Universidad ORT Uruguay
Facultad de Ingeniería

Diseño de aplicaciones 2

Obligatorio 2

Entregado como requisito para la obtención del título de Ingeniero en Sistemas

Lucía Dabezies – 200604
Sebastián Uriarte – 194973

Docentes: Gabriel Piffaretti

2017

Índice

1. Descripción general **3**
1.1 Nuevas funcionalidades del sistema 3
1.1.1 Venta de vehículos 3
1.1.2 Definir flujo para la venta 3
1.1.3 Registro de acciones del sistema 4
1.1.4 Importación de vehículos 4

2. Descripción del diseño **5**
2.1 Diagrama de paquetes 5
2.2 Diagramas de clases 7
2.2.1 VehicleTracking_Data_Entities 7
2.2.2 VehicleTracking_Data_DataAccess 12
2.2.3 Api.Services 15
2.2.4 Web.Api 19
2.2.5 VehicleTracking_ConcreteImportingStrategies 21
2.2.6 VehicleTracking.Reflection 21
2.3 Diagramas de interacción 22
2.3.1 UsersController - AttemptToGetRegisteredUsers 22
2.3.2 VehicleServices - AttemptToAddVehicle 23
2.3.3 BaseController - ExecuteActionAndReturnOutcome 23
2.3.4 UserServices - GetRegisteredUsers 23
2.4 Diagrama de componentes 24
2.5 Diagrama de entrega 25

3. Modelo de tablas **26**

4. Justificación de diseño **27**

5. Informe de métricas **31**
5.1 Cohesión relacional 31
5.2 Abstraccion 31
5.3 Inestabilidad 31
5.4 Inestabilidad vs. abstracción 32

6. Clean code **33**
6.1 Nombres 33
6.2 Funciones 33
6.3 Comentarios 34
6.4 Formato 34
6.5 Objetos y estructura de datos 34
6.6 Procesar errores 34
6.7 Límites 35
6.8 Pruebas unitarias 35

6.9 Clases 35

7. TDD (Test Driven Development) 36

7.1 Evidencia Nº1 37

7.1.1 Etapa Red 37

7.1.2 Etapa Green 38

7.1.3 Etapa Refactor 38

7.2 Evidencia N°2 39

7.2.1 Etapa Red 39

7.2.2 Etapa Green 40

7.2.3 Etapa Refactor 41

8. Cobertura de pruebas 41

9. Evaluación del proyecto 42

1. Descripción general

La aplicación representa un sistema de gestión de flujo para empresas automovilísticas. La misma permite a los usuarios hacer un seguimiento de los vehículos desde que llegan a un puerto hasta que están guardados para luego ser vendidos.

Permite registrar distintos tipos de usuarios los cuales pueden realizar diferentes acciones dependiendo de su rol. Los roles que existían en la versión anterior son administrador, operario de puerto, transportista y operario de patio, y en esta nueva versión se agrega un nuevo rol, vendedor.

Ahora se cuenta con dos aplicaciones, una windows desktop application a la cual solo pueden acceder los usuarios que sean de tipo administrador y una aplicación Angular (SAP) a la cual pueden acceder todos los usuarios.

La idea es que la desktop application sea utilizada para administrar los usuarios, vehículos, zonas, subzonas y ver los logs, así como definir el flujo que deben seguir los vehículos para luego poder ser vendidos mientras que en la aplicación Angular se realice el resto de las funcionalidades del sistema.

En esta entrega las funcionalidades pedidas fueron implementadas en su totalidad y no hay ningún bug que sea conocido.

1.1 Nuevas funcionalidades del sistema

En esta versión se han añadido nuevas funcionalidades. A continuación se listarán y se detallará cada una de ellas.

1.1.1 Venta de vehículos

En esta solución, el alcance del sistema es mayor que en la versión anterior. Ahora se puede definir si un vehículo está pronto para ser vendido, para que luego un usuario de tipo vendedor efectúe la venta. En el caso de que se realice la misma, el sistema almacenará los datos de la persona que compró dicho vehículo y el precio del mismo para llevar un control de las ventas.

La venta de vehículos se realizará a través de la aplicación Angular y sólo por usuarios que tengan rol de administrador o de vendedor.

1.1.2 Definir flujo para la venta

Para que los vehículos puedan ser vendidos, previamente deben pasar por una serie de subzonas. Es por esto que se podrá definir un flujo conformado por

una cantidad ilimitada de subzonas las cuales el vehículo tiene que atravesar en el orden definido para que pueda ser vendido.

La definición de dicho flujo solo la podrán realizar los usuarios que sean de tipo administrador y desde la windows form application.

1.1.3 Registro de acciones del sistema

El sistema registrará automáticamente un log para algunas de las funcionalidades que presenta el sistema. Se guardarán los inicios de sesión a ambas aplicaciones y las importaciones de vehículos.

Luego, los usuarios que tengan el rol de administrador podrán, desde la windows form application, acceder a los registros de dichos logs.

1.1.4 Importación de vehículos

Se le permitirá a los usuarios la importación de vehículos a través de diferentes formatos. La aplicación soporta archivos de tipo XML y JSON. Sin embargo, existe la posibilidad de cargar una librería que contengan estrategias de cargado que soportan otros tipos de archivos.

Una vez que el usuario cargó la librería, el sistema soportará la importación de vehículos a través de archivos que sean del tipo soportado por la misma.

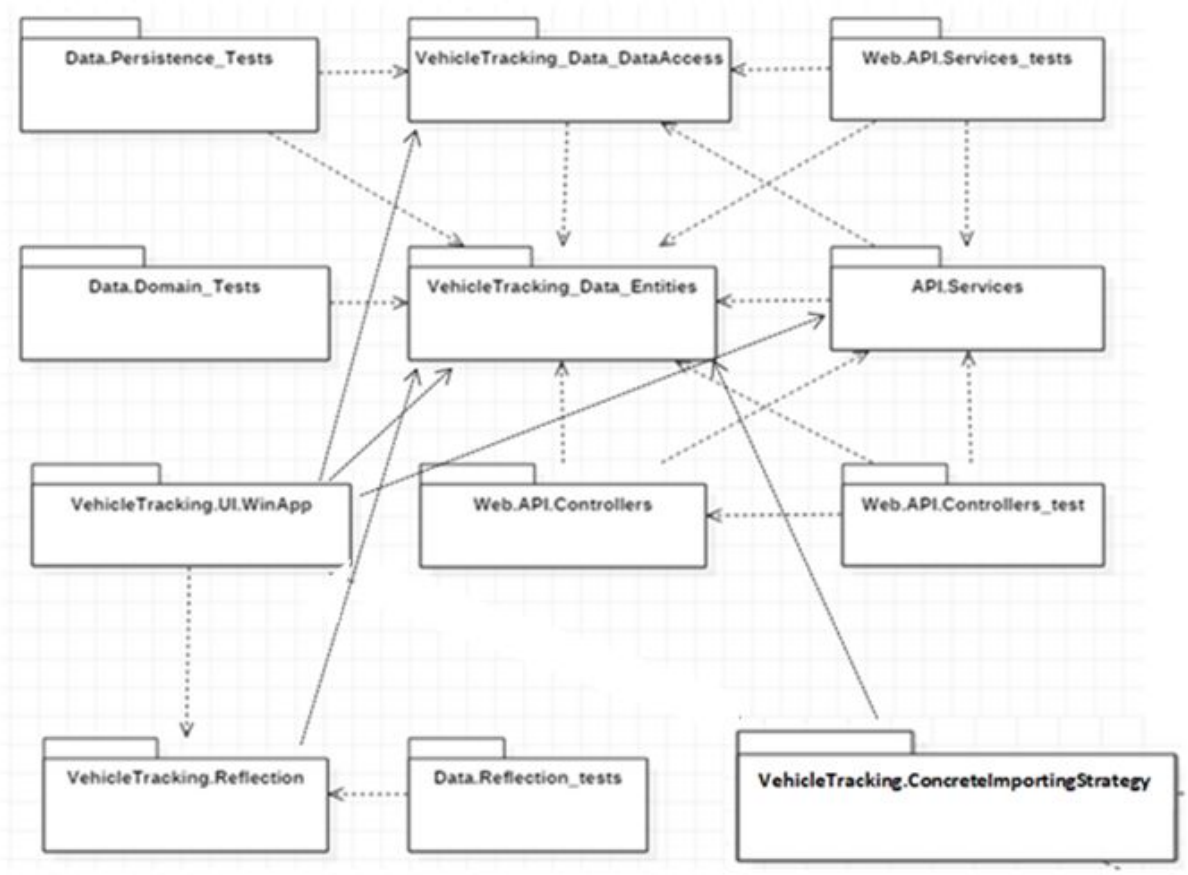
Todo esto se podrá realizar a través de la windows form application por lo que solo usuarios que sean administradores podrán tener acceso a esta nueva funcionalidad.

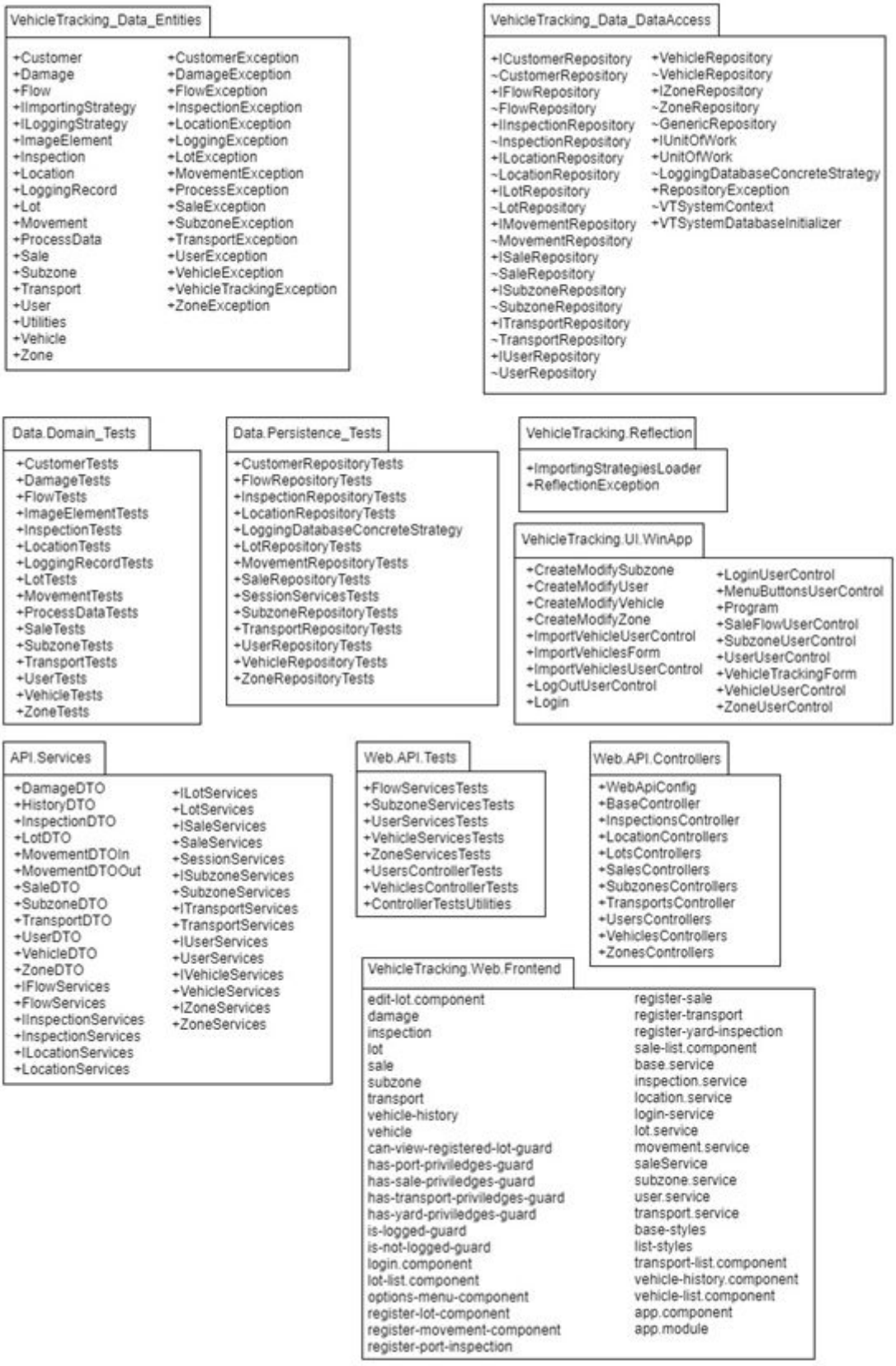
2. Descripción del diseño

Para mostrar la forma en la que fue diseñada la solución introduciremos una serie de diagramas UML.

2.1 Diagrama de paquetes

Para que los diagramas sean más fáciles de leer tomamos la decisión de, al igual que hicimos con los diagramas de clases el obligatorio pasado, mostrar las relaciones entre los paquetes sin las clases que están contenidas en ellos y luego mostrar los paquetes por separado con sus correspondientes clases.




VehicleTracking_Data_Entities
+Customer
+Damage
+Flow
+IImportingStrategy
+ILoggingStrategy
+ImageElement
+Inspection
+Location
+LoggingRecord
+Lot
+Movement
+ProcessData
+Sale
+Subzone
+Transport
+User
+Utilities
+Vehicle
+Zone
+CustomerException
+DamageException
+FlowException
+InspectionException
+LocationException
+LoggingException
+LotException
+MovementException
+ProcessException
+SaleException
+SubzoneException
+TransportException
+UserException
+VehicleException
+VehicleTrackingException
+ZoneException
VehicleTracking_Data_DataAccess
+ICustomerRepository
~CustomerRepository
+IFlowRepository
~FlowRepository
+IInspectionRepository
~InspectionRepository
+ILocationRepository
~LocationRepository
+ILotRepository
~LotRepository
+IMovementRepository
~MovementRepository
+ISaleRepository
~SaleRepository
+ISubzoneRepository
~SubzoneRepository
+ITransportRepository
~TransportRepository
+IUserRepository
~UserRepository
+VehicleRepository
~VehicleRepository
+IZoneRepository
~ZoneRepository
~GenericRepository
+IUnitOfWork
+UnitOfWork
~LoggingDatabaseConcreteStrategy
+RepositoryException
~VTSystemContext
+VTSystemDatabaseInitializer
Data.Domain_Tests
+CustomerTests
+DamageTests
+FlowTests
+ImageElementTests
+InspectionTests
+LocationTests
+LoggingRecordTests
+LotTests
+MovementTests
+ProcessDataTests
+SaleTests
+SubzoneTests
+TransportTests
+UserTests
+VehicleTests
+ZoneTests
Data.Persistence_Tests
+CustomerRepositoryTests
+FlowRepositoryTests
+InspectionRepositoryTests
+LocationRepositoryTests
+LoggingDatabaseConcreteStrategy
+LotRepositoryTests
+MovementRepositoryTests
+SaleRepositoryTests
+SessionServicesTests
+SubzoneRepositoryTests
+TransportRepositoryTests
+UserRepositoryTests
+VehicleRepositoryTests
+ZoneRepositoryTests
VehicleTracking.Reflection
+ImportingStrategiesLoader
+ReflectionException
VehicleTracking.UI.WinApp
+CreateModifySubzone
+CreateModifyUser
+CreateModifyVehicle
+CreateModifyZone
+ImportVehicleUserControl
+ImportVehiclesForm
+ImportVehiclesUserControl
+LogOutUserControl
+Login
+LoginUserControl
+MenuButtonsUserControl
+Program
+SaleFlowUserControl
+SubzoneUserControl
+UserUserControl
+VehicleTrackingForm
+VehicleUserControl
+ZoneUserControl
API.Services
+DamageDTO
+HistoryDTO
+InspectionDTO
+LotDTO
+MovementDTOIn
+MovementDTOOut
+SaleDTO
+SubzoneDTO
+TransportDTO
+UserDTO
+VehicleDTO
+ZoneDTO
+IFlowServices
+FlowServices
+IInspectionServices
+InspectionServices
+ILocationServices
+LocationServices
+ILotServices
+LotServices
+ISaleServices
+SaleServices
+SessionServices
+ISubzoneServices
+SubzoneServices
+ITransportServices
+TransportServices
+IUserServices
+UserServices
+IVehicleServices
+VehicleServices
+IZoneServices
+ZoneServices
Web.API.Tests
+FlowServicesTests
+SubzoneServicesTests
+UserServicesTests
+VehicleServicesTests
+ZoneServicesTests
+UsersControllerTests
+VehiclesControllerTests
+ControllerTestsUtilities
Web.API.Controllers
+WebApiConfig
+BaseController
+InspectionsController
+LocationControllers
+LotsControllers
+SalesControllers
+SubzonesControllers
+TransportsController
+UsersControllers
+VehiclesControllers
+ZonesControllers
VehicleTracking.Web.Frontend
edit-lot.component
damage
inspection
lot
sale
subzone
transport
vehicle-history
vehicle
can-view-registered-lot-guard
has-port-priviledges-guard
has-sale-priviledges-guard
has-transport-priviledges-guard
has-yard-priviledges-guard
is-logged-guard
is-not-logged-guard
login.component
lot-list.component
options-menu-component
register-lot-component
register-movement-component
register-port-inspection
register-sale
register-transport
register-yard-inspection
sale-list.component
base.service
inspection.service
location.service
login-service
lot.service
movement.service
saleService
subzone.service
user.service
transport.service
base-styles
list-styles
transport-list.component
vehicle-history.component
vehicle-list.component
app.component
app.module

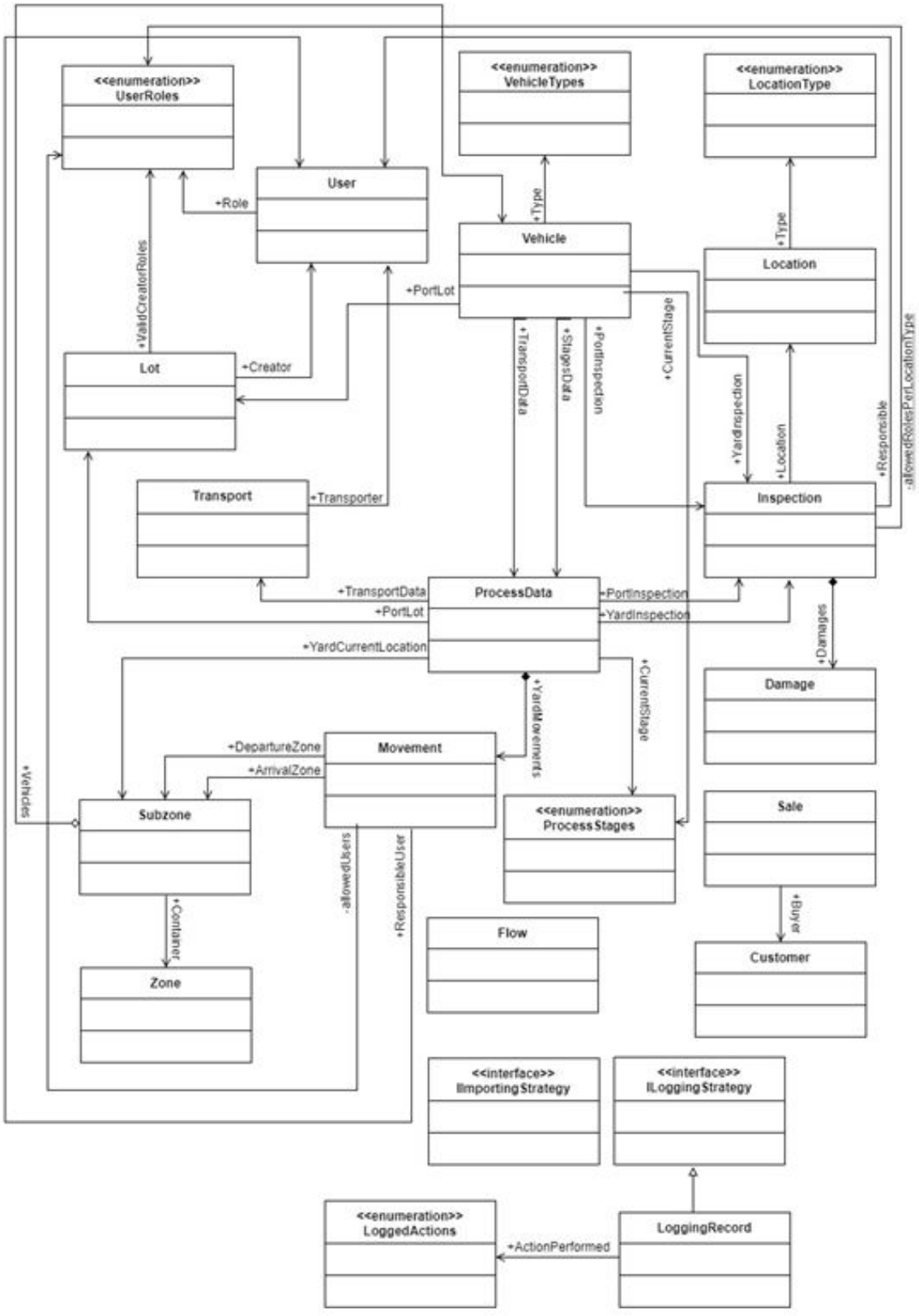
2.2 Diagramas de clases

Decidimos que los diagramas de clases que representan los paquetes que contienen las pruebas no eran relevantes al caso y optamos por no incluirlos.

