

Universidad ORT

Ingeniería en Sistemas

Diseño de aplicaciones 2 – Obligatorio 2 “Vehicle Tracking”

Gastón Donadio - 192203
Dan Blanco - 194544

Grupo:
Docentes: Ignacio Valle, Ramiro Visca
Octubre 2017

Indice

Introducción	4
Supuestos efectuados	4
Tecnologías utilizadas	4
Componentes físicos del sistema	5
Generalidades del sistema	6
Metricas	8
Base de datos	9
Entidades	9
Objetos de acceso a datos	11
Componentes	12
Paquetes	12
Clases	13
Lógica del negocio	20
Estados posibles por los que pueden pasar los vehículos	20
Servicios	21
Componentes	21
Paquetes	22
Clases	22
Interacciones	25
Excepciones	33
Flujo	34
Contenedor	35
Handler	35
Controladores	35
Endpoints	36
Componentes	38
Paquetes	38
Clases	39
Aplicación web	40
Funcionalidad por rol:	40
Maquina de estados	41
Diagrama de actividades	42
Arquitectura	43
Manejo de errores	46
Aplicación windows	46

Funcionalidades	47
Manejo de errores	47
Patrones de diseño	47
Logs	47
Patrones de diseño	48
Paquetes	48
Interacciones	49
Importador de archivos	49
Paquetes	50
Interacciones	50
Tests	52
TDD	52
Cobertura	54
Clases	55
Bugs conocidos y futuras mejoras	56
Clean Code	57
Capítulo 1: Código Limpio	57
Capítulo 2: Nombres con sentido	57
Capítulo 3: Funciones	58
Capítulo 4: Comentarios	59
Capítulo 5: Formato	59
Capítulo 6: Objetos y estructuras de datos	59
Capítulo 7: Procesar errores	60
Capítulo 8: Límites	61
Capítulo 9: Pruebas unitarias	61
Capítulo 10: Clases de tamaño reducido	61
Capítulo 12: Limpieza	61
Conclusión	61
Instalacion del sistema	62
Instalación de service:	62
Instalación de aplicación web angular	62
Instalación de aplicación de escritorio	62
Datos de prueba	63
Usuarios:	63
Roles:	63
Zonas:	63
Tipo de SubZona:	64
Items de flujo:	64
Vehículos:	64

Lotes:	66
Transportes:	66
Histórico de vehículo:	66
Ubicación:	70
Inspecciones:	71
Daños:	71
Bibliografía	72

Introducción

El nombre del sistema a desarrollar es “Vehicle Tracking”. Permite gestionar desde el arribo de vehículos al puerto, hasta la ubicación de los mismos en subzonas de patios, pudiendo controlar todas las fases intermedias por las que los vehículos pasan, y además controlar si algún vehículo ha sufrido algún daño en el trayecto.

Estos vehículos luego de arribados a un patio, pasan por varias subzonas en un orden específico previamente definido para que puedan estar listos para la venta, donde posteriormente se venden.

El sistema contempla varios tipos de usuarios, los cuales cuentan con distintas responsabilidades, y permite controlar que solo los usuarios con permisos puedan avanzar el flujo, dependiendo en que lugar se encuentre ubicado el vehículo, ya que el sistema cuenta con la trazabilidad del mismo.

Supuestos efectuados

- Las zonas tienen como máximo un nivel de anidación, llamado subzona.
- Los lugares donde se efectúan los incidentes son fijos y provistos por el sistema.
- Las operaciones de mantenimiento de cada una de las entidades no son necesarias en la mayoría de los casos (actualización y borrado) y cuando son de interés, el único que tiene acceso a ellas es un administrador, al igual que para crear vehículos. Las entidades que cuentan con alta, baja y modificación, son vehículos y zonas por requerimiento del sistema.
- Los vehículos son ingresados solo por un administrador desde la aplicación de windows.

Tecnologías utilizadas

- **C#:** Código del sistema.
- **Moq:** Mock de pruebas unitarias.
- **Unity:** Inyección de dependencias.
- **Entity Framework:** Mapeo de objetos con entidades, conexión y creación de la base de datos.
- **SQL Server:** Base de datos.
- **WebApi:** Servicios web.
- **Angular:** Lógica del cliente web.
- **Typescript:** Lógica del cliente web.
- **CSS:** Estilos del cliente web.
- **Ng-Bootstrap:** Lógica de modales.
- **Bootstrap:** Estilos y componentes html de la página web.

- **JQuery:** Dependencia de librería ng-bootstrap.
- **CoreJS:** Dependencia de librería ng-bootstrap.
- **RxJS:** Dependencia de librería ng-bootstrap.
- **ZoneJS:** Dependencia de librería ng-bootstrap.
- **Popper:** Dependencia de librería ng-bootstrap.
- **WindowsForm:** Ventanas de la aplicación de windows.
- **AspNet.Cors:** Permitir la aplicación web comunicarse con los servicios web desde distintos servidores a través del explorador.
- **Agency Template:** Template con algunos estilos y algunos htmls prediseñados.

Justificación:

Las tecnologías utilizadas forman parte del requerimiento del obligatorio, las tecnologías que forman parte de la aplicación web serán y no eran obligatorias se detallan a continuación.

Ng-Bootstrap: Usamos esta librería para facilitar el uso de los modales y la prolijidad del front end, ya que es una tecnología reconocida y muy bien testada, y con una fuerte estabilidad y retrocompatibilidad en angular 2, la misma depende de JQuery, CoreJS, ZoneJS, RxJS, y Popper que también son estables y por eso las incluimos en nuestro sistema.

Link de referencia: <https://ng-bootstrap.github.io/#/home>

Agency Template: Es un template que facilita el diseño de la página web, con templates ya probados.

Link de referencia: <https://startbootstrap.com/>

Bootstrap: Lo utilizamos para los estilos y algunos componentes de html con el fin de mejorar la apariencia de la página y la usabilidad de la misma, ya que es uno de los frameworks más usados y probados para el diseño de páginas web.

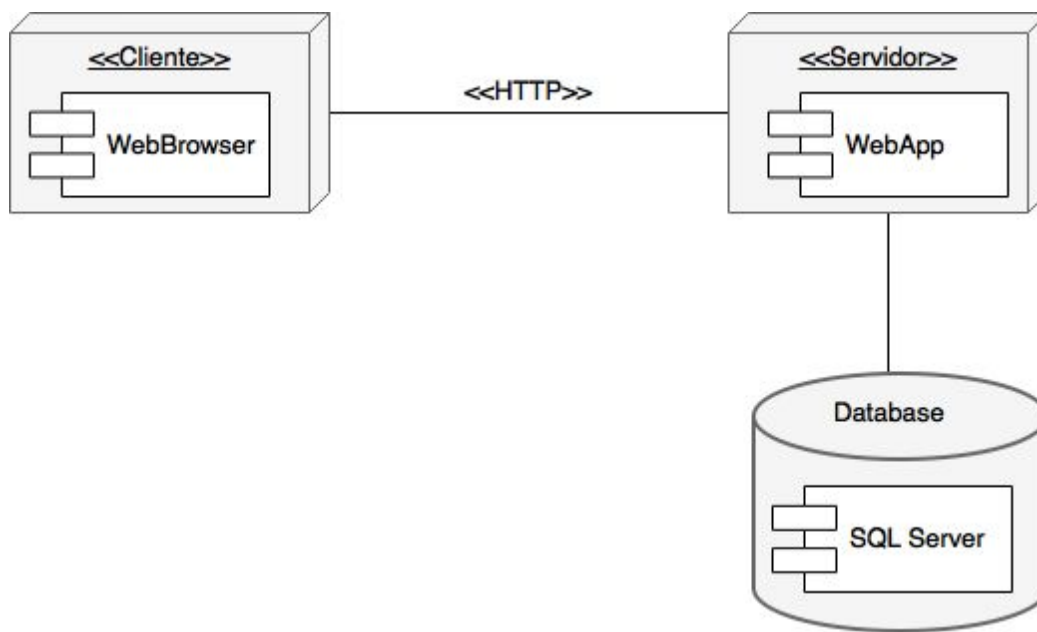
Componentes físicos del sistema

La arquitectura general consiste en múltiples clientes, que a través de una aplicación web en un servidor, se comunica con servicios web en otro del servidor, y éste almacena los datos en una base de datos.

La aplicación web es una página web, que ofrece una interfaz de usuario para que el usuario utilice el sistema.

Los servicios web son una API REST que ofrece un conjunto de operaciones capaces de resolver todo lo necesario, es donde se implementa la lógica de negocio.

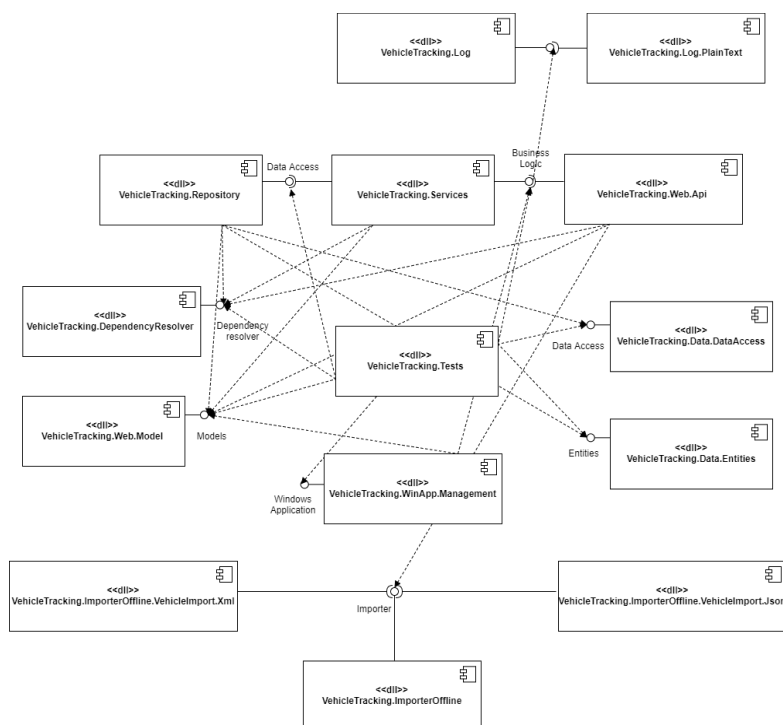
Se utiliza una base de datos relacional (SQL Server) para almacenar los datos de la aplicación.



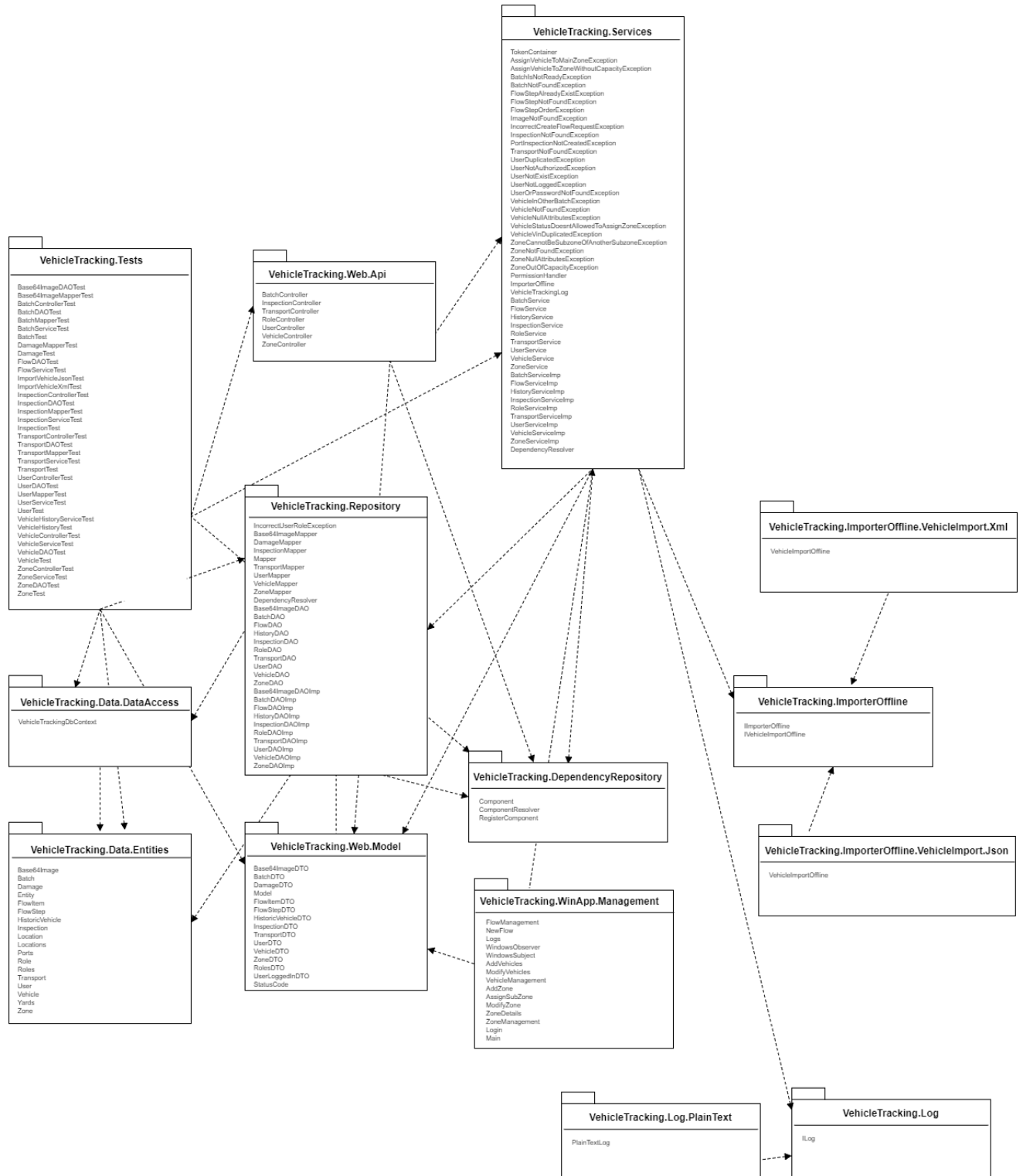
Generalidades del sistema

Exponemos los diagramas completos tanto de paquetes como de componentes, para mostrar cómo se comporta el sistema en su totalidad.

Componentes del sistema:



Paquetes del sistema:



Metricas

Exponemos las métricas ofrecidas por la herramienta de visual studio.

Resultados de métricas del código						
Filtro: Ninguno	Min.:	Máx.:				
Jerarquía	Índice de mantenimiento	Complejidad ciclomática	Profundidad de he...	Acoplamiento de clases	Líneas de código	
VehicleTracking.Data.DataAccess (Debug)	92	27	2	15	27	
VehicleTracking.Data.Entities (Debug)	92	156	2	15	191	
VehicleTracking.DependencyResolver (Debug)	86	20	1	30	31	
VehicleTracking.ImporterOffline (Debug)	100	5	0	3	0	
VehicleTracking.ImporterOffline.VehicleImport.Json (Debug)	86	26	1	15	36	
VehicleTracking.ImporterOffline.VehicleImport.Xml (Debug)	87	30	1	21	43	
VehicleTracking.Log (Debug)	94	14	2	6	16	
VehicleTracking.Log.PlainText (Debug)	59	15	1	27	42	
VehicleTracking.Repository (Debug)	75	299	2	134	1,227	
VehicleTracking.Services (Debug)	90	380	2	101	604	
VehicleTracking.Tests (Debug)	61	327	1	217	3,407	
VehicleTracking.Web.Api (Debug)	68	243	2	90	772	
VehicleTracking.Web.Models (Debug)	91	138	2	15	158	
VehicleTracking.WinApp.Management (Debug)	58	231	7	113	2,049	

A continuación se mostraran varias métricas de los paquetes y se hará una conclusión de los resultados encontrados.

Nombre el paquete	N	R	Ce	Ca	Na	H	I	A
VehicleTracking.Tests	38	0	127	0	0	0.02	1	0
VehicleTracking.Web.Api	7	0	34	0	0	0.14	1	0
VehicleTracking.Services	50	22	54	32	9	0.46	0.62	0.18
VehicleTracking.Repository	30	22	12	26	10	0.76	0.31	0.33
VehicleTracking.Data.DataAccess	1	0	13	10	0	1	0.56	0
VehicleTracking.Data.Entities	18	25	0	73	0	1.44	0	0
VehicleTracking.Web.Model	15	20	0	90	0	1.33	0	0
VehicleTracking.DependencyResolver	3	3	0	4	2	1.33	0	0.66
VehicleTracking.WinApp.Management	15	16	25	0	1	1.13	1	0.06
VehicleTracking.ImporterOffline	2	0	0	3	2	0.5	0	1
VehicleTracking.ImporterOffline.VehicleImport.Json	1	0	1	0	0	1	1	0
VehicleTracking.ImporterOffline.VehicleImport.Xml	1	0	1	0	0	1	1	0

VehicleTracking.Log.PlainText	1	0	1	0	0	1	1	0
VehicleTracking.Log	1	0	0	3	1	1	0	1

Conclusión

Las métricas de visual estudio nos indica que los resultados son muy buenos.

Sin embargo utilizando las métricas de paquetes, observamos que ninguna llega al mínimo aceptable teórico para tener buena cohesión, sin embargo la mayoría si esta muy cerca de eso, esto puede ser porque algunos paquetes tienen demasiados ejes de cambio, y que además se podrían reusar por separado las clases correspondientes a ellos, por lo que es necesario en un futuro rever esta situación y refactorizar con el fin de mejorar la cohesión de los paquetes.

En cuanto al acoplamiento y la estabilidad, la mayoría de los paquetes son super estables, y tienen buena abstracción lo que es positivo, algunos de ellos sin embargos son muy estables pero no tienen mucha abstracción, esto se debe a que existen paquetes como entities y models, que solo tienen estructuras de datos, por lo que creemos que estan correctos de todas formas.

Los paquetes más inestables dependen de paquetes más estables, por lo que creemos que la mantenibilidad del proyecto en general es bastante buena.

Base de datos

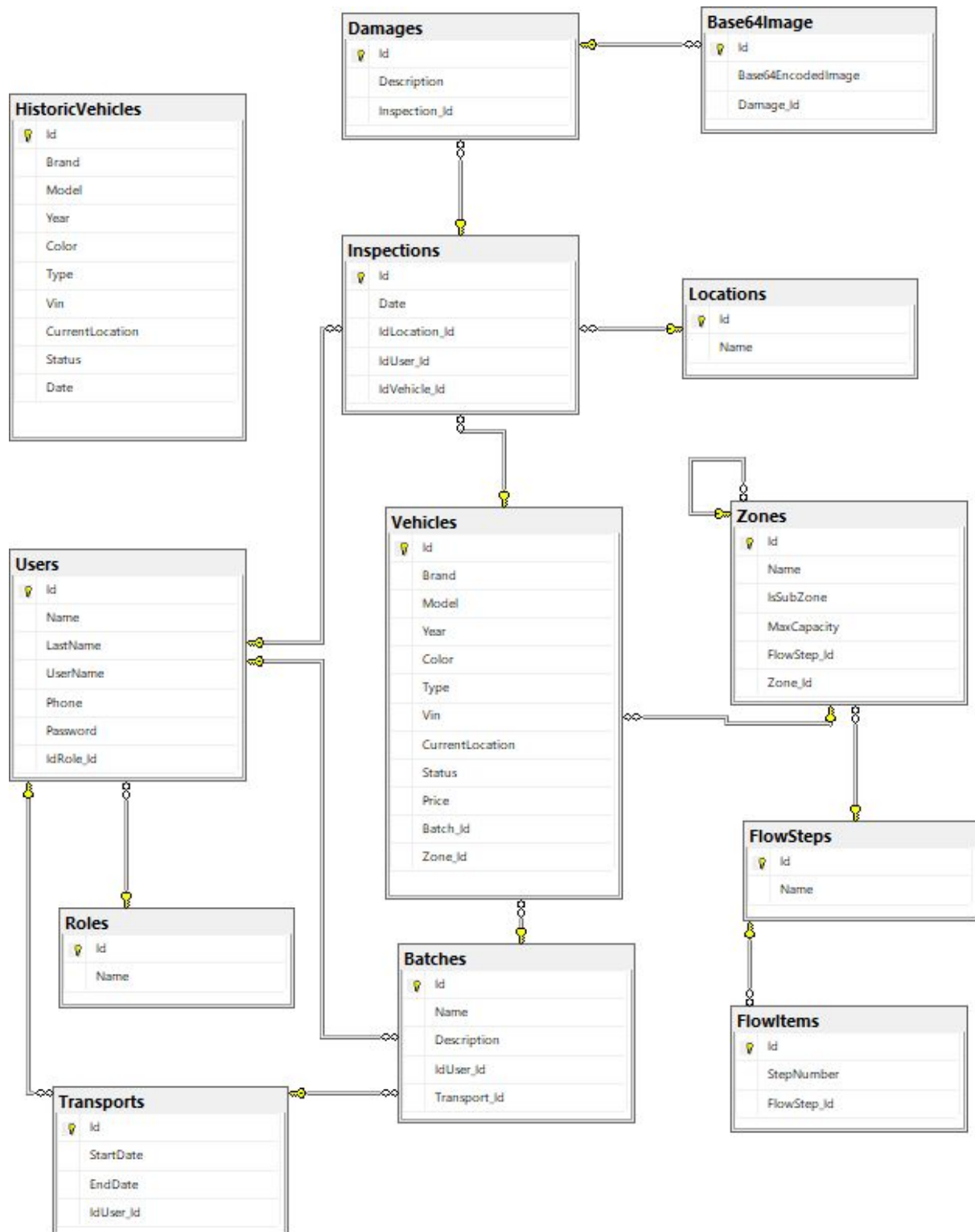
A continuación se detallara el contenido de la base de datos utilizada para el sistema "Vehicle Tracking", cómo se accede a ella, y que operaciones expone a la lógica de negocio para interactuar la misma.

