoGarantía. - El sistema recordó temporalmente el Precio Total.

Operación	confirmarAlquiler()
Responsabilidades	Registrar definitivamente el alquiler que se está manteniendo en la memoria temporal.
Salida	No aplica.

Pre-condiciones	- El sistema recuerda temporalmente el Alquiler A
	- El Alquiler A está vinculado al cliente C.
	- El Alquiler A está vinculado al vehículo V.
	- El Alquiler A está vinculado a la Sucursal de inicio S.
	- Están seteados los atributos
	  - fechaInicioAlq.
	  - fechaDevPrevistaAlq.
	  - seguro
	  - garantia

Post-condiciones	- Se seteó el valor activoAlq en verdadero. - Se recordó definitivamente el Alquiler A. - Se liberó la memoria temporal del sistema.

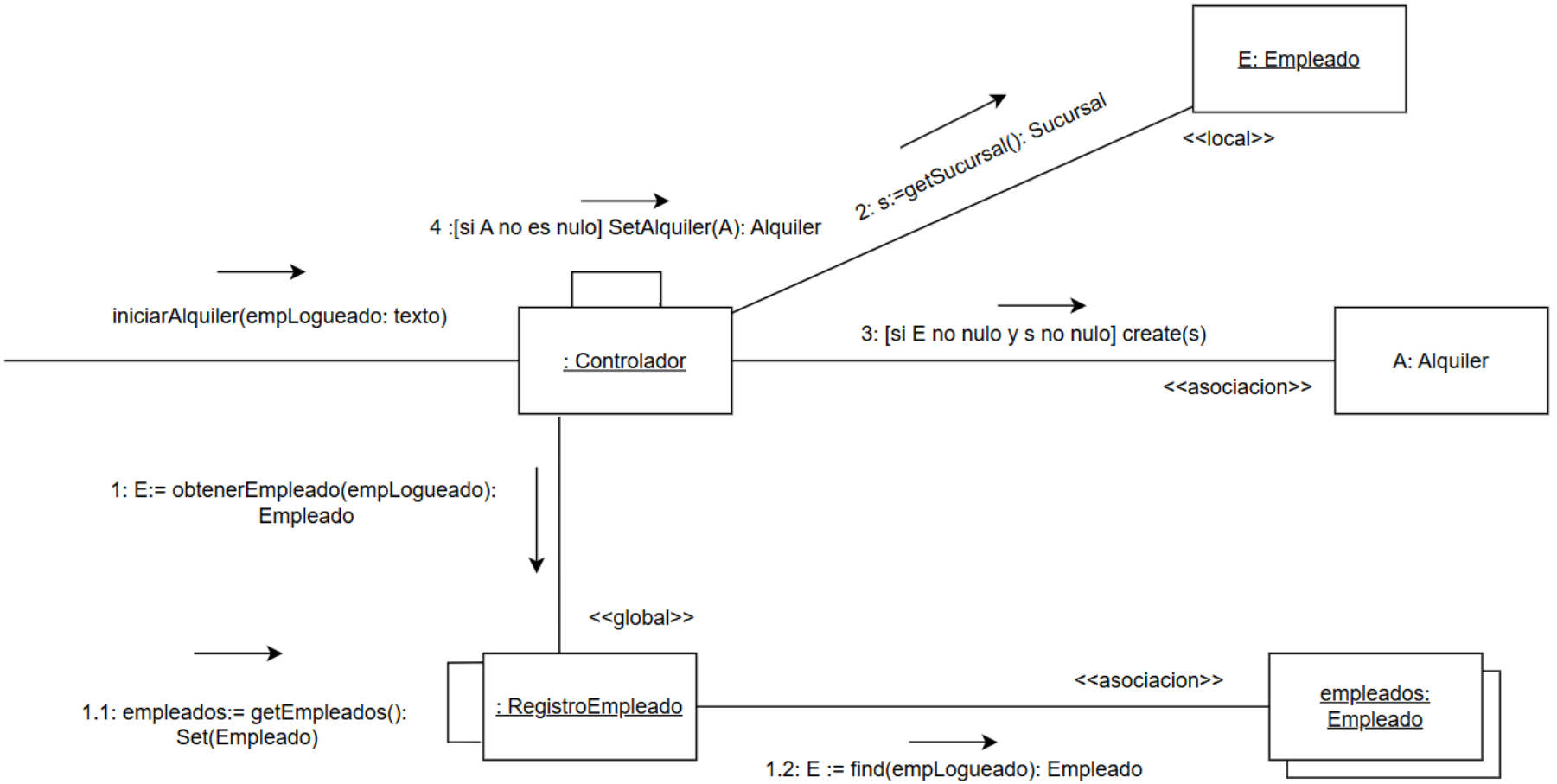
3.1. Diagramas de Comunicación del Sistema

En esta sección se mostrarán los diagramas de comunicación de las operaciones del sistema. Tomando como punto de partida y fuente principal para diseñar el artefacto al Diagrama de Secuencia del Sistema en donde se muestran de manera ordenada y secuencial los mensajes (operaciones del sistema) que se envían desde la capa de presentación a la capa lógica. También en menor medida nos apoyamos en el Modelo conceptual para tener una comprensión estructural del problema

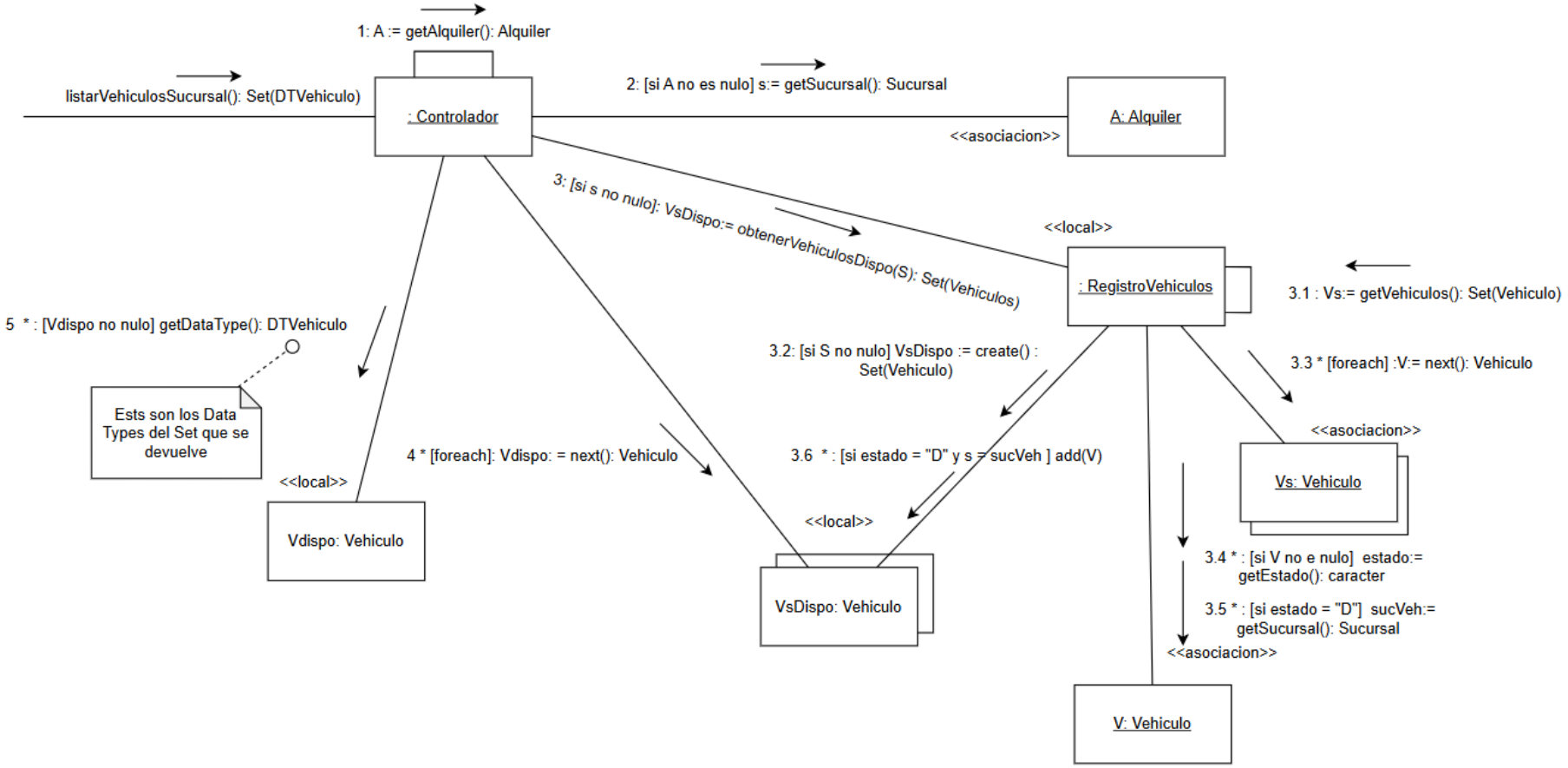
Este es un tipo de diagrama que muestra la interacción de forma visual entre el sistema (controlador) y sus distintos colaboradores para llevar a cabo la operación que se le pide. En este punto ya nos adentramos en el diseño a nivel de algoritmos. Todo diagrama de comunicación empieza con el controlador recibiendo la operación de sistema pertinente. Los mensajes entre el controlador y las clases y objetos colaboradores están numerados en orden de ejecución.