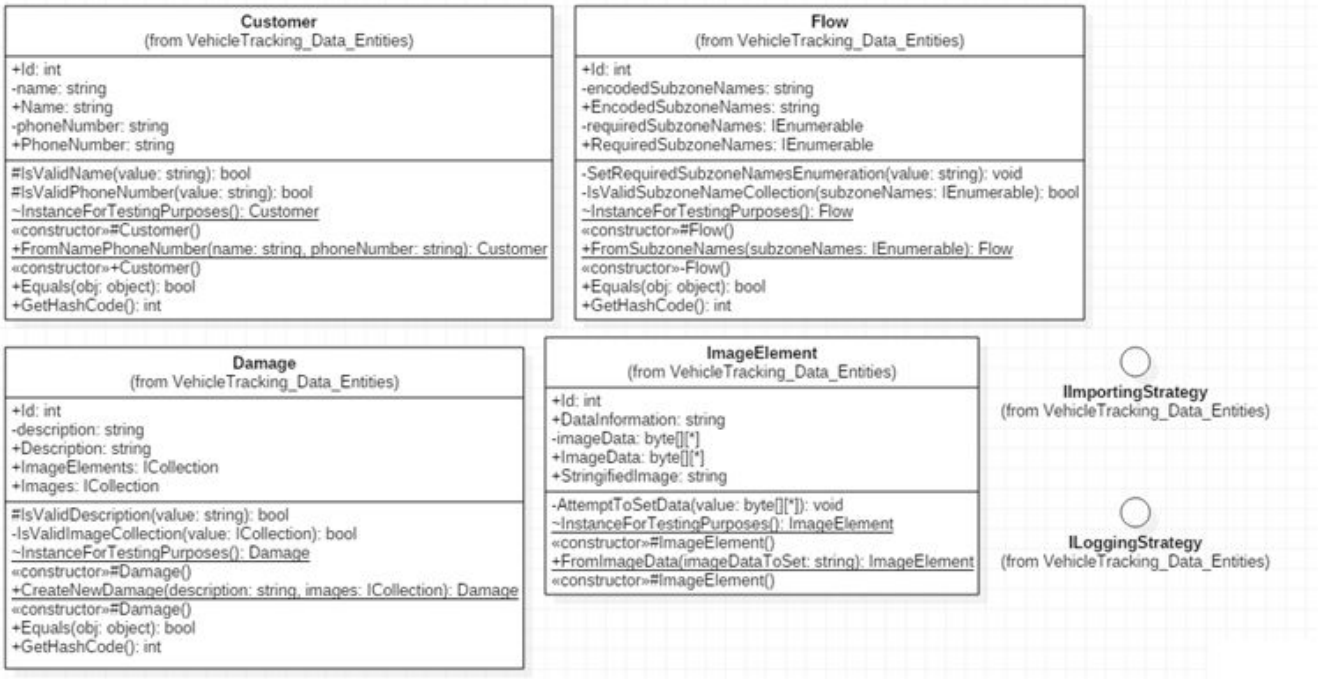
2.2.1 VehicleTracking_Data_Entities

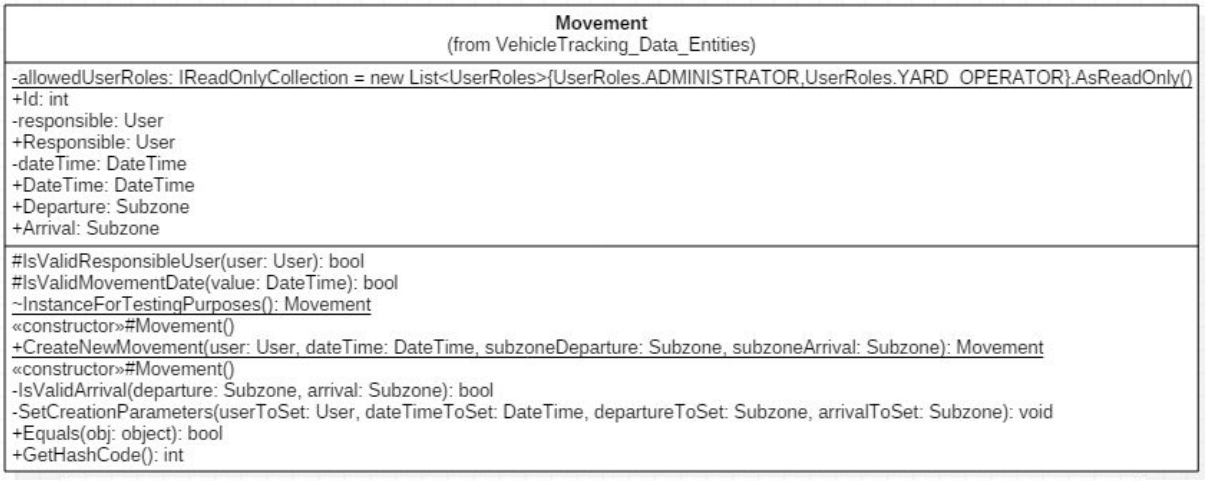
Para el diagrama de clases de este paquete decidimos mostrar las clases con sus relaciones por un lado y luego cada clase por separado con sus propiedades y métodos correspondientes. Se optó por esto ya que si lo realizamos todo en uno, el mismo no entra en una hoja obteniendo como resultado un diagrama confuso.

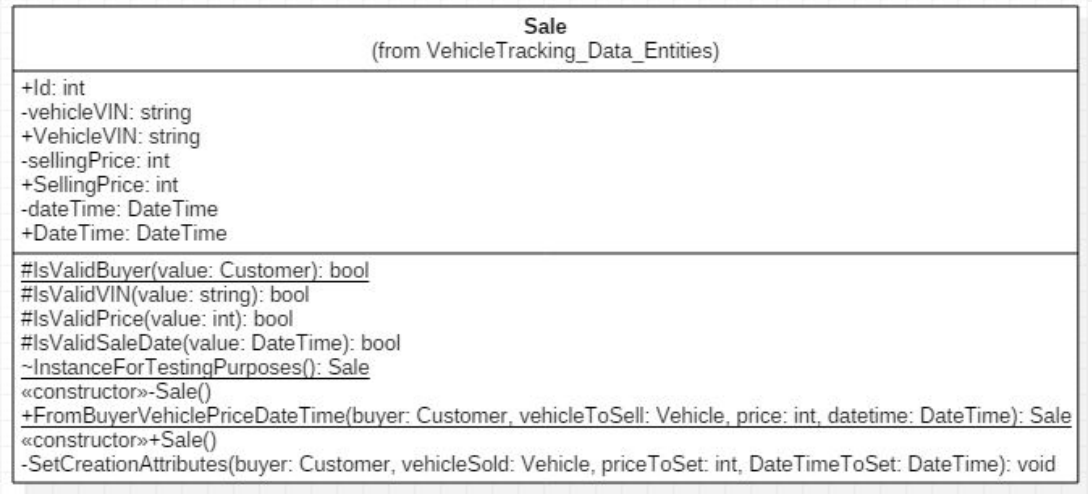
Asimismo, en el diagrama con las relaciones no incluimos las clase Utilities y las clases de excepciones ya que como todas las demás clases dependen de ellas el esquema iba a quedar con muchas flechas solapadas haciendo la lectura del mismo casi imposible.

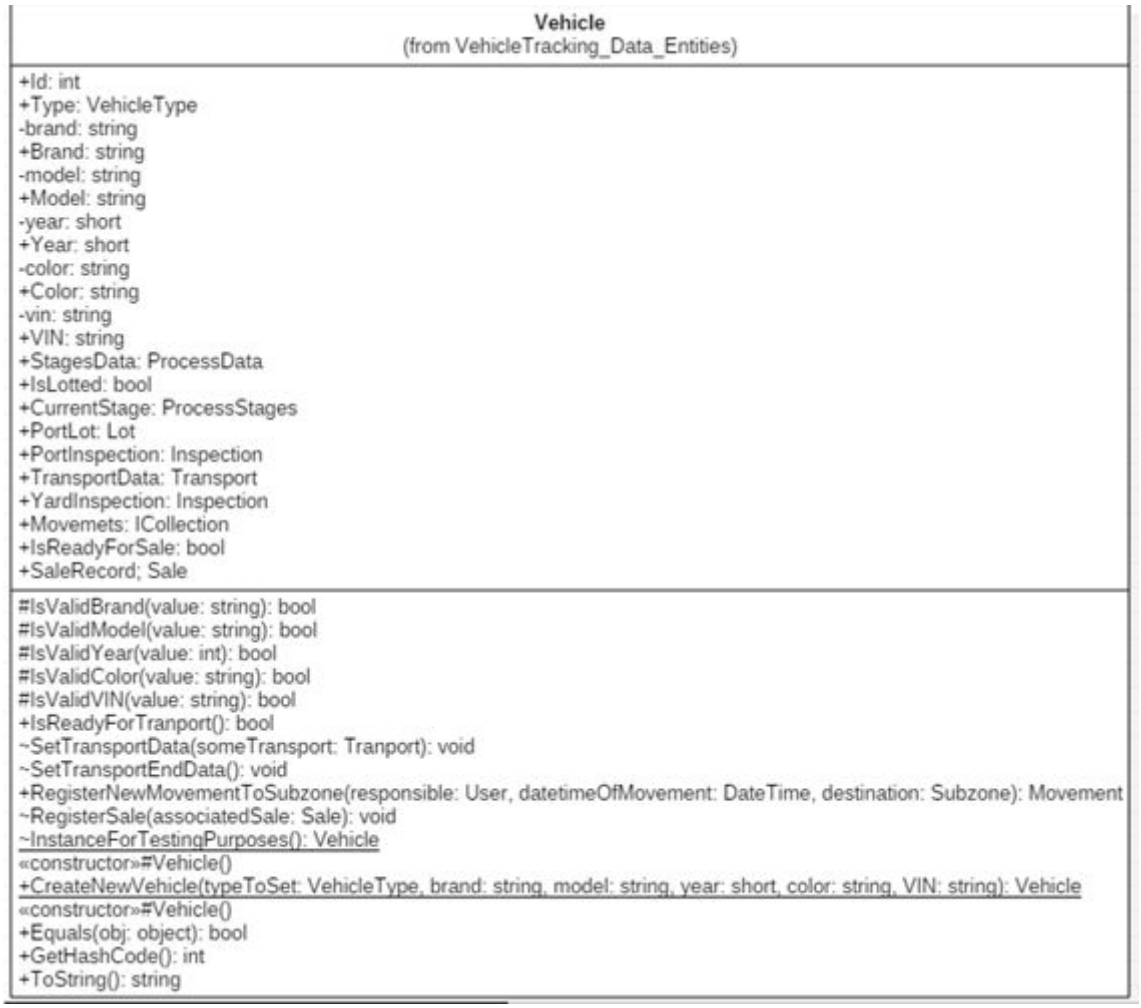

<<enumeration>>
UserRoles
<<enumeration>>
VehicleTypes
<<enumeration>>
LocationType
User
+Role
+Type
+Type
Vehicle
Location
+PortLot
+ValidCreatorRoles
+TransportData
+StageDate
+PortInspection
+CurrentStage
+YardInspection
+Location
+Responsible
-allowedRolesPerLocationType
Lot
+Creator
Transport
+Transporter
Inspection
+TransportData
ProcessData
+PortInspection
+PortLot
+YardInspection
+YardCurrentLocation
+Damages
Damage
+YardMovements
+CurrentStage
+DepartureZone
+ArrivalZone
Movement
+Vehicles
Subzone
<<enumeration>>
ProcessStages
Sale
-allowedUsers
+ResponsibleUser
+Container
+Buyer
Flow
Customer
Zone
<<interface>>
IImportingStrategy
<<interface>>
ILoggingStrategy
<<enumeration>>
LoggedActions
+ActionPerformed
LoggingRecord

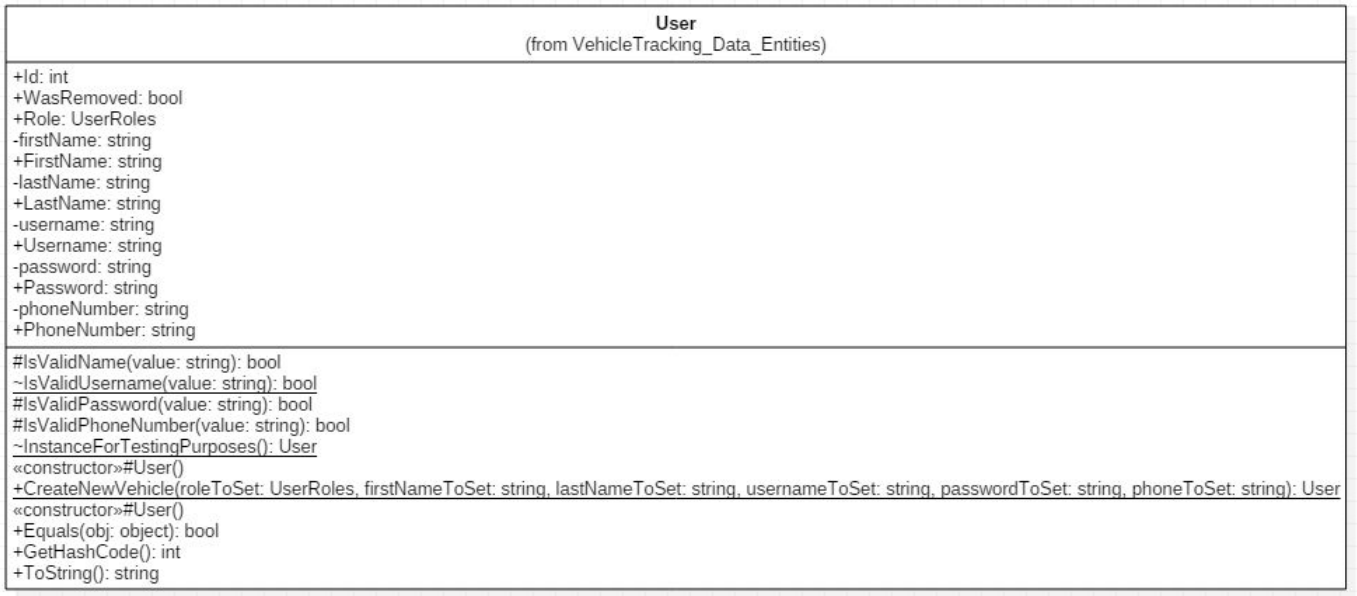
A continuación se muestra cada clase del paquete en cuestión por separado, con sus correspondientes métodos y propiedades:

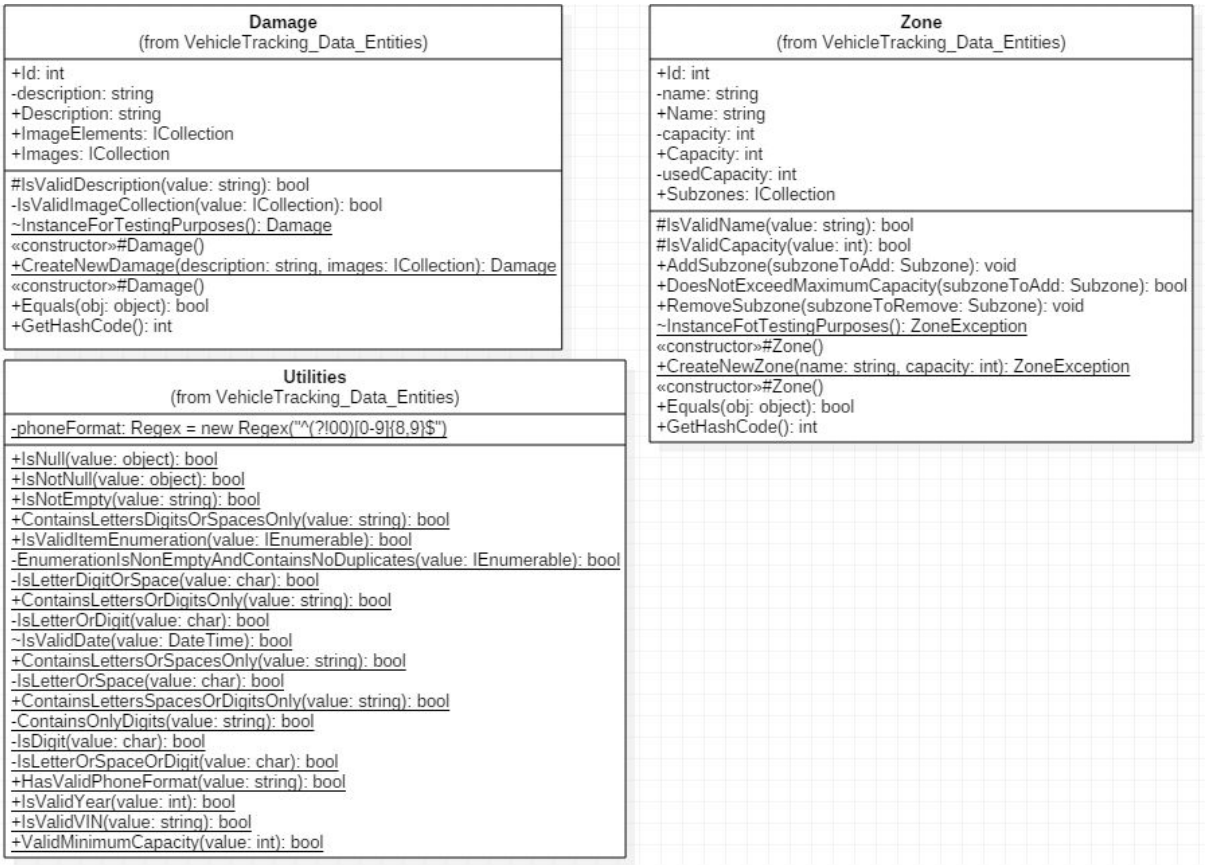


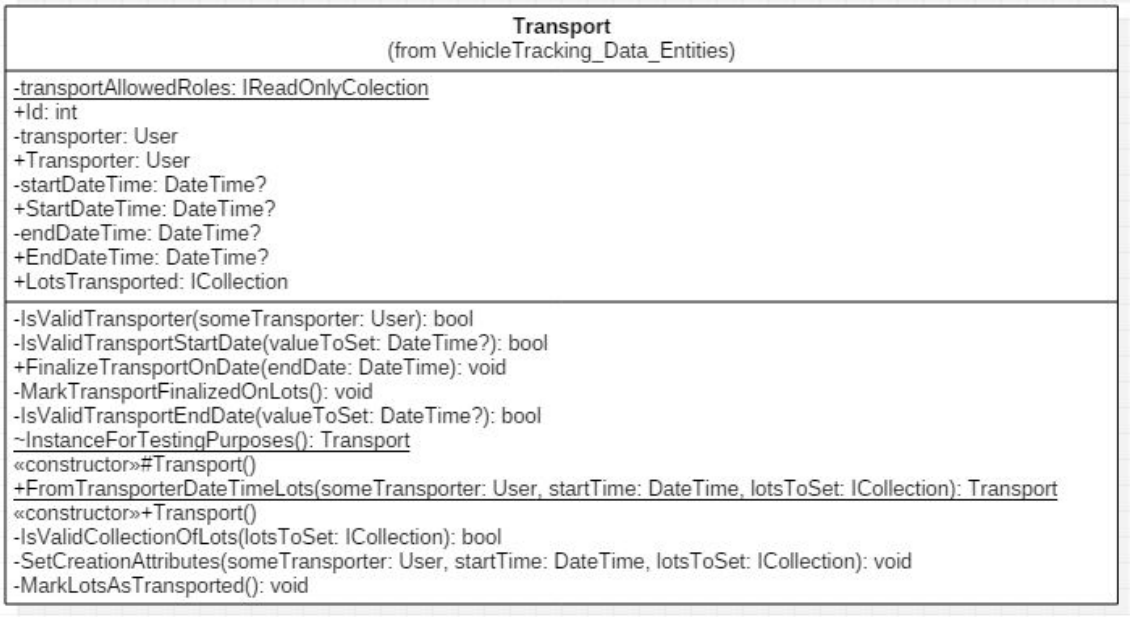




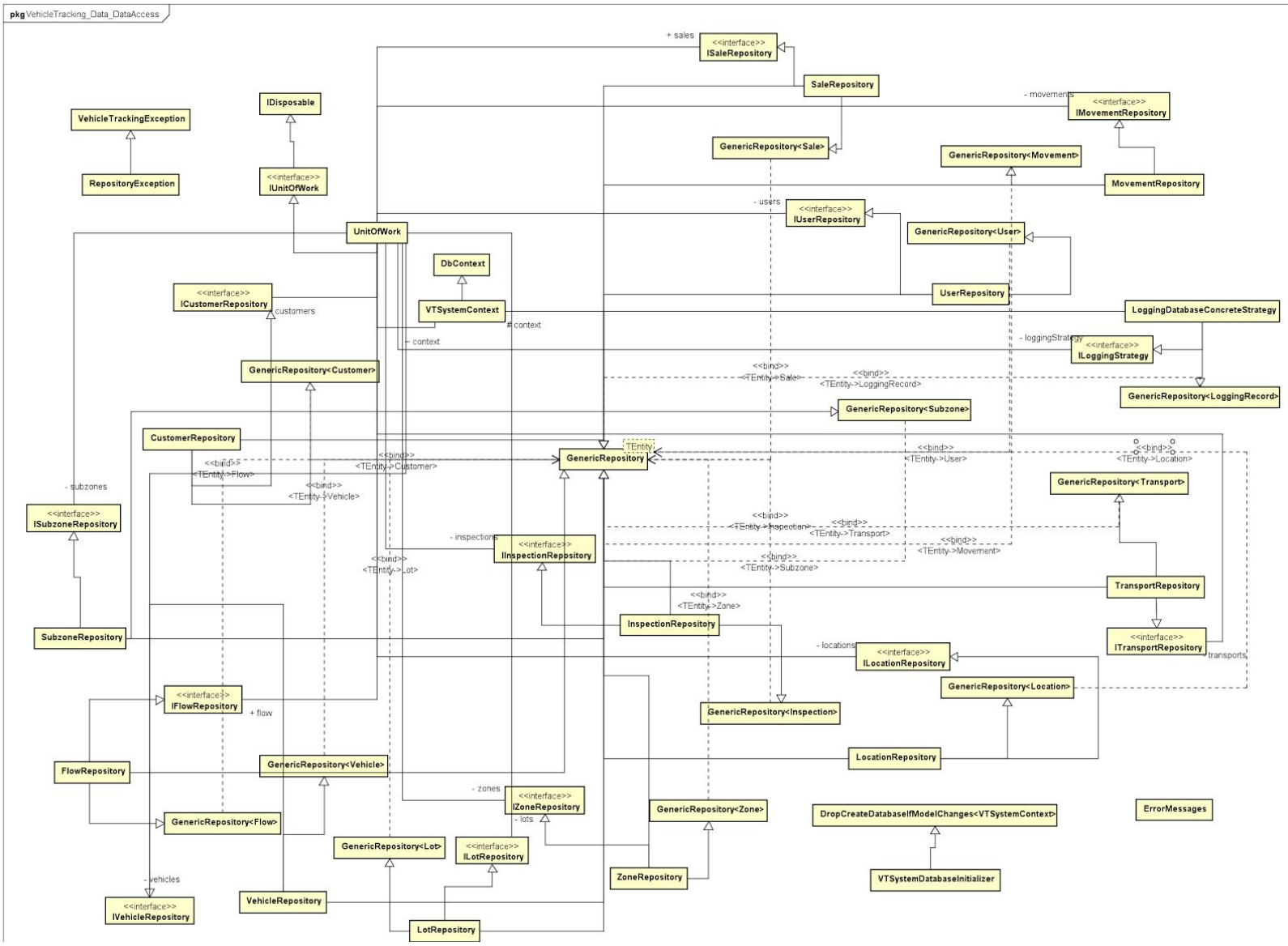

Vehicle
(from VehicleTracking_Data_Entities)
+Id: int
+Type: VehicleType
-brand: string
+Brand: string
-model: string
+Model: string
-year: short
+Year: short
-color: string
+Color: string
-vin: string
+VIN: string
+StagesData: ProcessData
+IsLotted: bool
+CurrentStage: ProcessStages
+PortLot: Lot
+PortInspection: Inspection
+TransportData: Transport
+YardInspection: Inspection
+Movemets: ICollection
+IsReadyForSale: bool
+SaleRecord; Sale
#IsValidBrand(value: string): bool
#IsValidModel(value: string): bool
#IsValidYear(value: int): bool
#IsValidColor(value: string): bool
#IsValidVIN(value: string): bool
+IsReadyForTranport(): bool
~SetTransportData(someTransport: Tranport): void
~SetTransportEndData(): void
+RegisterNewMovementToSubzone(responsible: User, datetimeOfMovement: DateTime, destination: Subzone): Movement
~RegisterSale(associatedSale: Sale): void
~InstanceForTestingPurposes(): Vehicle
«constructor»#Vehicle()
+CreateNewVehicle(typeToSet: VehicleType, brand: string, model: string, year: short, color: string, VIN: string): Vehicle
«constructor»#Vehicle()
+Equals(obj: object): bool
+GetHashCode(): int
+ToString(): string

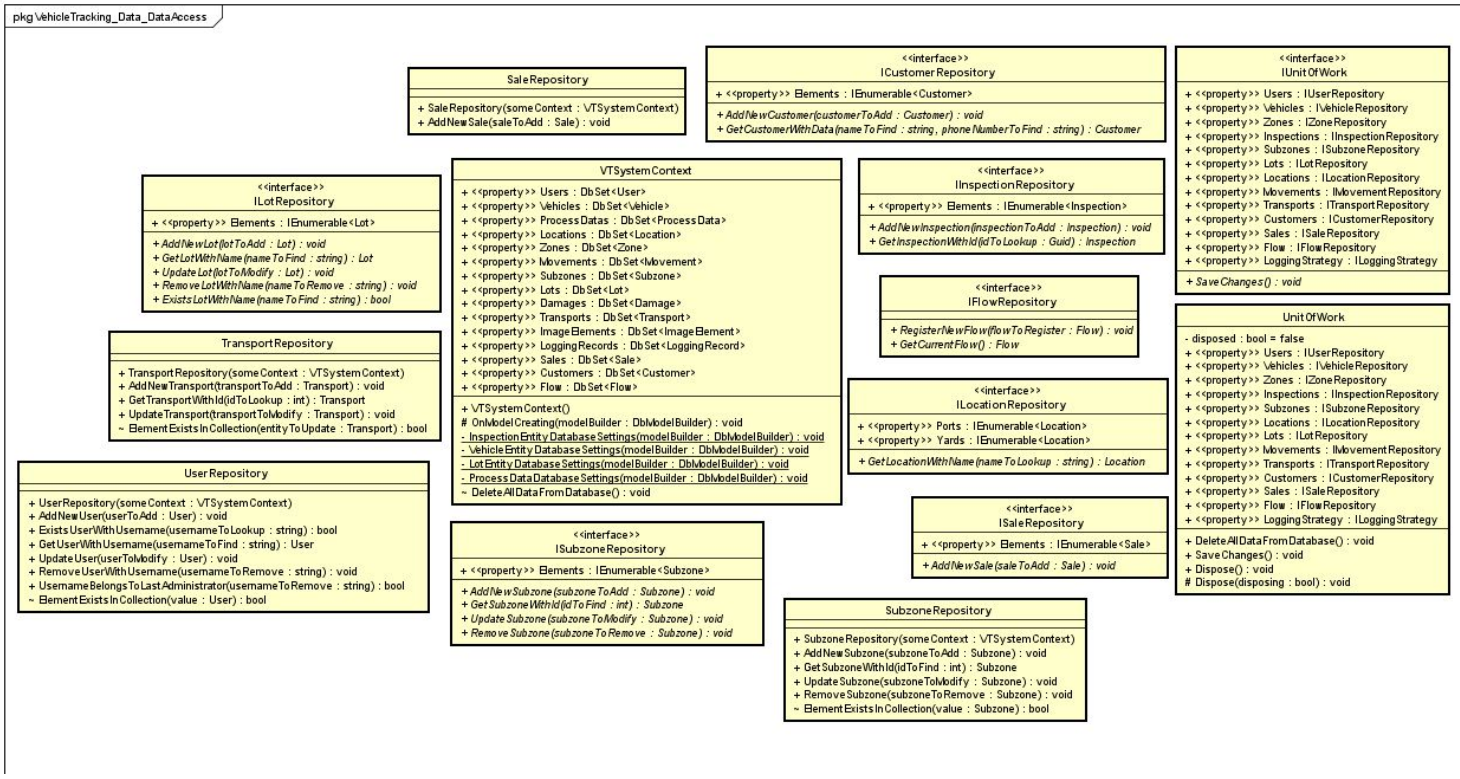

User
(from VehicleTracking_Data_Entities)
+Id: int
+WasRemoved: bool
+Role: UserRoles
-firstName: string
+FirstName: string
-lastName: string
+LastName: string
-username: string
+Username: string
-password: string
+Password: string
-phoneNumber: string
+PhoneNumber: string
#IsValidName(value: string): bool
~IsValidUsername(value: string): bool
#IsValidPassword(value: string): bool
#IsValidPhoneNumber(value: string): bool
~InstanceForTestingPurposes(): User
«constructor»#User()
+CreateNewVehicle(roleToSet: UserRoles, firstNameToSet: string, lastNameToSet: string, usernameToSet: string, passwordToSet: string, phoneToSet: string): User
«constructor»#User()
+Equals(obj: object): bool
+GetHashCode(): int
+ToString(): string

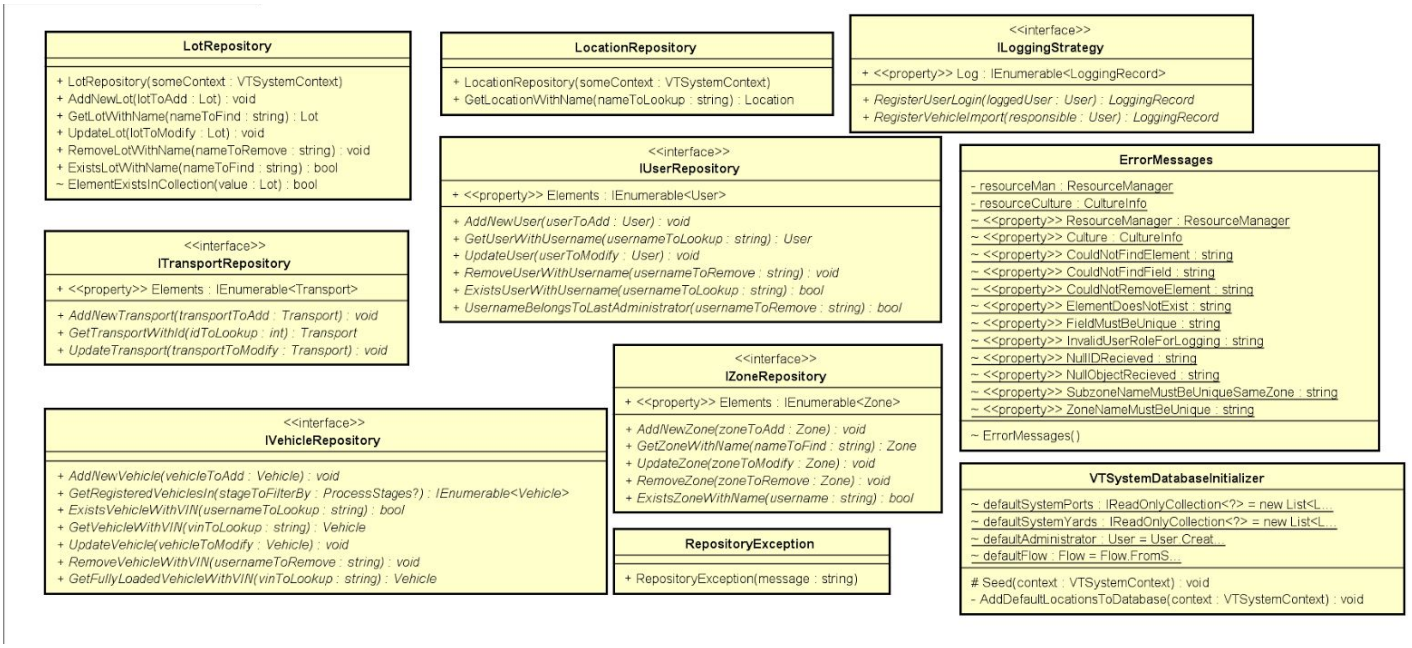

Damage
(from VehicleTracking_Data_Entities)
+Id: int
-description: string
+Description: string
+ImageElements: ICollection
+Images: ICollection
#IsValidDescription(value: string): bool
-IsValidImageCollection(value: ICollection): bool
~InstanceForTestingPurposes(): Damage
«constructor»#Damage()
+CreateNewDamage(description: string, images: ICollection): Damage
«constructor»#Damage()
+Equals(obj: object): bool
+GetHashCode(): int
Zone
(from VehicleTracking_Data_Entities)
+Id: int
-name: string
+Name: string
-capacity: int
+Capacity: int
-usedCapacity: int
+Subzones: ICollection
#IsValidName(value: string): bool
#IsValidCapacity(value: int): bool
+AddSubzone(subzoneToAdd: Subzone): void
+DoesNotExceedMaximumCapacity(subzoneToAdd: Subzone): bool
+RemoveSubzone(subzoneToRemove: Subzone): void
~InstanceFotTestingPurposes(): ZoneException
«constructor»#Zone()
+CreateNewZone(name: string, capacity: int): ZoneException
«constructor»#Zone()
+Equals(obj: object): bool
+GetHashCode(): int
Utilities
(from VehicleTracking_Data_Entities)
-phoneFormat: Regex = new Regex("^(?!00)[0-9]{8,9}$")
+IsNull(value: object): bool
+IsNotNull(value: object): bool
+IsNotEmpty(value: string): bool
+ContainsLettersDigitsOrSpacesOnly(value: string): bool
+IsValidItemEnumeration(value: IEnumerable): bool
-EnumerationIsNonEmptyAndContainsNoDuplicates(value: IEnumerable): bool
-IsLetterDigitOrSpace(value: char): bool
+ContainsLettersOrDigitsOnly(value: string): bool
-IsLetterOrDigit(value: char): bool
~IsValidDate(value: DateTime): bool
+ContainsLettersOrSpacesOnly(value: string): bool
-IsLetterOrSpace(value: char): bool
+ContainsLettersSpacesOrDigitsOnly(value: string): bool
-ContainsOnlyDigits(value: string): bool
-IsDigit(value: char): bool
-IsLetterOrSpaceOrDigit(value: char): bool
+HasValidPhoneFormat(value: string): bool
+IsValidYear(value: int): bool
+IsValidVIN(value: string): bool
+ValidMinimumCapacity(value: int): bool