Entidades

El sistema cuenta con las siguientes entidades:

- Zone
- Vehicle
- HistoricVehicle
- Batch
- User
- Role
- Inspection
- Base64Image
- Damage
- Location
- Transport
- FlowItem
- FlowStep

Ellas se relacionan de la siguiente manera:



Justificación:

Decidimos que el histórico de los vehículos era mejor separarlo de la entidad vehículos, ya que entendemos que no deberían haber varios vehículos con un el mismo VIN y distintos estados, y debido a que es relevante llevar un histórico de los vehículos, nos pareció mejor conceptualmente separarlos, además un vehículo no tendría porque tener una fecha que vaya cambiando en el correr del tiempo, simplemente el año en el que se creo es relevante,

y al hacer un histórico de vehiculos a parte, nos permite tener un control de cuando se efectuaron cambios en ese vehículo, agregando una fecha.

Las ubicaciones las separamos en una tabla aparte, y no lo utilizamos como un string, ya que creemos que sería imposible luego llevar un control del lugar en el que se creó una inspección, y debido a que por letra los lugares pueden ser una lista fija, preferimos separarlo de la entidad de inspección.

Los roles tambien los separamos de usuarios, ya que nos permite separarlos de acuerdo a sus distintas responsabilidades, además que es más fácil controlar posibles errores de tipeo, como por ejemplo que el rol de administrador este como admin, administrator, administrador, etc.

Para hacer el flujo lo separamos en FlowItems, y FlowSteps, el FlowItems tiene un numero que es el orden que ocupa en el flujo, y el FlowStep tiene un nombre, y a su vez se asocia a una subzona, ya que las subzonas tienen un FlowStep, según el orden de los FlowItem, el orden de la subzona.

Objetos de acceso a datos

Para el manejo de las entidades del sistema, utilizamos el patrón de diseño DAO (Data access object), el cual consiste básicamente en tener clases cuya responsabilidad sea manejar el acceso a la base de datos, y la gestión de las entidades del sistema, sin mostrar entidades en la capa de lógica, sino que una representación de ellas, con los datos interesantes a devolver, estas representaciones las llamamos DTO (Data transfer object), y serán las que utilizaremos más adelante, desde la capa de controladores hasta los servicios de la lógica de negocio.

El sistema cuenta con los siguientes DAO:

Interfaz provista	Implementación
VehicleDAO	VehicleDAOImp
BatchDAO	BatchDAOImp
UserDAO	UserDAOImp
RoleDAO	RoleDAOImp
HistoryDAO	HistoryDAOImp
InspectionDAO	InspectionDAOImp
ZoneDAO	ZoneDAOImp
TransportDAO	TransportDAOImp
FlowDAO	FlowDAOImp

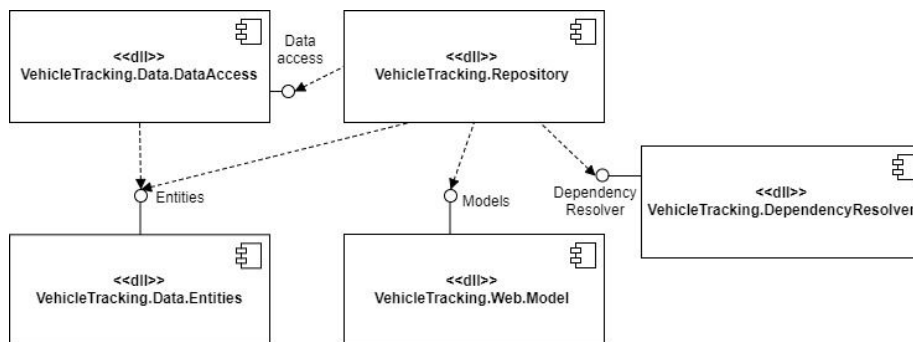
Varios de esos DAO (los que se consideran que tienen entidades más complejas) cuentan con un Mapper para pasar de entidad a modelo, y viceversa, todos implementan una interfaz llamada Mapper, que espera ser definida por un modelo y una entidad, con el fin de saber qué entidad corresponde a que modelo.

Mappers:

- Base64ImageMapper
- BatchMapper
- DamageMapper
- InspectionMapper
- TransportMapper
- UserMapper
- VehicleMapper
- ZoneMapper

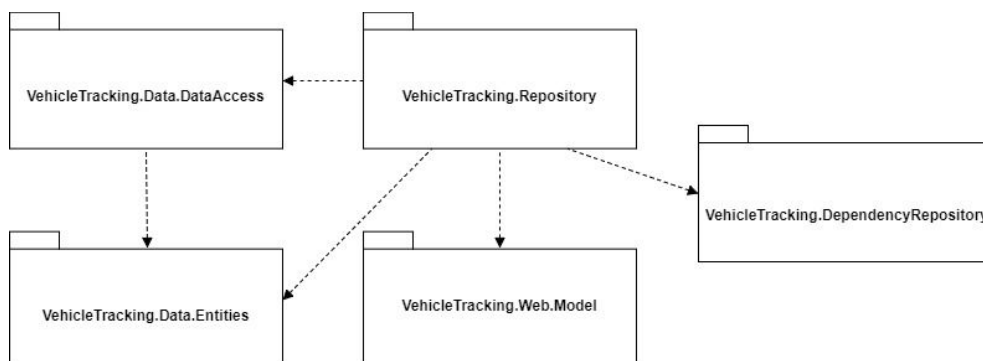
Componentes

Los componentes del sistema que interactúan entre sí en esta parte son:



Paquetes

Y se encuentran en los siguientes paquetes a grandes rasgos, ya que los mismos fueron detallados en las generalidades del sistema:



Clases

A continuación se muestran las distintas clases del sistema a través de diagramas.

Diagrama de modelos

Las siguientes clases son los modelos y dto que viajan desde la capa de controlador, pasando por la capa de la lógica del negocio, y llegando a la capa de repositorio.

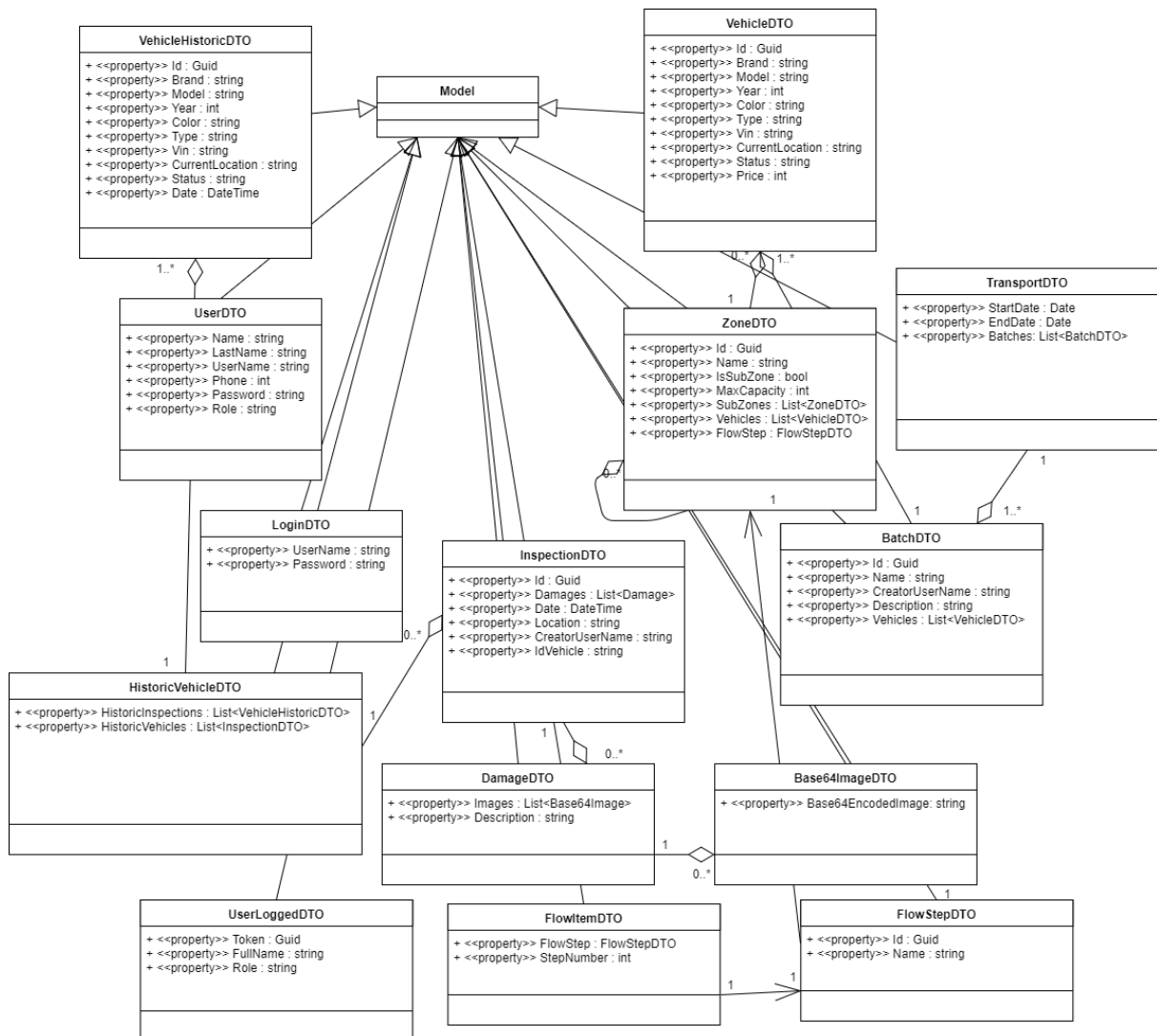


Diagrama de entidades

Luego se exponen las clases de las Entidades que se utilizan solo en la capa de repositorio, que es la que maneja los objetos reales del sistema.

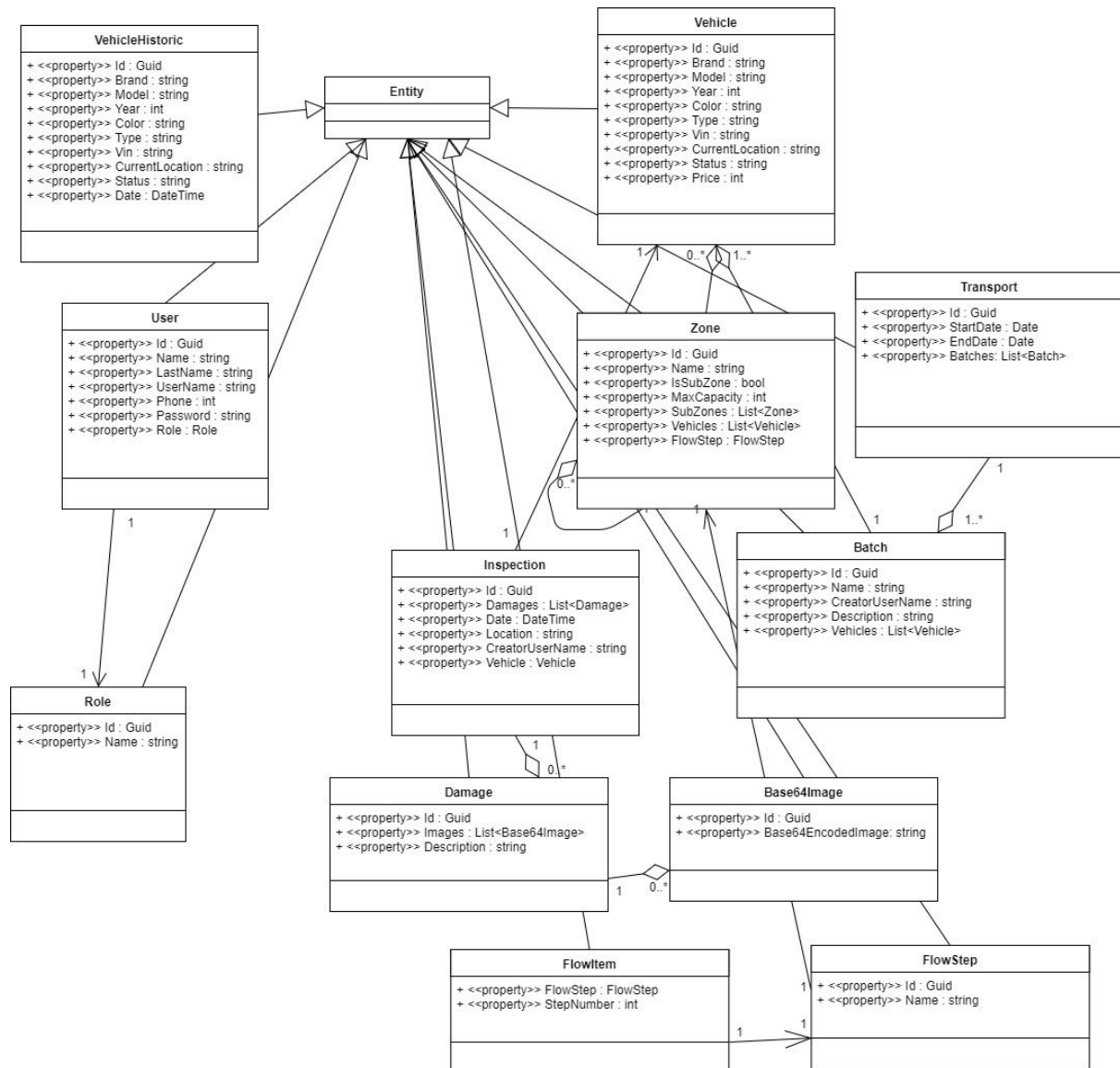


Diagrama de Mapper

Además para pasar de los DTO a Entity (y viceversa) que consideramos más complejos del sistema, se crearon los mapper que cuentan con las siguientes clases:

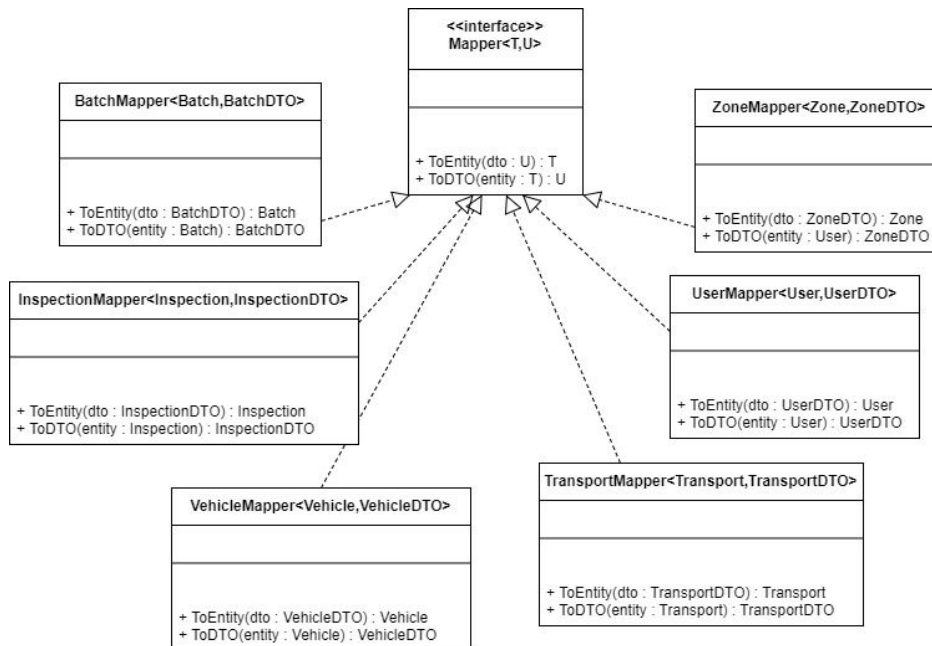
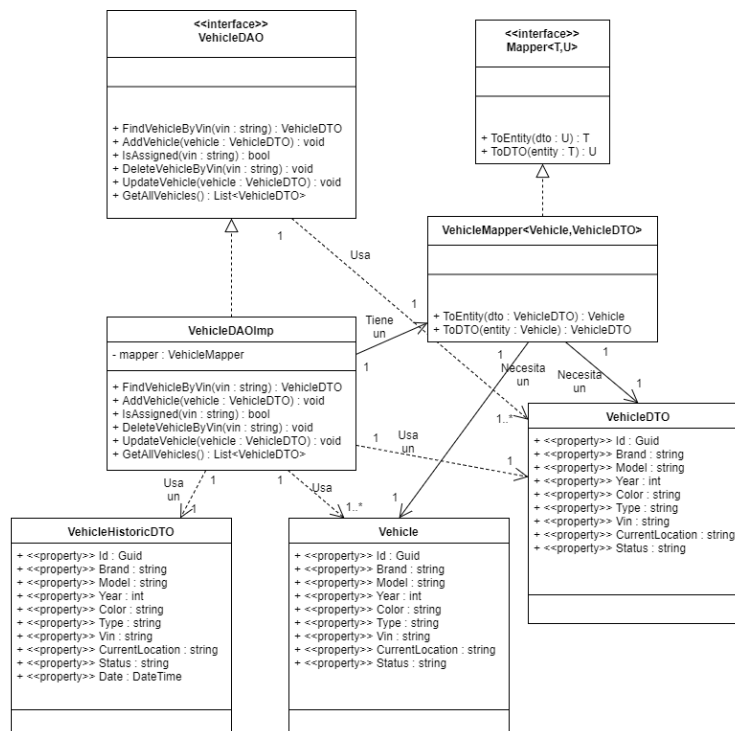


Diagrama de DAO de vehículo e interacciones con otras clases

Para ver cómo interactúan juntas estas clases exponemos el siguiente diagrama de clases con vehículos



El DAO de lotes junto a la entidad, modelo y mapper correspondiente, que se pueden observar abajo:

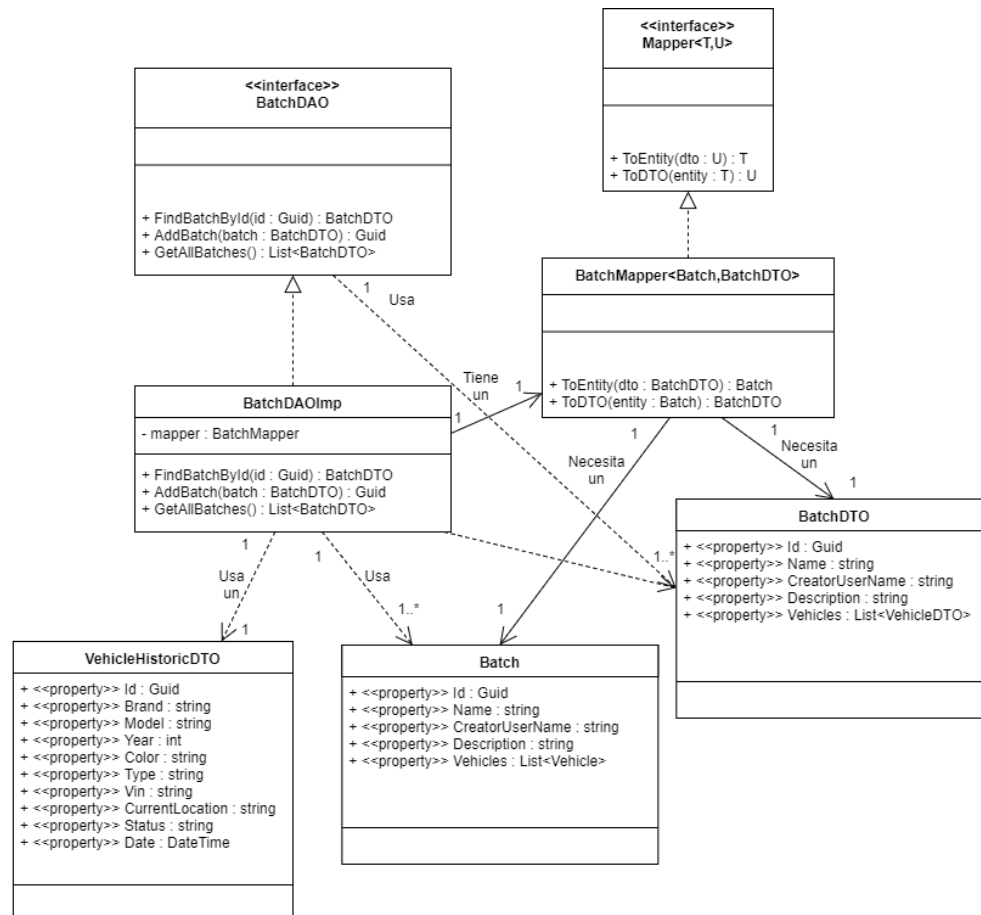


Diagrama de DAO de usuarios e interacciones con otras clases

Luego tenemos el DAO de usuario junto a la entidad, modelo y mapper correspondiente, que se pueden observar abajo:

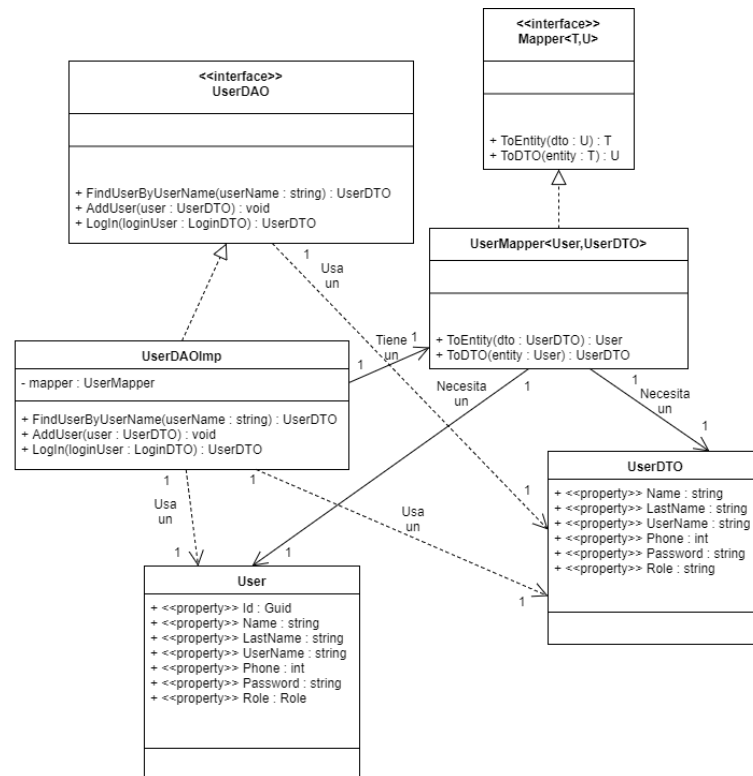


Diagrama de DAO de inspección e interacciones con otras clases

Después está el DAO de inspección junto a la entidad, modelo y mapper correspondiente, que se pueden observar abajo:

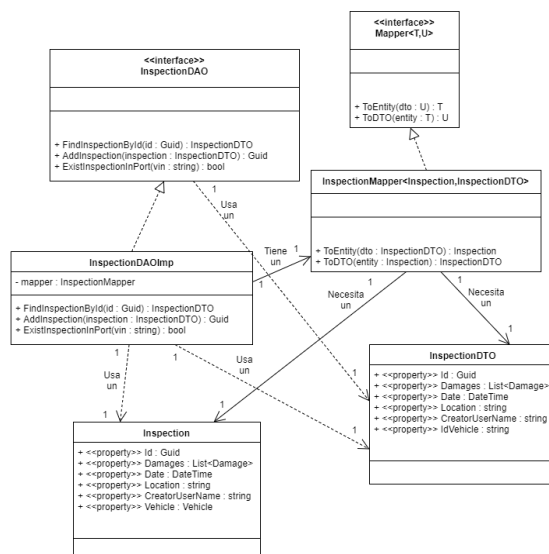


Diagrama de DAO de transporte e interacciones con otras clases

Siguiendo con el DAO de transporte junto a la entidad, modelo y mapper correspondiente se pueden observar abajo:

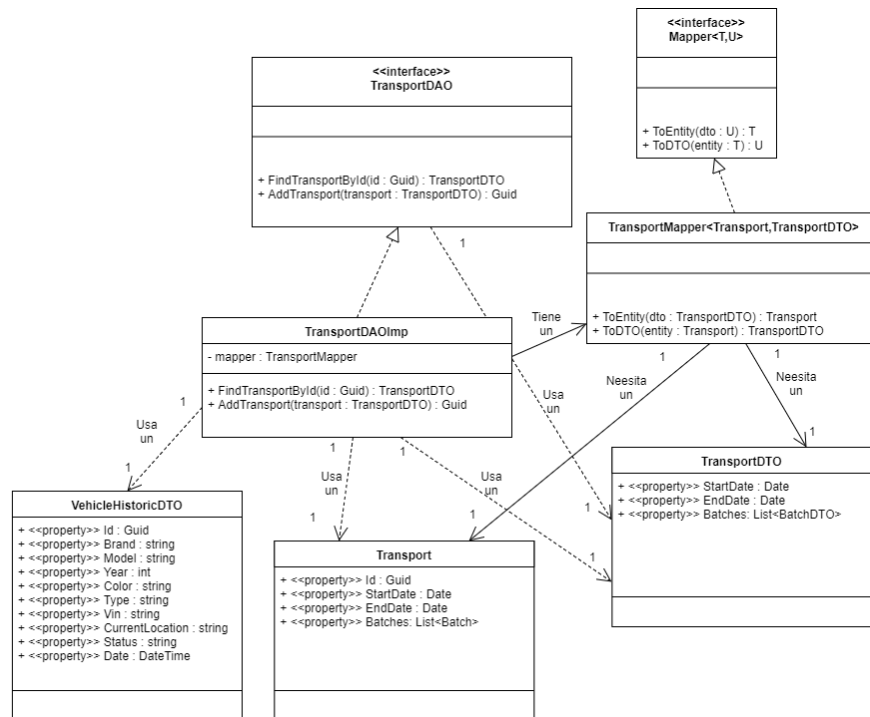


Diagrama de DAO de zonas e interacciones con otras clases

Continuando con el DAO de zonas, junto a la entidad, modelo y mapper, que correspondiente se pueden observar a continuación:

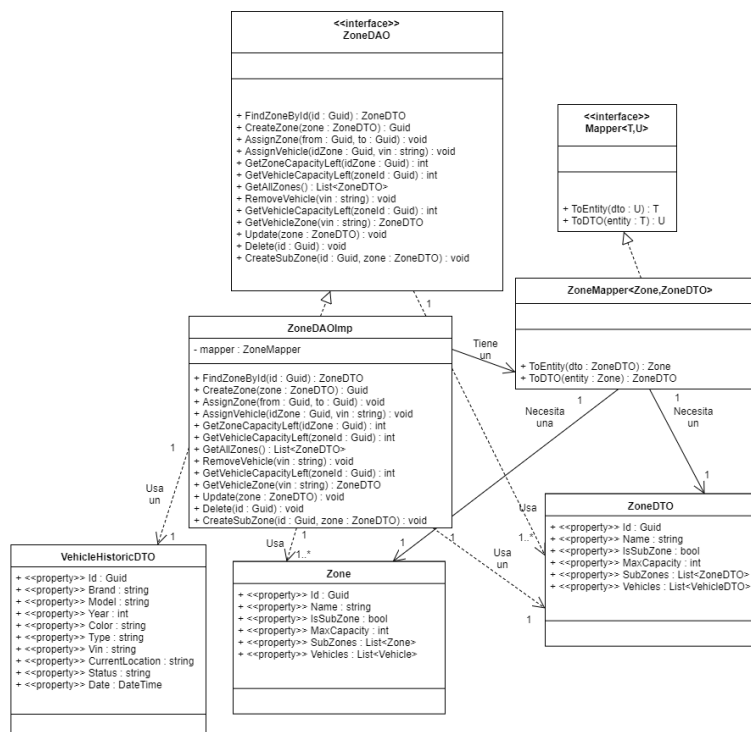


Diagrama de DAO de historial e interacciones con otras clases

Tenemos a su vez el DAO del historico de vehículos, junto a la entidad, y modelo, que correspondiente se pueden observar a continuación:

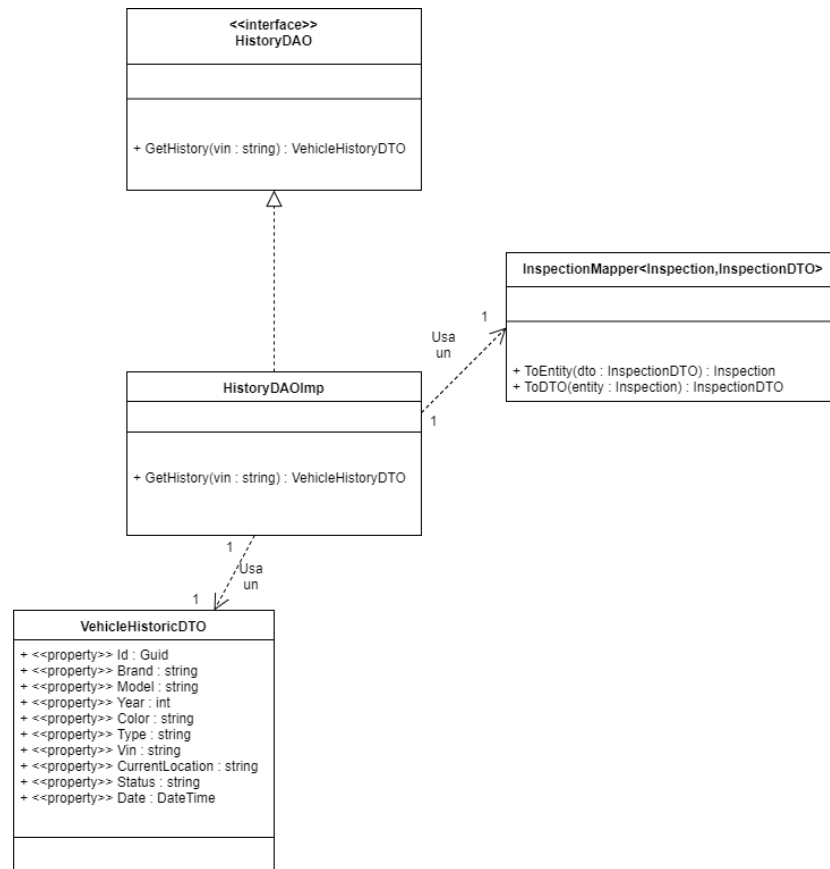
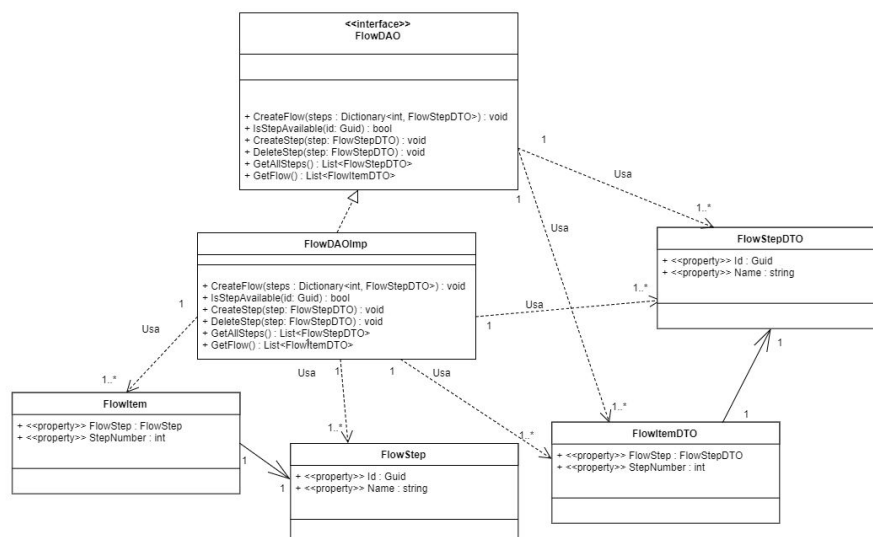


Diagrama de DAO de flujo e interacciones con otras clases

Y por último, el DAO de flujo, junto a la entidad, y modelo, que correspondiente se pueden observar a continuación:



Justificación:

Creamos 2 clases Model y Entity, con la responsabilidad de diferenciar el tipo de objetos de entidades y las representaciones de ellas, además, en la interfaz de Mapper, forzamos a que el objeto que se le pasa como T extienda de Entity y el objeto U extienda de Model, de este modo obligamos a que el que implementa la interfaz lo hace siempre partiendo de una entidad y un modelo, para poder pasar de una a otra.

Además se decidió separar las entidades y los modelos, y darle a los DAOs el manejo de acceso a datos , para que no se trabaje sobre las propias entidades en capas superiores y en lugar se trabajen con representaciones de las mismas.

Mappers: Se utilizaron mappers para desacoplarse de la responsabilidad de saber como pasar de un DTO a un Model y viceversa, ya que agrega complejidad y tamaño de forma innecesaria a los DAOs.

Lógica del negocio

Se detalla a continuación la lógica del sistema, donde se podrá distinguir las distintas interacciones entre los diversos modelos que mostramos antes, y cuales son las reglas de negocio que aplican a cada uno.

Se utilizaron distintas interfaces de servicios, cada implementación responsable de respetar las reglas que debían cumplir los recursos del sistema.

Estados posibles por los que pueden pasar los vehículos

En puerto	Cuando se crea un vehículo, este es su estado inicial.
Inspeccionado en puerto	Vehículo luego de ser inspeccionado por primera vez en el puerto.
Listo para partir en puerto	Vehículo inspeccionado y dentro de un lote esperando para partir en el puerto.
En transito	El vehículo partió del puerto en un lote.
Esperando por inspección patio	El vehículo espera en el patio para ser inspeccionado.
Listo para ser ubicado en zona	El vehículo ya fue inspeccionado por segunda vez en el patio.
Ubicado en zona	El vehículo fue ubicado en alguna subzona.
Listo para la venta	El vehículo pasó por todas las subzonas que establece el flujo definido en ese momento, y quedó pronto para ser vendido.

Vendido	El vehículo se vendió.
----------------	------------------------

Justificación:

Agregamos algunos estados más que no se especifican en la letra con la intención de facilitar el uso del propio sistema, con el fin de saber con certeza la condición actual de un vehículo.

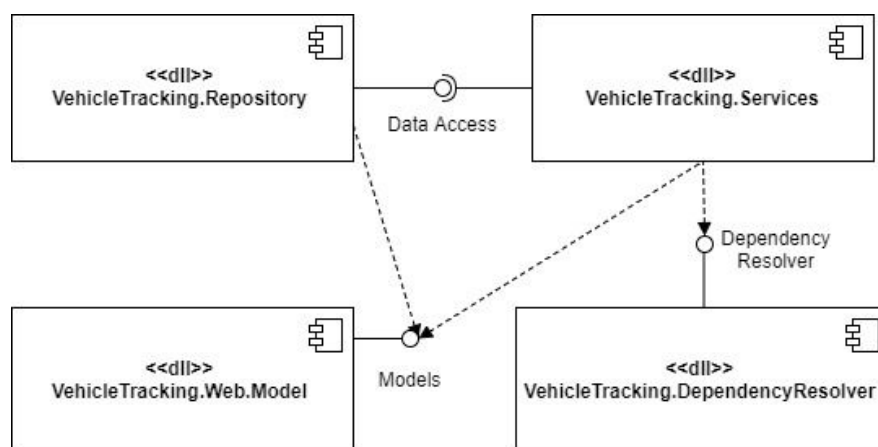
Servicios

Comenzaremos por listar las distintas interfaces y servicios de los cuales se compone nuestra capa de lógica del negocio.

Interfaz provista	Implementación
UserService	UserServiceImp
BatchService	BatchServiceImp
VehicleService	VehicleServiceImp
HistoryService	HistoryServiceImp
InspectionService	InspectionServiceImp
ZoneService	ZoneServiceImp
TransportService	TransportServiceImp
RoleService	RoleServiceImp
FlowService	FlowServiceImp

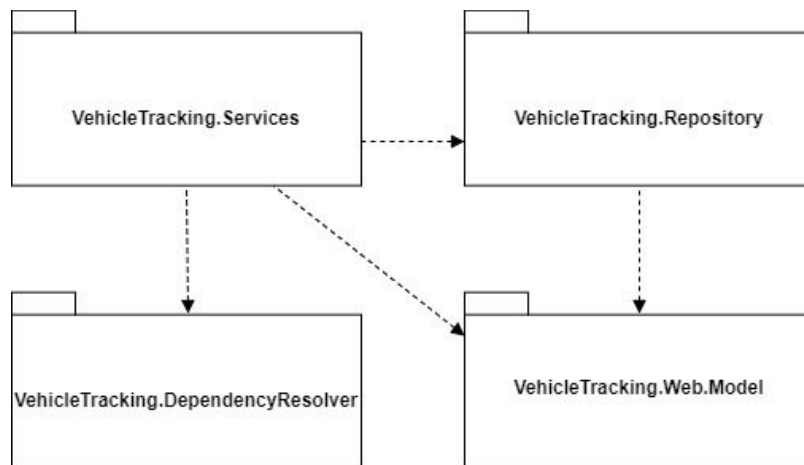
Componentes

Cuenta con los siguientes componentes:



Paquetes

Y contiene además los siguientes paquetes, a grandes rasgos, ya que los mismos fueron detallados en las generalidades del sistema.



Clases

A continuación se presentan los diagramas de clases más significativos de la capa de servicios:

Diagrama de servicio de usuarios

El siguiente diagrama muestra las clases que interactúan en el servicio de usuarios

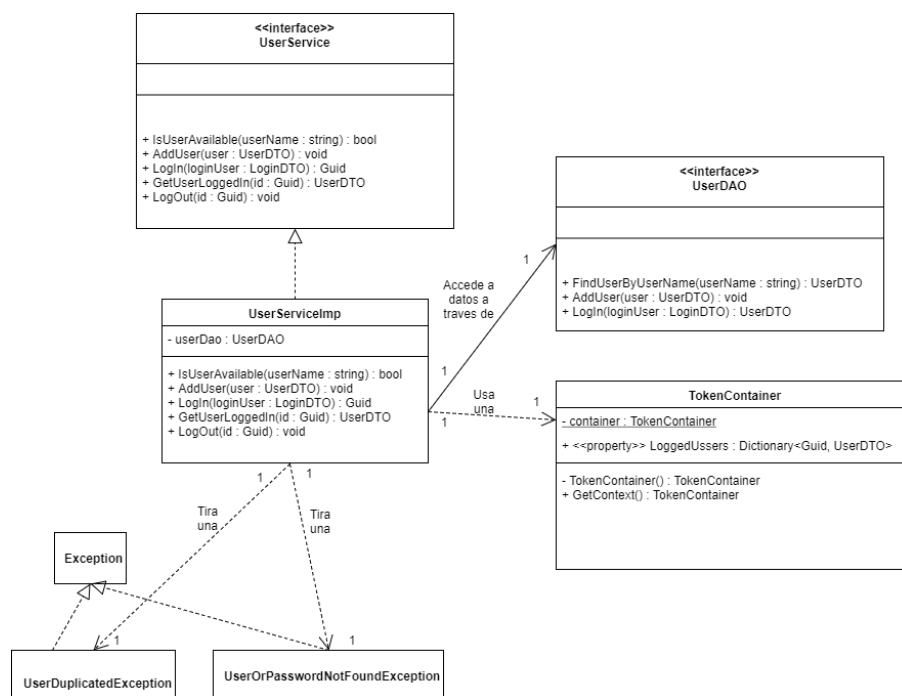


Diagrama de servicio de vehículos

Luego se muestra el siguiente diagrama que tiene las clases que interactúan en el servicio de vehículos

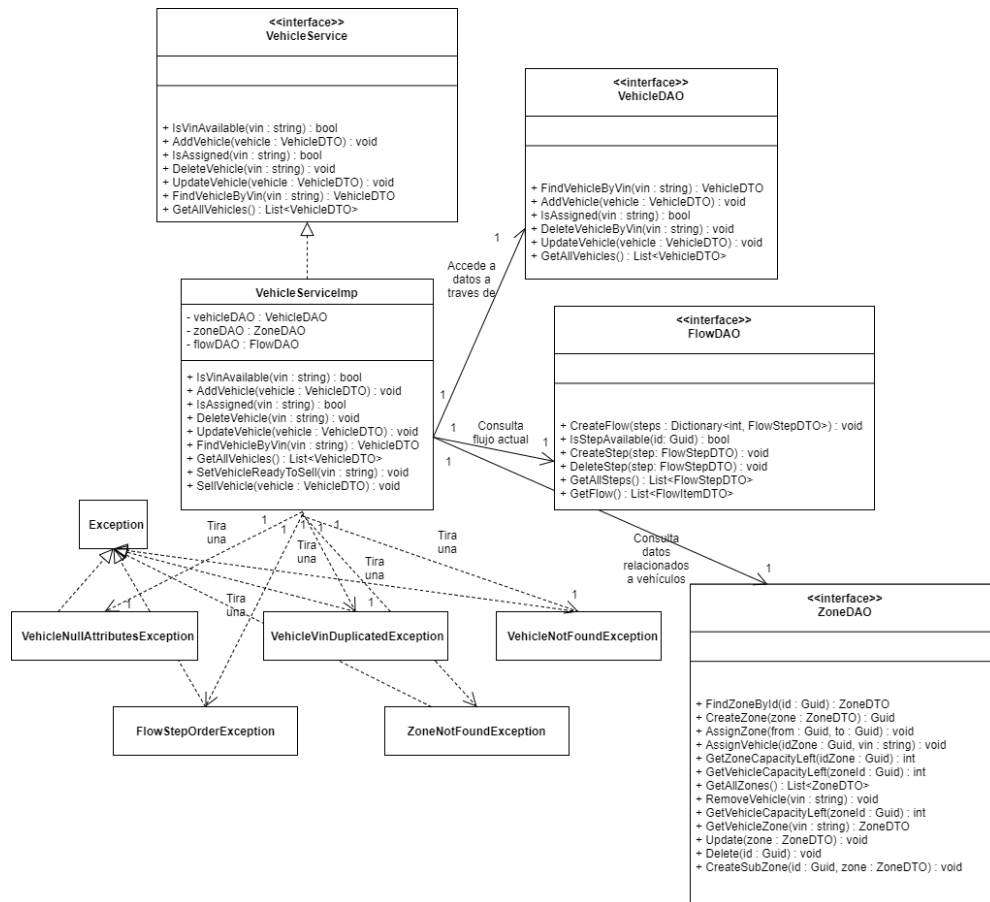


Diagrama de servicio de lotes

Seguimos por el siguiente diagrama que tiene las clases que interactúan en el servicio de lotes, este es uno de los más sencillos y nos interesa mostrarlo para ver su simplicidad

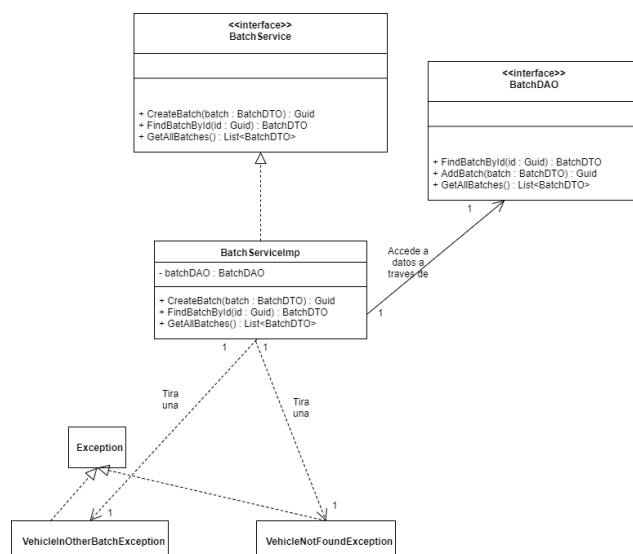


Diagrama de servicio de zonas

Aquí mostramos el diagrama de clases del servicio con más complejidad del sistema, por lo tanto uno de los más interesantes, se pueden observar sus interacciones con las distintas clases.