En este punto se siguen teniendo en cuenta los principios del GRASP Controller indicado en la sección de Definición de Interfaces del Sistema y Controladores

iniciarAlquiler(empLogueado: texto)



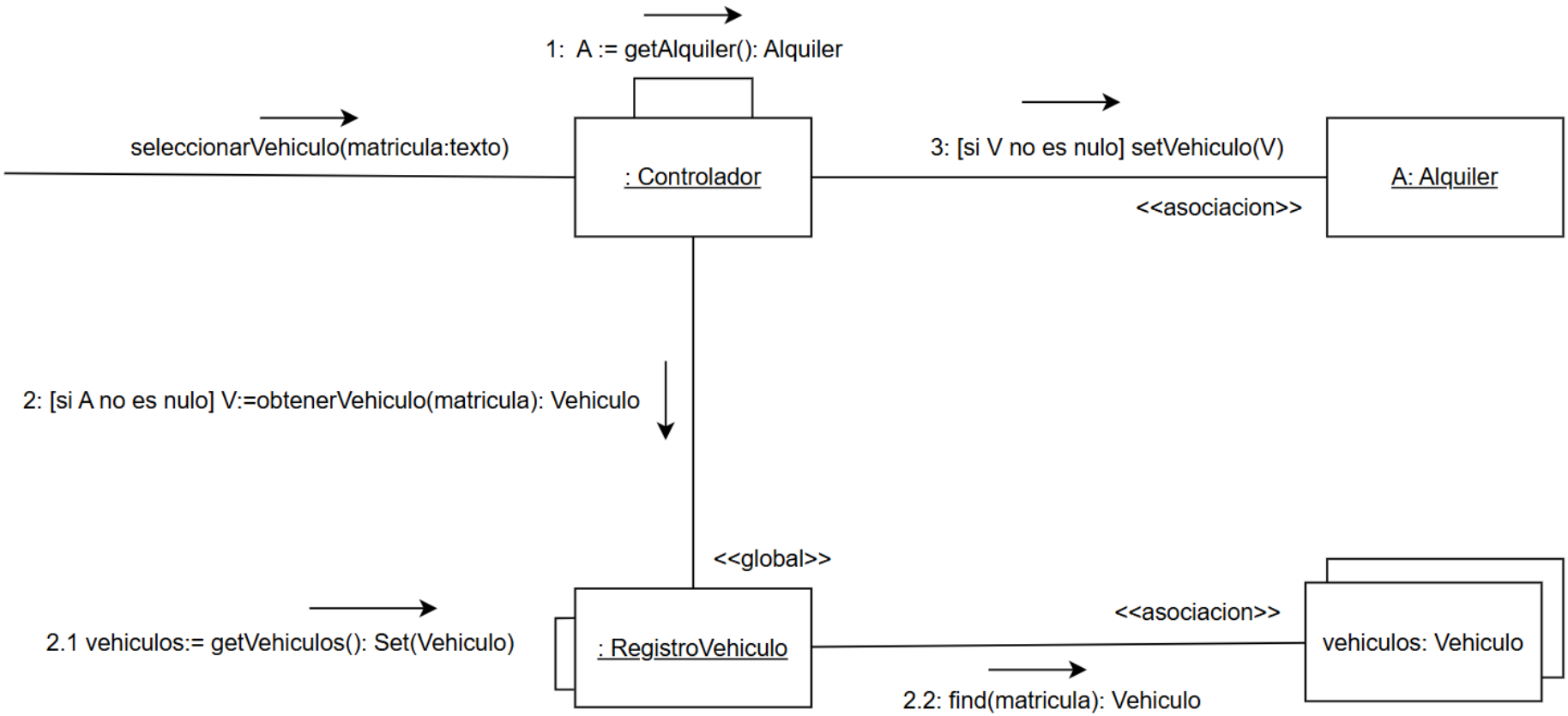
listarVehiculosSucursal(): Set(DTVehiculo)



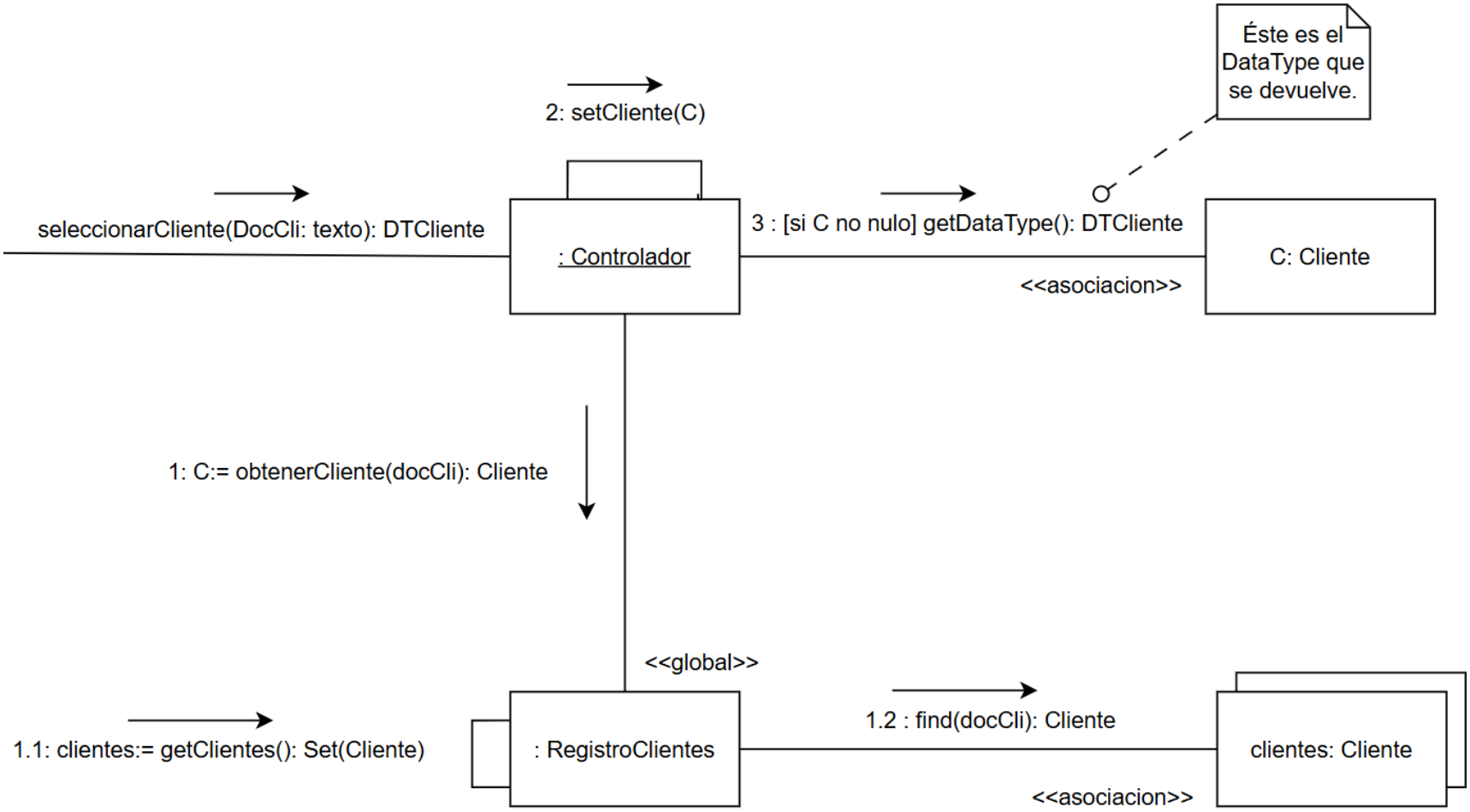
seleccionarVehiculo(matricula: texto)