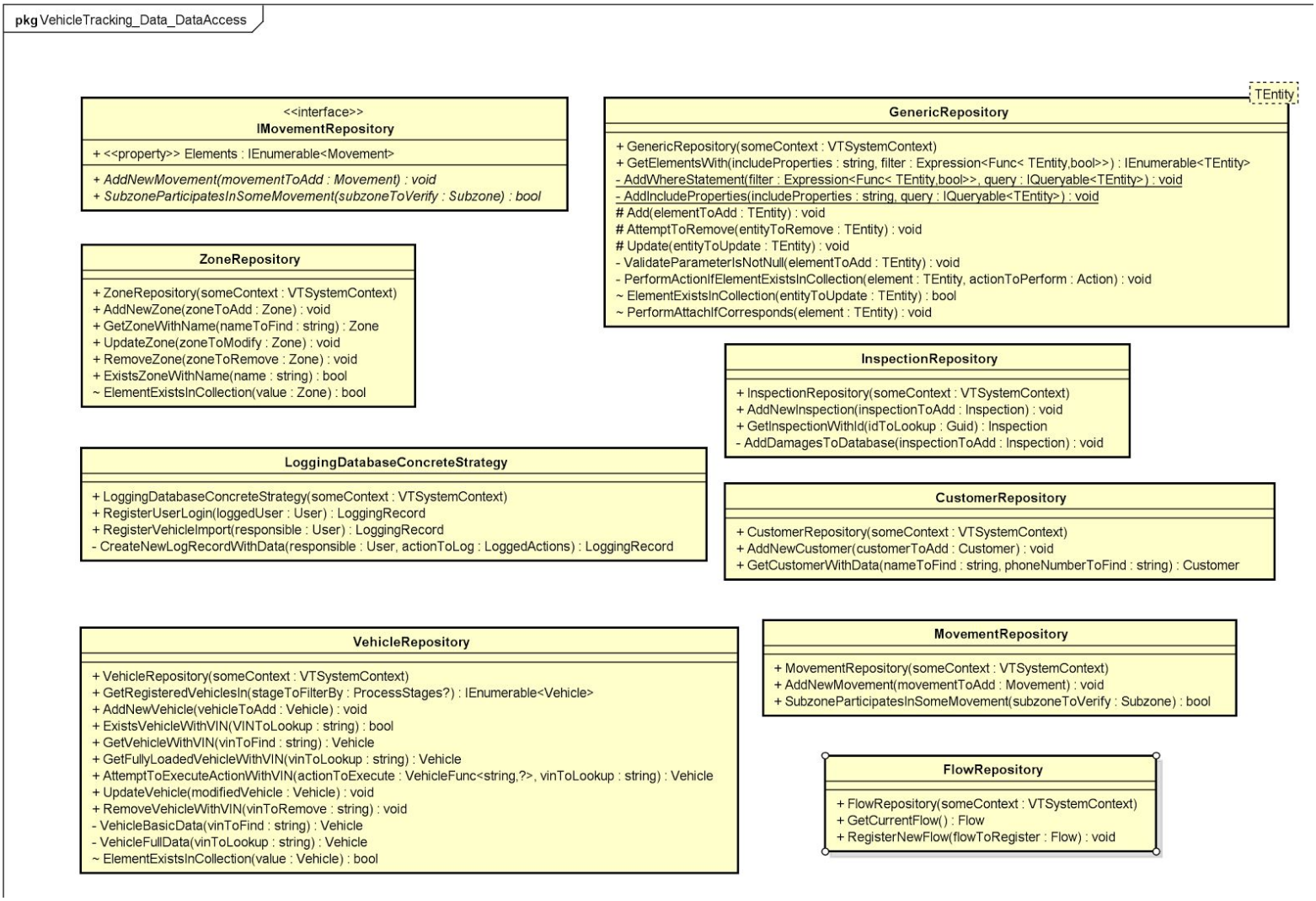

Transport
(from VehicleTracking_Data_Entities)
-transportAllowedRoles: IReadOnlyColection
+Id: int
-transporter: User
+Transporter: User
-startDateTime: DateTime?
+StartDateTime: DateTime?
-endDateTime: DateTime?
+EndDateTime: DateTime?
+LotsTransported: ICollection
-IsValidTransporter(someTransporter: User): bool
-IsValidTransportStartDate(valueToSet: DateTime?): bool
+FinalizeTransportOnDate(endDate: DateTime): void
-MarkTransportFinalizedOnLots(): void
-IsValidTransportEndDate(valueToSet: DateTime?): bool
~InstanceForTestingPurposes(): Transport
«constructor»#Transport()
+FromTransporterDateTimeLots(someTransporter: User, startTime: DateTime, lotsToSet: ICollection): Transport
«constructor»+Transport()
-IsValidCollectionOfLots(lotsToSet: ICollection): bool
-SetCreationAttributes(someTransporter: User, startTime: DateTime, lotsToSet: ICollection): void
-MarkLotsAsTransported(): void

2.2.2 VehicleTracking_Data_DataAccess




pkg VehicleTracking_Data_DataAccess
SaleRepository
+ SaleRepository(someContext : VTSystemContext)
+ AddNewSale(saleToAdd : Sale) : void
<<interface>>
ICustomerRepository
+ <<property>> Elements : IEnumerable<Customer>
+ AddNewCustomer(customerToAdd : Customer) : void
+ GetCustomerWithData(nameToFind : string, phoneNumberToFind : string) : Customer
<<interface>>
IUnitOfWork
+ <<property>> Users : IUserRepository
+ <<property>> Vehicles : IVehicleRepository
+ <<property>> Zones : IZoneRepository
+ <<property>> Inspections : IInspectionRepository
+ <<property>> Subzones : ISubzoneRepository
+ <<property>> Lots : ILotRepository
+ <<property>> Locations : ILocationRepository
+ <<property>> Movements : IMovementRepository
+ <<property>> Transports : ITransportRepository
+ <<property>> Customers : ICustomerRepository
+ <<property>> Sales : ISaleRepository
+ <<property>> Flow : IFlowRepository
+ <<property>> LoggingStrategy : ILoggingStrategy
+ SaveChanges() : void
<<interface>>
ILotRepository
+ <<property>> Elements : IEnumerable<Lot>
+ AddNewLot(lotToAdd : Lot) : void
+ GetLotWithName(nameToFind : string) : Lot
+ UpdateLot(lotToModify : Lot) : void
+ RemoveLotWithName(nameToRemove : string) : void
+ ExistsLotWithName(nameToFind : string) : bool
VTSystemContext
+ <<property>> Users : DbSet<User>
+ <<property>> Vehicles : DbSet<Vehicle>
+ <<property>> ProcessDatas : DbSet<ProcessData>
+ <<property>> Locations : DbSet<Location>
+ <<property>> Zones : DbSet<Zone>
+ <<property>> Movements : DbSet<Movement>
+ <<property>> Subzones : DbSet<Subzone>
+ <<property>> Lots : DbSet<Lot>
+ <<property>> Damages : DbSet<Damage>
+ <<property>> Transports : DbSet<Transport>
+ <<property>> ImageElements : DbSet<ImageElement>
+ <<property>> LoggingRecords : DbSet<LoggingRecord>
+ <<property>> Sales : DbSet<Sale>
+ <<property>> Customers : DbSet<Customer>
+ <<property>> Flow : DbSet<Flow>
+ VTSystemContext()
# OnModelCreating(modelBuilder : DbModelBuilder) : void
- InspectionEntityDatabaseSettings(modelBuilder : DbModelBuilder) : void
- VehicleEntityDatabaseSettings(modelBuilder : DbModelBuilder) : void
- LotEntityDatabaseSettings(modelBuilder : DbModelBuilder) : void
- ProcessDataDatabaseSettings(modelBuilder : DbModelBuilder) : void
~ DeleteAllDataFromDatabase() : void
<<interface>>
IInspectionRepository
+ <<property>> Elements : IEnumerable<Inspection>
+ AddNewInspection(inspectionToAdd : Inspection) : void
+ GetInspectionWithId(idToLookup : Guid) : Inspection
<<interface>>
IFlowRepository
+ RegisterNewFlow(flowToRegister : Flow) : void
+ GetCurrentFlow() : Flow
Unit Of Work
- disposed : bool = false
+ <<property>> Users : IUserRepository
+ <<property>> Vehicles : IVehicleRepository
+ <<property>> Zones : IZoneRepository
+ <<property>> Inspections : IInspectionRepository
+ <<property>> Subzones : ISubzoneRepository
+ <<property>> Locations : ILocationRepository
+ <<property>> Lots : ILotRepository
+ <<property>> Movements : IMovementRepository
+ <<property>> Transports : ITransportRepository
+ <<property>> Customers : ICustomerRepository
+ <<property>> Sales : ISaleRepository
+ <<property>> Flow : IFlowRepository
+ <<property>> LoggingStrategy : ILoggingStrategy
+ DeleteAllDataFromDatabase() : void
+ SaveChanges() : void
+ Dispose() : void
# Dispose(disposing : bool) : void
TransportRepository
+ TransportRepository(someContext : VTSystemContext)
+ AddNewTransport(transportToAdd : Transport) : void
+ GetTransportWithId(idToLookup : int) : Transport
+ UpdateTransport(transportToModify : Transport) : void
~ ElementExistsInCollection(entityToUpdate : Transport) : bool
<<interface>>
ILocationRepository
+ <<property>> Ports : IEnumerable<Location>
+ <<property>> Yards : IEnumerable<Location>
+ GetLocationWithName(nameToLookup : string) : Location
UserRepository
+ UserRepository(someContext : VTSystemContext)
+ AddNewUser(userToAdd : User) : void
+ ExistsUserWithUsername(usernameToLookup : string) : bool
+ GetUserWithUsername(usernameToFind : string) : User
+ UpdateUser(userToModify : User) : void
+ RemoveUserWithUsername(usernameToRemove : string) : void
+ UsernameBelongsToLastAdministrator(usernameToRemove : string) : bool
~ ElementExistsInCollection(value : User) : bool
<<interface>>
ISaleRepository
+ <<property>> Elements : IEnumerable<Sale>
+ AddNewSale(saleToAdd : Sale) : void
<<interface>>
ISubzoneRepository
+ <<property>> Elements : IEnumerable<Subzone>
+ AddNewSubzone(subzoneToAdd : Subzone) : void
+ GetSubzoneWithId(idToFind : int) : Subzone
+ UpdateSubzone(subzoneToModify : Subzone) : void
+ RemoveSubzone(subzoneToRemove : Subzone) : void
SubzoneRepository
+ SubzoneRepository(someContext : VTSystemContext)
+ AddNewSubzone(subzoneToAdd : Subzone) : void
+ GetSubzoneWithId(idToFind : int) : Subzone
+ UpdateSubzone(subzoneToModify : Subzone) : void
+ RemoveSubzone(subzoneToRemove : Subzone) : void
~ ElementExistsInCollection(value : Subzone) : bool

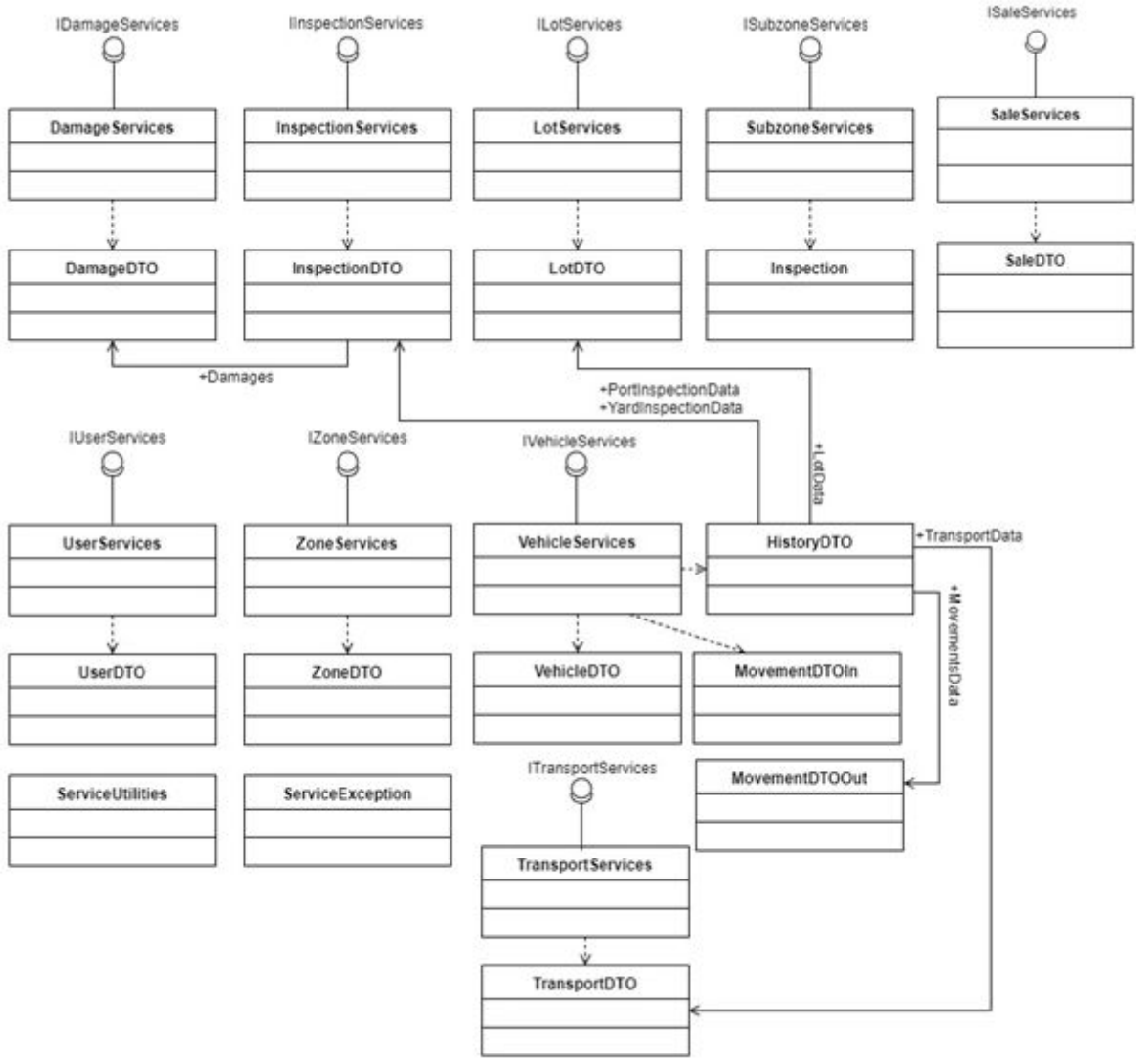

LotRepository
+ LotRepository(someContext : VTSystemContext)
+ AddNewLot(lotToAdd : Lot) : void
+ GetLotWithName(nameToFind : string) : Lot
+ UpdateLot(lotToModify : Lot) : void
+ RemoveLotWithName(nameToRemove : string) : void
+ ExistsLotWithName(nameToFind : string) : bool
~ ElementExistsInCollection(value : Lot) : bool
LocationRepository
+ LocationRepository(someContext : VTSystemContext)
+ GetLocationWithName(nameToLookup : string) : Location
<<interface>>
ILoggingStrategy
+ <<property>> Log : IEnumerable<LoggingRecord>
+ RegisterUserLogin(loggedUser : User) : LoggingRecord
+ RegisterVehicleImport(responsible : User) : LoggingRecord
<<interface>>
IUserRepository
+ <<property>> Elements : IEnumerable<User>
+ AddNewUser(userToAdd : User) : void
+ GetUserWithUsername(usernameToLookup : string) : User
+ UpdateUser(userToModify : User) : void
+ RemoveUserWithUsername(usernameToRemove : string) : void
+ ExistsUserWithUsername(usernameToLookup : string) : bool
+ UsernameBelongsToLastAdministrator(usernameToRemove : string) : bool
ErrorMessages
- resourceMan : ResourceManager
- resourceCulture : CultureInfo
~ <<property>> ResourceManager : ResourceManager
~ <<property>> Culture : CultureInfo
~ <<property>> CouldNotFindElement : string
~ <<property>> CouldNotFindField : string
~ <<property>> CouldNotRemoveElement : string
~ <<property>> ElementDoesNotExist : string
~ <<property>> FieldMustBeUnique : string
~ <<property>> InvalidUserRoleForLogging : string
~ <<property>> NullIDRecieved : string
~ <<property>> NullObjectRecieved : string
~ <<property>> SubzoneNameMustBeUniqueSameZone : string
~ <<property>> ZoneNameMustBeUnique : string
~ ErrorMessages()
<<interface>>
ITransportRepository
+ <<property>> Elements : IEnumerable<Transport>
+ AddNewTransport(transportToAdd : Transport) : void
+ GetTransportWithId(idToLookup : int) : Transport
+ UpdateTransport(transportToModify : Transport) : void
<<interface>>
IZoneRepository
+ <<property>> Elements : IEnumerable<Zone>
+ AddNewZone(zoneToAdd : Zone) : void
+ GetZoneWithName(nameToFind : string) : Zone
+ UpdateZone(zoneToModify : Zone) : void
+ RemoveZone(zoneToRemove : Zone) : void
+ ExistsZoneWithName(username : string) : bool
<<interface>>
IVehicleRepository
+ AddNewVehicle(vehicleToAdd : Vehicle) : void
+ GetRegisteredVehiclesIn(stageToFilterBy : ProcessStages?) : IEnumerable<Vehicle>
+ ExistsVehicleWithVIN(usernameToLookup : string) : bool
+ GetVehicleWithVIN(vinToLookup : string) : Vehicle
+ UpdateVehicle(vehicleToModify : Vehicle) : void
+ RemoveVehicleWithVIN(usernameToRemove : string) : void
+ GetFullyLoadedVehicleWithVIN(vinToLookup : string) : Vehicle
RepositoryException
+ RepositoryException(message : string)
VTSystemDatabaseInitializer
~ defaultSystemPorts : IReadOnlyCollection<?> = new List<L...
~ defaultSystemYards : IReadOnlyCollection<?> = new List<L...
~ defaultAdministrator : User = User.Creat...
~ defaultFlow : Flow = Flow.FromS...
# Seed(context : VTSystemContext) : void
- AddDefaultLocationsToDatabase(context : VTSystemContext) : void