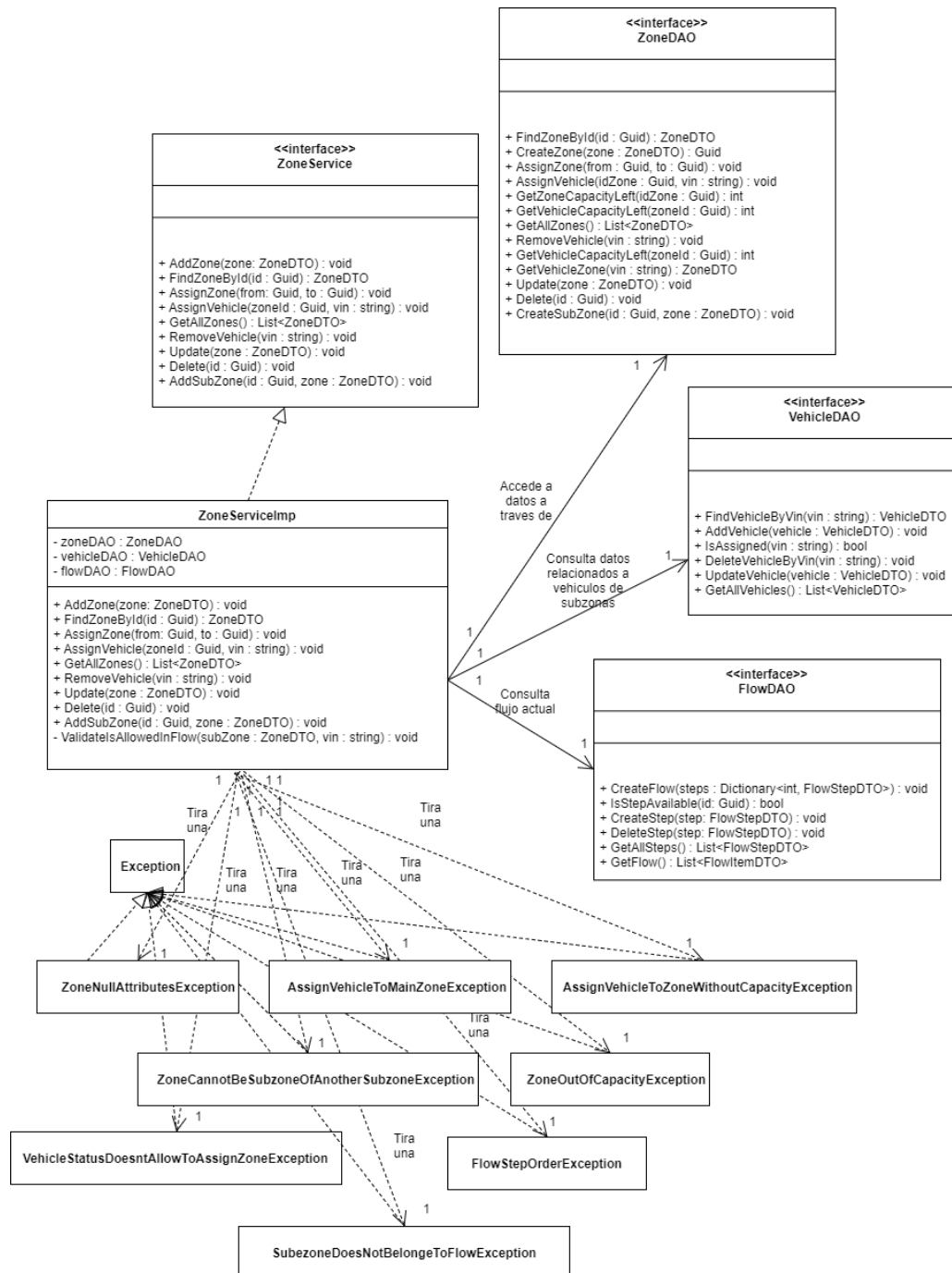
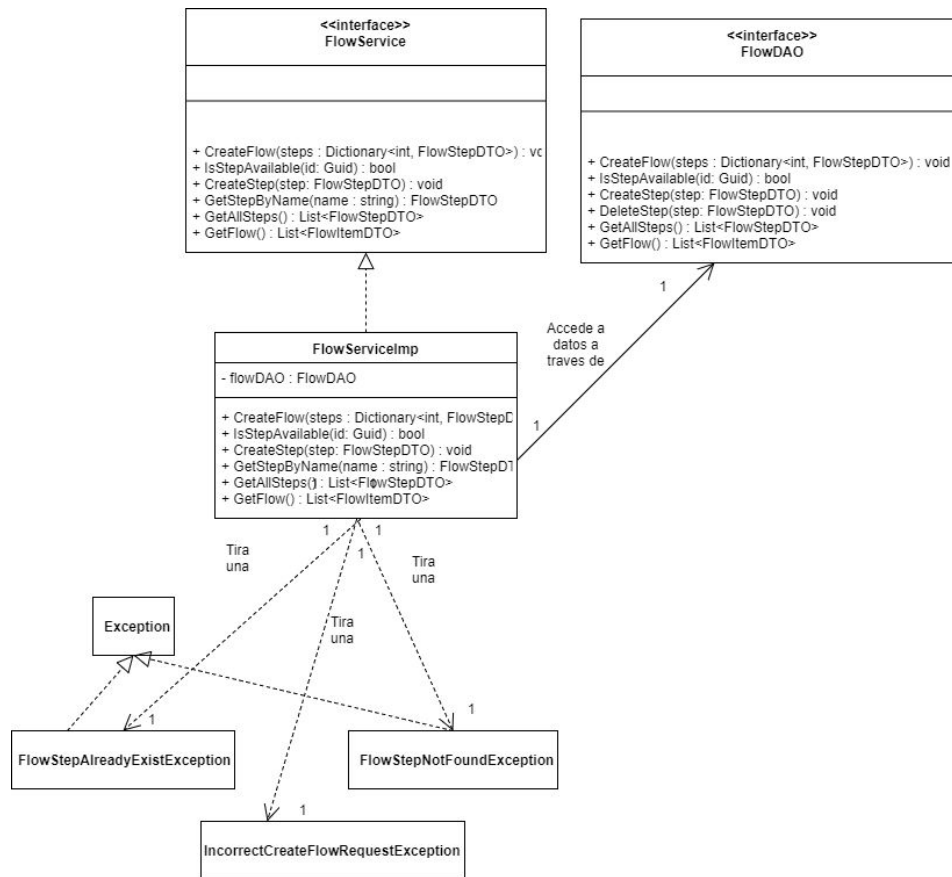


Diagrama de servicio de flujo

Por último mostramos como funciona el servicio de flujo a través de su diagrama de clases para ver con que clases interactúa.



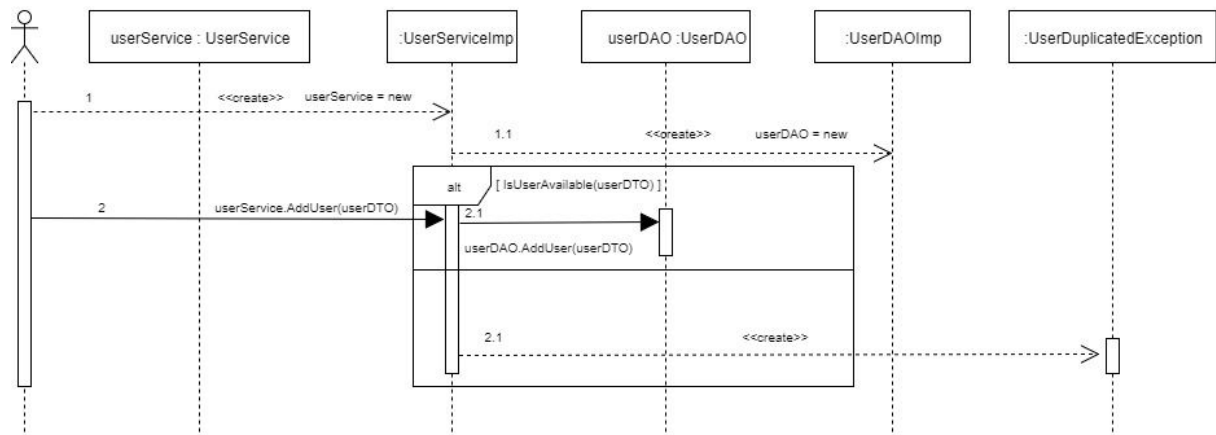
Interacciones

A continuación se verán las interacciones en la capa de servicios, las que no aparezcan se entiende que son análogas a las presentadas, o que no aportan a entender la lógica del sistema:

Crear usuario

Se muestra qué interacción tiene que hacer el usuario con el sistema para crear un usuario.

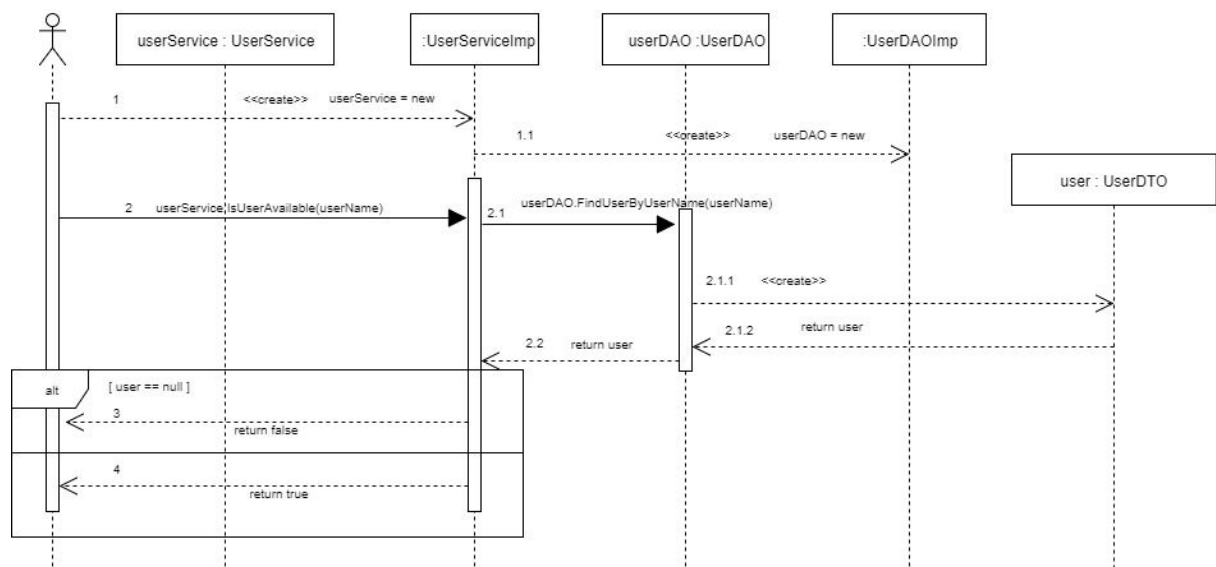
Crear Usuario



Obtener un usuario

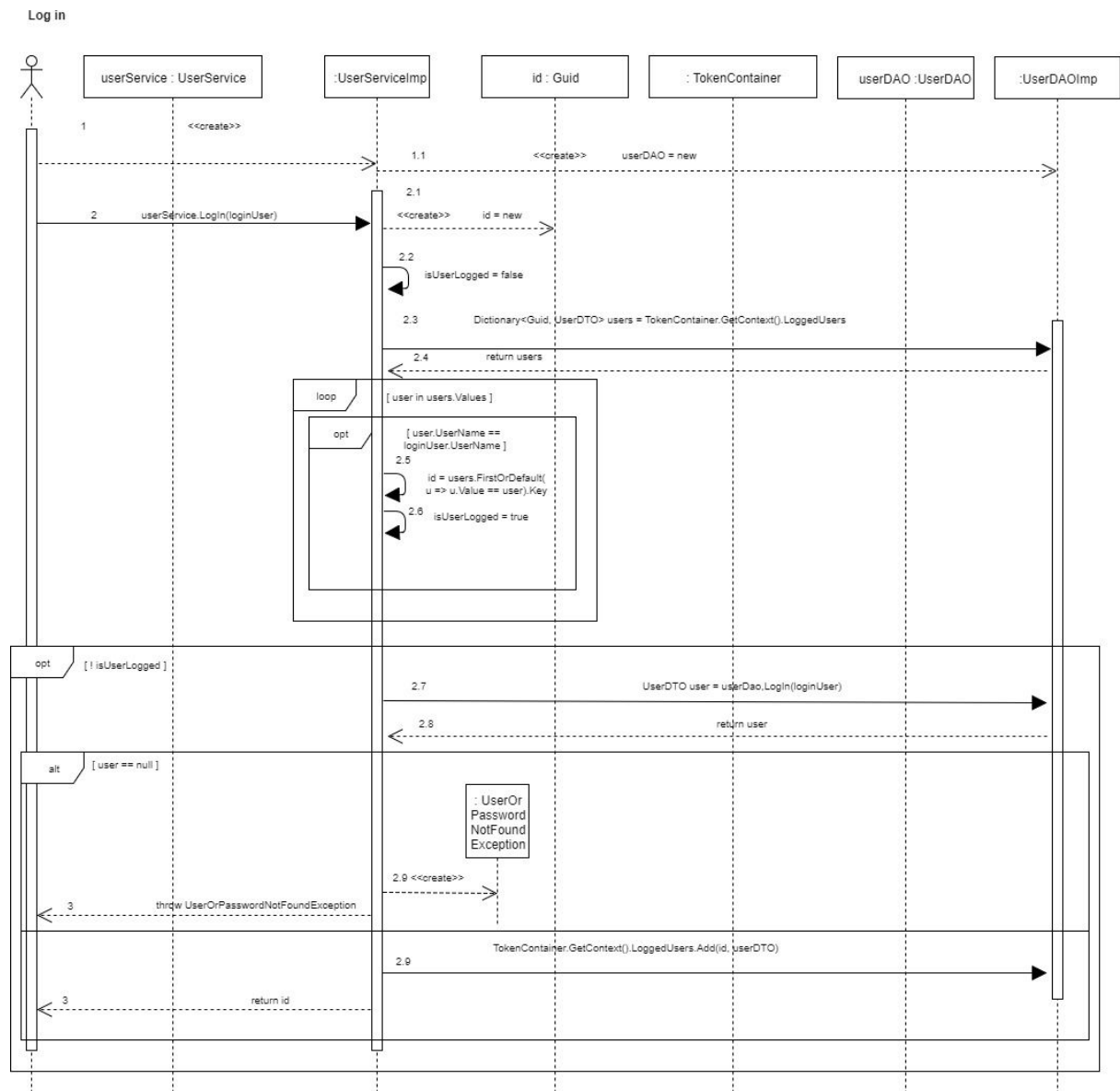
Se muestra qué interacción tiene que hacer un administrador si quiere saber si un usuario se encuentra o no en el sistema.

Averiguar si un usuario se encuentra en el sistema



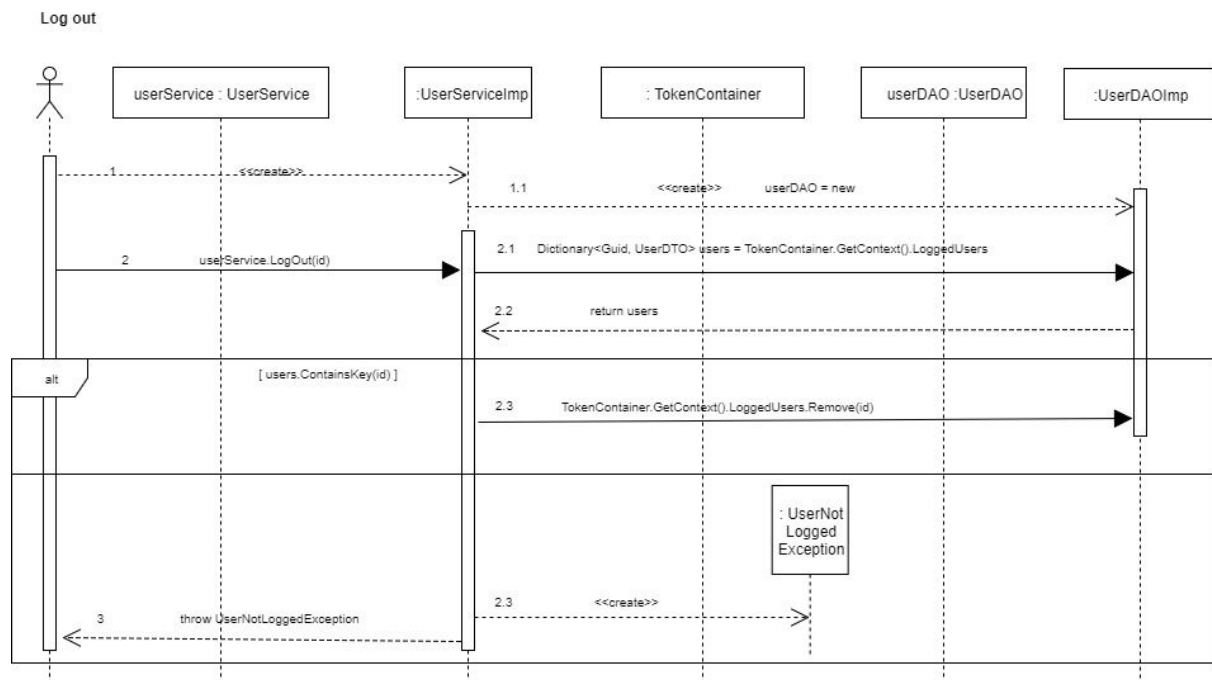
Log in

A continuación se puede observar cómo interactúa el sistema cuando un usuario quiere ingresar en él.



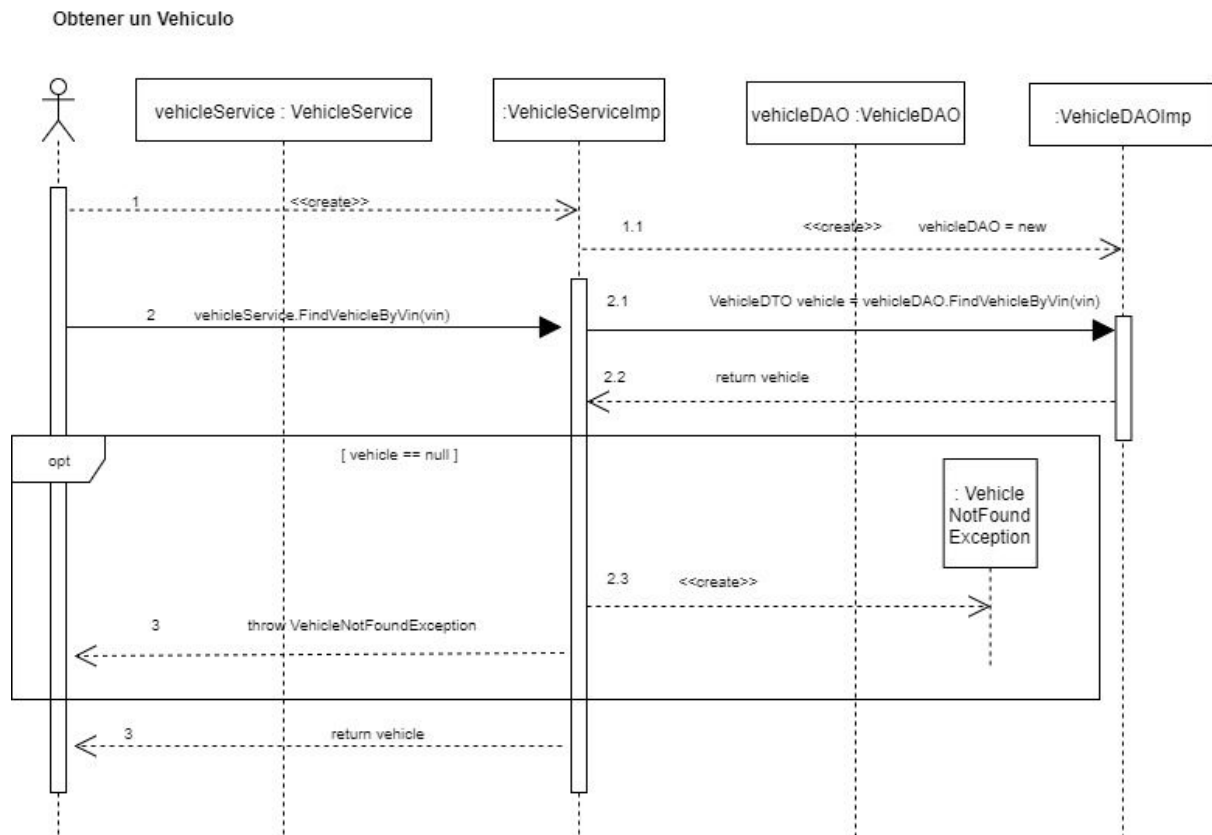
Log in

Luego se puede observar cómo interactúa el sistema si el usuario ya no quiere estar logueado en la aplicación.



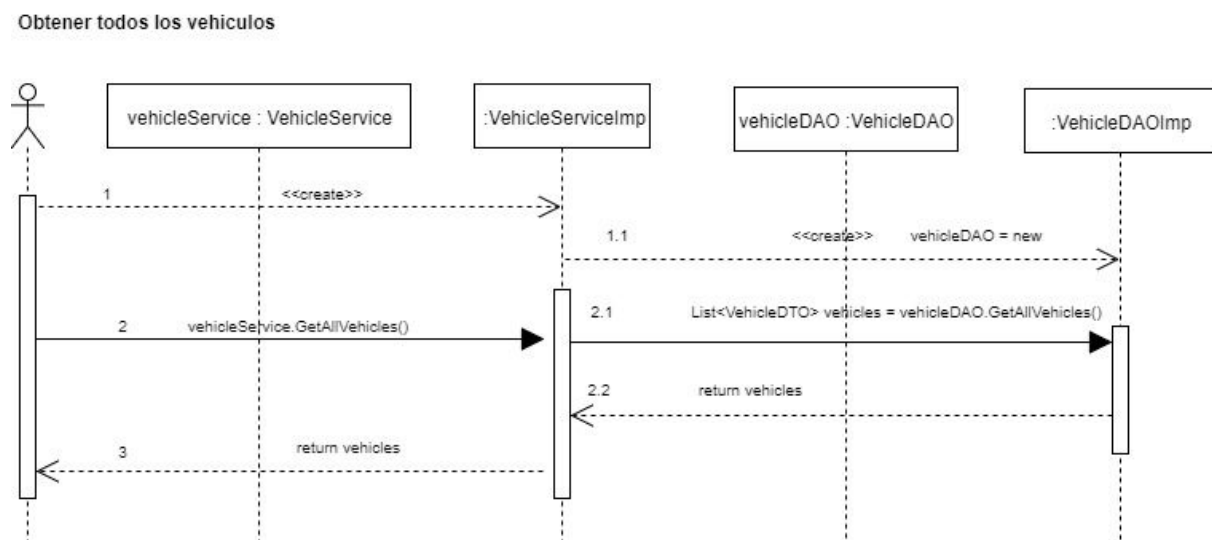
Obtener un vehículo

Se muestran las interacciones que debe tener un usuario para obtener un vehículo específico en el sistema.