Nicolás Dagys
Alexander Lemos
Camilo Pereyra

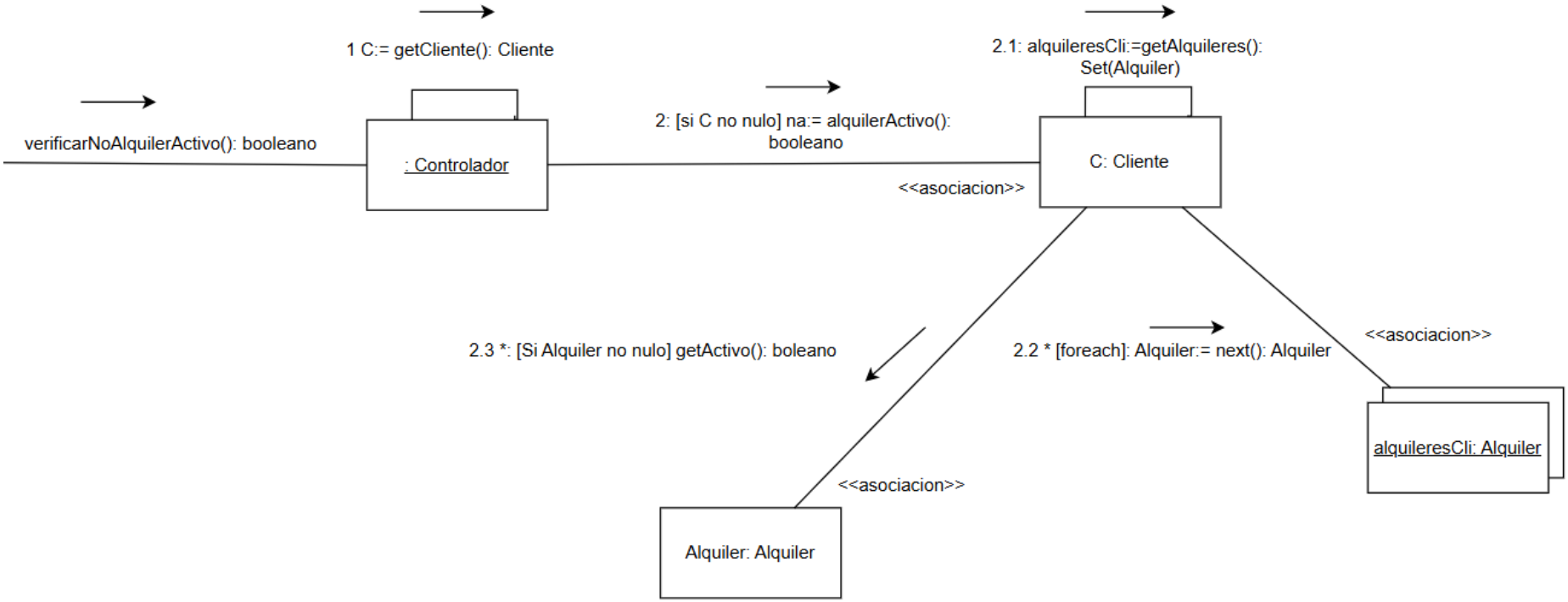
04/06/2025



seleccionarCliente (DocCli: texto): DTCliente



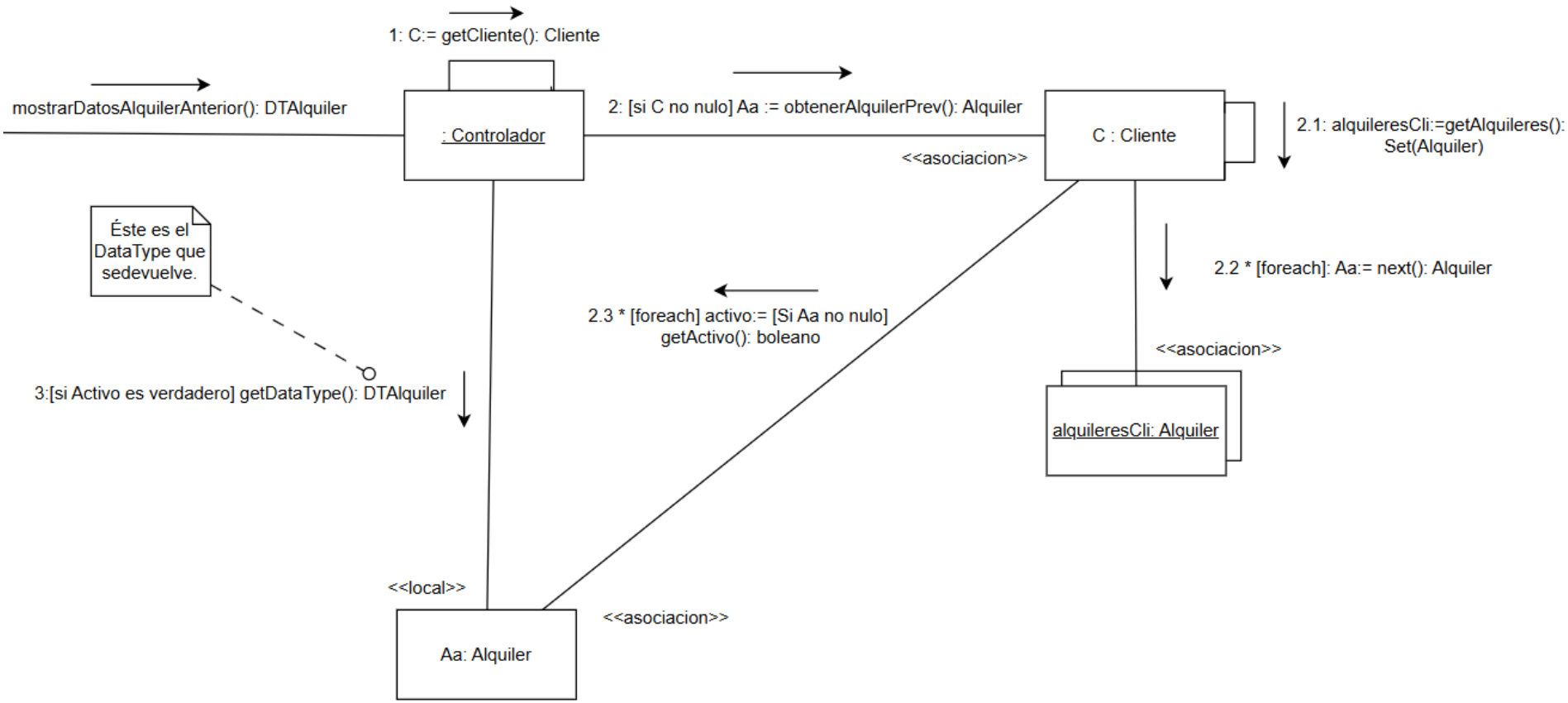
verificarNoAlquilerActivo(): booleano



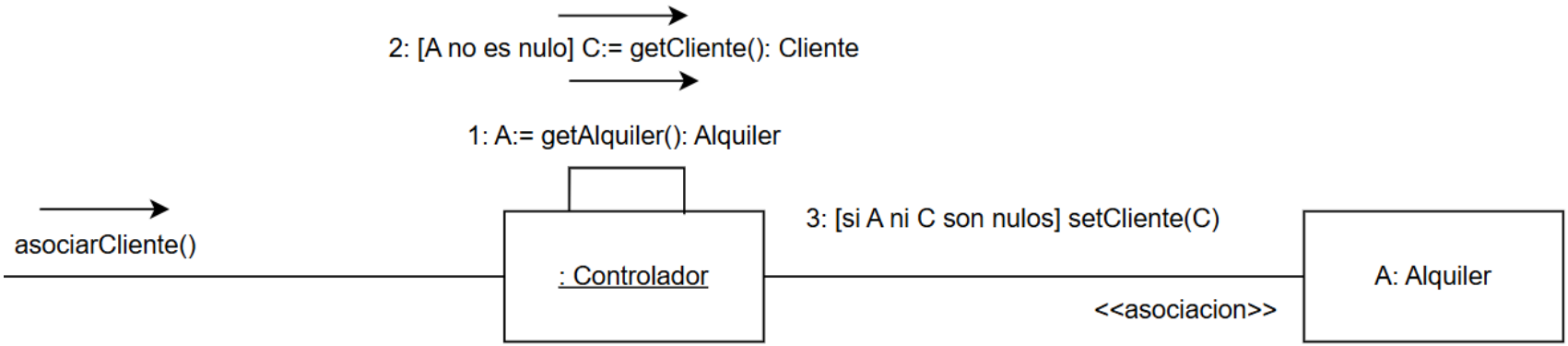
MostrarDatosAlquilerAnterior(): DTAlquiler

Nicolás Dagys
Alexander Lemos
Camilo Pereyra

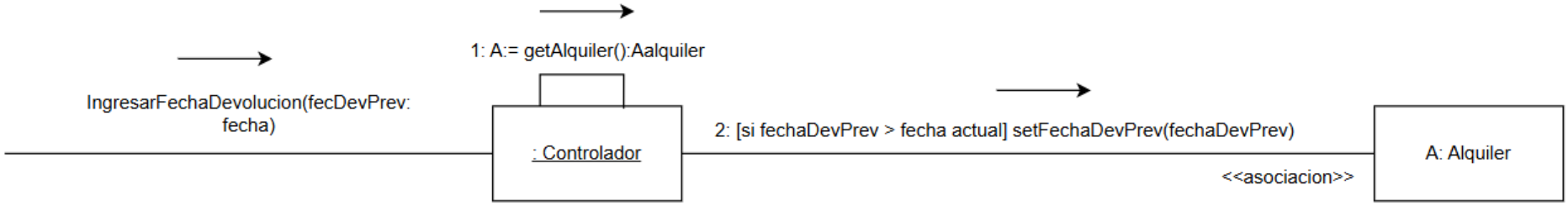
04/06/2025



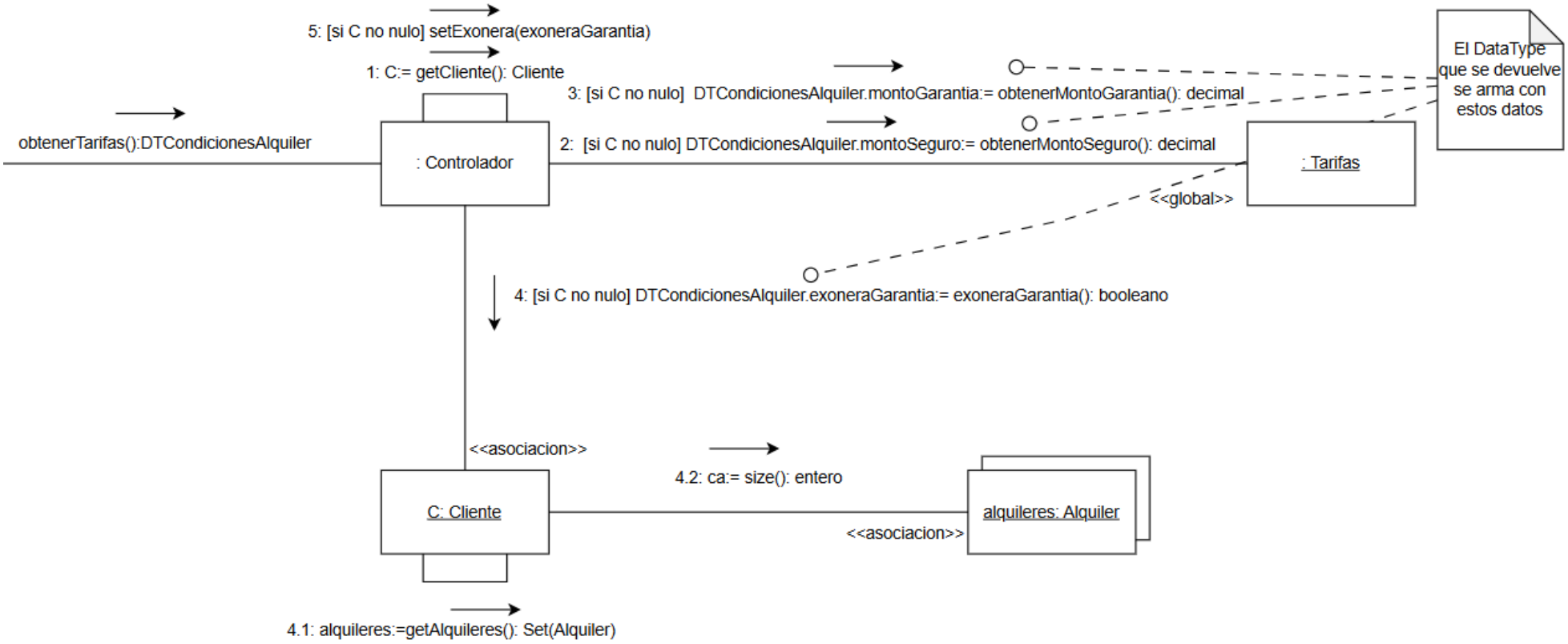
asociarCliente()



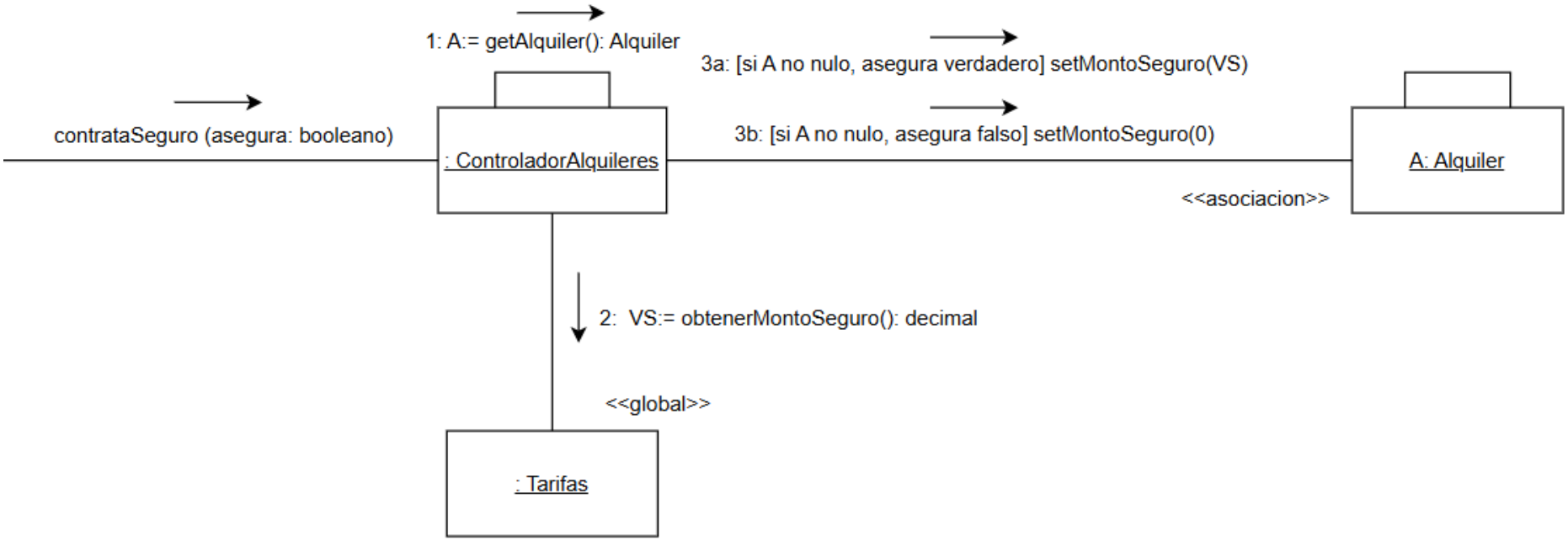
IngresarFechaDevolucion(fecDevPrev: fecha)