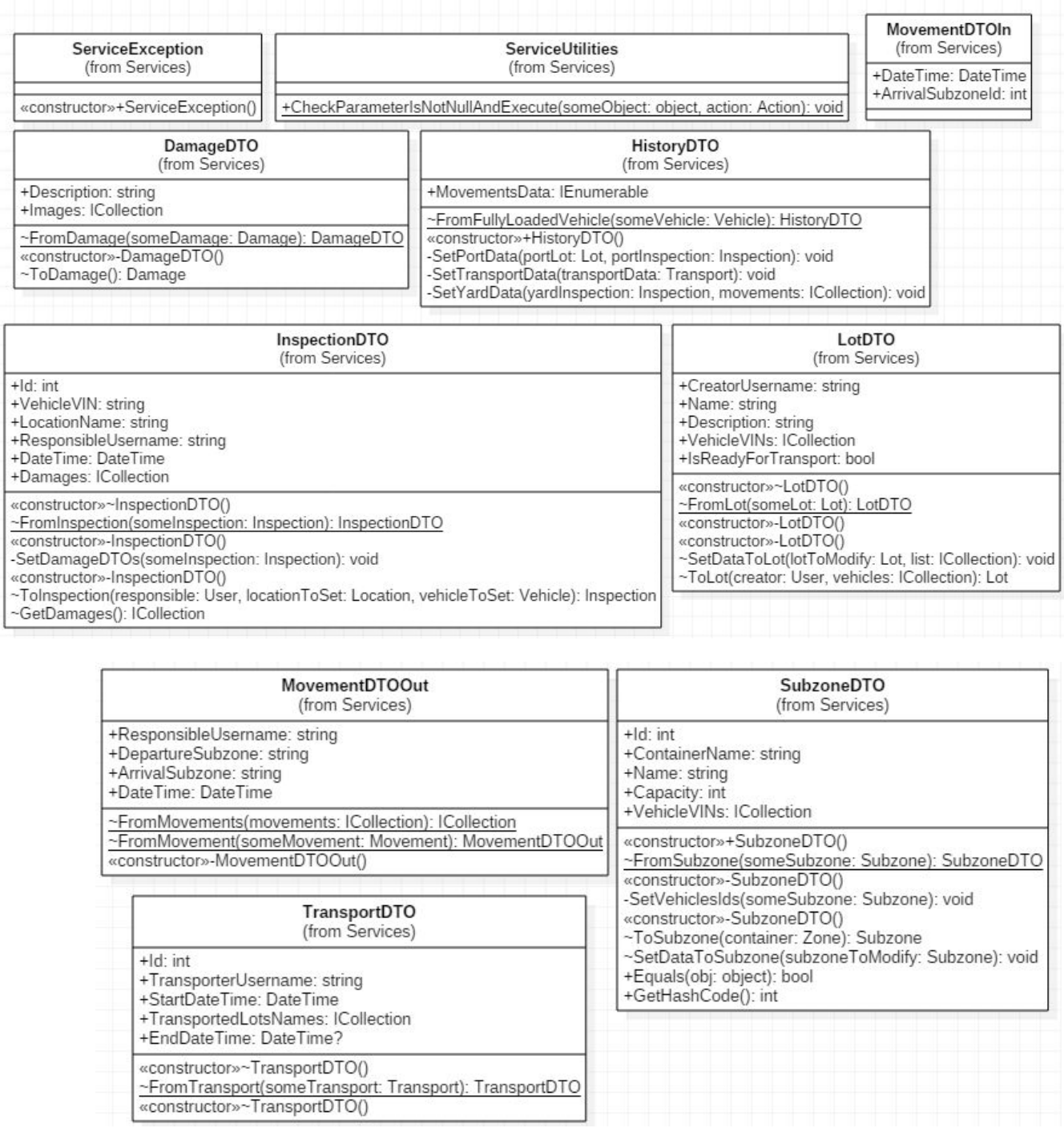

pkg VehicleTracking_Data_DataAccess
<<interface>>
IMovementRepository
+ <<property>> Elements : IEnumerable<Movement>
+ AddNewMovement(movementToAdd : Movement) : void
+ SubzoneParticipatesInSomeMovement(subzoneToVerify : Subzone) : bool
ZoneRepository
+ ZoneRepository(someContext : VTSystemContext)
+ AddNewZone(zoneToAdd : Zone) : void
+ GetZoneWithName(nameToFind : string) : Zone
+ UpdateZone(zoneToModify : Zone) : void
+ RemoveZone(zoneToRemove : Zone) : void
+ ExistsZoneWithName(name : string) : bool
~ ElementExistsInCollection(value : Zone) : bool
LoggingDatabaseConcreteStrategy
+ LoggingDatabaseConcreteStrategy(someContext : VTSystemContext)
+ RegisterUserLogin(loggedUser : User) : LoggingRecord
+ RegisterVehicleImport(responsible : User) : LoggingRecord
- CreateNewLogRecordWithData(responsible : User, actionToLog : LoggedActions) : LoggingRecord
VehicleRepository
+ VehicleRepository(someContext : VTSystemContext)
+ GetRegisteredVehiclesIn(stageToFilterBy : ProcessStages?) : IEnumerable<Vehicle>
+ AddNewVehicle(vehicleToAdd : Vehicle) : void
+ ExistsVehicleWithVIN(VINToLookup : string) : bool
+ GetVehicleWithVIN(vinToFind : string) : Vehicle
+ GetFullyLoadedVehicleWithVIN(vinToLookup : string) : Vehicle
+ AttemptToExecuteActionWithVIN(actionToExecute : VehicleFunc<string,?>, vinToLookup : string) : Vehicle
+ UpdateVehicle(modifiedVehicle : Vehicle) : void
+ RemoveVehicleWithVIN(vinToRemove : string) : void
- VehicleBasicData(vinToFind : string) : Vehicle
- VehicleFullData(vinToLookup : string) : Vehicle
~ ElementExistsInCollection(value : Vehicle) : bool
TEntity
GenericRepository
+ GenericRepository(someContext : VTSystemContext)
+ GetElementsWith(includeProperties : string, filter : Expression<Func< TEntity,bool>>) : IEnumerable<TEntity>
- AddWhereStatement(filter : Expression<Func< TEntity,bool>>, query : IQueryable<TEntity>) : void
- AddIncludeProperties(includeProperties : string, query : IQueryable<TEntity>) : void
Add(elementToAdd : TEntity) : void
AttemptToRemove(entityToRemove : TEntity) : void
Update(entityToUpdate : TEntity) : void
- ValidateParameterIsNotNull(elementToAdd : TEntity) : void
- PerformActionIfElementExistsInCollection(element : TEntity, actionToPerform : Action) : void
~ ElementExistsInCollection(entityToUpdate : TEntity) : bool
~ PerformAttachIfCorresponds(element : TEntity) : void
InspectionRepository
+ InspectionRepository(someContext : VTSystemContext)
+ AddNewInspection(inspectionToAdd : Inspection) : void
+ GetInspectionWithId(idToLookup : Guid) : Inspection
- AddDamagesToDatabase(inspectionToAdd : Inspection) : void
CustomerRepository
+ CustomerRepository(someContext : VTSystemContext)
+ AddNewCustomer(customerToAdd : Customer) : void
+ GetCustomerWithData(nameToFind : string, phoneNumberToFind : string) : Customer
MovementRepository
+ MovementRepository(someContext : VTSystemContext)
+ AddNewMovement(movementToAdd : Movement) : void
+ SubzoneParticipatesInSomeMovement(subzoneToVerify : Subzone) : bool
FlowRepository
+ FlowRepository(someContext : VTSystemContext)
+ GetCurrentFlow() : Flow
+ RegisterNewFlow(flowToRegister : Flow) : void

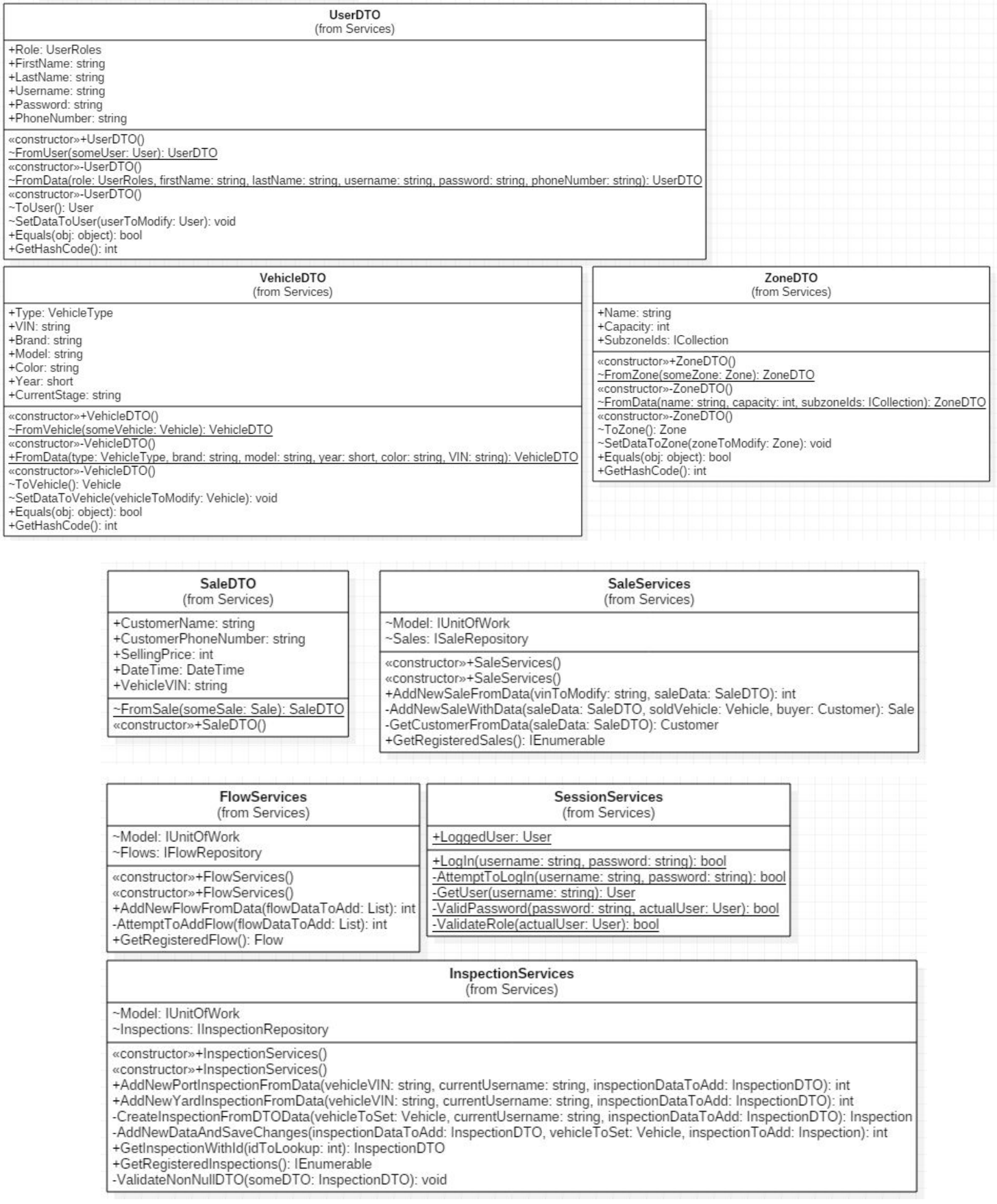
2.2.3 Api.Services

Para este paquete se optó por hacer lo mismo que para los anteriores. Además, en el primer diagrama no están representadas las relaciones entre *ServiceUtilities* y *ServiceException* con el resto de las clases.

Todas las clases tienen una relación de dependencia con las dos mencionadas anteriormente.




ServiceException
(from Services)
«constructor»+ServiceException()
ServiceUtilities
(from Services)
+CheckParameterIsNotNullAndExecute(someObject: object, action: Action): void
MovementDTOIn
(from Services)
+DateTime: DateTime
+ArrivalSubzoneId: int
DamageDTO
(from Services)
+Description: string
+Images: ICollection
~FromDamage(someDamage: Damage): DamageDTO
«constructor»-DamageDTO()
~ToDamage(): Damage
HistoryDTO
(from Services)
+MovementsData: IEnumerable
~FromFullyLoadedVehicle(someVehicle: Vehicle): HistoryDTO
«constructor»+HistoryDTO()
-SetPortData(portLot: Lot, portInspection: Inspection): void
-SetTransportData(transportData: Transport): void
-SetYardData(yardInspection: Inspection, movements: ICollection): void
InspectionDTO
(from Services)
+Id: int
+VehicleVIN: string
+LocationName: string
+ResponsibleUsername: string
+DateTime: DateTime
+Damages: ICollection
«constructor»~InspectionDTO()
~FromInspection(someInspection: Inspection): InspectionDTO
«constructor»-InspectionDTO()
-SetDamageDTOs(someInspection: Inspection): void
«constructor»-InspectionDTO()
~ToInspection(responsible: User, locationToSet: Location, vehicleToSet: Vehicle): Inspection
~GetDamages(): ICollection
LotDTO
(from Services)
+CreatorUsername: string
+Name: string
+Description: string
+VehicleVINs: ICollection
+IsReadyForTransport: bool
«constructor»~LotDTO()
~FromLot(someLot: Lot): LotDTO
«constructor»-LotDTO()
«constructor»-LotDTO()
~SetDataToLot(lotToModify: Lot, list: ICollection): void
~ToLot(creator: User, vehicles: ICollection): Lot
MovementDTOOut
(from Services)
+ResponsibleUsername: string
+DepartureSubzone: string
+ArrivalSubzone: string
+DateTime: DateTime
~FromMovements(movements: ICollection): ICollection
~FromMovement(someMovement: Movement): MovementDTOOut
«constructor»-MovementDTOOut()
SubzoneDTO
(from Services)
+Id: int
+ContainerName: string
+Name: string
+Capacity: int
+VehicleVINs: ICollection
«constructor»+SubzoneDTO()
~FromSubzone(someSubzone: Subzone): SubzoneDTO
«constructor»-SubzoneDTO()
-SetVehiclesIds(someSubzone: Subzone): void
«constructor»-SubzoneDTO()
~ToSubzone(container: Zone): Subzone
~SetDataToSubzone(subzoneToModify: Subzone): void
+Equals(obj: object): bool
+GetHashCode(): int
TransportDTO
(from Services)
+Id: int
+TransporterUsername: string
+StartDateTime: DateTime
+TransportedLotsNames: ICollection
+EndDateTime: DateTime?
«constructor»~TransportDTO()
~FromTransport(someTransport: Transport): TransportDTO
«constructor»~TransportDTO()


UserDTO
(from Services)
+Role: UserRoles
+FirstName: string
+LastName: string
+Username: string
+Password: string
+PhoneNumber: string
«constructor»+UserDTO()
~FromUser(someUser: User): UserDTO
«constructor»-UserDTO()
~FromData(role: UserRoles, firstName: string, lastName: string, username: string, password: string, phoneNumber: string): UserDTO
«constructor»-UserDTO()
~ToUser(): User
~SetDataToUser(userToModify: User): void
+Equals(obj: object): bool
+GetHashCode(): int
VehicleDTO
(from Services)
+Type: VehicleType
+VIN: string
+Brand: string
+Model: string
+Color: string
+Year: short
+CurrentStage: string
«constructor»+VehicleDTO()
~FromVehicle(someVehicle: Vehicle): VehicleDTO
«constructor»-VehicleDTO()
+FromData(type: VehicleType, brand: string, model: string, year: short, color: string, VIN: string): VehicleDTO
«constructor»-VehicleDTO()
~ToVehicle(): Vehicle
~SetDataToVehicle(vehicleToModify: Vehicle): void
+Equals(obj: object): bool
+GetHashCode(): int
ZoneDTO
(from Services)
+Name: string
+Capacity: int
+SubzoneIds: ICollection
«constructor»+ZoneDTO()
~FromZone(someZone: Zone): ZoneDTO
«constructor»-ZoneDTO()
~FromData(name: string, capacity: int, subzoneIds: ICollection): ZoneDTO
«constructor»-ZoneDTO()
~ToZone(): Zone
~SetDataToZone(zoneToModify: Zone): void
+Equals(obj: object): bool
+GetHashCode(): int
SaleDTO
(from Services)
+CustomerName: string
+CustomerPhoneNumber: string
+SellingPrice: int
+DateTime: DateTime
+VehicleVIN: string
~FromSale(someSale: Sale): SaleDTO
«constructor»+SaleDTO()
SaleServices
(from Services)
~Model: IUnitOfWork
~Sales: ISaleRepository
«constructor»+SaleServices()
«constructor»+SaleServices()
+AddNewSaleFromData(vinToModify: string, saleData: SaleDTO): int
-AddNewSaleWithData(saleData: SaleDTO, soldVehicle: Vehicle, buyer: Customer): Sale
-GetCustomerFromData(saleData: SaleDTO): Customer
+GetRegisteredSales(): IEnumerable
FlowServices
(from Services)
~Model: IUnitOfWork
~Flows: IFlowRepository
«constructor»+FlowServices()
«constructor»+FlowServices()
+AddNewFlowFromData(flowDataToAdd: List): int
-AttemptToAddFlow(flowDataToAdd: List): int
+GetRegisteredFlow(): Flow
SessionServices
(from Services)
+LoggedUser: User
+LogIn(username: string, password: string): bool
-AttemptToLogIn(username: string, password: string): bool
-GetUser(username: string): User
-ValidPassword(password: string, actualUser: User): bool
-ValidateRole(actualUser: User): bool
InspectionServices
(from Services)
~Model: IUnitOfWork
~Inspections: IInspectionRepository
«constructor»+InspectionServices()
«constructor»+InspectionServices()
+AddNewPortInspectionFromData(vehicleVIN: string, currentUsername: string, inspectionDataToAdd: InspectionDTO): int
+AddNewYardInspectionFromData(vehicleVIN: string, currentUsername: string, inspectionDataToAdd: InspectionDTO): int
-CreateInspectionFromDTOData(vehicleToSet: Vehicle, currentUsername: string, inspectionDataToAdd: InspectionDTO): Inspection
-AddNewDataAndSaveChanges(inspectionDataToAdd: InspectionDTO, vehicleToSet: Vehicle, inspectionToAdd: Inspection): int
+GetInspectionWithId(idToLookup: int): InspectionDTO
+GetRegisteredInspections(): IEnumerable
-ValidateNonNullDTO(someDTO: InspectionDTO): void

LotServices
(from Services)

~Model: IUnitOfWork
~Lots: ILotRepository

«constructor»+LotServices()
«constructor»+LotServices()
+AddNewLotFromData(activeUsername: string, lotDataToAdd: LotDTO): Guid
-GetVehicleList(vinsToFind: ICollection): ICollection
-AttemptToAddLot(creator: User, vehicles: ICollection, lotData: LotDTO): Guid
-CreateAndAddLot(creator: User, vehicles: ICollection, lotData: LotDTO): Guid
+GetRegisteredLots(): IEnumerable
+GetLotByName(nameToFind: string): LotDTO
+ModifyLotWithName(nameToModify: string, lotDataToSet: LotDTO): void
-AttemptToPerformModification(nameToModify: string, lotData: LotDTO): void
-ModifyLotWithData(nameToModify: string, lotData: LotDTO, lotFound: Lot): void
-SetLotPropertiesAndSaveChanges(lotData: LotDTO, lotFound: Lot): void
-MarkAddedAndRemovedVehiclesAsModified(lotFound: Lot, vehiclesToSet: ICollection): void
-MarkVehicleCollectionAsModified(vehicles: IEnumerable): void
-ChangeCausesRepeatedNames(nameToModify: string, lotData: LotDTO): bool
+RemoveLotWithName(nameToModify: string): void

SubzoneServices
(from Services)

~Model: IUnitOfWork
~Zones: IZoneRepository
~Subzones: ISubzoneRepository

«constructor»+SubzoneServices()
«constructor»+SubzoneServices()
+AddNewSubzoneFromData(containerName: string, zoneDataToAdd: SubzoneDTO): int
-AttemptToAddSubzone(container: Zone, subzoneData: SubzoneDTO): int
+GetRegisteredSubzones(): IEnumerable
+GetSubzoneWithId(idToFind: int): SubzoneDTO
+ModifySubzoneWithId(idToModify: int, subzoneDataToSet: SubzoneDTO): void
-AttemptToPerformModification(idToModify: int, subzoneData: SubzoneDTO): void
+RemoveSubzoneWithId(idToRemove: int): void

UserServices
(from Services)

~Model: IUnitOfWork
~Users: IUserRepository

«constructor»+UserServices()
«constructor»+UserServices()
+AddNewUserFromData(userDataToAdd: UserDTO): int
-AttemptToAddUser(userDataToAdd: UserDTO): int
+GetRegisteredUsers(): IEnumerable
+GetUserWithUsername(usernameToLookup: string): UserDTO
+ModifyUserWithUsername(usernameToModify: string, userDataToSet: UserDTO): void
-AttemptToPerformModification(usernameToModify: string, userData: UserDTO): void
-ChangeCausesRepeatedUsernames(currentUsername: string, userData: UserDTO): bool
+RemoveUserWithUsername(usernameToRemove: string): void