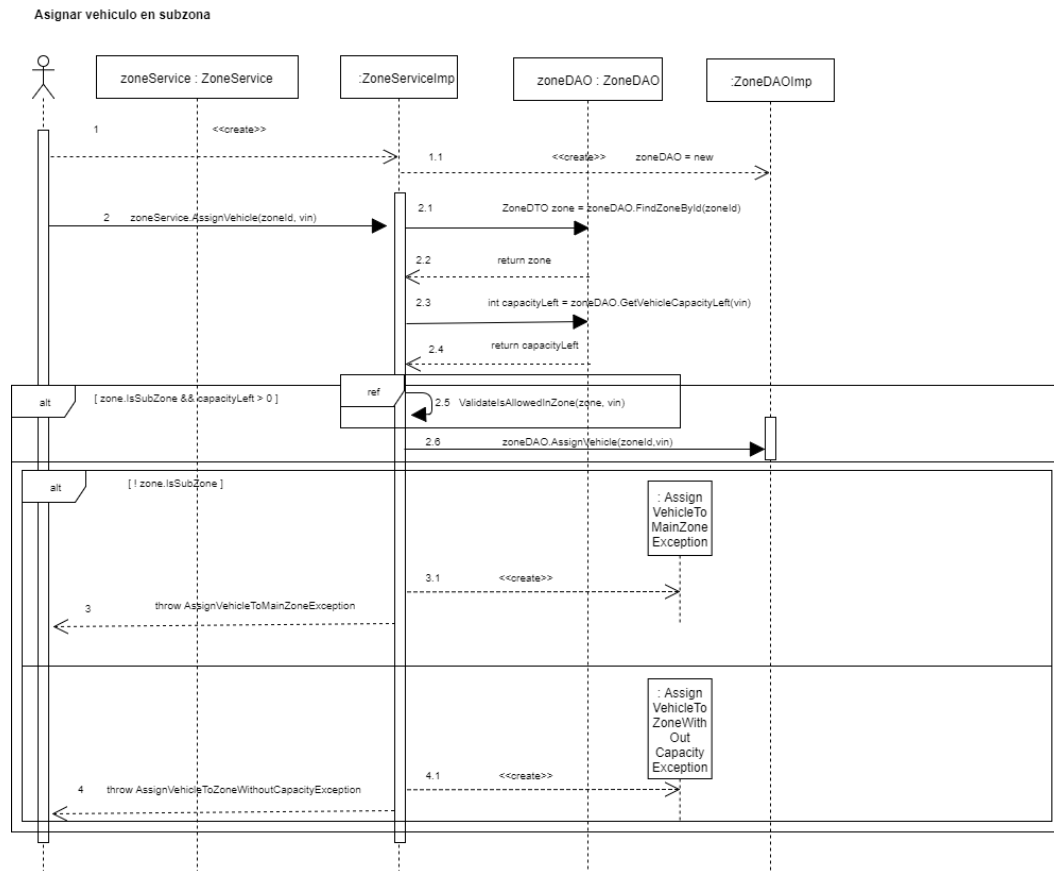
Obtener todos los vehículos

Se muestran las interacciones que debe tener un usuario si quiere obtener todos los vehículos del el sistema.

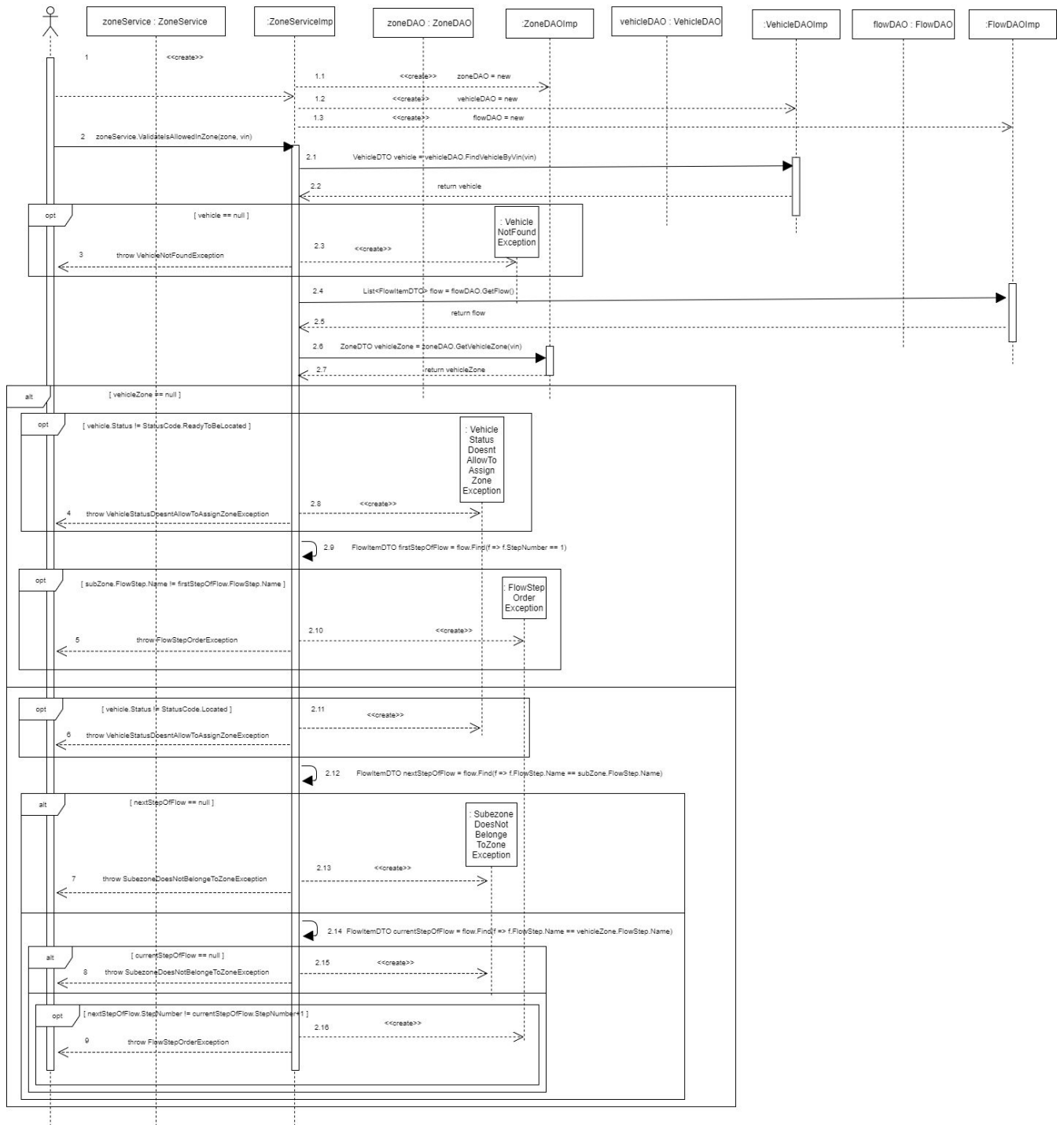


Asignar vehículo en zona

Está es una de las funcionalidades más complejas de la aplicación y por ende de las más interesantes para mostrar sus interacciones, para este caso se va a componer de dos diagramas, el segundo que se muestra es al que se le hace referencia en el primer diagrama.

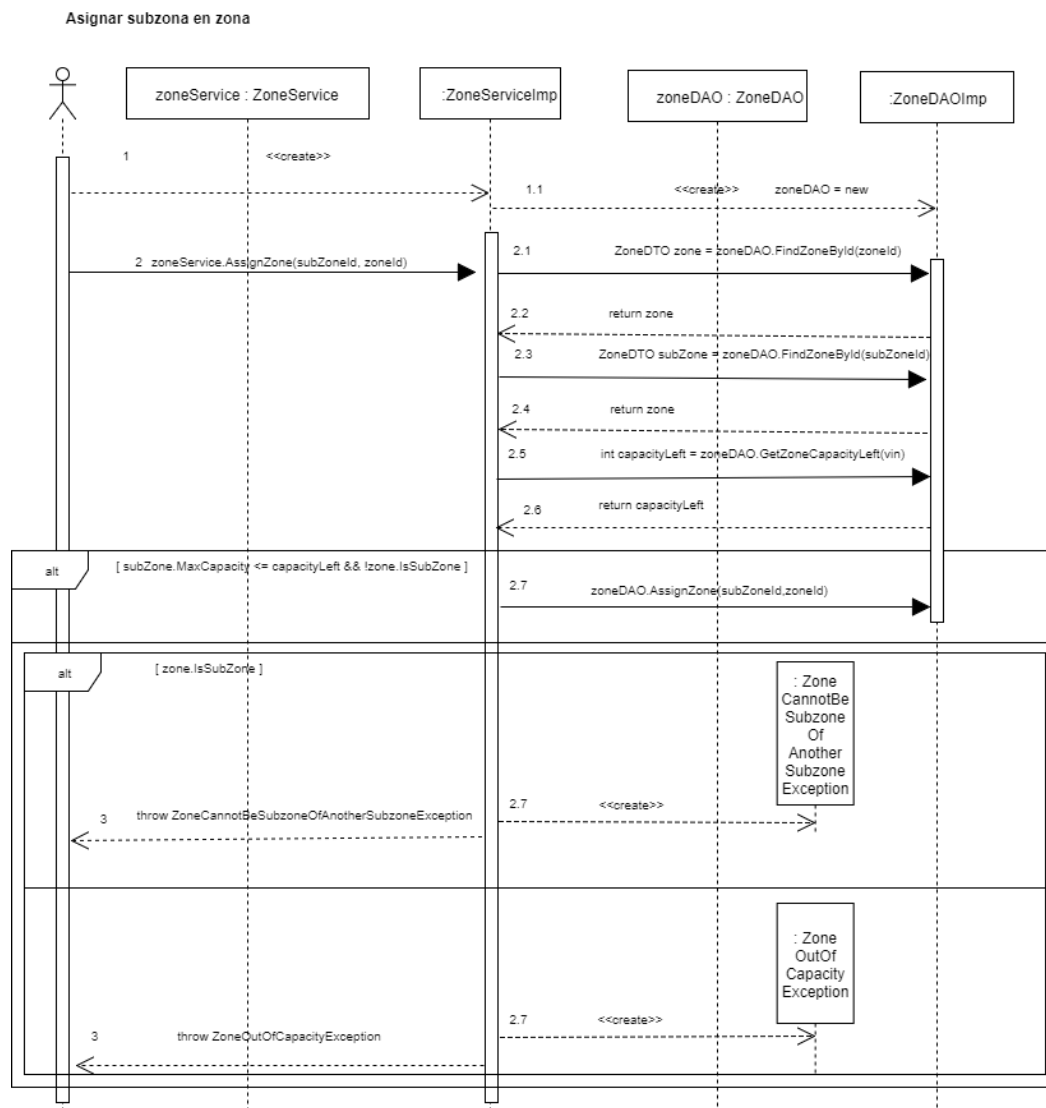


Validar si un vehiculo puede ingresar a una subzona segun el flujo



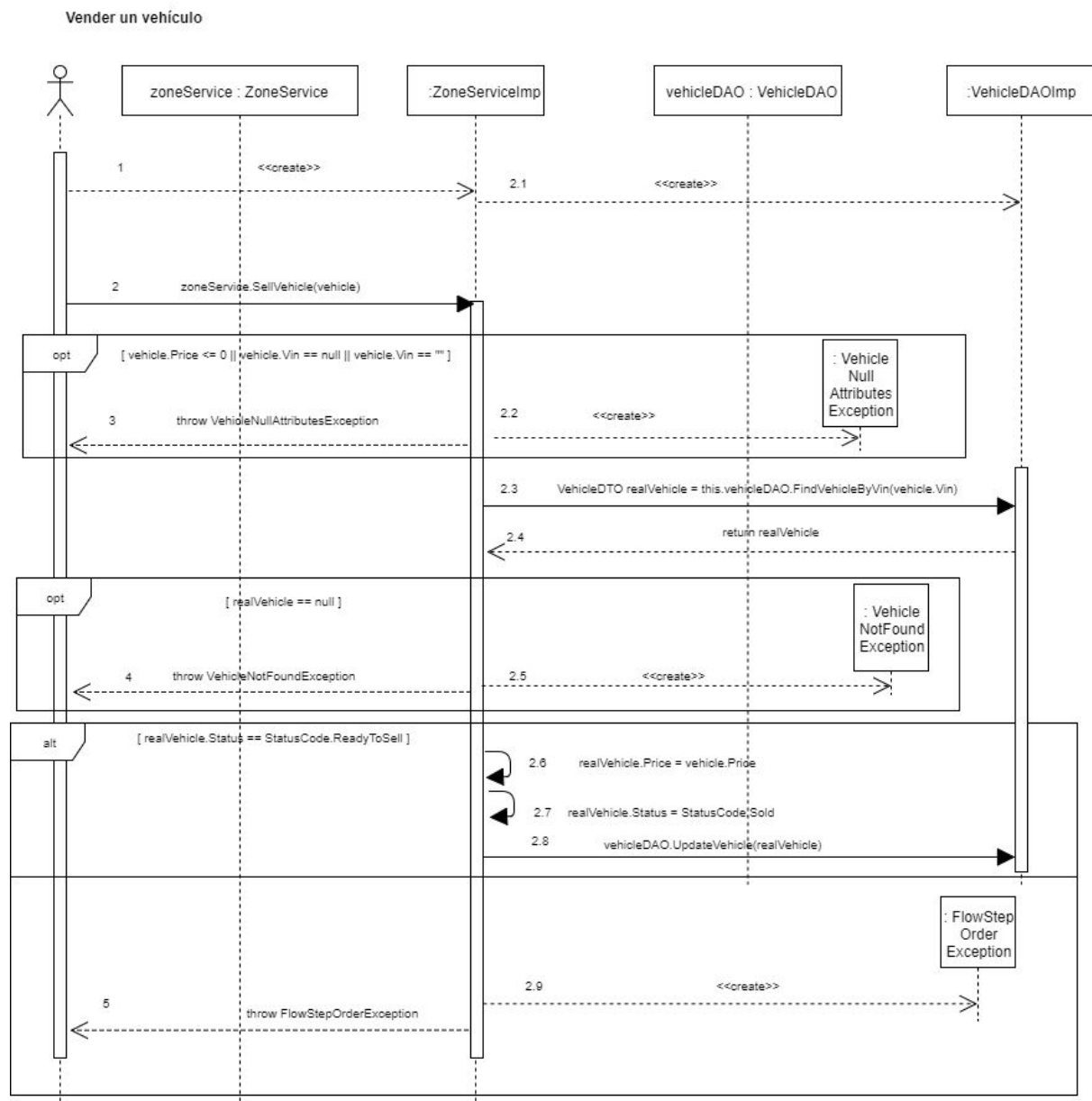
Asignar subzona en zona

Se muestran las interacciones que debe tener un usuario si quiere asignar una subzona a una zona dada.



Vender vehículo

Finalmente se muestra la interacción de vender vehículo, ya que creemos que es un poco diferente a las otras interacciones y nos interesa mostrar cómo funciona.



Excepciones

Se muestra a continuación un listado de las excepciones que son lanzadas por esta capa:

- AssignVehicleToMainZoneException
- AssignVehicleToZoneWithoutCapacityException
- BatchIsNotReadyException
- BatchNotFoundException
- FlowStepAlreadyExistException
- FlowStepNotFoundException
- FlowStepOrderException

- ImageNotFoundException
- IncorrectCreateFlowRequestException
- InspectionNotFoundException
- PortInspectionNotCreatedException
- TransportNotFoundException
- UserDuplicatedException
- UserNotAuthorizedException
- UserNotExistException
- UserNotLoggedException
- UserOrPasswordNotFoundException
- VehicleInOtherBatchException
- VehicleNotFoundException
- VehicleNullAttributesException
- VehicleStatusDoesntAllowedToAssignZoneException
- VehicleVinDuplicatedException
- ZoneCannotBeSubzoneOfAnotherSubzoneException
- ZoneNotFoundException
- ZoneNullAttributesException
- ZoneOutOfCapacityException

Flujo

Para crear el flujo en el sistema, se crearon dos clases, FlowItem y FlowStep, el FlowItem está compuesto por un número, que indica el orden en el que debe ir ese item en el flujo, y un FlowStep que dice el tipo de paso que es, luego **un flujo se define por una lista de FlowItem**.

Ejemplo: Una subzona tiene un FlowStep asignado, ese FlowStep tiene como atributos id y nombre, el nombre de ese FlowStep podría ser lavado, y a su vez, uno de los item del flujo tiene ese FlowStep de lavado, esto resulta en la posibilidad de que un vehículo pase por esa subzona al atravesar el flujo predefinido por algún administrador, siempre y cuando esté en el orden correspondiente, es decir, el vehículo se encontraba en una subzona en el paso 3, y el paso 4 es el de lavado, entonces puede asignarse a la subzona del ejemplo.

Justificación:

Creamos un servicio de flujo ya que el flujo en si, es un concepto muy complejo de nuestro sistema y necesitábamos servicios para acceder a él, consultarlo, y crearlo de manera dinámica por un administrador, además los datos del flujo debían permanecer almacenados en la base de datos, ya que era de sumo interés que no se perdiera una vez la aplicación no estuviera corriendo, por eso mismo también creamos un DAO para persistir y traer los datos del flujo.

Era necesario que las subzonas fueran tipificadas, ya que una de las restricciones de nuestro sistema es que los vehículos sean ubicados en subzonas respetando un flujo que puede variar con el tiempo, por lo que compartimos esa tipificación entre una SubZona y lo que es el flujo en si, y además a esa tipificación en el flujo le asignamos un orden, para poder validar, que la subzona de la que viene, sea el anterior paso en el flujo, y a la que se quiere asignar el vehículo sea el siguiente, en caso de que el vehículo ya estuviera

asignado a una subzona, en caso que la subzona que se encuentra el vehículo, sea la última de nuestro flujo definido, dejamos que el vehículo se ponga listo para venderse, y en caso de que el vehículo no esté asignado a una subzona aun, verificamos que a la subzona que se le quiere asignar, corresponda al primer paso del flujo.

Esta implementación nos permitio que varias subzonas puedan tener el mismo tipo, y por lo tanto, un vehículo podría tener más de una posible subzona para moverse y a su vez respete el flujo.

Contenedor

Se construyó un singleton TokenContainer para almacenar en memoria los tokens de los usuarios loggeados, este contenedor es un singleton, ya que interesa que haya una sola instancia de el, y este mismo conserve todos los tokens de los usuarios loggeados mientras el servidor esté funcionando.

Justificación:

No almacenamos en la base de datos los token de los usuarios logueados, ya que entendemos que la carga de este servidor no será lo suficientemente grande como para tener varios cluster y un balanceador de carga, de manera que la información de los usuarios logueados siempre estará centralizada en un servidor, y como no se espera que use el sistema más de 10 usuarios al mismo tiempo, no consideramos que afecte la performance del sistema, almacenar una cantidad tan poco significativa de tokens en memoria.

Handler

Se creó una clase PermissionHandler, cuya responsabilidad es proveer un set de funciones que dado un rol, diga si el mismo tiene o no permisos para la funcionalidad deseada, es usado por todos los controladores del sistema para restringir el acceso a los mismos.

Justificación:

Necesitábamos tener una clase responsable de saber exactamente cuales roles tenían permiso de acceder a las funciones específicas del sistema, de esta forma no le agregabamos una responsabilidad extra a los controladores.

Controladores

El sistema se comunica a través de una API de servicios RESTful, cada uno de estos servicios fueron creados utilizando buenas prácticas, con el fin del uso intuitivo del sistema en los clientes externos a nuestra aplicación.

A continuación se podrá ver de manera detallada, los distintos endpoints que provee el sistema y como se manejan las distintas interacciones de los mismos.

Endpoints

Listamos los endpoint según los diferentes recursos:

Usuarios

Detalle	Metodo	URI
Crear	POST	/api/users
Log in	POST	/api/users/login
Log out	POST	/api/users/logout

Vehículos

Detalle	Metodo	URI
Listar todos	GET	/api/vehicles
Listar uno (por vin)	GET	/api/vehicles/vin
Listar por grupo	GET	/api/vehicles/group/state
Crear	POST	/api/vehicles
Borrar	DELETE	/api/vehicles
Actualizar	PUT	/api/vehicles
Obtener historial	GET	/api/vehicles/vin/history

Lotes

Detalle	Metodo	URI
Listar todos	GET	/api/batches
Listar uno (por id)	GET	/api/batches/id
Crear	POST	/api/batches

Inspecciones

Detalle	Metodo	URI
---------	--------	-----

Listar todos	GET	/api/inspections
Listar uno (por id)	GET	/api/inspections/id
Crear	POST	/api/inspections

Transporte

Detalle	Metodo	URI
Listar todos	GET	/api/transportes
Listar uno (por id)	GET	/api/transportes/id
Comenzar transporte	POST	/api/transportes/start
Finalizar transporte	POST	/api/transportes/finish

Roles

Detalle	Metodo	URI
Listar todos	GET	/api/roles

Zonas

Detalle	Metodo	URI
Listar todos	GET	/api/zones
Listar uno (por id)	GET	/api/zones/id
Crear	POST	/api/zones
Borrar vehículo	DELETE	/api/zones/vehicle/vin
Asignar vehículo a subzona	POST	/api/zones/vehicle/assign?zoneId=id&vin=vin
Asignar subzona a zona	POST	/api/zones/subzone/assign?subZoneId=subZoneId&zoneId=zoneId
Eliminar vehículo de una subzona	DELETE	/api/zones/vehicles/vin
Obtener flujo actual	GET	/api/zones/flow
Obtener todos los posibles tipos de subzonas	GET	/api/zones/steps

Justificación:

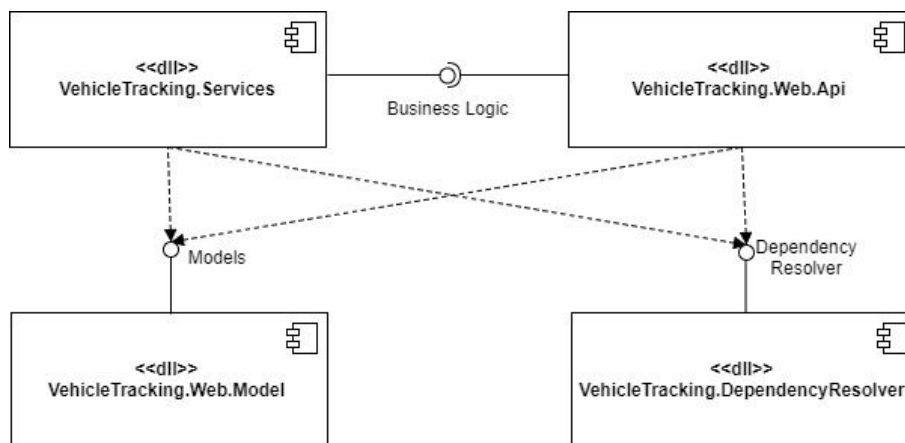
Para el log in y el log out del sistema, entendemos que son usuarios los que ingresan y egresan del mismo, por lo que decidimos que era responsabilidad no solo del servicio, sino que del controlador de usuarios encargarse de esta funcionalidad, por lo que decidimos dejarlo dentro del endpoint /users/.