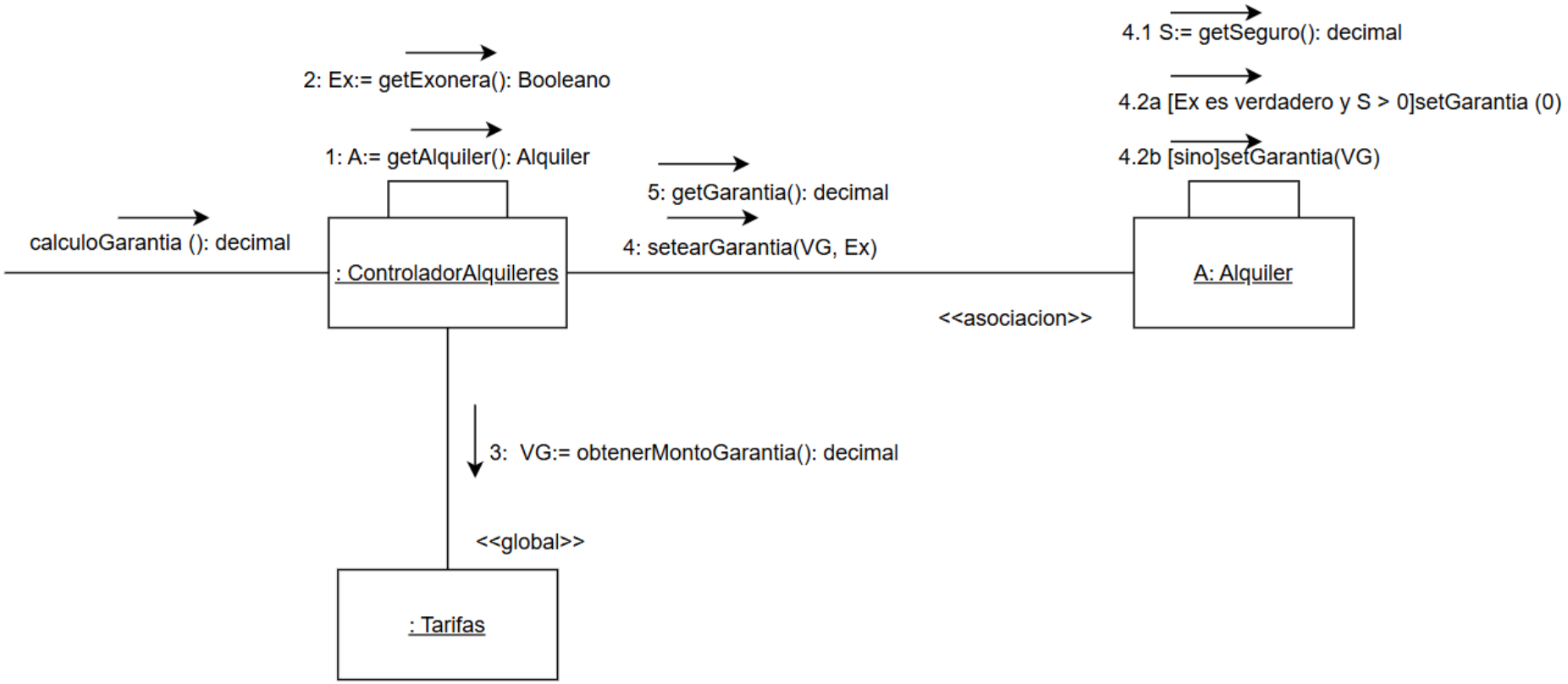
obtenerTarifas(): DTCondicionesAlquiler



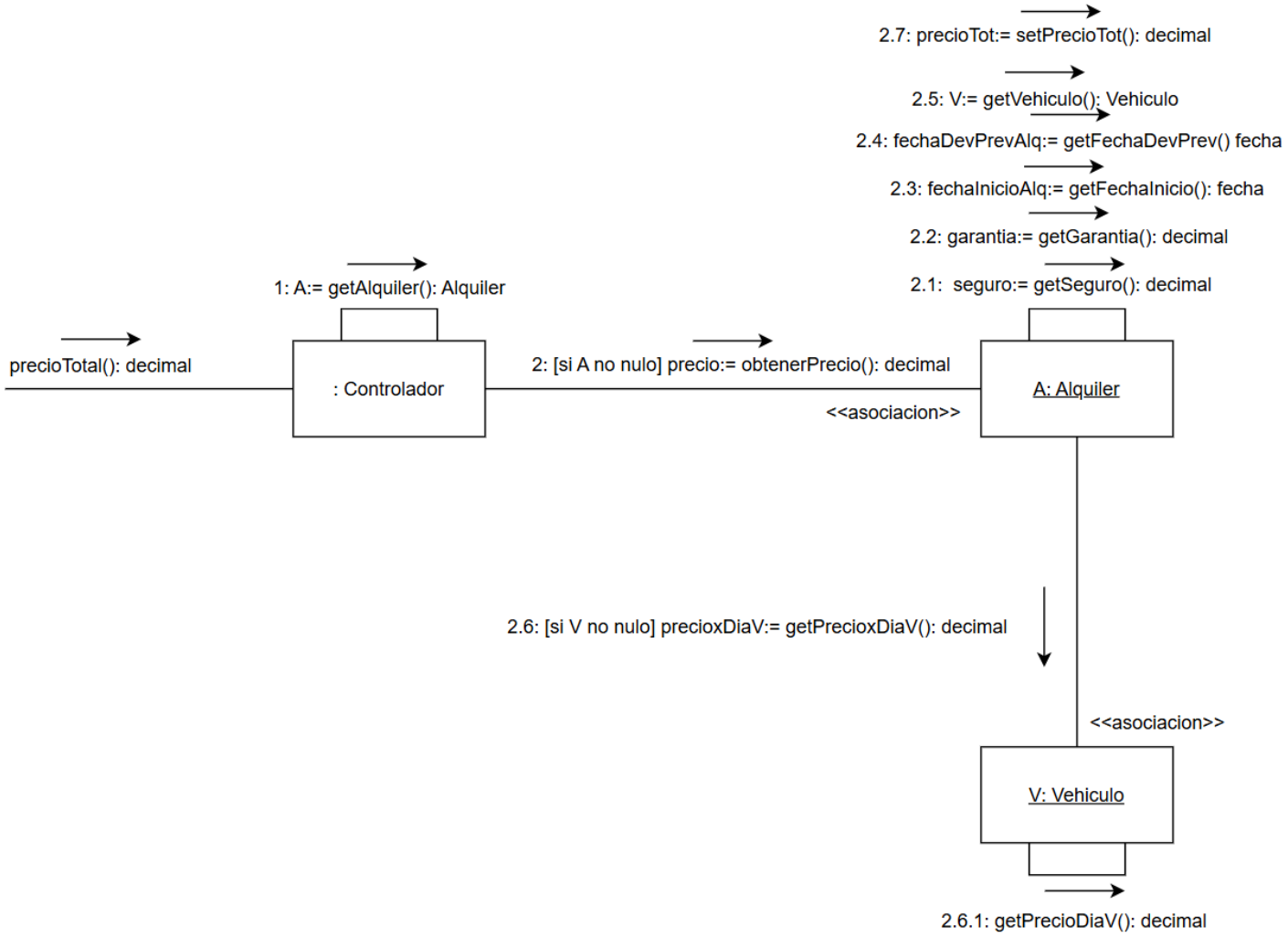
contrataSeguro(asegura: booleano, tarifaSeguro: decimal): decimal



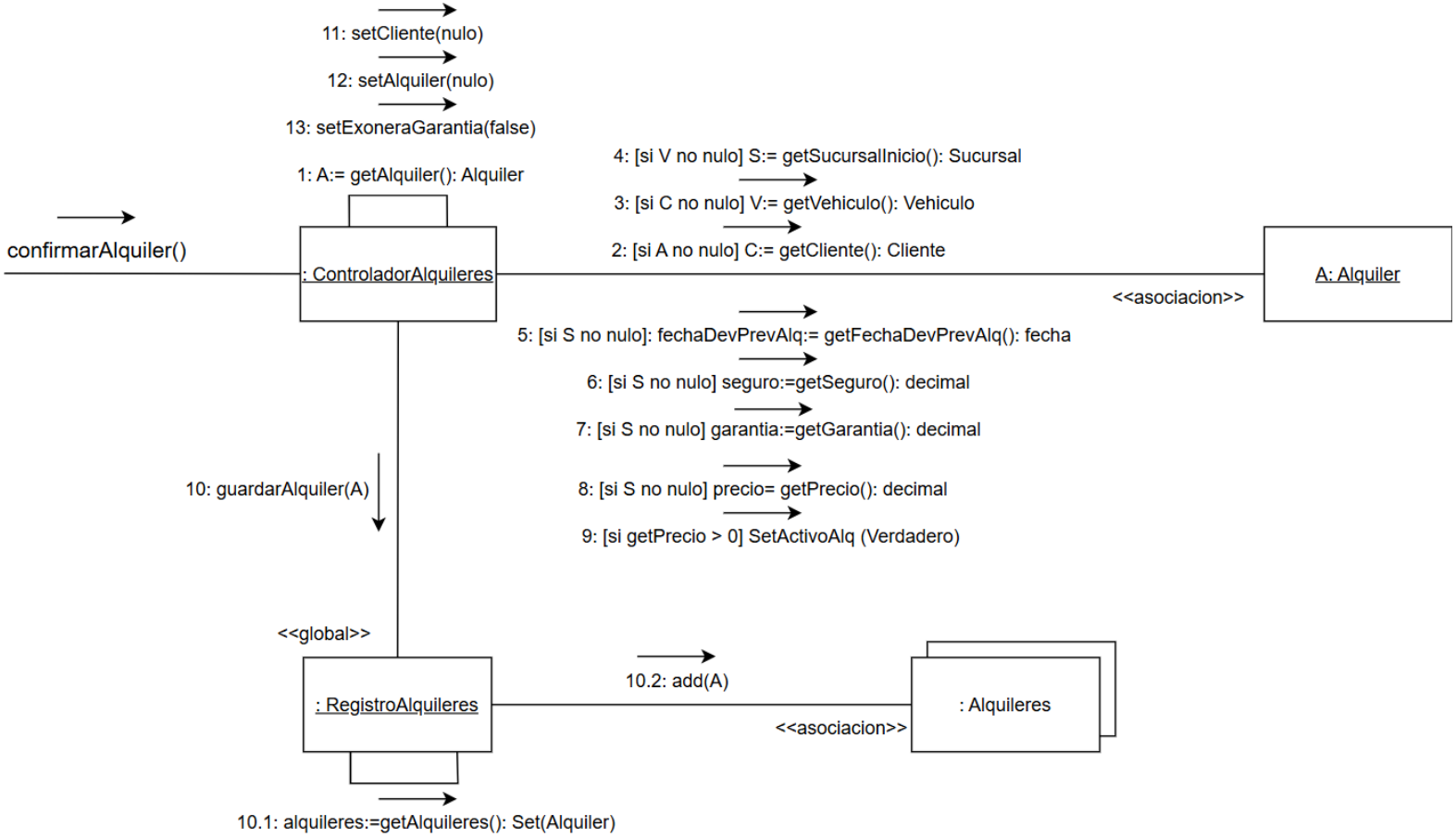
calculoGarantia (tarifaGarantia: decimal): decimal