TransportServices
(from Services)

~Model: IUnitOfWork
~Transports: ITransportRepository

«constructor»+TransportServices()
«constructor»+TransportServices()
+StartNewTransportFromData(activeUsername: string, transportData: TransportDTO): int
-GetLotsFromNames(transportedLotsNames: ICollection): ICollection
+GetRegisteredTransports(): ICollection
+FinalizeTransport(activeUsername: string, transportIdToFinalize: int, finalizationDateTime: DateTime): void
-SetFinalizationData(finalizationDateTime: DateTime, transportToFinalize: Transport): void
-MarkLotsAndVehiclesAsModified(lotsTransported: ICollection): void
-MarkVehiclesInLotAsModified(vehicles: ICollection): void

VehicleServices
(from Services)

~Model: IUnitOfWork
~Vehicles: IVehicleRepository

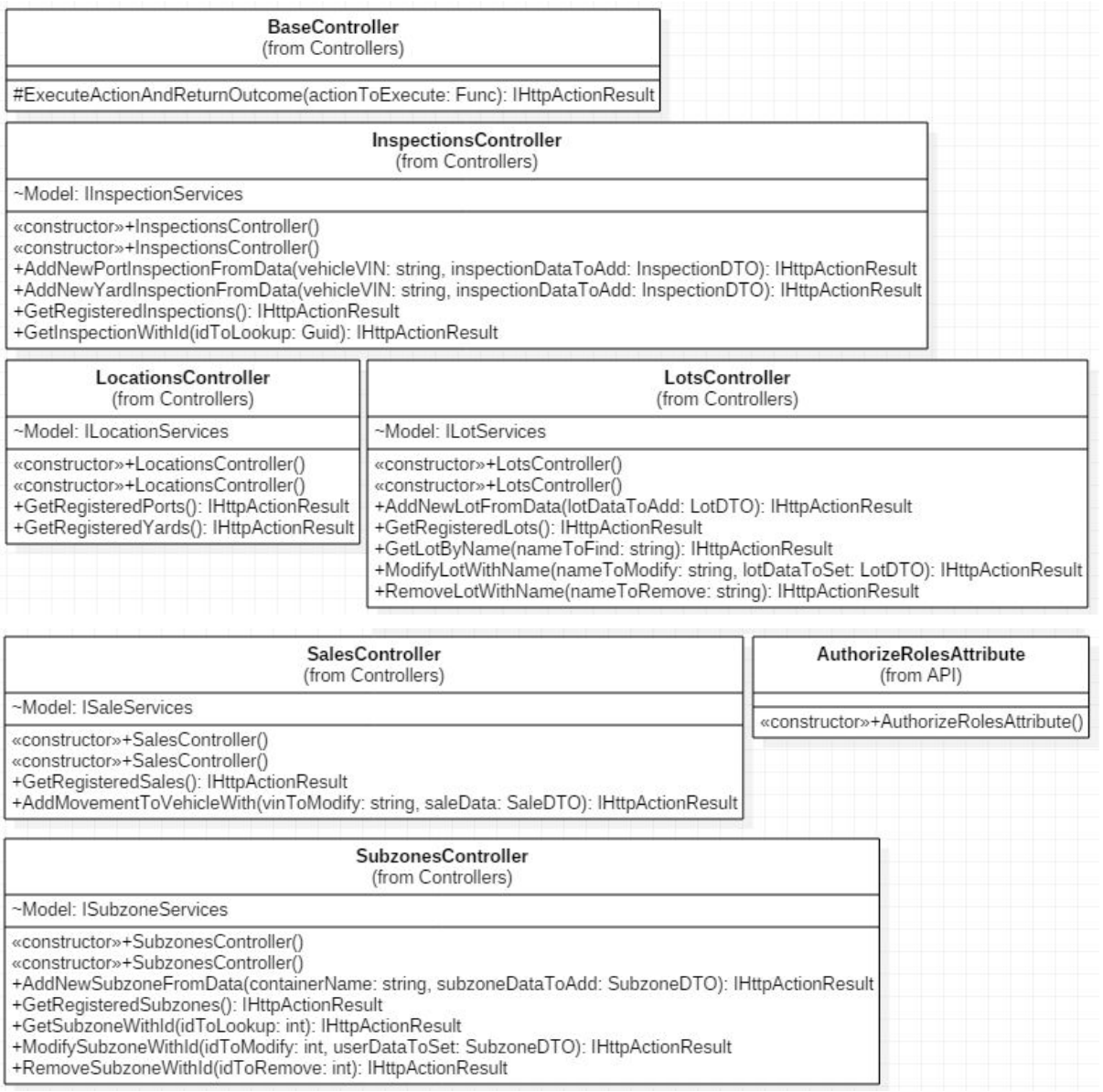
«constructor»+VehicleServices()
«constructor»+VehicleServices()
+AddNewVehicleFromData(vehicleDataToAdd: VehicleDTO): int
-AttemptToAddVehicle(vehicleDataToAdd: VehicleDTO): int
+GetRegisteredVehicles(): IEnumerable
+GetVehicleWithVIN(vinToLookup: string): VehicleDTO
+ModifyVehicleWithVIN(vinToModify: string, vehicleDataToSet: VehicleDTO): void
-AttemptToPerformModification(vinToModify: string, vehicleData: VehicleDTO): void
-ChangeCausesRepeatedVINs(currentVIN: string, vehicleData: VehicleDTO): bool
+RemoveVehicleWithVIN(vinToRemove: string): void
+AddNewMovementFromData(responsibleUsername: string, vinToModify: string, movementDataToAdd: MovementDTOIn): int
-AttemptToAddNewMovementFromData(responsibleUsername: string, vinToModify: string, movementData: MovementDTOIn): int
-AddNewMovement(movedVehicle: Vehicle, movementToAdd: Movement): int
+GetHistoryForVehicleWithVIN(vinToLookup: string): HistoryDTO

ZoneServices
(from Services)

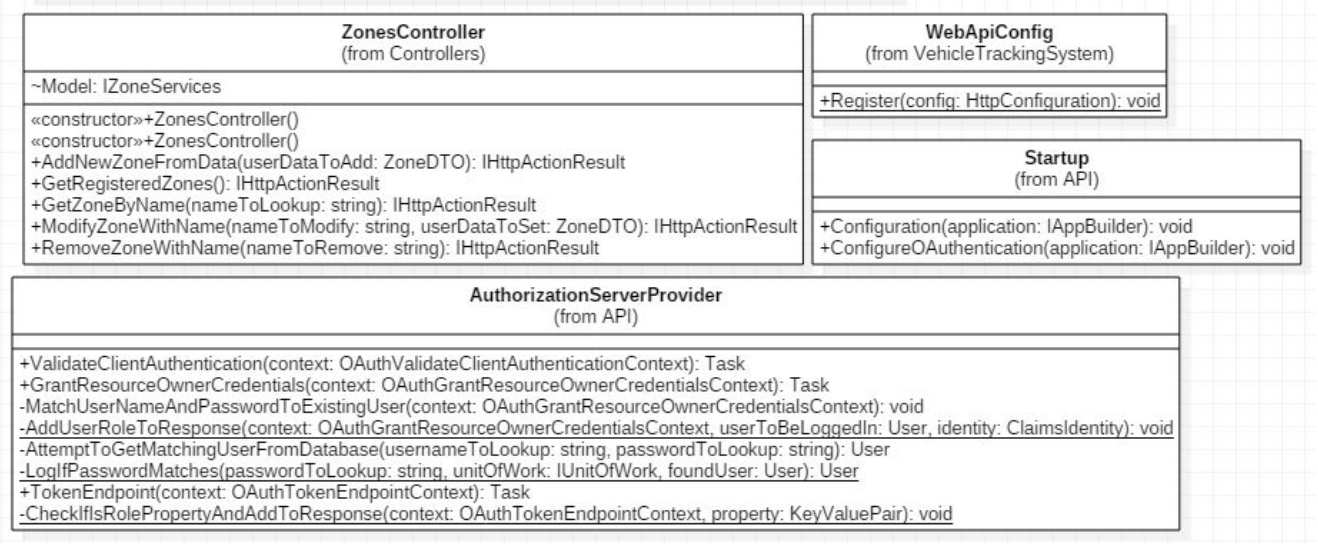
~Model: IUnitOfWork
~Zones: IZoneRepository

«constructor»+ZoneServices()
«constructor»+ZoneServices()
+AddNewZoneFromData(zoneDataToAdd: ZoneDTO): int
-AttemptToAddZone(zoneDataToAdd: ZoneDTO): int
+GetRegisteredZones(): IEnumerable
+GetZoneWithName(nameToLookup: string): ZoneDTO
+ModifyZoneWithName(nameToModify: string, zoneDataToSet: ZoneDTO): void
-AttemptToPerformModification(nameToModify: string, zoneData: ZoneDTO): void
-ChangeCausesRepeatedNames(currentName: string, zoneData: ZoneDTO): bool
+RemoveZoneWithName(nameToRemove: string): void

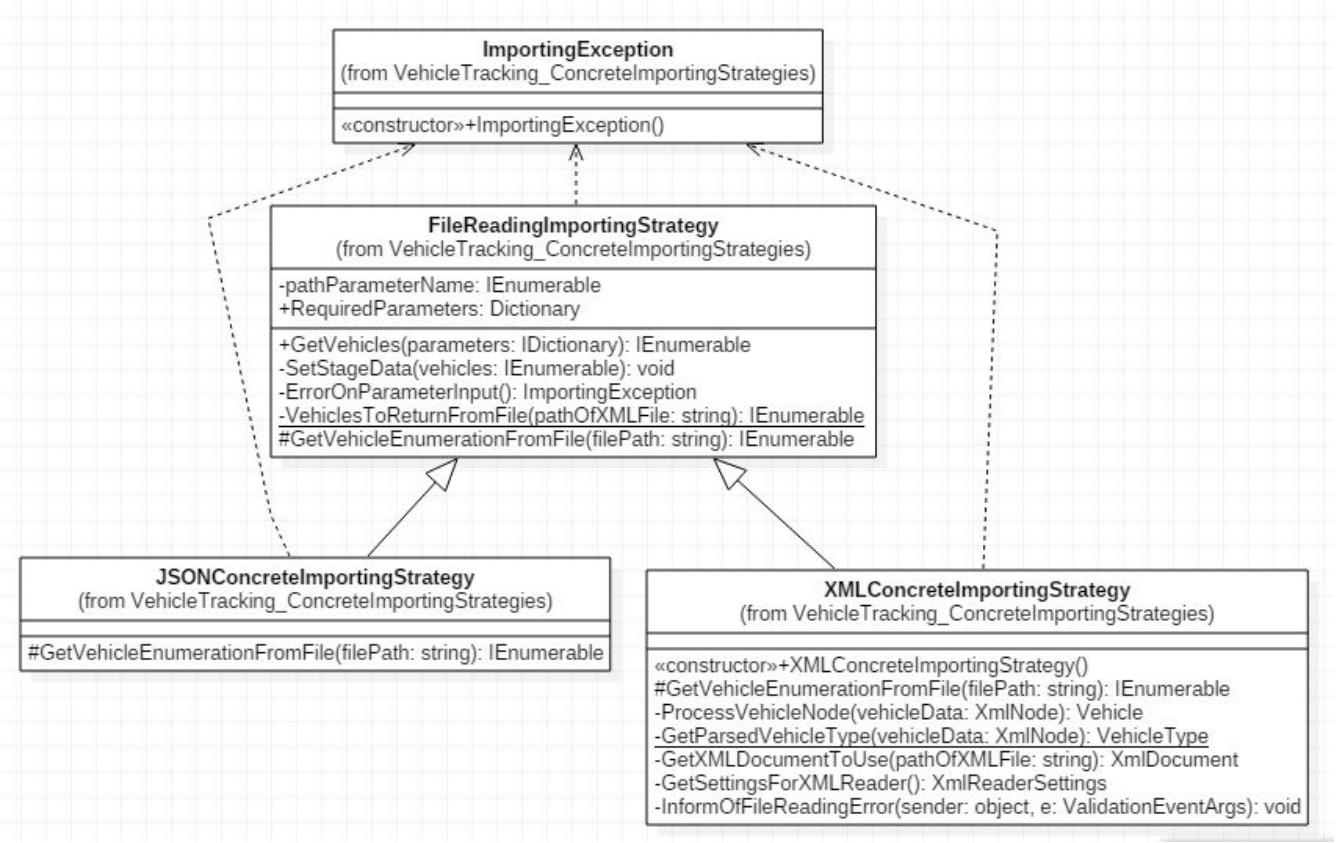
2.2.4 Web.Api


BaseController
(from Controllers)
#ExecuteActionAndReturnOutcome(actionToExecute: Func): IHttpActionResult
InspectionsController
(from Controllers)
~Model: IInspectionServices
«constructor»+InspectionsController()
«constructor»+InspectionsController()
+AddNewPortInspectionFromData(vehicleVIN: string, inspectionDataToAdd: InspectionDTO): IHttpActionResult
+AddNewYardInspectionFromData(vehicleVIN: string, inspectionDataToAdd: InspectionDTO): IHttpActionResult
+GetRegisteredInspections(): IHttpActionResult
+GetInspectionWithId(idToLookup: Guid): IHttpActionResult
LocationsController
(from Controllers)
~Model: ILocationServices
«constructor»+LocationsController()
«constructor»+LocationsController()
+GetRegisteredPorts(): IHttpActionResult
+GetRegisteredYards(): IHttpActionResult
LotsController
(from Controllers)
~Model: ILotServices
«constructor»+LotsController()
«constructor»+LotsController()
+AddNewLotFromData(lotDataToAdd: LotDTO): IHttpActionResult
+GetRegisteredLots(): IHttpActionResult
+GetLotByName(nameToFind: string): IHttpActionResult
+ModifyLotWithName(nameToModify: string, lotDataToSet: LotDTO): IHttpActionResult
+RemoveLotWithName(nameToRemove: string): IHttpActionResult
SalesController
(from Controllers)
~Model: ISaleServices
«constructor»+SalesController()
«constructor»+SalesController()
+GetRegisteredSales(): IHttpActionResult
+AddMovementToVehicleWith(vinToModify: string, saleData: SaleDTO): IHttpActionResult
AuthorizeRolesAttribute
(from API)
«constructor»+AuthorizeRolesAttribute()
SubzonesController
(from Controllers)
~Model: ISubzoneServices
«constructor»+SubzonesController()
«constructor»+SubzonesController()
+AddNewSubzoneFromData(containerName: string, subzoneDataToAdd: SubzoneDTO): IHttpActionResult
+GetRegisteredSubzones(): IHttpActionResult
+GetSubzoneWithId(idToLookup: int): IHttpActionResult
+ModifySubzoneWithId(idToModify: int, userDataToSet: SubzoneDTO): IHttpActionResult
+RemoveSubzoneWithId(idToRemove: int): IHttpActionResult

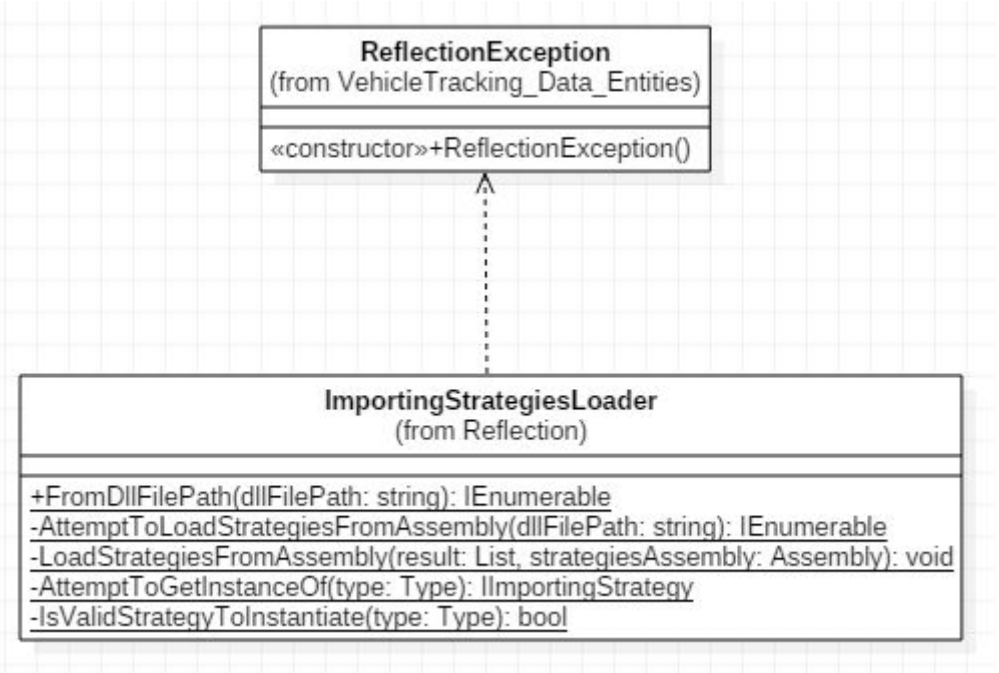

TransportsController
(from Controllers)
~Model: ITransportServices
«constructor»+TransportsController()
«constructor»+TransportsController()
+StartNewTransportFromData(transportDataToAdd: TransportDTO): IHttpActionResult
+GetRegisteredTransports(): IHttpActionResult
+ModifyUserWithUsername(transportIdToFinalize: int, finalizationDateTime: DateTime): IHttpActionResult
UsersController
(from Controllers)
~Model: IUserServices
«constructor»+UsersController()
«constructor»+UsersController()
+AddNewUserFromData(userDataToAdd: UserDTO): IHttpActionResult
+GetRegisteredUsers(): IHttpActionResult
+GetUserByUsername(usernameToLookup: string): IHttpActionResult
+ModifyUserWithUsername(usernameToModify: string, userDataToSet: UserDTO): IHttpActionResult
+RemoveUserWithUsername(usernameToRemove: string): IHttpActionResult
VehiclesController
(from Controllers)
~Model: IVehicleServices
«constructor»+VehiclesController()
«constructor»+VehiclesController()
+AddNewVehicleFromData(vehicleDataToAdd: VehicleDTO): IHttpActionResult
+GetRegisteredVehicles(): IHttpActionResult
-AttemptToGetRegisteredVehicles(): IHttpActionResult
-GetRoleOfActiveUser(): UserRoles
+GetVehicleWithVIN(vinToLookup: string): IHttpActionResult
+ModifyVehicleWithVIN(vinToModify: string, vehicleDataToSet: VehicleDTO): IHttpActionResult
+RemoveVehicleWithVIN(vinToRemove: string): IHttpActionResult
+AddMovementToVehicleWith(vinToModify: string, movementData: MovementDTOIn): IHttpActionResult
+GetFullHistoryOfVehicleWithVIN(vinToLookup: string): IHttpActionResult