La asignación de subzonas en zonas, y vehículos en las mismas, como no existe un recurso en si para esto, y es una funcionalidad, decidimos que se pudiera hacer dentro del controller de zonas, con query params, siguiendo la documentación de las buenas prácticas de RESTful services.

Agregamos los servicios para consultar el flujo actual y los posibles tipos de subzonas dentro del controlador de zonas ya que creemos que es responsabilidad de este controlador manejar todo lo relacionado a las zonas, y el flujo está fuertemente enlazado a las zonas.

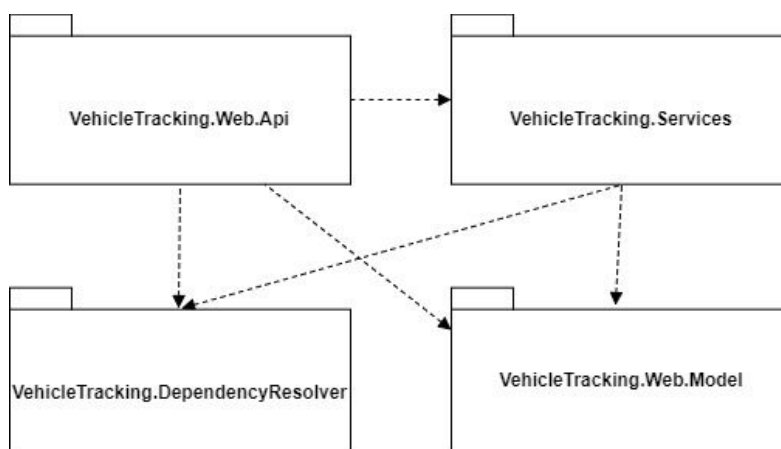
Componentes

Los siguientes componentes son utilizados en la capa de controladores:



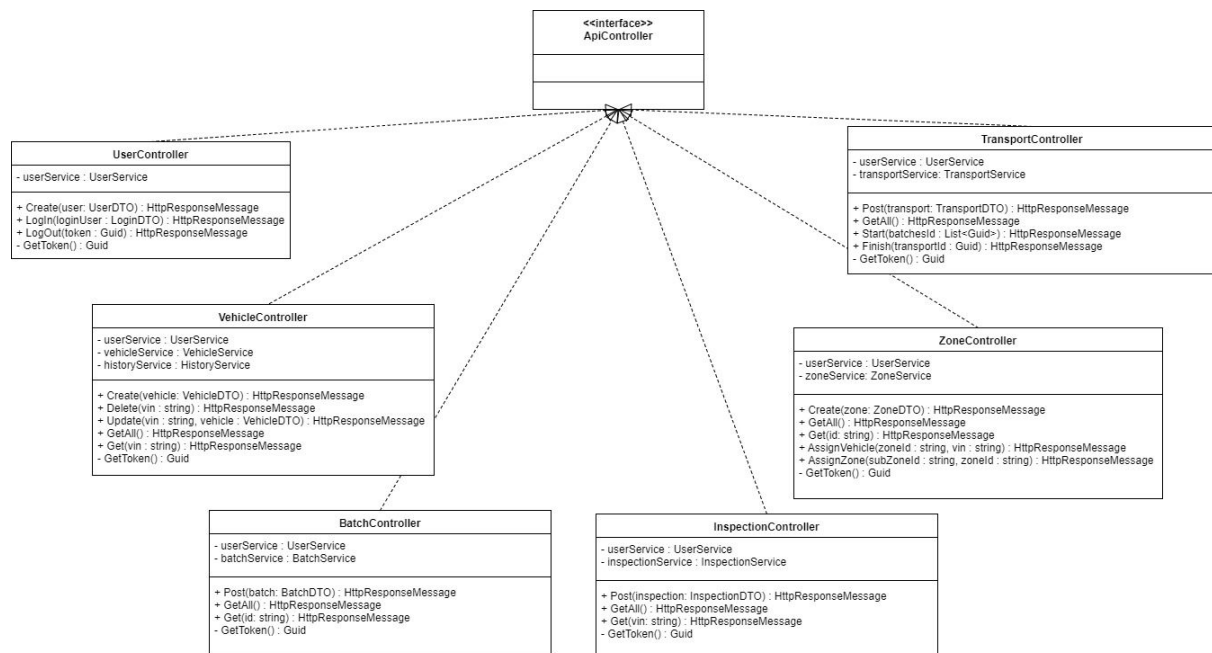
Paquetes

Y los siguientes paquetes que interactúan en esta capa, a grandes rasgos, ya que los mismos fueron detallados en las generalidades del sistema:



Clases

Se puede observar el siguiente diagrama de clases, donde aparecen los distintos controllers de la API REST.



Aplicación web

Para el sistema de Vehicle tracking se creó una aplicación web, con el fin de facilitar el uso del sistema al usuario final, la misma está orientada a los usuarios no administradores de la aplicación, sin embargo cuenta con todas las funcionalidades para que utilicen también los administradores.

Funcionalidad por rol:

- Administrador: Puede utilizar todas las funcionalidades de la aplicación web.
- Operario del puerto:
 - Puede listar los vehiculos que estan en estado de:
 - En puerto
 - Inspeccionado en puerto
 - Listo para partir en puerto
 - Puede crear inspecciones a los vehículos en puerto
 - Puede crear lotes con los vehículos que estén en condiciones de ser ubicados en los mismos.
 - Acceso al historial de los vehículos
- Transportista:
 - No puede listar vehículos ya que solo le interesa listar los lotes que contienen los vehículos.
 - Puede comenzar a transportar un set de lotes.
 - Puede terminar de transportar lotes.
 - Acceso al historial de los vehículos
- Operario del patio:
 - Puede listar los vehiculos que estan en estado de:
 - Esperando inspección en patio
 - Listo para ser ubicado en zona
 - Ubicado en zona
 - Crear zona
 - Crear subzona
 - Mover subzonas
 - Asignar vehículos a subzonas
 - Preparar el vehículo para la venta
 - Acceso al historial de los vehículos
- Vendedor:
 - Puede listar los vehiculos que estan en estado de:
 - Listo para la venta
 - Vendido
 - Vender vehículos

Maquina de estados

Se muestra el siguiente diagrama de estados, donde se observa por las diversas acciones que debe ir pasando un vehículo para ir cambiando los estados, hasta llegar al estado final.

Maquina de estados de un Vehículo

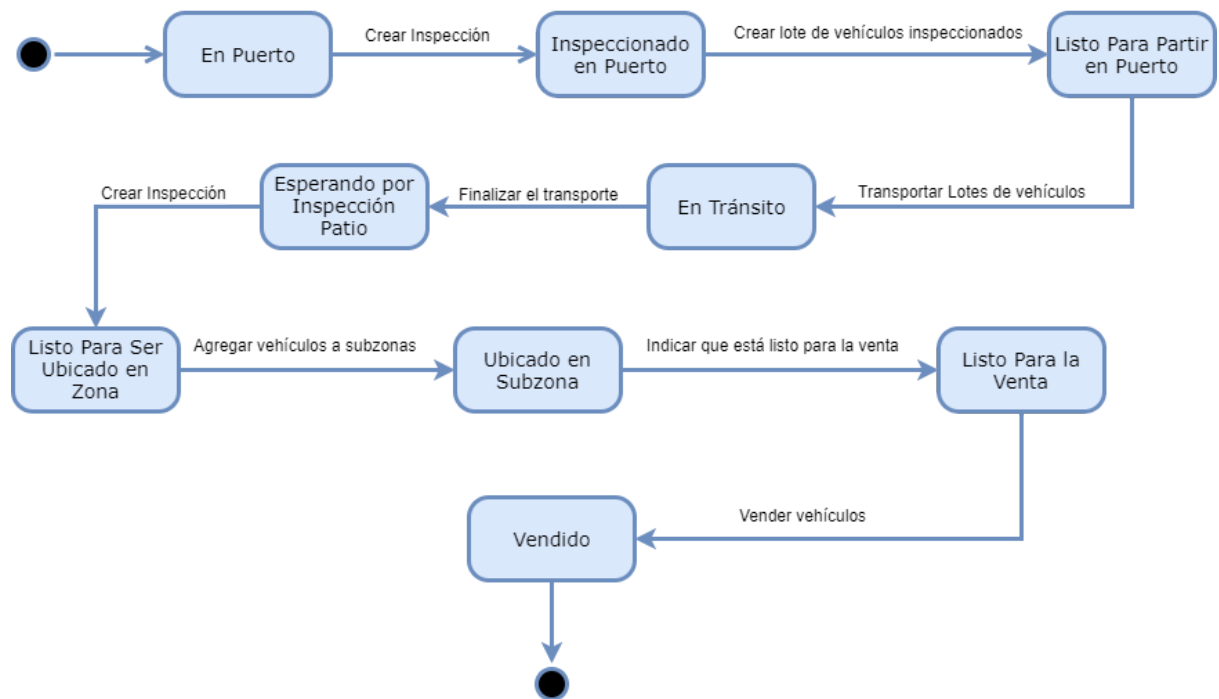
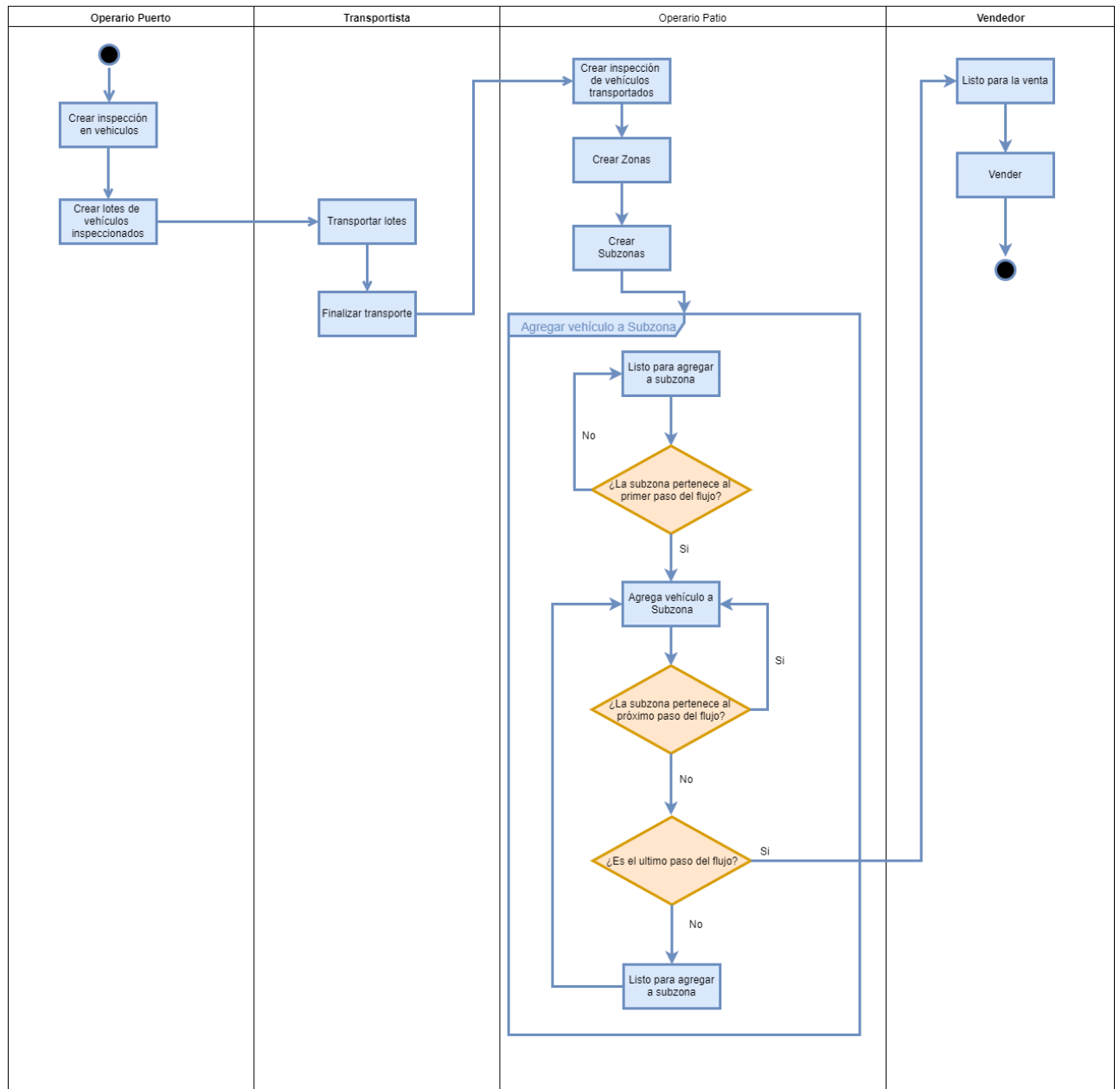


Diagrama de actividades

Aquí se muestra en forma más detallada las actividades del sistema, para que un vehículo pase por todos sus estados, y que rol interactúa en cada parte del sistema.

Diagrama de actividad del sistema



Arquitectura

Se muestra una descripción de los elementos interesantes de la aplicación web (Angular) que se creó con la siguiente estructura:

Carpeta “src” principal del proyecto que contiene

- index.html: donde se inicia la pagina y se llama al componente principal
- style.css: contiene los estilos de toda la pagina
- carpeta js: contiene el javascript del template utilizado
- carpeta img: contiene el logo e imágenes de la página
- carpeta css: contiene los estilos del template y de bootstrap
- assets: contiene un archivo de configuración con la url de los servicios web.
- modulo app: contiene la lógica y los html de toda la aplicación web.
 - dentro de ella se encuentra el modulo principal de la aplicación y el html correspondiente a dicho módulo, donde se conectan todos los otros módulos de la aplicación.
 - modulo zone
 - modulo vehicle
 - modulo transport
 - modulo inspection
 - modulo common
 - módulo batch

Cada una de estos módulos contiene varios módulos que se detallan a continuación:

Common:

Modulo	Contenido	Descripción
Components	loading	Muestra el contenido de una animación y un letrero que indica que se están cargando datos.
	loading-modal	Abre un modal donde muestra el contenido de una animación y un letrero que indica que se están cargando datos.
	login	Contenido que se abre dentro de un modal para poder ingresar al sistema.
	modal	Es el componente que se usa cada vez que se abre un modal en la aplicación, lo suficientemente genérico para poder ingresarle cualquier contenido, y que el mismo sea independiente del modal.
Services	login	Tiene un servicio con las llamadas a los endpoint del login.

	config	Es un servicio que lee un archivo de configuración, obtiene la url principal de los servicios rest y la devuelve.
--	--------	---

Zone:

Modulo	Contenido	Descripción
Components	assign-vehicles	Contenido que se abre dentro de un modal con una lista de vehiculos para asignar a una subzona.
	create-subzone	Contenido que se abre dentro de un modal para crear una subzona dentro de otra zona.
	create-zone	Contenido que se abre dentro de un modal para crear una zona principal.
	display-zones	Muestra todo el contenido de las zonas y subzonas del sistema, botones y se comunica con los distintos componentes del módulo zone.
	move-zone	Asigna una subzona a otra zona principal (la mueve).
Pipes	search-zone	Filtra las zonas segun el Id de la misma.
Services	zone	Es un servicio con las llamadas a los distintos endpoint de la capa de servicios rest del controlador de zona.

Vehicle:

Modulo	Contenido	Descripción
Components	display-vehicles	Muestra todo el contenido de los vehiculos del sistema, botones y se comunica con los distintos componentes del módulo vehicle.
	display-historic	Muestra el historial de un vehículo buscado por vin.
	create-vehicle	Contenido que se abre dentro de un modal para crear un vehículo.
Pipes	search-vehicle	Filtra los vehículos según el vin del mismo.
Services	vehicle	Es un servicio con las llamadas a los distintos endpoint de la capa de servicios rest del controlador de vehiculos.

Transport:

Modulo	Contenido	Descripción
Components	display-transport	Muestra todo el contenido de los transportes del sistema, y botones.
Pipes	search-transport	Filtra los transportes segun el id del mismo.
Services	transport	Es un servicio con las llamadas a los distintos endpoint de la capa de servicios rest del controlador de transportes.

Inspection:

Modulo	Contenido	Descripción
Components	display-inspections	Muestra todo el contenido de las inspecciones del sistema, botones y se comunica con los distintos componentes del módulo inspection.
	create-inspection	Contenido que se abre dentro de un modal para crear una inspección..
Pipes	search-inspection	Filtra las inspecciones según el id de las mismas.
Services	inspection	Es un servicio con las llamadas a los distintos endpoint de la capa de servicios rest del controlador de inspections.

Batch:

Modulo	Contenido	Descripción
Components	display-batches	Muestra todo el contenido de los lotes del sistema, botones y se comunica con los distintos componentes del módulo batch.
	create-batch	Contenido que se abre dentro de un modal para crear un lote..
Pipes	search-batch	Filtra los lotes segun el id de los mismos.
Services	batch	Es un servicio con las llamadas a los distintos endpoint de la capa de servicios rest del controlador de lotes.

A fuera de la carpeta “src” hay un archivo llamado package.json que contiene las versiones y los distintos componentes de librerías de terceros que se usan en la aplicación, junto con la versión actual del proyecto.

Justificación:

Archivo de configuración con url de servicios web del sistema, se hizo ya que el dominio y el puerto de los mismos podría cambiar, y de esta forma no se necesita recompilar la solución para seguir conectando a la página.

Elegimos modularizar la aplicación de la forma descrita arriba ya que creemos que separa de forma correcta las responsabilidades del contenido de cada uno de los módulos y sus correspondientes sub módulos, por ejemplo, en el modulo de vehículos, está todo el contenido referente a la seccion vehiculos de la aplicación web, pero además, cada uno de los componentes que tienen responsabilidades distintas, están separados, y bajo el modulo de componentes, se separan los filtros en distintos pipe, uno por filtro, bajo el modulo de pipes, y se separan los servicios en distintos services, dentro del módulo de services, cada uno de ellos cuenta con responsabilidades concretas y todas referentes a lo que se trabaja en la seccion vehiculos.

Manejo de errores

Debido a que la aplicación web es para usuarios finales, decidimos que el manejo de errores sea usable para un usuario final, por esto entendemos que por ejemplo si un vehículo no puede introducirse dentro de un lote sin ser anteriormente inspeccionado, no se intente agregar y mostrar un error al usuario, sino que directamente listamos los vehículos que el usuario si puede agregar dentro de ese lote.

Por lo que el usuario siempre deberá encontrarse dentro del “happy path” (camino feliz), contando con doble validación, tanto del front end, como del back end, para ingresar datos e interactuar con la aplicación.

Justificación:

Nos basamos en las heurísticas de usabilidad de Nielsen para manejar los errores de una forma que no sea complicada para entender para un usuario y se abstraiga un poco de la lógica del sistema en sí, ya que el usuario a priori no tiene porque saber las reglas del sistema.

“Prevención de errores: mucho mejor que un buen diseño de mensajes de error es realizar un diseño cuidadoso que prevenga la ocurrencia de problemas.”

Aplicación windows

Para el uso del mantenimiento del sistema se creó una aplicación windows para el uso exclusivo de administradores, en la que se puede hacer alta baja y modificacion de vehiculos y zonas, además de crear de manera dinámica el flujo de trabajo por el cual debe pasar un vehículo para poder ser vendido.

Funcionalidades

- Alta, baja y modificación de zonas.
- Alta, baja y modificación de vehículos.
- Creación de tipos de subzona.
- Creación de flujo de forma dinámica, según los tipos de subzonas habidas en el sistema.
- Ver logs del sistema entre dos fechas seleccionables por el administrador.
- Importar vehículos desde xml y csv

Manejo de errores

Dado que el usuario objetivo para esta aplicación es un administrador, el manejo de errores que utilizamos es indicarle si algo no se puede hacer, con un mensaje claro que indique el porque no se puede hacer, y lo dejamos revertir su accionar.

Patrones de diseño

- Observer: Para refrescar la información entre pantallas, cuando se usan varias a la vez.

Logs

La aplicación de logs fue diseñada con el fin de controlar las acciones que pasan en el sistema.

En primer lugar se decidió crear una interfaz ILog con el fin de que futuros mecanismos de logueo puedan implementarla para poder definir su manera de guardar los datos de eventos ocurridos en el sistema.

En principio contamos con los eventos de Login y VehicleImport, pero de esta manera se podrán agregar otros eventos que se desee loguear.

Justificación:

La solución planteada fue diseñada con el fin de ofrecer extensibilidad.