precioTotal(): decimal


2.7: precioTot:= setPrecioTot(): decimal
2.5: V:= getVehiculo(): Vehiculo
2.4: fechaDevPrevAlq:= getFechaDevPrev() fecha
2.3: fechaInicioAlq:= getFechaInicio(): fecha
2.2: garantia:= getGarantia(): decimal
2.1: seguro:= getSeguro(): decimal
1: A:= getAlquiler(): Alquiler
precioTotal(): decimal
: Controlador
2: [si A no nulo] precio:= obtenerPrecio(): decimal
<<asociacion>>
A: Alquiler
2.6: [si V no nulo] precioxDiaV:= getPrecioxDiaV(): decimal
<<asociacion>>
V: Vehiculo
2.6.1: getPrecioDiaV(): decimal

confirmarAlquiler()



3.2. Patrones de diseño

A partir de problemas que se repetían con el tiempo surgieron los patrones de diseño como una forma de sistematizar, organizar y documentar soluciones aplicadas en experiencias previas definidas como patrones.
Los patrones ofrecen soluciones generales que deben adaptarse para su aplicación concreta.
En nuestro sistema, se incorporarán algunos de estos patrones para abordar desafíos puntuales que surgen durante el desarrollo.
El patrón Adapter se utilizará para la integración de nuestra aplicación con el servicio externo de geolocalización, contemplando posibles cambios en el sistema del proveedor actual o ante la eventual sustitución por un nuevo proveedor.
Se utilizará el patrón State para gestionar el comportamiento de los vehículos en función de su estado operativo, controlando la lógica de las operaciones según su estado disponible, alquilado o fuera de servicio.
Como se describe en el Apartado II “Definición de Interfaces del Sistema y Controladores” se utiliza el patrón Fábrica conjuntamente con Interfaces para romper con la dependencia y el acoplamiento entre la Capa Presentación y los Controladores.

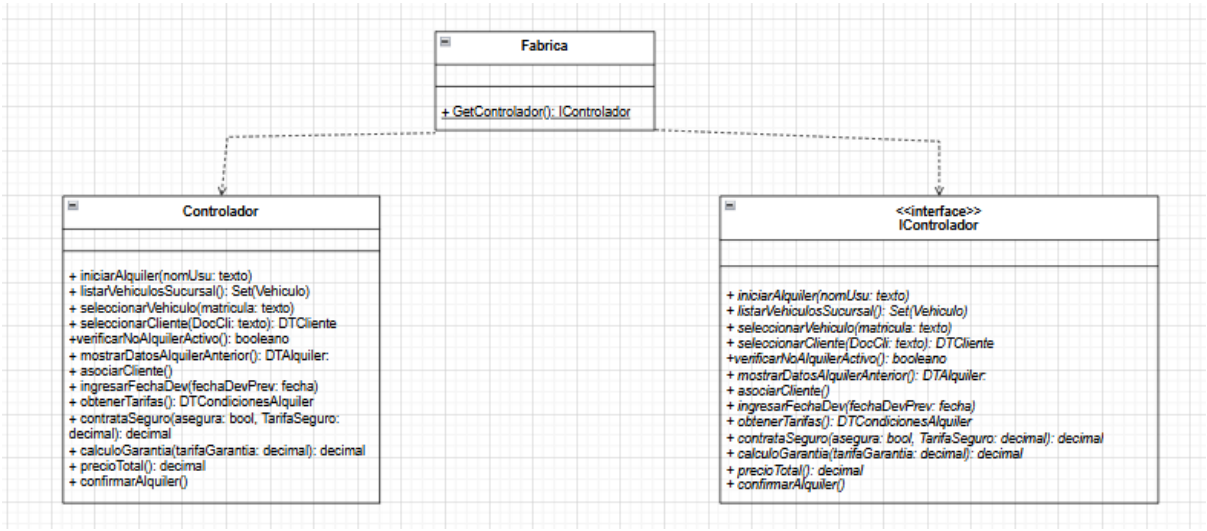
Patrón Fabrica

Este patrón centraliza la creación de objetos en una clase especializada (Fábrica).
En lugar de instanciar directamente los objetos desde el controlador, se delega esta responsabilidad en la fábrica, situación que permite desacoplar la lógica de creación del resto del sistema.
La utilización del patrón Fábrica facilita la posibilidad de modificar la implementación interna de los objetos creados sin afectar al resto del sistema.
Se aplicará este patrón a cada controlador del sistema.

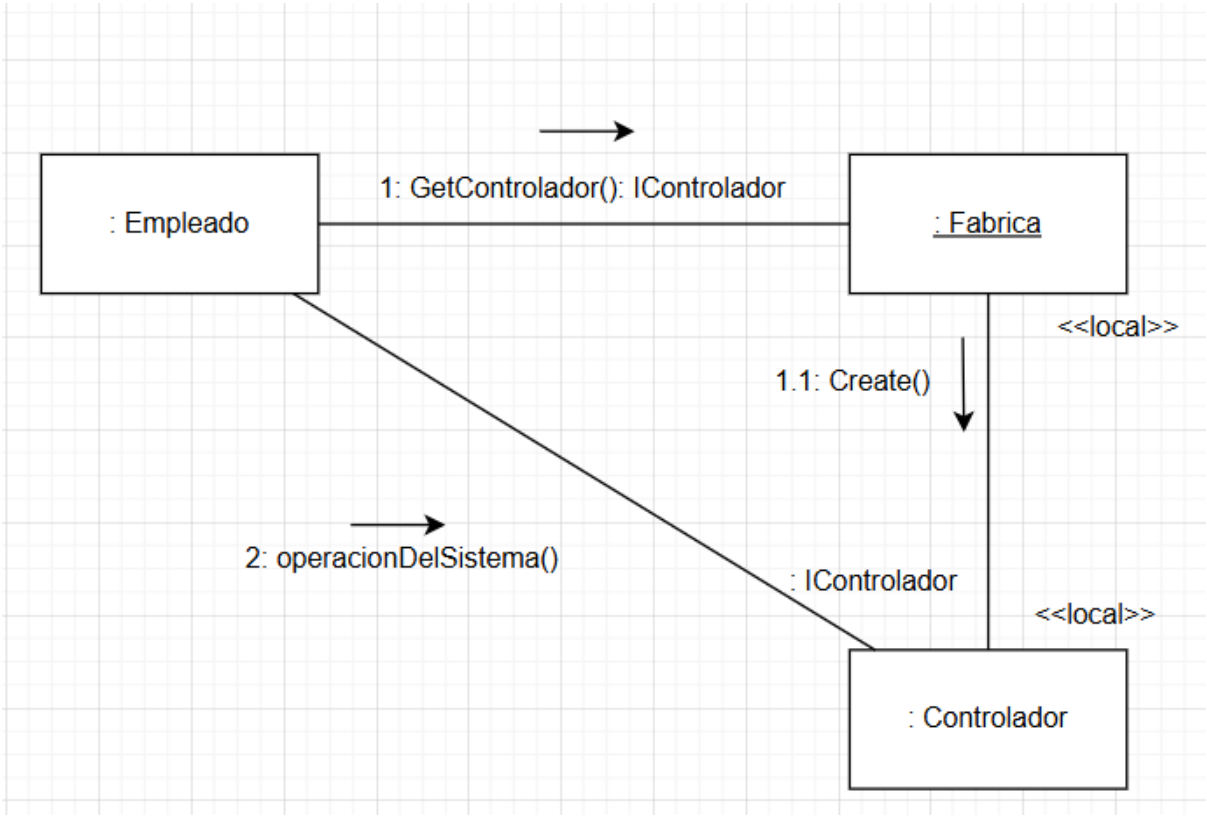
Se define el siguiente componente:

<u>Fábrica</u>
La fábrica tendrá una operación estática GetControlador() que retorna la instancia del controlador como tipo interfaz IControlador. El controlador implementa la Interfaz IControlador. De esta manera se ejecutan las operaciones del sistema a través de la instancia del controlador enmascarado por la interfaz

<u>Estructura</u>



Comportamiento (Diagrama de comunicación)



Patrón Adapter

Utilizamos el patrón Adapter para facilitar la comunicación entre nuestro componente LocalizarVehiculo y el ServicioGeolocalización provisto por la empresa externa.
Este servicio nos proveerá de datos de latitud y longitud de un vehículo como par de números enteros.
Teniendo en cuenta que estos datos se obtienen de una empresa externa que podría cambiar o modificar sus servicios, se utilizan adaptadores:

Se definen los siguientes componentes:
LocalizadorVehiculo
Se comunica con la interface IAdaptadorLocVehiculo y utilizará sus objetos (adaptadores) para obtener la información requerida (latitud y longitud del vehículo).
IAdaptadorLocVehiculo
Conforma la interfaz de la que dependerán todos los adaptadores. Cuenta con la operación abstracta obtenerCoordenadas que recibe por parámetro la matrícula del vehículo y devuelve el par de números enteros.
SistemaGeolocalizacion
Define los servicios que provee la empresa externa, los cuales necesitan ser adaptados para poder ser utilizados por LocalizadorVehiculo. Cuenta con una operación llamada obtenerCoordenadasDestino que recibe por parámetro una matrícula y devuelve un par de números enteros.
Adaptador
Implementa la operación obtenerCoordanadas de la interfaz IAdaptadorLocVehiculo adaptando los datos proporcionados por SistemaGeolocalizacion para poder enviarlos a LocalizadorVehiculo.
La operación **obtenerCoordenadas** llama a la **operacionDestino**, le pasa por **parámetro** la **matrícula** del vehículo a localizar (caracteres) y **ésta devolverá la información de latitud y longitud.**
Ejemplo:
Suponemos que LocalizarVehiculos necesita para trabajar los datos de latitud y longitud como par de decimales.
Contrata a una Empresa de Geolocalización devuelve datos de latitud y longitud en ese formato de par de datos.