ZonesController
(from Controllers)
~Model: IZoneServices
«constructor»+ZonesController()
«constructor»+ZonesController()
+AddNewZoneFromData(userDataToAdd: ZoneDTO): IHttpActionResult
+GetRegisteredZones(): IHttpActionResult
+GetZoneByName(nameToLookup: string): IHttpActionResult
+ModifyZoneWithName(nameToModify: string, userDataToSet: ZoneDTO): IHttpActionResult
+RemoveZoneWithName(nameToRemove: string): IHttpActionResult
WebApiConfig
(from VehicleTrackingSystem)
+Register(config: HttpConfiguration): void
Startup
(from API)
+Configuration(application: IAppBuilder): void
+ConfigureOAuthentication(application: IAppBuilder): void
AuthorizationServerProvider
(from API)
+ValidateClientAuthentication(context: OAuthValidateClientAuthenticationContext): Task
+GrantResourceOwnerCredentials(context: OAuthGrantResourceOwnerCredentialsContext): Task
-MatchUserNameAndPasswordToExistingUser(context: OAuthGrantResourceOwnerCredentialsContext): void
-AddUserRoleToResponse(context: OAuthGrantResourceOwnerCredentialsContext, userToBeLoggedIn: User, identity: ClaimsIdentity): void
-AttemptToGetMatchingUserFromDatabase(usernameToLookup: string, passwordToLookup: string): User
-LogIfPasswordMatches(passwordToLookup: string, unitOfWork: IUnitOfWork, foundUser: User): User
+TokenEndpoint(context: OAuthTokenEndpointContext): Task
-CheckIfIsRolePropertyAndAddToResponse(context: OAuthTokenEndpointContext, property: KeyValuePair): void

2.2.5 VehicleTracking_ConcreteImportingStrategies

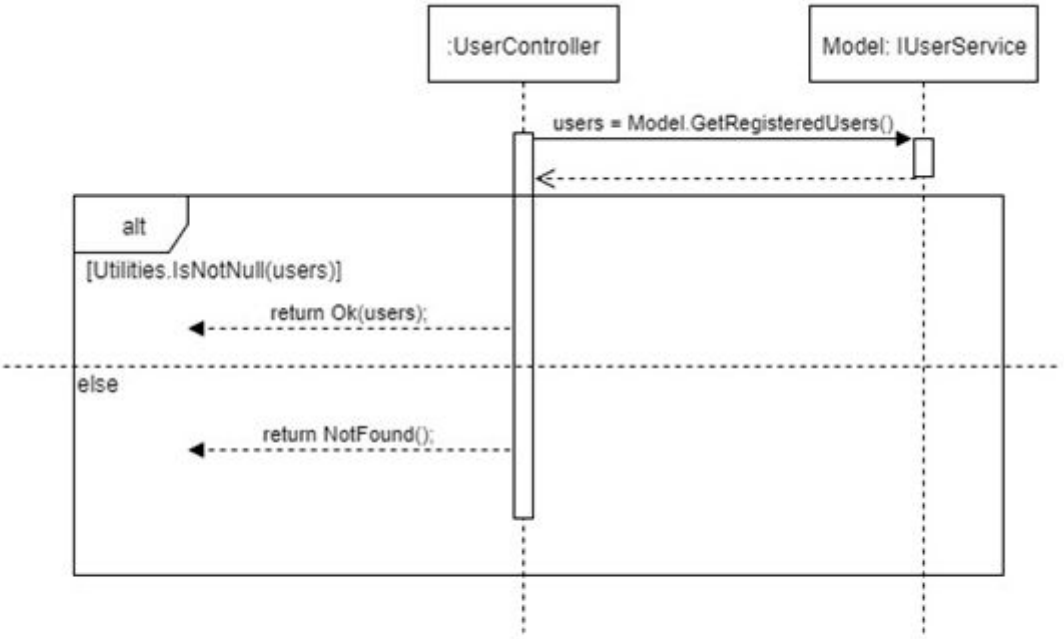


2.2.6 VehicleTracking.Reflection

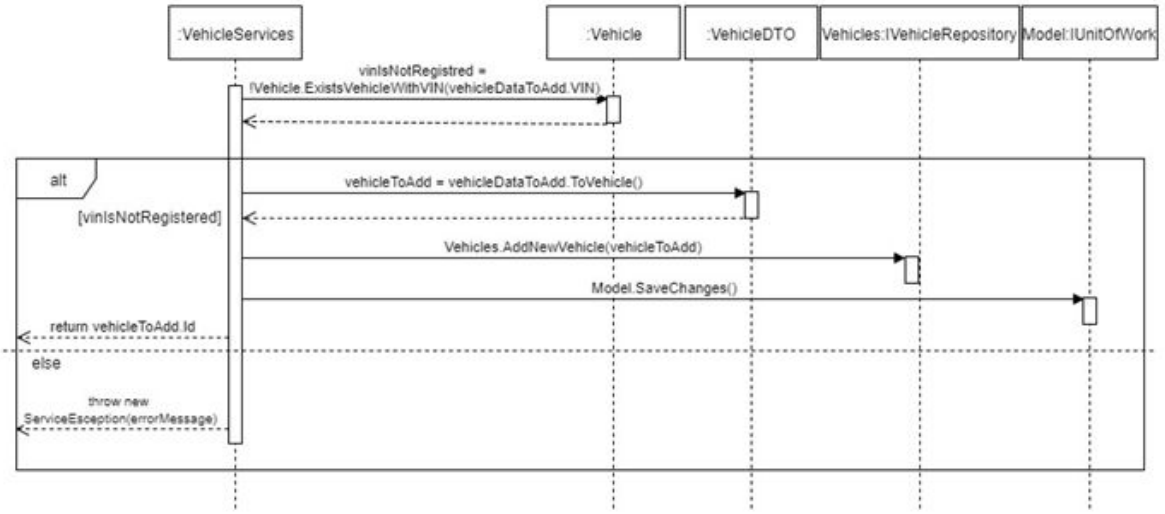


2.3 Diagramas de interacción

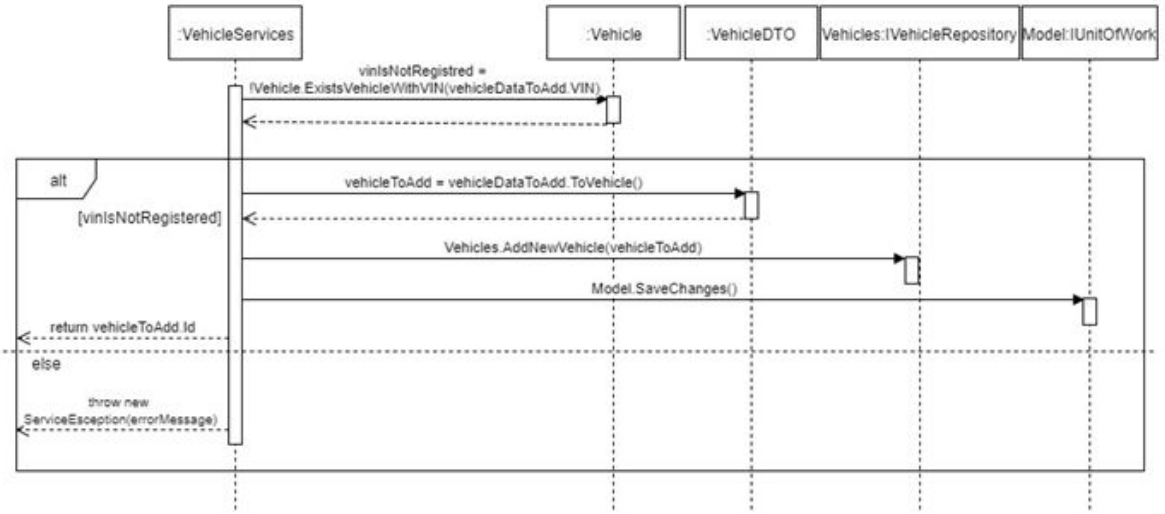
2.3.1 UsersController - AttemptToGetRegisteredUsers



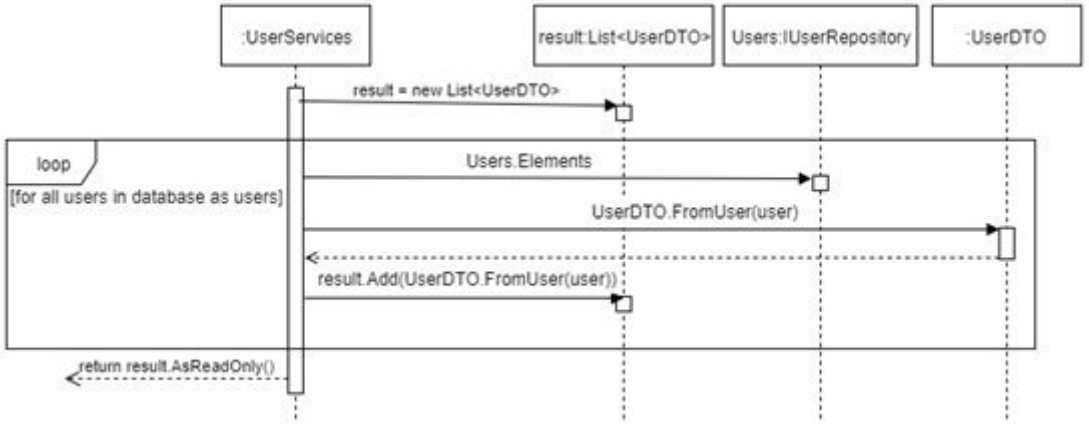
2.3.2 VehicleServices - AttemptToAddVehicle



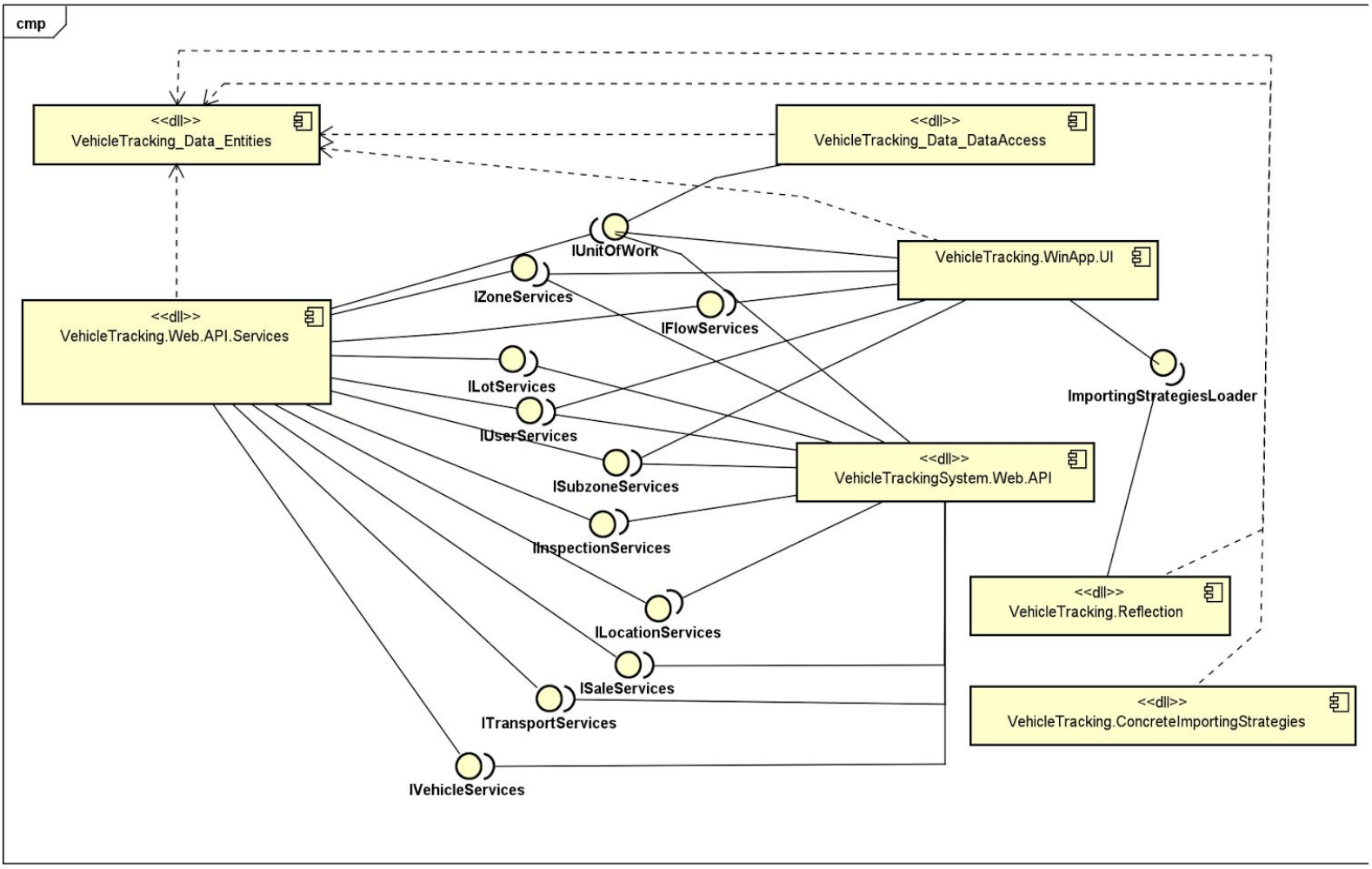
2.3.3 BaseController - ExecuteActionAndReturnOutcome



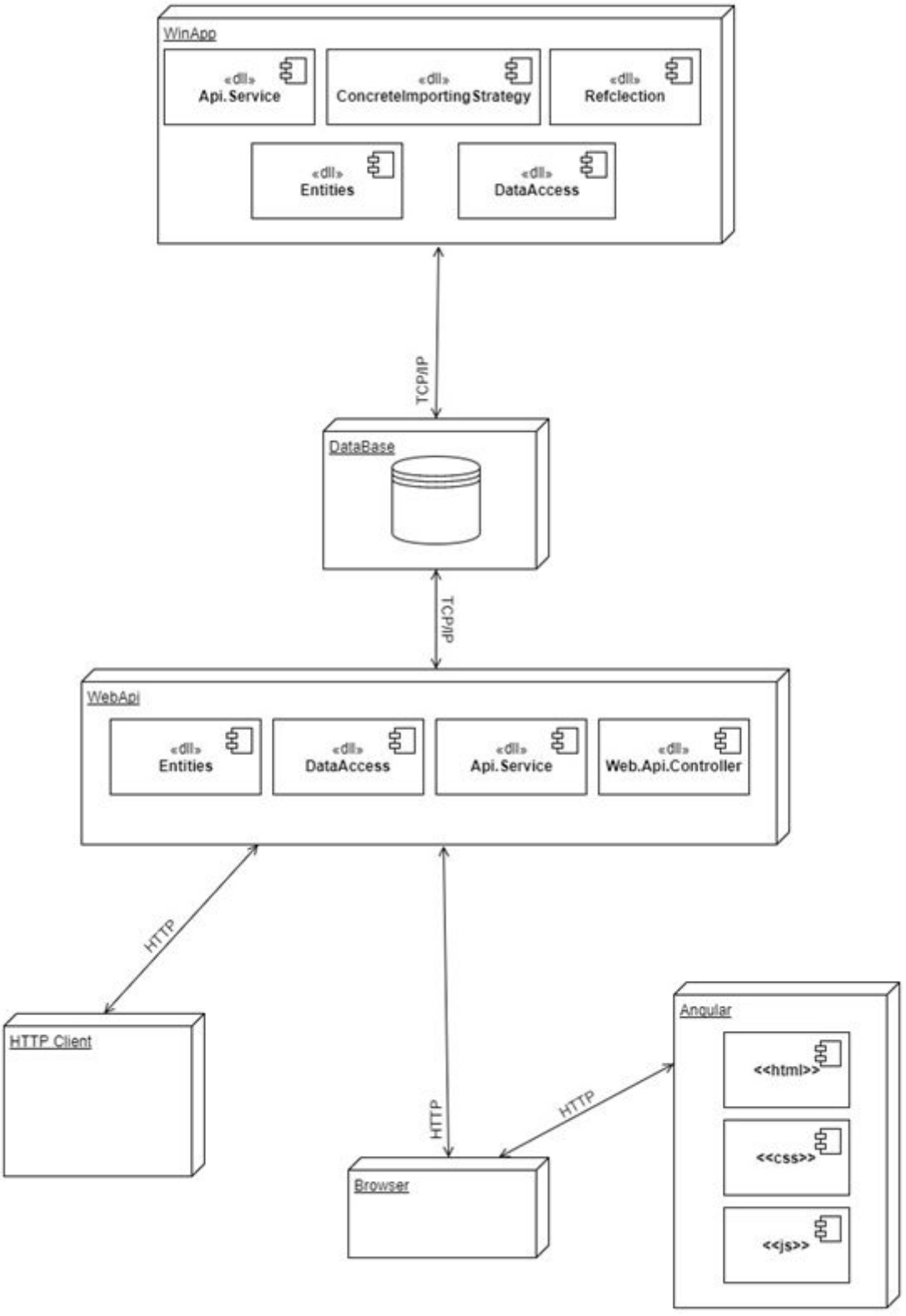
2.3.4 UserServices - GetRegisteredUsers



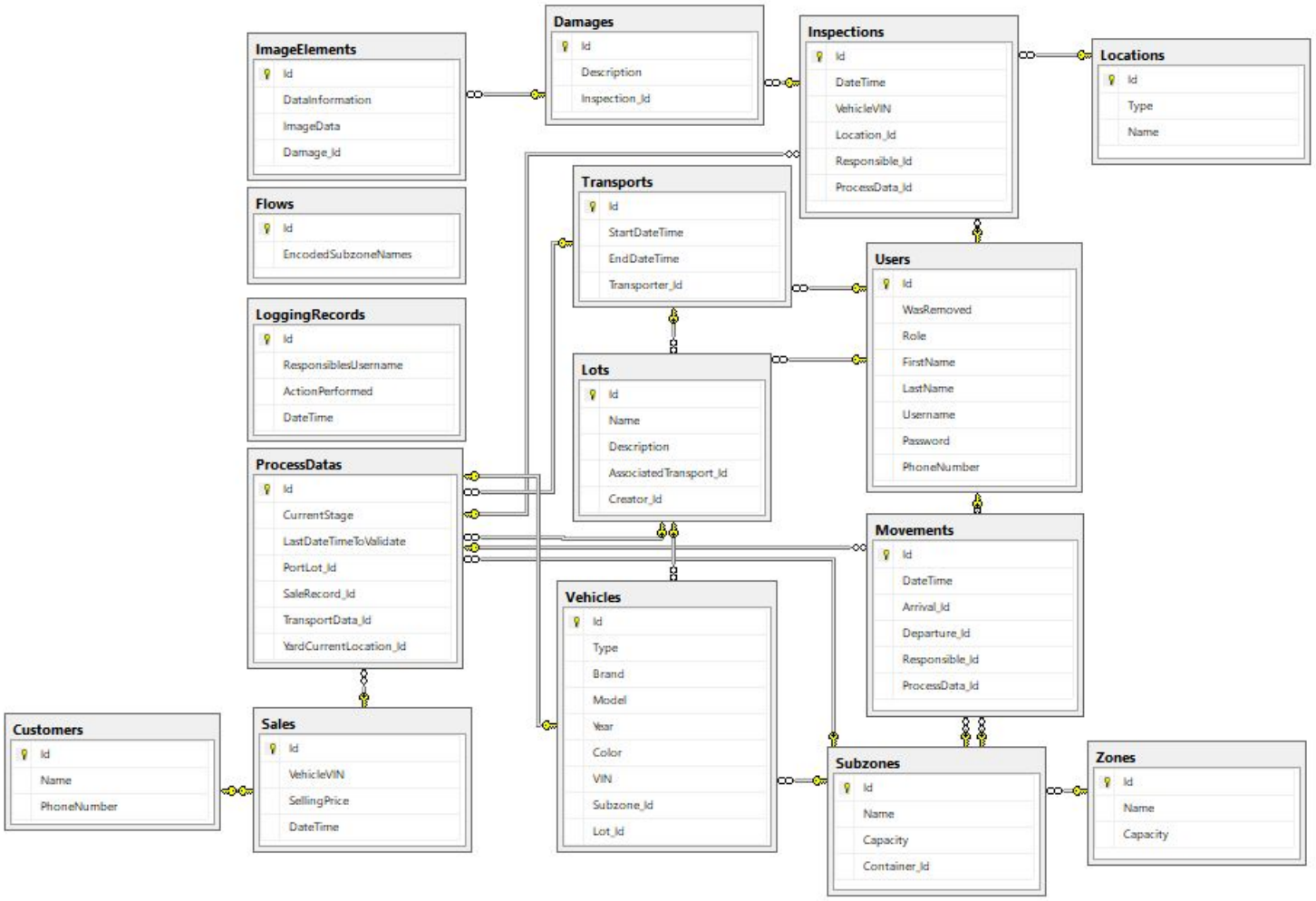
2.4 Diagrama de componentes



2.5 Diagrama de entrega



3. Modelo de tablas



4. Justificación de diseño

En el caso de este obligatorio, como se mencionó anteriormente, el principal objetivo del mismo sería, en primera instancia, generar la interfaz gráfica asociada a la funcionalidad implementada en la primer mitad del trabajo, y en segundo lugar extender la misma para abarcar nuevos requerimientos introducidos para esta entrega. Así, naturalmente, se reutilizaría la solución ya implementada al punto de la instancia anterior, manteniéndose válidas por tanto las consideraciones a nivel del diseño del sistema expuestas en la documentación del obligatorio anterior.