Para este obligatorio se definió un proyecto aparte con la realización de la interfaz anteriormente mencionada ILog, que nos permite guardar y obtener logs en un archivo de texto plano.

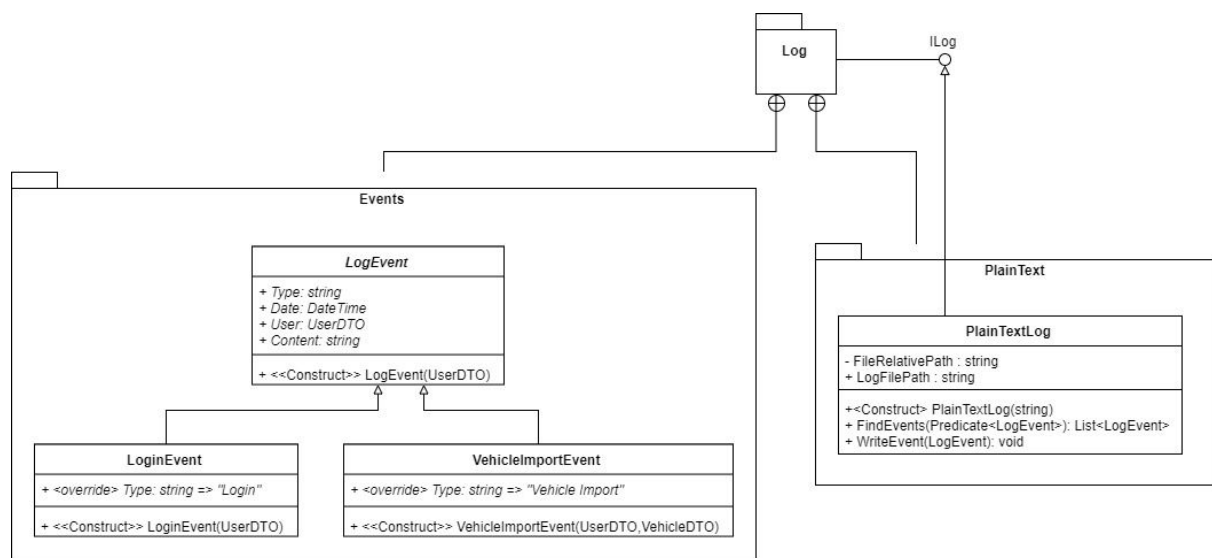
Se decidió separar la lógica de logs en un proyecto aparte, de manera que se pueda reutilizar en otros sistemas este mecanismo de logs. Simplemente copiando y llevando el proyecto de logs a otro sistema ya estaremos reutilizándolo.

Patrones de diseño

- Strategy: Para los eventos se decidió crear una clase abstracta LogEvent del cual serán derivados los eventos particulares que se desea loguear. Para ello se utilizó éste patrón de diseño.

Paquetes

A continuación mostramos el diagrama de paquetes bien detallado de esta parte de la solución, con el fin de entender cómo interactúan los paquetes entre si.

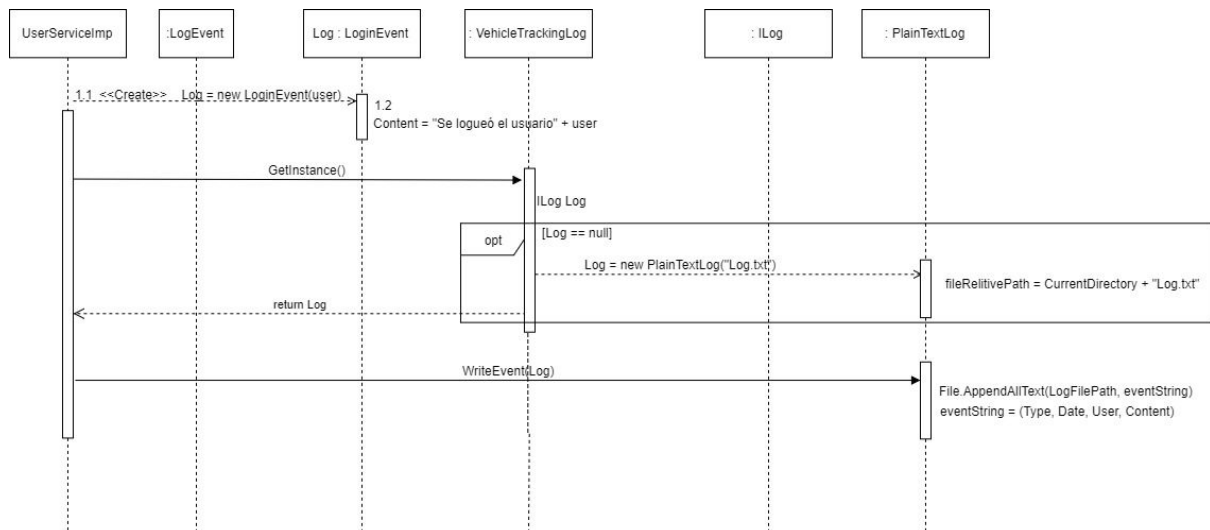


Interacciones

Se muestra la interacción más interesante de la solución de log.

Logueo de información en texto plano

En ella se puede observar como se escriben los logs en texto plano.



Importador de archivos

El sistema de importers está diseñado con el fin de poder importar vehículos desde archivos en tiempo de ejecución.

Para ello se define un interfaz `IImporterOffline` con el fin de establecer cómo deberán ser los importers que se agregaran al sistema. De esta forma agregamos extensibilidad al sistema ya que realizando esta interfaz se podrán definir los distintos tipos de importers que necesitemos.

Justificación:

El sistema de importers está diseñado con el fin de ofrecer extensibilidad al sistema en tiempo de ejecución es decir que un tercero pueda agregar funcionalidades sin necesidades de recompilar la solución.

Para este obligatorio implementamos la interfaz `IVehicleImportOffline` que extenderá de `IImporterOffline` y será la que se realizará por los importers externos de forma de desacoplar la implementación del importador concreto con nuestra lógica de negocio. De la misma forma se podrán definir otros importers.

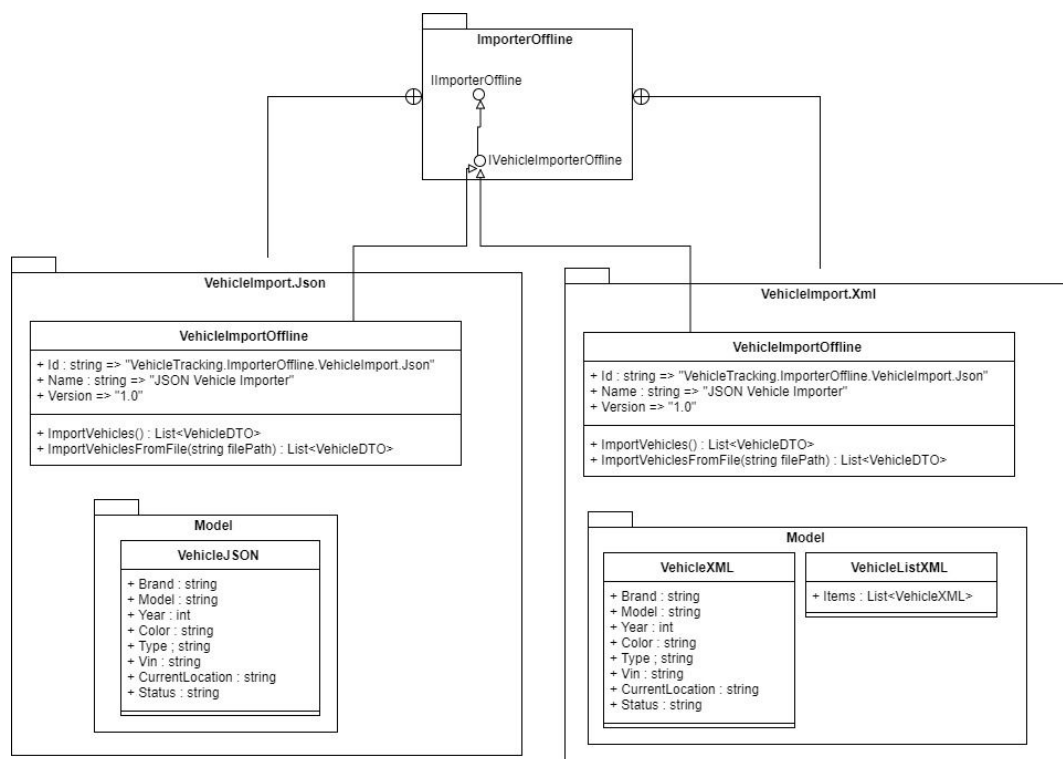
Se implementa además un `ImporterOffline` en la capa de servicios ya que nos interesa que las funcionalidades que agreguemos se puedan utilizar en toda la capa de negocio y no solo en la interfaz (WebAPI, Forms, u otra). Aquí se cargarán automáticamente los importers, a través de una ruta configurable.

Se decidió separar esta lógica en un proyecto aparte para poder reutilizar el mecanismo de importar, de manera que simplemente sea necesario copiar el proyecto y llevarlo a otro sistema.

A la hora de implementar un importador particular (como por ejemplo VehicleImport.Json) es necesario implementar la interfaz IImporterOffline como ya mencionamos, pero además es necesario definirse un proyecto Model, donde se simule la entidad con la que se trabaja (en este caso Vehiculo). Esto es para no tener que acoplarse al sistema actual y poder reutilizarlo en otro sistema.

Paquetes

A continuación se muestra un diagrama de paquetes detallado para ver qué interacciones tiene el mismo, con que clases cuenta dentro de él y como se relacionan las clases entre los paquetes.

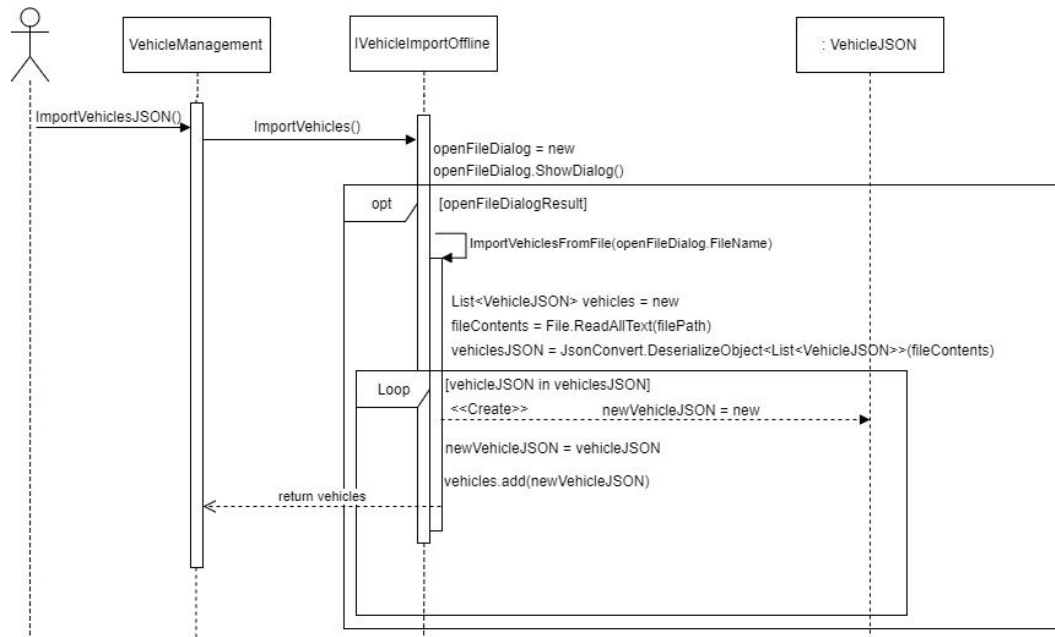


Interacciones

Aquí se mostrarán las interacciones más interesantes del importador de archivos.

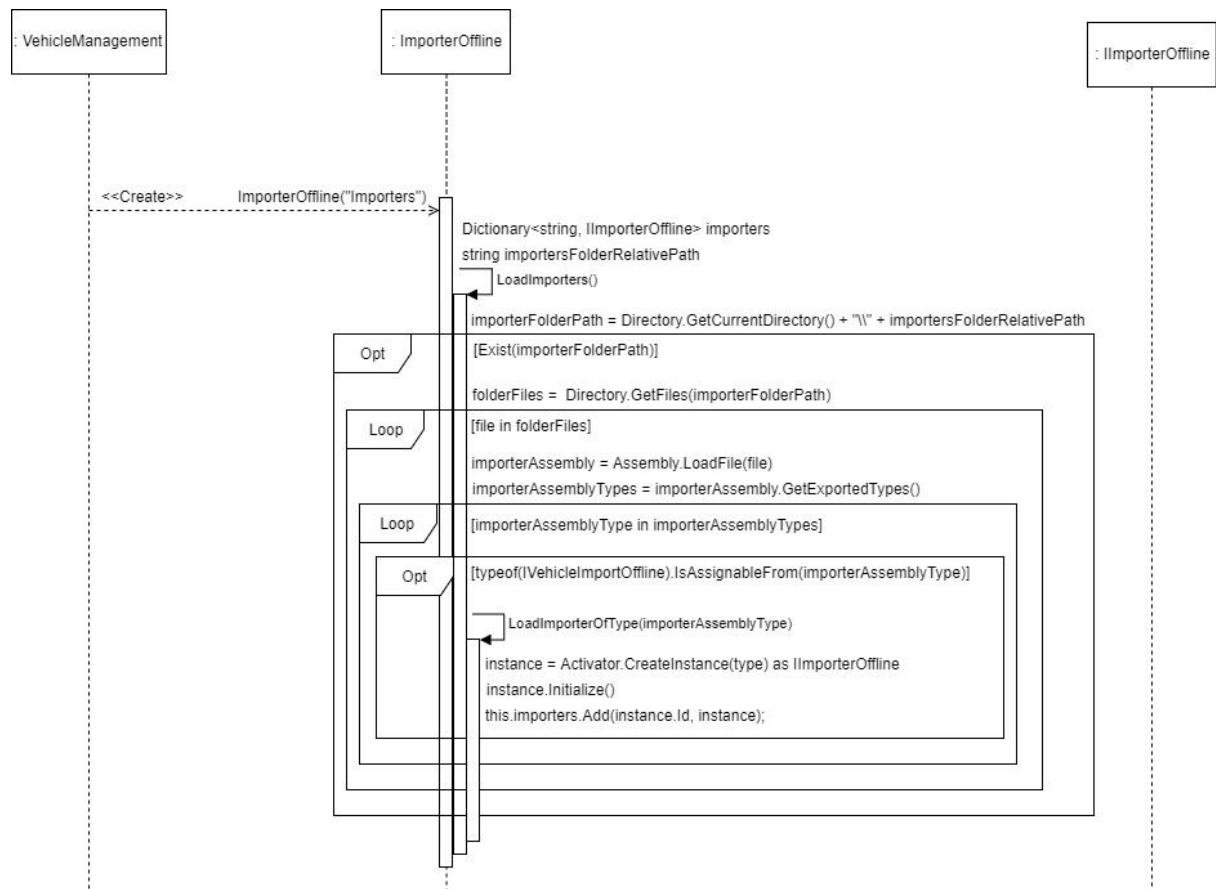
Importación del archivo JSON

Aquí se muestran las interacciones necesarias para importar vehículos a través de un archivo JSON.



Instanciación dinámica de importers

Se muestra como se debe interactuar con el sistema para instanciar un importer



Tests

Se ha utilizado la metodología de trabajo TDD (Test driven development) para la creación del sistema, con la finalidad de disminuir posibles bugs.

En esta sección se podrá observar el estado actual del sistema.


TDD

Como mencionamos anteriormente el desarrollo se realizó utilizando la metodología de TDD, y aquí mostraremos las evidencias del mismo:

-En principio fuimos trabajando cumpliendo las 3 etapas de TDD claras (Se crean las pruebas, se implementan, y se realizar un refactor).



Merge pull request #2 from ORT-DA2/feature/CreateUsers

 origin/feature/CreateUsers Refactor, agrego constructor para User, pruebas funcionando ok

Implementacion de la creacion de usuario, todas las pruebas dieron ok

Prueba para la creacion de usuario, sin funcionar

Merge pull request #1 from ORT-DA2/feature/CreacionVehiculos

 origin/feature/CreacionVehiculos Refactor de creacion de vehiculo

Prueba de creacion de vehiculo funcionando


Prueba para la creacion de vehiculos, sin compilar

Creacion del proyecto

-Una vez que ya teníamos la estructura armada no fue necesario muchas veces el tercer paso (refactor). Mostramos a continuación cómo fue



Merge pull request #6 from ORT-DA2/feature/CreateRole

 origin/feature/CreateRole Se implementa la prueba para crear rol

Se crea prueba para la creación de un rol

Merge branch 'feature/PersistUser' into develop

Cobertura

Se muestran los resultados obtenidos por el analizador de pruebas de Visual Studio, la cantidad de pruebas y las pruebas superadas:

Pruebas: superadas (142)	
✓ AssignSubZoneToAnotherSubZoneErrorTest	11 ms
✓ AssignSubZoneToAZoneWithoutCapacityTest	13 ms
✓ AssignVehicleErrorFlowStepOrderTest	15 ms
✓ AssignVehicleErrorVehicleStatusTest	17 ms
✓ AssignVehicleIsNotSubZoneTest	8 ms
✓ AssignVehicleOutOfCapacityTest	8 ms
✓ AssignVehicleSuccessfullyTest	28 ms
✓ AssignVehicleTest	8 ms
✓ AssignVehicleTest	6 ms
✓ AssignZoneSuccessfullyTest	14 ms
✓ AssignZoneTest	27 ms
✓ AssignZoneTest	6 ms
✓ CreateBatch	1 ms
✓ CreateBatchSuccessfullyAsAdministratorTest	2 ms
✓ CreateBatchSuccessfullyTest	10 ms
✓ CreateBatchTest	6 ms
✓ CreateDamageTest	< 1 ms
✓ CreateFlowSuccessfullyTest	4 ms
✓ CreateFlowTest	31 ms
✓ CreateInspectionPortTest	1 ms
✓ CreateInspectionSuccessfullyTest	16 ms
✓ CreateInspectionTest	7 ms
✓ CreateListAllBatchesTest	6 ms
✓ CreateLogLoginTest	< 1 ms
✓ CreateLogVehicleImportTest	< 1 ms
✓ CreateRoleTest	< 1 ms

Jerarquía	No cubiertos (bloques)	No cubiertos (% de bloques)	Cubiertos (bloques)	Cubiertos (% de bloques)
Tonga TONGA-PC 2017-11-23 1...	2407	21,33 %	8876	78,67 %
vehicletracking.data.dataac...	0	0,00 %	26	100,00 %
vehicletracking.data.entities...	2	0,93 %	212	99,07 %
vehicletracking.importeroffl...	11	18,03 %	50	81,97 %
vehicletracking.importeroffl...	11	15,07 %	62	84,93 %
vehicletracking.log.dll	0	0,00 %	24	100,00 %
vehicletracking.log.plaintext...	9	8,82 %	93	91,18 %
vehicletracking.repository.dll	751	31,21 %	1655	68,79 %
vehicletracking.services.dll	467	39,15 %	726	60,85 %
vehicletracking.tests.dll	324	5,58 %	5479	94,42 %
vehicletracking.web.api.dll	815	66,10 %	418	33,90 %
vehicletracking.web.models...	17	11,49 %	131	88,51 %

Se tiene un 79% de la cobertura debido a que se decidió no testear algunas funcionalidades como ser el DependencyResolver o archivos de configuración. Esto hizo que nuestra

cobertura bajara. No fue necesario agregar pruebas ya que la mayoría se generaron a medida que se codificaba gracias a la metodología de TDD.

Clases

Se listaran las clases que contienen a las pruebas unitarias del sistema:

Vehículos

- VehicleTest
- VehicleServiceTest
- VehicleDAOTest
- VehicleControllerTest

Usuarios

- UserTest
- UserServiceTest
- UserMapperTest
- UserDAOTest
- UserControllerTest

Zonas

- ZoneTest
- ZoneServiceTest
- ZoneDAOTest
- ZoneControllerTest

Historial

- VehicleHistoryTest
- VehicleHistoryServiceTest

Transporte

- TransportTest
- TransportServiceTest
- TransportMapperTest
- TransportDAOTest
- TransportControllerTest

Lotes

- BatchTest
- BatchServiceTest
- BatchMapperTest
- BatchDAOTest
- BatchControllerTest

Inspecciones

- InspectionTest

- InspectionServiceTest
- InspectionDAOTest
- InspectionMapperTest
- InspectionControllerTest

Bugs conocidos y futuras mejoras

- Bug: No se ha detectado ningún comportamiento ajeno a los requerimientos del sistema.
- Mejora: Persistir los token de loggeo.
- Mejora: Aumentar la cantidad de niveles de subzonas.
- Mejora: Proveer una interfaz gráfica para el uso del sistema.
- Mejora: Mayor cantidad de patios y puertos para elegir a la hora de crear un lote.
- Mejora: Extraer las excepciones que tiramos de la capa de controlador para que no queden tan largos los métodos.
- Mejora: Extraer el manejo de permisos para la capa de negocio y dejar de controlarlo en el controlador.

Clean Code

Mencionaremos capítulo por capítulo que practicas de clean code se han utilizado para el sistema.

Capítulo 1: Código Limpio

Se desarrolla un código que es simple y directo, se lee como un texto bien escrito.

Por ejemplo: en la implementación del servicio de usuario

```
public void AddUser(UserDTO user)
{
    if (this.IsUserAvailable(user.UserName))
    {
        this.userDAO.AddUser(user);
    }
    else
    {
        throw new UserDuplicatedException("No es posible crear un usuario con este
        userName, el userName: " + user.UserName + " ya se encuentra
        registrado.");
    }
}
```

Se muestra el problema de forma clara, utilizando nombres mnemotécnicos y agrupando en funciones de manera de manera que ninguna resulte demasiado extensa.

Capítulo 2: Nombres con sentido

Los nombres de las variables y métodos son mnemotécnicos, no se necesita un comentario que lo explique.

Ejemplo:

Clase VehicleHistoryDTO con atributos: "VehicleHistory", "InspectionHistory".