- El Adaptdor llamará a la operación correspondiente del Servicio externo, le pasa por parámetro la matrícula y recibe el par de datos que podrá utilizar LocalizarVehiculo.

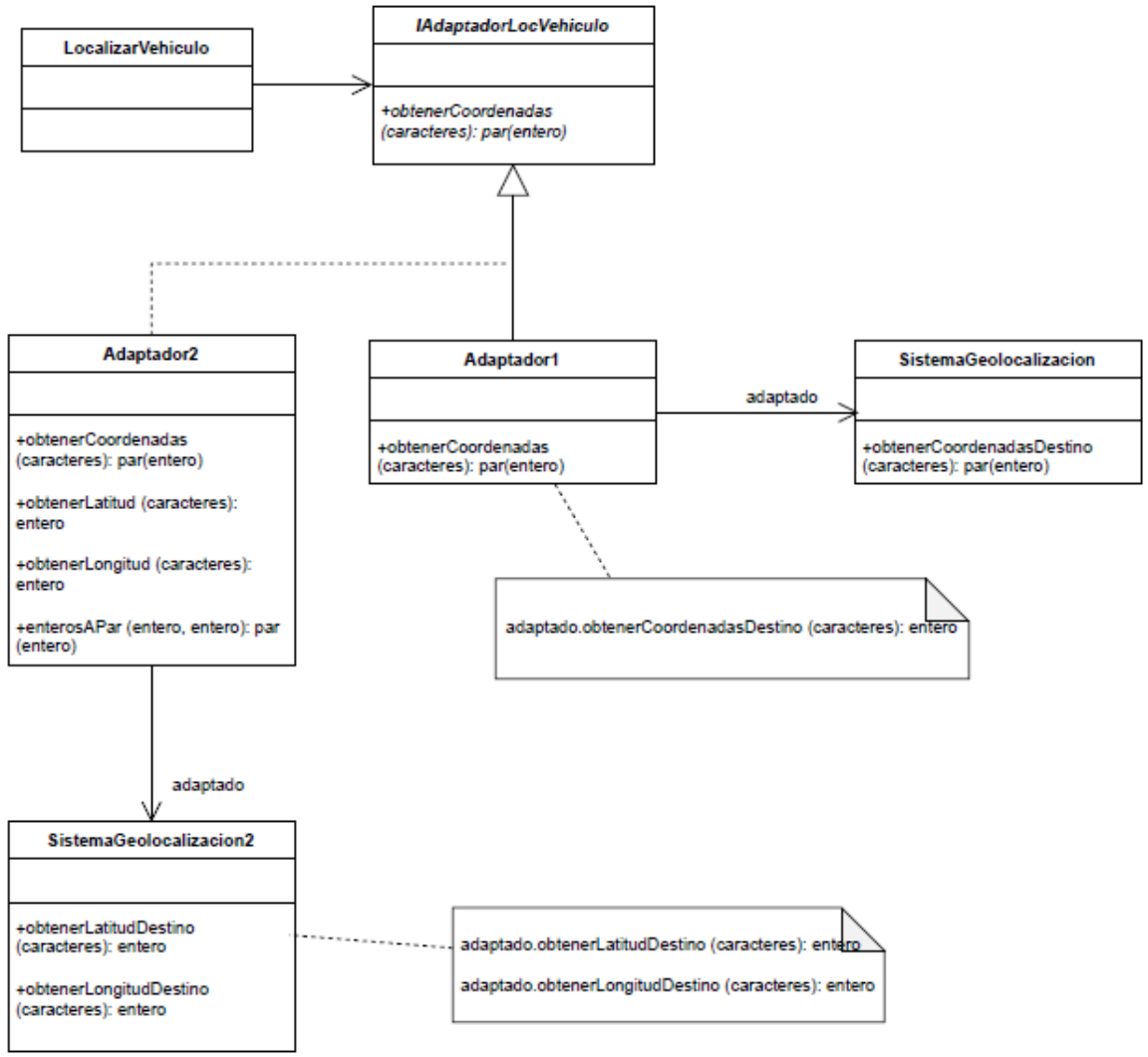
Luego esta Empresa de Geolocalización cambia por otra que devuelve los datos de latitud y longitud como números enteros separados.

- Se define otro Adaptador (Adaptador2) que tendrá operaciones internas que al recibir la información de la Empresa de Geolocalización la adaptan al formato de par de enteros que requiere LocalizarVehículo.
- Adaptador2 tendrá las siguientes operaciones:
    - **obtenerLatitud** que recibe por parámetro la matrícula y devuelve la latitud del vehículo.

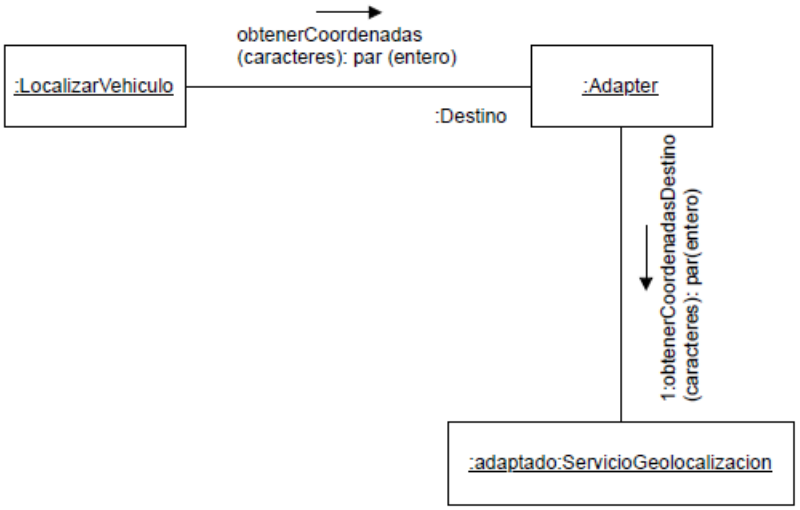
- **obtenerLongitud** que recibe por parámetro la matrícula y devuelve la longitud del vehículo.
- **operacionEnterosAPares:** Conforma el par por los números enteros recibidos como latitud y longitud
- **operación obtenerCoordenadas** recibe la matrícula y devuelve el par de enteros conformado en la operación anterior

Estructura (Diagrama de Clases)

Obs. Solo a efectos de ilustrativos se incluye en el diagrama lo que sería un segundo adaptador con la posibilidad de otro sistema externo



Comportamiento (Diagrama de Comunicación)



Patrón State con singleton

El patrón State se utiliza para permitir que, ante una misma solicitud, un objeto adopte diferentes comportamientos según su estado interno.
En el sistema BIOS Rent, este patrón se aplicará a los vehículos, donde según su estado interno tendrá un comportamiento diferente.
A su vez estos estados serán Singleto, para que solo haya una instancia de estado por vehículo. Los estados son: Disponible, Alquilado o Fuera de Servicio.
El comportamiento del vehículo frente a determinadas operaciones variará en función de su estado.

Ejemplo descriptivo: Devolución de vehículo (finalizado el alquiler)

- Estado Alquilado: Se realizan acciones relacionadas a la devolución de vehículo, entre ellas el cambio en su estado: Si el vehículo está en condiciones pasa al estado de Disponible, en caso contrario pasa a Fuera de Servicio.
- Estado Disponible: Se lanza una excepción con el mensaje: “No se puede devolver un vehículo que no está alquilado”.
- Fuera de Servicio:Se lanza una excepción con el mensaje: “No se puede devolver un vehículo que no está alquilado y se encuentra fuera de servicio”.

Diagrama de Clases

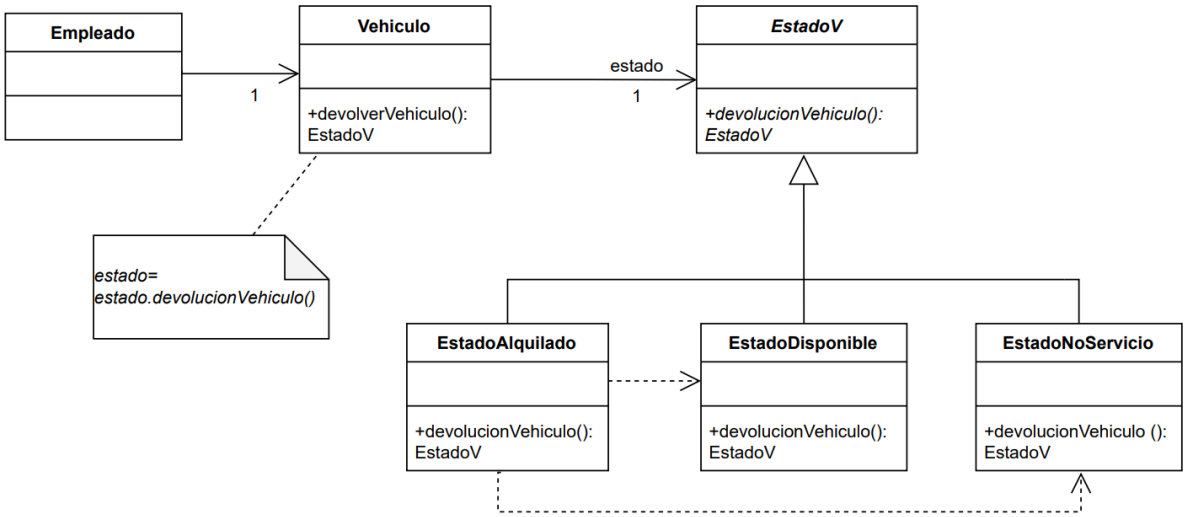
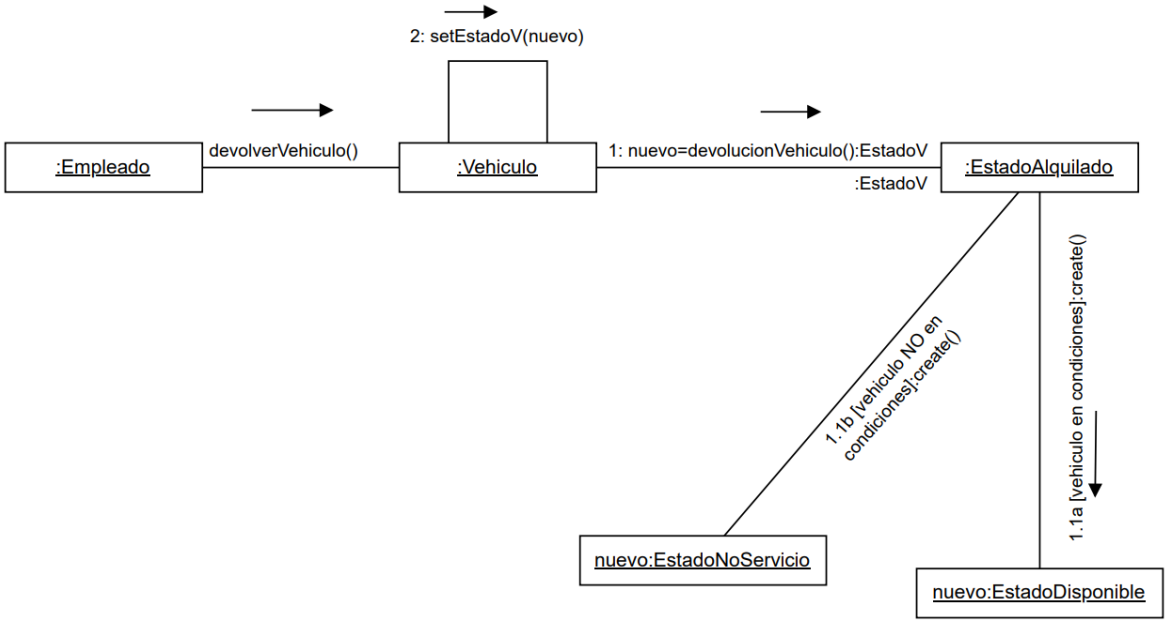


Diagrama de Comunicación



También se aplica el patrón State para operaciones como:
Localizar vehículo

- Estado Alquilado: Se realiza la conexión con el sistema de geolocalización externo.

- Estado Disponible: Se informa que el vehículo está disponible y se indica en qué sucursal se encuentra.
- Estado Fuera de Servicio: Se informa que el vehículo se encuentra fuera de servicio.

Eliminar Vehículo
- Estado Disponible o Fuera de Servicio: El vehículo se elimina del sistema.
- Estado Alquilado: Se lanza una excepción con el mensaje: “No se puede eliminar un vehículo que actualmente se encuentra alquilado”.

Seleccionar Vehículo para Alquilar:
- Estado Disponible: Se asocia el vehículo a un alquiler y cambia su estado a Alquilado.
- Estado Alquilado: Se lanza un mensaje donde se indica: “No se puede alquilar este vehículo porque se encuentra alquilado.”
- Estado Fuera de servicio: Se lanza un mensaje donde se indica: “No se puede alquilar este vehículo porque se encuentra fuera de servicio.”

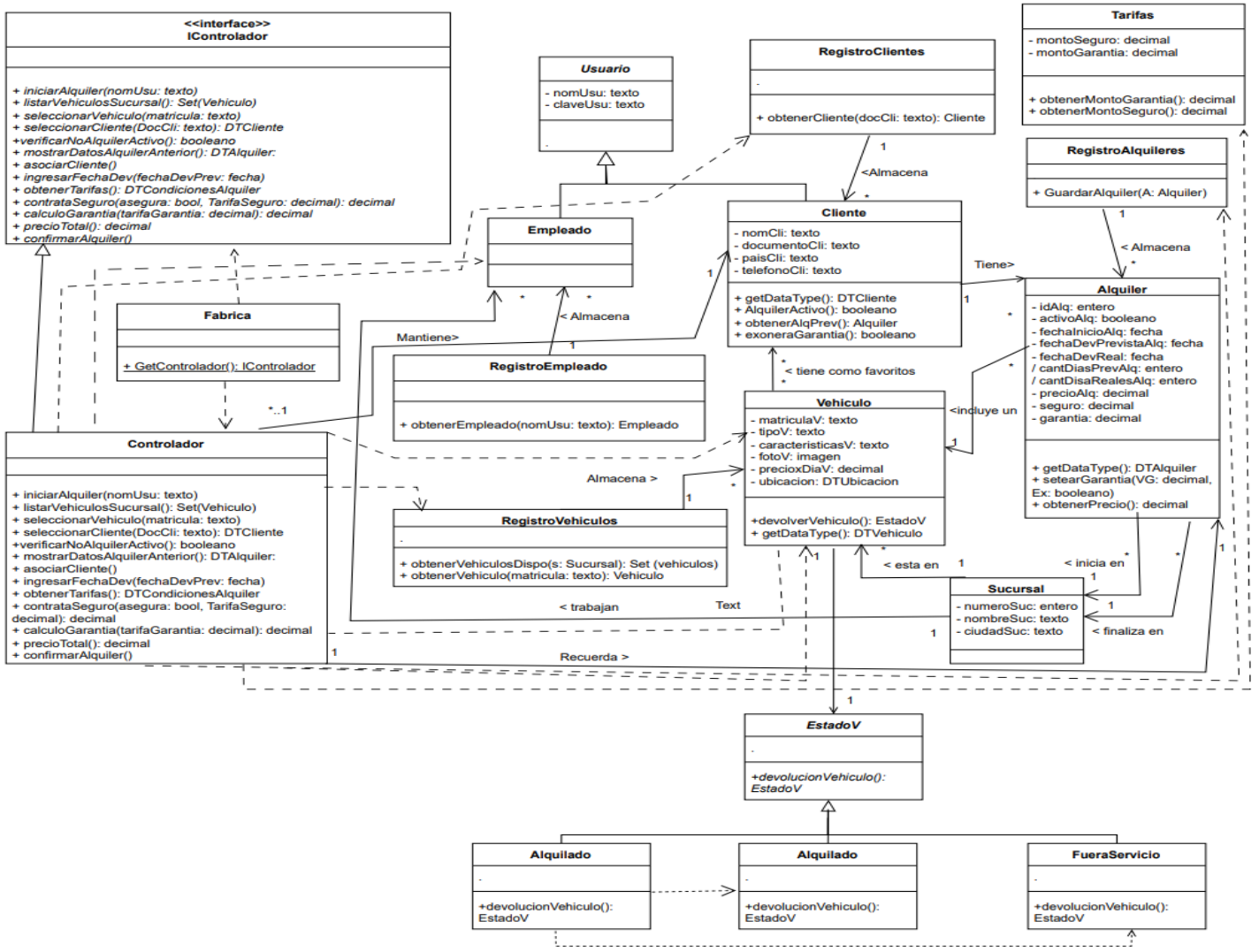
IV. DISEÑO DE ESTRUCTURA

El diseño de estructura del Sistema BIOS Rent tiene como objetivo definir los elementos estructurales del sistema y las relaciones permanentes que existen entre ellos. Se define de esta manera la estructura que da soporte a las interacciones entre los objetos -las cuales fueron definidas previamente en los diagramas de comunicación-.
A través del Diagrama de Estructura Estática (Diagrama de Clases en UML) se modela la organización del sistema en sus distintos niveles de abstracción, representando las clases principales, sus atributos, operaciones, relaciones de asociación, herencia y composición.

4.1 Diagrama de Clases de Diseño

Se presenta un diagrama de clases parcial centrado en el caso de uso “Registrar Alquiler”.
La creación de este Diagrama de clases de diseño toma como base los diagramas de Comunicación desarrollados en el punto 3.1 y el Modelo Conceptual del Sistema.

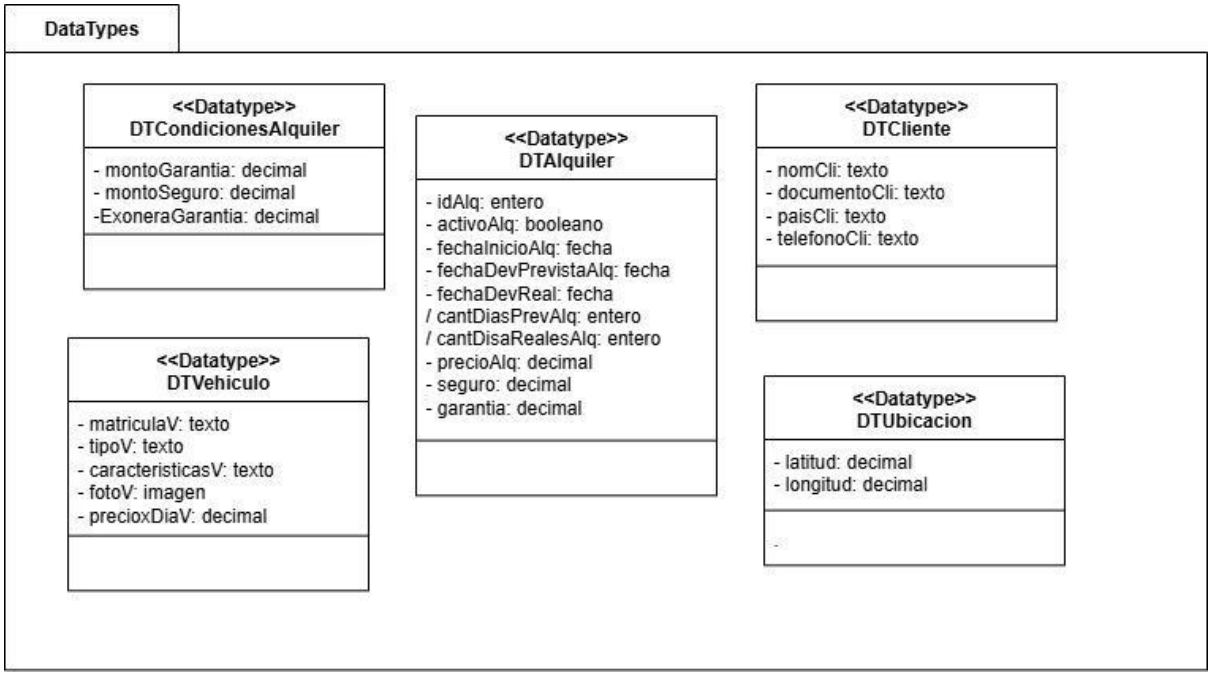
Lógica (Componente Gestión Alquileres y devoluciones)



Nicolás Dagys
Alexander Lemos
Camilo Pereyra

04/06/2025

DataTypes del sistema



V. DISCUSIONES E INVESTIGACIONES

El siguiente capítulo consta de tres apartados en los que se presentan tres temas relevantes que ameritaron discusiones e investigaciones en la elaboración del sistema.
En una primera parte se presenta el Plan de Testing a aplicar en busca de garantizar un producto de calidad.
La segunda parte está relacionada a la relación de los componentes de nuestro sistema con los servicios provistos por la Empresa de Geolocalización externa.
La tercera se centra en el análisis de posibles aplicaciones de patrones para mejorar el vínculo con el Sistema de Geolocalización externo y mejorar las prestaciones.

5.1. Plan de Testing

En este capítulo se define la planificación de la garantía de calidad a través del testing. El objetivo es ofrecer una visión completa y justificada de las elecciones adoptadas para asegurar que el sistema cumpla con los requerimientos funcionales y no funcionales definidos en las etapas iniciales.

La calidad del software es una prioridad para el sistema BIOS Rent, sobre todo cuando nos referimos a operaciones de mayor complejidad que involucran gestión de alquileres, gestión de clientes, control de vehículos, etc. Contar con un Plan de Testing robusto es fundamental para garantizar el correcto funcionamiento y llevar a la mínima expresión posibles defectos en la producción.