Los nombres de las variables y métodos ofrecen información al que lee el código sobre lo que las mismas hacen. No hay variables que con mismo nombre en una misma clase ya que eso podría confundir.

Los nombres que se utilizan se pueden pronunciar desde el más sencillo hasta el más complejo. Los nombres de las clases no son verbos.

Capítulo 3: Funciones

Las funciones que se utilizan tienen un tamaño en cantidad de líneas reducido.

Si bien hay funciones de 1 o 2 líneas, las mismas son para hacer más mnemotécnico su comportamiento.

Ejemplos:

```
public void AddVehicle(VehicleDTO vehicle)
{
    if (this.IsVinAvailable(vehicle.Vin) && !this.ContainsNullAttributes(vehicle))
    {
        vehicle.Status = StatusCode.InPort;
        vehicle.CurrentLocation = null;
        this.vehicleDAO.AddVehicle(vehicle);
    }
    else if(this.ContainsNullAttributes(vehicle))
    {
        throw new VehicleNullAttributesException("Faltan datos en el vehículo que
        está intentando ingresar");
    }
    else
    {
        throw new VehicleVinDuplicatedException("No es posible crear un vehiculo
        con este vin, el vin: " + vehicle.Vin + " ya se encuentra registrado.");
    }
}

public bool IsVinAvailable(string vin)
{
    VehicleDTO vehicle = this.vehicleDAO.FindVehicleByVin(vin);
    if(vehicle != null)
    {
        return false;
    } else
    {
        return true;
    }
}
```

Se utilizan excepciones como retorno en lugar de error.

Capítulo 4: Comentarios

No se incluyen comentarios en el código ya que cuando se vio la necesidad de tener ningún comentario se solucionó con nombres más mnemotécnicos o refactorizando código en los casos de que el comentario fuera por código confuso o complicado.

Capítulo 5: Formato

Para darle un buen formato al código se utilizan líneas en blanco para separar conceptos independientes. Con esto se mejora la legibilidad del código.

El ancho de las líneas es de máximo 120 caracteres de manera que la línea se pueda visualizar entera (sin tener que deslizarse) en un monitor estándar.

Capítulo 6: Objetos y estructuras de datos

Se aplica la ley de *Demeter* acerca de no hablar con desconocidos, ya que asignamos responsabilidades a las clases, utilizando interfaces, de manera de evitar conocer la estructura interna de objetos indirectos.

Ejemplo:

```
public class InspectionServiceImpl : InspectionService
{
    InspectionDAO inspectionDAO;

    public InspectionServiceImpl(InspectionDAO inspectionDAO)
    {
        this.inspectionDAO = inspectionDAO;
    }

    public Guid CreateInspection(InspectionDTO inspection)
    {
        inspectionDAO.AddInspection(inspection);

        return inspection.Id;
    }
}
```

...

No nos interesa saber el comportamiento interno de InspectionDAO y nuestros métodos no deberían variar en base a su implementación

Capítulo 7: Procesar errores

En todas las clases que se necesita devolver errores, estas tiran excepciones a la siguiente capa para transmitir esos errores, y son controladas en la capa de controlador, para poder indicarle al usuario cual fue el problema.

Ejemplo:

```
[Route("vehicles/{vin}/history")]
[HttpGet]
public HttpResponseMessage History(string vin)
{
    HttpResponseMessage response = new HttpResponseMessage();

    try
    {
        Guid token = this.GetToken();
        this.userService.GetUserLoggedIn(token);
        VehicleHistoryDTO historyDTO = this.historyService.GetHistory(vin);
        this.vehicleService.FindVehicleByVin(vin);
        response = this.Request.CreateResponse(HttpStatusCode.OK, historyDTO);
    }
    catch (VehicleNotFoundException e)
    {
        response = this.Request.CreateResponse(HttpStatusCode.BadRequest,
            e.Message);
    }
    catch (UserNotLoggedInException e)
    {
        response = this.Request.CreateResponse(HttpStatusCode.BadRequest,
            e.Message);
    }
    catch (InvalidOperationException)
    {
        string message = "No se ha enviado header de autenticación.";
        response = this.Request.CreateResponse(HttpStatusCode.BadRequest,
            message);
    }
    catch (FormatException)
    {
        string message = "El token enviado no tiene un formato valido.";
        response = this.Request.CreateResponse(HttpStatusCode.BadRequest,
            message);
    }

    return response;
}
```

}

Capítulo 8: Límites

El código de terceros utilizado, son los propios frameworks de Microsoft, como Entity Framework, Web Api, Unity, Moq, por lo que asumimos su correcto funcionamiento.

Capítulo 9: Pruebas unitarias

Se realizan pruebas unitarias a la mayor parte de las pruebas del sistema.
Las mismas cumplen con los principios F.I.R.S.T.

Son rápidas ya que no manipulamos grandes cargas de información en la Base de Datos (utilizamos Moq).

Son independientes ya que no necesitan de nada más que lo que cada prueba se crea.

Se pueden repetir y se verifica que siempre da el mismo resultado.

Se validan automáticamente ya que tienen un resultado booleano.

No llevan mucho tiempo correrlas.

Capítulo 10: Clases de tamaño reducido

No hay clases muy grandes ya que sino tendría demasiadas responsabilidades esa clase.

Es por esto que decidimos tener varios servicios, varios controladores, varios objetos de acceso a datos, cada uno con responsabilidades puntuales, siempre buscando el bajo acoplamiento y la alta cohesión de las mismas.

Capítulo 12: Limpieza

Como se puede ver en los commits hechos en el repositorio, se ha hecho limpieza de código durante todo el desarrollo, se ha refactorizado código, eliminado código duplicado, separando en funciones más chicas el código poco legible, entre otros.

Conclusión

Se codificó de manera sencilla, expresiva y fácil de leer. Los nombres de los métodos, variables y funciones son mnemotécnicos, de manera de poder deducir para qué se utiliza cada una. Además, los métodos y funciones son utilizados para un único propósito.

La mayor parte del código fue testeado a través de pruebas automáticas. Esto facilita el trabajo a la hora de refactorizar código, ya que corriendo los test automáticos se puede garantizar que el código sigue funcionando tal cual funcionaba antes de introducir los cambios.

Se utilizan interfaces para todos los servicios y clases que tienen un comportamiento el cual podría ser implementado de diferentes maneras en un futuro. Estas interfaces nos permiten desacoplar el código, quedando acoplado a interfaces y no a implementaciones particulares.

Instalacion del sistema

Para realizar la instalación y puesta en marcha del sistema hay que seguir cuatro sencillos pasos.

- Volcar el script de la base de datos (VehicleTrancking.sql) en nuestra instancia de SQL Server.

Instalación de service:

- Copiar la carpeta contenida dentro de Deploy/Services (VehicleTracking) dentro de la ruta especificada en IIS para alojar las aplicaciones (por lo general C:\inetpub\wwwroot).
- Crear la aplicación desde IIS (en el grupo de aplicaciones DefaultAppPool, seteando la ruta a donde acabamos de pegar la carpeta VehicleTracking) y modificar la Cadena de conexión, apuntándole hacia el servidor de SQL Server y configurando el usuario que tendrá los permisos necesarios sobre la base de datos VehicleTracking.

Instalación de aplicación web angular

- Copiar la carpeta contenida dentro de Deploy/Angular dentro de la ruta especificada en IIS para alojar las aplicaciones (por lo general C:\inetpub\wwwroot).
- Crear la aplicación desde IIS (en el grupo de aplicaciones DefaultAppPool, seteando la ruta a donde acabamos de pegar la carpeta Angular) y modificar el archivo api-config.json, apuntándole hacia el servicio levantado anteriormente
- Iniciar desde IIS el grupo de aplicaciones DefaultAppPool.
- Acceder desde un navegador <http://localhost/Angular>.

Nota: se asume que se parte desde el punto en el que tanto .NET Framework, SQL Server e IIS están instalados y correctamente configurados.

Instalación de aplicación de escritorio

- Se deberá configurar el connectionString que se encuentra en el archivo App.config ubicado en la carpeta Deploy/WinApp. En dicha configuración se deberá establecer la base de datos a la que hacemos referencia.
- Ejecutar el archivo VehicleTrakcin.exe ubicado en la carpeta Deploy/WinApp

Si se desea agregar nuevos mecanismos para importaciones se deberá copiar la .dll del proyecto de la implementación de dicho mecanismo dentro de la carpeta Importers ubicada en Deploy/WinApp/Importers.

Datos de prueba

A continuación se detallan los datos de prueba utilizados para comprobar el correcto funcionamiento del sistema:

Se decidió no agregar los datos de prueba para las imágenes debido a que los datos en base 64 son demasiado extensos como para mostrarlos en el documento

Usuarios:

Nombre	Apellido	Usuario	Teléfono	Clave	Rol
admin	admin	admin	1234	admin	Administrador
patio	patio	patio	1234	patio	Operario del Patio
trans	trans	trans	1234	trans	Transportista
vend	vend	vend	1234	vend	Vendedor
puerto	puerto	puerto	1234	puerto	Operario del Puerto

Roles:

Nombre
Administrador
Operario del Puerto
Transportista
Operario del Patio
Vendedor

Zonas:

Nombre	Capacidad	Tipo de SubZona	Zona padre
--------	-----------	-----------------	------------

Zona alternativa	40	-	-
Mecanica	15	Mecanica ligera	Zona gigante
Mecanica auxiliar	13	Mecanica ligera	Zona gigante
Lavado primario	20	Lavado	Zona gigante
Zona gigante	100	-	-
Lavados intensos	5	Lavado	Zona alternativa
Centro de pintura	46	Pintura	Zona gigante

Tipo de SubZona:

Nombre
Mecanica ligera
Lavado
Pintura

Items de flujo:

Número de paso	Tipo de SubZona
3	Lavado
2	Pintura
1	Mecanica ligera

Vehículos:

Marca	Modelo	Año	Color	Tipo	Vin	Estado	Precio	Lote	Zona
-------	--------	-----	-------	------	-----	--------	--------	------	------

Mitsubishi	L200	2009	Gris	Camioneta	VKQ-2412	Listo para la venta	0	Lote puro	-
Hyundai	Creta	2016	Amarillo	Auto	WOF-2491	En puerto	0	-	-
Hyundai	Creta	2011	Verde	Auto	WRI-4891	Listo para partir en puerto	0	Lote primario	-
Mercedes	Atego 1518	2001	Azul	Camion	LKD-4231	En puerto	0	-	-
Mitsubishi	L200	2008	Rojo	Camioneta	BNW-8927	Listo para ser ubicado en zona	0	Lote puro	-
Volkswagen	Bora	2012	Verde	Auto	MVO-1294	Listo para partir en puerto	0	Lote primario	-
Chevrolet	Onyx	2010	Rojo	Auto	FDK-4912	En puerto	0	-	-
Fiat	Uno	2014	Azul	Auto	ASD-1423	En puerto	0	-	-
Chevrolet	Onyx	2013	Gris	Auto	MCV-3521	Listo para partir en puerto	0	Lote primario	-
Mitsubishi	L200	2006	Amarillo	Camioneta	JKW-8926	Listo para ser ubicado en zona	0	Lote puro	-
Chevrolet	Onyx	2000	Azul	Auto	KVX-8931	En puerto	0	-	-
Hyundai	Creta	2009	Azul	Auto	MVL-3841	Vendido	1000	Lote pequeño	-
Volkswagen	Bora	2008	Rojo	Auto	NKE-2412	En puerto	0	-	-
Volkswagen	Bora	2003	Azul	Auto	MCO-8592	Vendido	928	Lote pequeño	-
Fiats	Uno	2016	Amarillo	Auto	LDF-2141	En puerto	0	-	-
Mercedes	Atego	2006	Gris	Camion	CVI-	En puerto	0	-	-

	1518				3810				
--	------	--	--	--	------	--	--	--	--

Lotes:

Nombre	Descripción	Usuario creador
Lote pequeño	Lote con dos vehiculos con daños	puerto
Lote primario	Lote con tres vehiculos dañados	puerto
Lote puro	Lote sin daños	admin

Transportes:

Fecha de Inicio	Fecha de Fin	Usuario
2017-11-20 22:44:28	2017-11-20 22:44:36	trans
2017-11-20 22:56:29	2017-11-20 22:56:38	admin

Histórico de vehículo:

Marca	Modelo	Año	Color	Tipo	Vin	Estado	Fecha
Mitsubishi	L200	2008	Rojo	Camioneta	BNW-8927	Listo para ser ubicado en zona	2017-11-20 22:57:05
Chevrolet	Onyx	2000	Azul	Auto	KVX-8931	En puerto	2017-11-20 22:28:22
Mitsubishi	L200	2008	Rojo	Camioneta	BNW-8927	En puerto	2017-11-20 22:33:18
Mitsubishi	L200	2006	Amarillo	Camioneta	JKW-8926	En transito	2017-11-20 22:56:29
Mitsubishi	L200	2009	Gris	Camioneta	VKQ-2412	Esperando por inspeccion patio	2017-11-20 22:56:38

Mitsubishi	L200	2006	Amarillo	Camioneta	JKW-8926	Listo para ser ubicado en zona	2017-11-20 22:57:11
Volkswagen	Bora	2003	Azul	Auto	MCO-8592	En puerto	2017-11-20 22:35:21
Hyundai	Creta	2009	Azul	Auto	MVL-3841	Ubicado en zona	2017-11-20 22:50:29
Volkswagen	Bora	2012	Verde	Auto	MVO-1294	En puerto	2017-11-20 22:35:44
Mitsubishi	L200	2009	Gris	Camioneta	VKQ-2412	En transito	2017-11-20 22:56:28
Hyundai	Creta	2009	Azul	Auto	MVL-3841	En transito	2017-11-20 22:44:28
Hyundai	Creta	2009	Azul	Auto	MVL-3841	Ubicado en zona	2017-11-20 22:51:02
Hyundai	Creta	2009	Azul	Auto	MVL-3841	Ubicado en zona	2017-11-20 22:49:58
Volkswagen	Bora	2003	Azul	Auto	MCO-8592	Ubicado en zona	2017-11-20 22:50:05
Volkswagen	Bora	2003	Azul	Auto	MCO-8592	En transito	2017-11-20 22:44:28
Chevrolet	Onyx	2013	Gris	Auto	MCV-3521	Listo para partir en puerto	2017-11-20 22:43:32
Volkswagen	Bora	2008	Rojo	Auto	NKE-2412	En puerto	2017-11-20 22:35:07
Hyundai	Creta	2009	Azul	Auto	MVL-3841	En puerto	2017-11-20 22:36:47
Fiat	Uno	2014	Azul	Auto	ASD-1423	En puerto	2017-11-20 22:26:50
Chevrolet	Onyx	2013	Gris	Auto	MCV-3521	Inspeccionado en puerto	2017-11-20 22:41:46

Mitsubishi	L200	2006	Amarillo	Camioneta	JKW-8926	Listo para partir en puerto	2017-11-20 22:56:14
Hyundai	Creta	2011	Verde	Auto	WRI-4891	En puerto	2017-11-20 22:36:30
Fiat	Uno	2016	Amarillo	Auto	LDF-2141	En puerto	2017-11-20 22:27:10.3943882
Mitsubishi	L200	2009	Gris	Camioneta	VKQ-2412	Inspeccionado en puerto	2017-11-20 22:55:33
Mitsubishi	L200	2009	Gris	Camioneta	VKQ-2412	Ubicado en zona	2017-11-20 22:58:07
Chevrolet	Onyx	2013	Gris	Auto	MCV-3521	En puerto	2017-11-20 22:27:37
Mitsubishi	L200	2009	Gris	Camioneta	VKQ-2412	En puerto	2017-11-20 22:32:44
Mercedes	Atego 1518	2006	Gris	Camion	CVI-3810	En puerto	2017-11-20 22:30:24
Mitsubishi	L200	2009	Gris	Camioneta	VKQ-2412	Ubicado en zona	2017-11-20 22:57:33
Volkswagen	Bora	2003	Azul	Auto	MCO-8592	Vendido	2017-11-20 22:53:17
Volkswagen	Bora	2003	Azul	Auto	MCO-8592	Esperando por inspeccion patio	2017-11-20 22:44:37
Mitsubishi	L200	2008	Rojo	Camioneta	BNW-8927	Listo para partir en puerto	2017-11-20 22:56:14
Volkswagen	Bora	2003	Azul	Auto	MCO-8592	Ubicado en zona	2017-11-20 22:51:07
Hyundai	Creta	2011	Verde	Auto	WRI-4891	Inspeccionado en puerto	2017-11-20 22:39:07
Mitsubishi	L200	2008	Rojo	Camioneta	BNW-8927	En transito	2017-11-20 22:56:28
Hyundai	Creta	2009	Azul	Auto	MVL-3841	Vendido	2017-11-20 22:53:10

Mitsubishi	L200	2006	Amarillo	Camioneta	JKW-8926	Esperando por inspeccion patio	2017-11-20 22:56:38
Hyundai	Creta	2009	Azul	Auto	MVL-3841	Listo para la venta	2017-11-20 22:51:10
Volkswagen	Bora	2003	Azul	Auto	MCO-8592	Listo para ser ubicado en zona	2017-11-20 22:47:52
Volkswagen	Bora	2003	Azul	Auto	MCO-8592	Inspeccionado en puerto	2017-11-20 22:42:32
Mercedes	Atego 1518	2001	Azul	Camion	LKD-4231	En puerto	2017-11-20 22:30:09
Mitsubishi	L200	2006	Amarillo	Camioneta	JKW-8926	Inspeccionado en puerto	2017-11-20 22:55:45
Mitsubishi	L200	2009	Gris	Camioneta	VKQ-2412	Ubicado en zona	2017-11-20 22:57:24
Volkswagen	Bora	2012	Verde	Auto	MVO-1294	Inspeccionado en puerto	2017-11-20 22:39:56
Volkswagen	Bora	2012	Verde	Auto	MVO-1294	Listo para partir en puerto	2017-11-20 22:43:32
Mitsubishi	L200	2008	Rojo	Camioneta	BNW-8927	Inspeccionado en puerto	2017-11-20 22:55:37
Hyundai	Creta	2009	Azul	Auto	MVL-3841	Inspeccionado en puerto	2017-11-20 22:40:21
Volkswagen	Bora	2003	Azul	Auto	MCO-8592	Listo para partir en puerto	2017-11-20 22:44:00
Mitsubishi	L200	2009	Gris	Camioneta	VKQ-2412	Listo para partir en puerto	2017-11-20 22:56:14
Hyundai	Creta	2009	Azul	Auto	MVL-3841	Listo para ser ubicado en zona	2017-11-20 22:46:28
Volkswagen	Bora	2003	Azul	Auto	MCO-8592	Listo para la venta	2017-11-20 22:51:12.8063953

Hyundai	Creta	2009	Azul	Auto	MVL-3841	Esperando por inspeccion patio	2017-11-20 22:44:37
Volkswagen	Bora	2003	Azul	Auto	MCO-8592	Ubicado en zona	2017-11-20 22:50:48
Mitsubishi	L200	2009	Gris	Camioneta	VKQ-2412	Listo para la venta	2017-11-20 22:58:11
Mitsubishi	L200	2009	Gris	Camioneta	VKQ-2412	Listo para ser ubicado en zona	2017-11-20 22:56:56
Mitsubishi	L200	2006	Amarillo	Camioneta	JKW-8926	En puerto	2017-11-20 22:33:01
Hyundai	Creta	2011	Verde	Auto	WRI-4891	Listo para partir en puerto	2017-11-20 22:43:32
Hyundai	Creta	2009	Azul	Auto	MVL-3841	Listo para partir en puerto	2017-11-20 22:44:00
Chevrolet	Onyx	2010	Rojo	Auto	FDK-4912	En puerto	2017-11-20 22:27:55
Hyundai	Creta	2016	Amarillo	Auto	WOF-2491	En puerto	2017-11-20 22:37:10
Mitsubishi	L200	2008	Rojo	Camioneta	BNW-8927	Esperando por inspeccion patio	2017-11-20 22:56:38

Ubicación:

Nombre
Puerto 1
Patio 1
Puerto 3
Puerto 2
Patio 2
Patio 3

Inspecciones:

Fecha	Ubicación	Usuario	Vehículo
2017-11-20 22:39:07	Puerto 2	puerto	WRI-4891
2017-11-20 22:39:56	Puerto 3	puerto	MVO-1294
2017-11-20 22:57:05	Patio 3	admin	BNW-8927
2017-11-20 22:47:52	Patio 1	patio	MCO-8592
2017-11-20 22:56:56	Patio 2	admin	VKQ-2412
2017-11-20 22:42:32	Puerto 2	puerto	MCO-8592
2017-11-20 22:57:11	Patio 2	admin	JKW-8926
2017-11-20 22:46:28	Patio 3	patio	MVL-3841
2017-11-20 22:55:45	Puerto 1	admin	JKW-8926
2017-11-20 22:55:33	Puerto 1	admin	VKQ-2412
2017-11-20 22:40:21	Puerto 1	puerto	MVL-3841
2017-11-20 22:41:46	Puerto 1	puerto	MCV-3521
2017-11-20 22:55:37	Puerto 2	admin	BNW-8927

Daños:

Descripción	Inspección
Completamente roto	Inspección 8
Pequeño daño en la zona delantera	Inspección 2
Destrozado completamente	Inspección 11
Parte trasera destrozada	Inspección 6
Daño trasero considerable	Inspección 4
Vidrios rotos	Inspección 12
Daño en la parte delantera	Inspección 1

Bibliografía

Utilizamos las siguientes fuentes para la investigación de tecnologías y patrones de diseño:

- Type-safe-enum pattern : <http://www.javacamp.org/designPattern/enum.html>
- Dao pattern :
https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm
- Dto pattern :
https://www.tutorialspoint.com/design_pattern/transfer_object_pattern.htm
- Bootstrap:
<http://getbootstrap.com/>
- Angular:
<https://angular.io/>
- Ng-bootstrap:
<https://ng-bootstrap.github.io/#/components/modal/examples>
- AspNet.Cors:
<https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/enabling-cross-origin-requests-in-web-api>
- Heurísticas de usabilidad de Nielsen
<http://www.braintive.com/10-reglas-heuristicas-de-usabilidad-de-jakob-nielsen/>