Propósito del plan de testing:

El presente plan define estrategias, tipos de pruebas, roles y responsabilidades necesarias para verificar el sistema BIOS Rent en sus diferentes etapas de desarrollo. El principal objetivo es detectar y corregir defectos de forma temprana, elementos que contribuyen a una myor calidad del producto final.

Tipos de testing a aplicar:

1. **Pruebas unitarias:**
   Se enfocan en validar de forma aislada las unidades más pequeñas de código. Este tipo de pruebas tienen varias ventajas: permiten la detección temprana de posibles errores, reducen el costo de corrección, fomentan una mejora en la calidad del diseño mediante la creación de componentes modulares. Tiene como desventaja, la imposibilidad de permitir identificar problemas de interacción entre componentes.
   Aplicabilidad en BIOS Rent: Estas pruebas serán esenciales para testear componentes de la capa de lógica de negocio, como GestionAlquileresyDevoluciones, GestionVehiculos, GestionUsuarios, etc. Nos

permitirán validar cada regla de negocio y que las operaciones de persistencia funcionen correctamente de forma independiente.

2. **Pruebas de Integración:**
   Estas pruebas permiten la validación de la interacción y comunicación entre los diferentes módulos o subsistemas luego de que hayan pasado las pruebas unitarias.
   Tienen la ventaja de permitirnos identificar defectos en el flujo de los datos entre componentes, asegurando que los módulos no solo funcionen bien de manera individual, sino también como conjunto.
   Presentan como desventaja la necesidad de un entorno de prueba más complejo.
   Aplicabilidad en BIOS Rent: Seon adecuadas para validar la comunicación entre el Controlador de Alquileres y los diferentes componentes y entre los componentes entre sí. También puede ser utilizada para verificar la correcta integración con el ServicioGeolocalizacion externo.

3. **Pruebas de Aceptación:**
   Se utilizarán pruebas de aceptación dirigidas a al testeo del sistema por los usuarios finales (empleados o clientes). Tienen la finalidad de garantizar que el sistema cumpla con las necesidades y expectativas de cada tipo de usuario.
   Presentan la ventaja de asegurarnos que el producto final cumpla cabalmente con los requerimientos y necesidades de los usuarios.
   Tienen como desventaja la necesidad de disponibilidad y compromiso por parte de los usuarios.
   Aplicabilidad en BIOS Rent: Serán ejecutadas por el personal de las sucursales y usuarios externos a la empresa, quienes validarán el sistema ejecutando escenarios de prueba basados en los casos de uso. Esto permitirá asegurar que el sistema responde adecuadamente a situaciones reales.

Como forma de trabajo adoptaremos un enfoque de testing continuo e iterativo, integrando pruebas en cada fase del ciclo de desarrollo. Las pruebas unitarias y de integración serán en lo posible automatizadas. Las pruebas de aceptación serán planificadas y ejecutadas de manera formal, involucrando a los usuarios.

5.2. Definición de Componentes para comunicación del Sistema con el Servicio de Geolocalización Externo

5.2.1. Descripción del Problema

La funcionalidad de Localizar Vehículo es esencial para BIOS Rent ya que nos permite obtener la ubicación en tiempo real de todos los vehículos que se encuentran

alquilados. Esta funcionalidad depende de la integración con un servicio de geolocalización externo, al cual se accede a través de una API REST proporcionada por un tercero.

Nos enfrentamos al desafío de incorporar esta interacción externa de forma efectiva y con bajos niveles de acoplamiento, considerando modificaciones internas en el sistema de la proveedora o posibilidades de cambio de empresa en un futuro.

5.2.2. Alternativas

1. Crear un componente especializado llamado ServicioGeolocalizacion La idea es hacer un modelo de software nuevo, al que llamaremos ServicioGeolocalizacion. Su tarea será comunicarse con la API externa de geolocalización. Sería como un “adaptador” que esconde los detalles de la API del resto de nuestro sistema.
2. Ponerlo directamente en GestionVehiculos, ya que este es el encargado de todo lo relacionado con los vehículos también podría tener la responsabilidad de hablar con el servicio de geolocalización externo. Podríamos utilizar una interfaz interna (ILocalizarVehiculo) dentro de GestionVehiculos para organizar esta comunicación.

5.2.3. Ventajas y desventajas

Opción	1. Componente dedicado (ServicioGeolocalizacion)	2. Integrar directamente en (GestionVehiculos)
Separación de responsabilidades	Ventaja: El nuevo componente solo se encarga de hablar con el servicio de geolocalización externa. GestionVehiculos solo se centra en los vehículos.	Desventaja: A GestionVehiculos se le sumaría la responsabilidad de manejar la comunicación externa.
Protección ante futuros cambios	Ventaja: Si la empresa de geolocalización cambia su API Rest, solo tendríamos que modificar este nuevo componente, no afectando al resto del sistema.	Desventaja: Si la API externa cambia, tendríamos que modificar GestionVehiculos.
Reutilizar la función	Ventaja: Si otra parte del sistema necesita saber la ubicación de un vehículo, podrá utilizar este servicio.	Desventaja: La función de geolocalización queda directamente relacionada a GestionVehiculos,

		dificultando a que otros módulos la utilicen.
Manejo de errores	Si el sistema de geolocalización externo falla, el componente puede manejar el error de forma aislada, no afectando el resto del sistema.	El manejo de los errores del sistema de geolocalización externo debería estar en GestionVehiculos, lo que vuelve este componente aún más complejo.
Testing	Se vuelve más sencillo el testing al tener los componentes separados.	Para probar GestionVehiculos tendríamos que simular la conexión externa.

5.2.4. Elección de Alternativa: Crear un Componente Dedicado

Luego de evaluadas las opciones, optamos por la creación de un nuevo componente específico, el ServicioGeolocalizacion, encargado de manejar la conexión con el servicio de geolocalización externo.

Tomamos esta decisión porque buscamos que cada componente del sistema tenga una responsabilidad específica, escondiendo los detalles de los servicios externos y logrando que el sistema sea más flexible ante eventuales cambios en un futuro.

Por otra parte, con esto logramos que GestionVehiculos pueda dedicarse únicamente a su función de administrar los vehículos.

5.2.5. Consecuencias

Positivas:

- Arquitectura más ordenada, cada módulo tiene su función específica. GestionVehiculos se encarga de los vehículos y ServicioGeolocalizacion de la localización externa.
- Si algún otro componente necesita saber la localización de un vehículo, pueden usar ServicioGeolocalizacion directamente.
- Si el servicio externo tiene algún problema, esto no va a afectar a GestionVehiculos.
- Facilita el testing tanto de ServicioGeolocalizacion como de ServicioGeolocalizacion, ya que permite hacerlo de forma independiente.

Negativas:

- Se suma un nuevo componente al sistema.

- Tendríamos un paso más para acceder a los datos, pero pensamos que los beneficios aportados compensan la complejidad.

5.3. Utilización de patrones para la comunicación del Sistema con el Servicio de Geolocalización externo

5.2.1. Descripción del Problema

La comunicación con el servicio externo de geolocalización se realiza a través de la API REST cuya funcionalidad no puede ser controlada desde nuestro sistema. Esta podría sufrir modificaciones o incluso ser reemplazada, por lo que debemos atender estas eventualidades.
En este contexto el análisis de la mejor solución para la comunicación del servicio externo con la lógica interna del sistema es una condicionante importante.
Con estos objetivos, se analizó la posibilidad de aplicar distintos patrones de diseño: Proxy, Adapter o una combinación de ambos.

5.2.2. Alternativas

1. **Patrón Proxy:** Una opción es la implementación del patrón Proxy mediante el cual se crea un intermediario entre nuestro sistema y la API REST del servicio externo. El Proxy actuaría como un representante del servicio y podría incluir funcionalidades adicionales como control de errores, reintentos automáticos, mensajes automatizados ante desconexión de la API externa.

2. **Patrón Adapter:** La segunda opción fue la utilización del patrón adapter mediante el cual se incorpora un adaptador entre nuestro sistema y el servicio externo. Este adaptador adecua la comunicación con el servicio en función de las necesidades de nuestro sistema. El Patrón Adapter facilita acciones ante cambios o reemplazo de la empresa externa que provee los servicios.

3. **Combinación de ambos patrones**: Una tercera posibilidad sería combinar ambos patrones. El Adapter dedicado a traducir el Servicio que llega desde la API REST externa a los requerimientos y formato de nuestro sistema, facilitando tareas de reemplazabilidad y el Proxy como capa intermedia para incorporar funcionalidades de control de errores, mensajes automatizados, etc.

5.2.3. Ventajas y desventajas

Patrón Proxy
Ventajas
- Permite añadir funcionalidades adicionales que podrían mejorar las prestaciones ante situaciones eventuales.
- Permite incorporar mayores controles, reintento de conexiones, mensajes predefinidos, etc.

Desventajas
- Su aporte para la adaptación del sistema ante cambios en el servicio externo presenta mayor complejidad.

Patrón Adapter
Ventajas
- Permite un correcto desacople entre componentes internos y externos.
- Traduce formatos de los datos a los requerimientos del sistema.
- Facilita adecuaciones ante el mantenimiento o cambio del proveedor.

Desventajas
- No incorpora funcionalidades ante situaciones eventuales como errores de comunicación, API REST externa fuera de servicio, etc.

Proxy + Adapter
Ventajas
- Combina las ventajas de ambos patrones. Desacople, facilidad para adaptación de la lógica del sistema y funcionalidades extra mediante el intermediario.

Desventajas
- Aumento en la complejidad general del diseño.
- Incremento en la cantidad de componentes y recursos utilizados.

5.2.4. Elección de Alternativa - Patrón Adapter

El Patrón Proxy se adecua a necesidades de control de acceso o incorporar funcionalidades que no presenta el objeto real. El patrón Adapter es adecuado para resolver problemas de compatibilidad entre sistemas. La combinación de ambos es una muy buena alternativa, pero agrega complejidad general.
Finalmente se opta por utilizar el patrón Adapter para traducir la comunicación con la API REST del proveedor.
Adapter ofrece una solución adecuada ante la situación concreta sin complejizar de manera excesiva la lógica del sistema.
Por medio de esta solución, con cambios de muy poca entidad podemos desacoplar completamente el sistema del proveedor externo, permitiendo uno de los requerimientos más importantes del sistema que es su fácil reemplazo.

Si bien el patrón Proxy o la combinación de Adapter con Proxy aumentan las prestaciones, generan una complejidad excesiva en comparación con el aporte tomando en consideración el objetivo buscado.

5.2.5. Consecuencias

Positivas:

- Permite lograr el desacople entre el sistema y el proveedor externo.
- Permite absorber cambios en el sistema externo y adecuarse a los mismos con cambios mínimos.

Negativas:

- El adaptador no permite incorporar funcionalidades que podrían ser útiles y mejorar nuestro sistema como controles, mensajes automatizados, reintento de conexiones entre otras.