En primer lugar, entonces, con respecto a la nueva funcionalidad pedida, se descubriría esta tendría impacto en distintos aspectos de la solución, debido a la naturaleza diversa de los nuevos requerimientos. Así, por ejemplo, se pediría extender el proceso atravesado por un vehículo con una nueva etapa de venta posterior a la de patio, en la órbita de un rol de usuario “vendedor” a manejarse. Este punto implicaría el manejo de un nuevo tipo de entidad en la lógica de dominio, encapsulado en la clase “*Sale*” generada, para la cual también sería necesario crear clases *Repository* y *Service* que lograran persistir la misma a partir de los datos introducidos por el cliente, en particular número de teléfono y nombre del comprador, y precio de venta. En cuanto a ésto, y si bien la propuesta no lo pedía explícitamente, se optaría por manejar los primeros datos mencionados como parte de un tipo “*Customer*”, como forma de brindar a futuro flexibilidad al momento de implementar, por ejemplo, consultas sobre los datos de clientes de la empresa, por ejemplo para determinar cuál es el cliente que mayor cantidad de compras ha realizado, así como forma de evitar introducir información duplicada de los mismos en la base de datos. Asimismo, como ya se adelantaría en la justificación de diseño del primer obligatorio, esto también conllevaría cambios en la clase denominada “*ProcessData*”, que se mantuvo para esta entrega, encargada justamente de los datos asociados al proceso asociado a un vehículo. Como forma de subsanar esto, queda pendiente la evaluación de una reestructura del diseño del sistema a algo más similar al patrón *State*, como forma de amortiguar el impacto de cambios de este tipo.

Asimismo, y si bien estrictamente no vinculado a nuevos requerimientos, con respecto a cambios asociados a entidades del dominio se descubrió que la solución entregada a la fecha del primer obligatorio no lograba persistir las imágenes asociadas a daños de inspecciones, puesto que estas eran guardadas en una *property* de tipo *ICollection<string>* que *Entity Framework* no maneja automáticamente puesto que no puede mapear éstos a una tabla al no poseer éstos una clave primaria identificable. Se crearía, así, un nuevo tipo “*ImageElement*” encargada del manejo de esta información, manejada y persistida por *Entity*

Framework, en la cual además se implementaría la funcionalidad de validar que la cadena de caracteres recibida es efectivamente una imagen, crítica recibida sobre el primer obligatorio. Esta sería entonces almacenada bajo la forma *byte[]*, que se presume es la que resulta más conveniente.

Con respecto a las restantes funcionalidades pedidas para esta segunda entrega, concretamente el manejo de *logs* del acceso a ciertas funcionalidades del sistema, y la importación de vehículos a partir de cierta información (pudiendo esta provenir de archivos en determinado formato, por ejemplo, más debiendo poderse extender estas en tiempo de ejecución), éstos no representarían mayores cambios a nivel de las entidades del dominio propiamente dichas.

La solución perseguida en ambos casos sería similar: generar una familia de tipos según lo define el patrón *Strategy*, que pudieran utilizarse de forma transparente unos u otros desde clases clientes. Especial cuidado hubo que tenerse respecto de la importación de vehículos, debido a el requerimiento anteirormente mencionado de ser extensible en tiempo de ejecución. Para esto, se empleó una solución basada en *Reflection*, que cargara archivos *.dll* contenedores de una interfaz provista (la estrategia genérica) para efectuar la importación. Tal funcionalidad se encapsularía en una clase estática *ImportingStrategiesLoader* en un proyecto independiente, denominado *VehicleTracking.WinApp.Reflection.* Asimismo, dado que a efectos de prueba sería necesario proveer a tal interfaz de implementaciones concretas, que importaran vehículos desde archivos en formato XML y JSON, según fue elegido, estas también serían creadas como parte de un proyecto denominado *VehicleTracking.ConcreteImportingStrategies*.

El funcionamiento (mas quizás no la implementación) de estas clases sería sencillo, simplemente creando y devolviendo una colección de objetos del tipo *Vehicle,* que deberían luego ser manejados (agregados al sistema mediante la utilización de la clase *VehicleServices*, por ejemplo) por la clase cliente. Finalmente, dado que los parámetros requeridos por cada estrategia (si bien la dos utilizadas necesitan la ruta del archivo asociado) podrían variar, se persiguió una solución que manejara dinámicamente tal variación, retornándose mediante un método implementado en la estrategia un diccionario de los mismos con su tipo, y buscándose luego generar en base al mismo la interfaz para su ingreso, estrategia que se considera funciona presumiblemente bien.

Por otro lado, tendríamos quizás la parte en la que se invirtió mayor cantidad de tiempo en el proyecto; esto es, la generación de la interfaz gráfica asociada a la funcionalidad pedida, que se dividiría en dos aplicaciones, una *Web* a realizarse utilizando *Angular 2* como tecnología, y otra administrativa, de escritorio, a generarse mediante la ya conocida *Windows Forms*. Con respecto a esta última, al igual que en Diseño de aplicaciones I se intentaría generar *UserControls*

independientes y reutilizables por cada funcionalidad identificable, intercambiándose éstos en una ventana principal. Esta aplicación, por otro lado, a efecto de la implementación de su lógica asociada, no interactuaría con la *Web API,* sino que haría uso de las clases internas *Services* directamente. Esto traería algunos problemas e ineficiencias debido a la firma de los métodos expuestos en los mismos, optimizados para el manejos de entidades DTO (*data transfer objects*) manejados por la API, instanciados directamente desde el cuerpo de las *requests HTTP.* Sin embargo, estos se crearían entonces a mano, solución considerada un tanto poco elegante, pero siendo ésta la que menor impacto causaba en el sistema.

Por otro lado, está la interfaz *Web* anteriormente mencionada, que sería la que mayores desafíos representaría a nivel de tecnología e implementación, debido a la poca experiencia de los desarrolladores con la misma. Con respecto a esta, ya desde una primera instancia la solución sería creada utilizando *Angular CLI*, que mediante su comando "*ng new*" genera un *template* de las mismas, lo cual resultaría conveniente luego al momento de realizar el *deploy* de la solución. Se seguiría la estructura de directorios recomendada, creando una carpeta por componente individual creado, automáticamente manejado esto al utilizarse el comando *ng generate component.* Asimismo, se intentaría al mayor grado posible respetar los lineamientos generales manejados hasta el momento para realizar piezas de *software* de alta calidad, por ejemplo implementándose la interacción con la API REST en distintas clases *Service* a ser llamadas desde los distintos componentes para obtener los datos requeridos por los mismos, por ejemplo, en el método *ngInit.* Éstas heredarían de un *BaseService* creado como forma de evitar código duplicado en cuanto al manejo de la generación de headers HTTP (necesarios para enviar en cada request el *token* de autenticación), o las potenciales respuestas indicando un error desde la API.

Finalmente, y como suele ser habitual, al estarse generando esta interfaz resultó necesario realizar necesarios ciertos cambios que, si bien no resultaban demasiado radicales ni rompían completamente la Web API utilizada hasta el momento, si impactaban positivamente en cuanto a la usabilidad de la aplicación. El primero y más grande de estos sería el filtrado de vehículos por etapa del proceso vinculada al rol del usuario; es decir, que al efectuar una request HTTP GET a la URL "*/api/Vehicles*" se retornen, por ejemplo, solamente en el puerto para un operario de puerto, que además era un requerimiento funcional y fue uno de los puntos señalados en la instancia de defensa del anterior obligatorio (que como se habrá visto fue bastante exhaustiva). Otros, sin embargo, serían un tanto menores, como por ejemplo enviar ciertos valores *booleanos* ya manejados por la entidad *Vehicle* pero que claramente en *Postman* no eran utilizados hasta el momento, por ejemplo si un vehículo pertenece a un lote, impactando esto en las clases de *data transfer objects, VehicleDTO* para este ejemplo concreto. Estos se utilizarían

principalmente para activar y desactivar botones cuando la selección de los mismos no resultara válida, dado que no es posible agregar un vehículo a un lote si este ya pertenece a uno, por ejemplo.

Como puntos a mejorar, para este caso, se mantienen los mencionados para la entrega anterior; concretamente, la implementación de un cargado de datos que utilice *Lazy Loading* (si bien se tienen ciertas dudas de que efectivamente resulte más eficiente que el implementado), y la utilización de inyección de dependencias entre clases *Services* y *Repositories,* y entre *Controllers* de la *Web API,* y los distintos *Services* asociados, dándose ésta en ambos casos directamente en los constructores sin parámetros llamados automáticamente al recibirse una *request HTTP.* Al no ser esto, así como la cobertura de pruebas unitarias, considerado una prioridad para la presente entrega, habiéndose ya de por sí llegado bastante justos a la misma en cuanto a la implementación de la funcionalidad requerida, que se presume cubierta completamente, permanece como trabajo a llevarse a cabo a futuro en meses venideros. Sin embargo, a pesar de ello se considera, a nivel general, el diseño generado es uno de buena calidad, en líneas generales correcto según lo manejado a nivel teórico en clase, si bien como se ha visto existen algunas posibles mejoras abordables a futuro.

5. Informe de métricas

Para realizar el informe de métricas utilizamos la herramienta NDepend. La misma nos calcula las métricas en función del código, además de hacer varias cosas mas.

Consideramos que hubiese sido beneficioso haber utilizado esto desde el comienzo del proyecto ya que a medida se van implementando las distintas funcionalidades, la herramienta evalúa las dependencias y calcula estadísticos de manera que si leemos bien la información obtendremos como resultado final un sistema en el cual el código es prolijo y estable.

5.1 Cohesión relacional

La cohesión relacional mide la relación entre las responsabilidades de las clases de un mismo paquete. Las clases dentro de un mismo paquete deberían de estar fuertemente relacionadas. Sin embargo, si las clases están muy relacionadas esto tampoco es bueno. Es por esto que se establece que la cohesión relacional debe estar entre 1.5 y 4.

En nuestro caso, el promedio de cohesión relacional de todos los paquetes fue 1,6; siendo VehicleTracking.Data.DataAccess el valor más alto con 3,25 y VehicleTracking.Data.Tests el valor más bajo con 0,03.

5.2 Abstraccion

La abstracción es una relación entre el número de clases de un paquete y el número de clases abstractas en el paquete. Representa qué tan abstracto es un componente. El rango de este componente va de 0 a 1, donde 0 representa que no tiene clases abstractas y 1 a un componente que solo tiene clases abstractas.

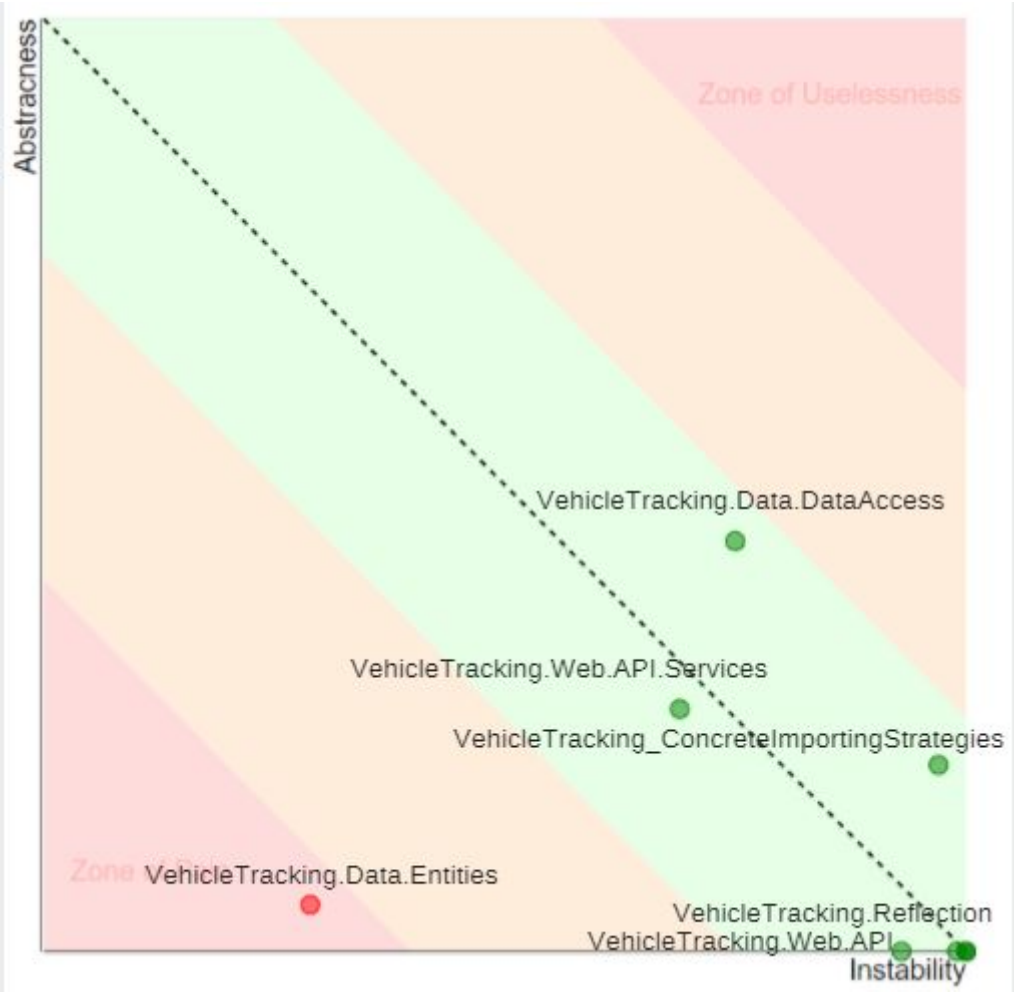
En el caso de nuestro obligatorio, todos los paquetes tienen una abstracción menor a 0.5, e incluso algunos tienen abstracción 0.

5.3 Inestabilidad

La inestabilidad representa la resistencia de un paquete a cambiar. El rango de esta métrica también va de 0 a 1, siendo 0 un paquete totalmente estable y 1 un paquete inestable.

En el caso de nuestro obligatorio, los paquetes de pruebas son los más inestables con un valor de 1. Esto se debe a que dependen de todos los paquetes para poder probar sus funcionalidades pero a su vez nadie depende de ellos. Por otro lado, el paquete más inestable es el de las entidades con un valor de 0.29. Esto se da debido a que la gran mayoría de los paquetes dependen de él.

5.4 Inestabilidad vs. abstracción



La gráfica anterior muestra la relación que hay entre la métrica de inestabilidad y la de abstracción. Lo deseable es que todos los componentes estén cerca de la secuencia principal, es decir, que estén balanceados en cuanto a su nivel de abstracción y estabilidad.

Si los componentes se encuentran en la llamada zona de dolor es porque son componentes estables pero no son abstractos, son rígidos. Esto quiere decir que son difícil de cambiar por el hecho de que son muy estables.

Si los componentes se encuentran en la llamada zona de la inutilidad, significa que son abstractos pero inestables. Estos componentes no son deseables porque por un lado son muy abstractos, incapaces de hacer nada por sí mismos, y por otro lado, no tiene responsabilidades por lo que no es referenciado por los demás componentes.

En el caso de nuestro obligatorio, todos los paquetes a excepción de uno caen dentro de la zona verde, dentro del equilibrio entre la inestabilidad y la abstracción. El que cae fuera está en la zona de dolor y es el paquete de las entidades. Suponemos que esto se debe a que al contener sólo las entidades, y de acuerdo a la forma en la cual decidimos implementar nuestra solución, no fue necesario utilizar abstracciones. De cualquier forma, al estar éstas asociadas a entidades del dominio del problema, se espera éstas cambien poco.

6. Clean code

En este proyecto hicimos uso de los estándares que establece "*Clean Code*", de Robert C. Martin, para que el código que se escribe sea más limpio y por lo tanto más mantenible. De esta manera, cuando otro desarrollador, o nosotros mismos, vuelve a leer el proyecto después de un tiempo, le será muy simple entender el código antes escrito.
Para evidenciar el uso de este libro, decidimos separarlo en los capítulos que el mismo contiene.

6.1 Nombres

Se utilizan nombres descriptivos, claros y legibles. Si es el nombre de un método, el mismo debe representar la funcionalidad del mismo.

```
private static void LoadStrategiesFromAssembly(List<IImportingStrategy> result,
    Assembly strategiesAssembly)
{
    foreach (var type in strategiesAssembly.GetExportedTypes())
    {
        if (IsValidStrategyToInstantiate(type))
        {
            IImportingStrategy strategyToReturn = AttemptToGetInstanceOf(type);
            result.Add(strategyToReturn);
        }
    }
}
```

6.2 Funciones

Se debe poder inferir su comportamiento de un vistazo. Por lo tanto, deben ser cortas, y realizar una única cosa. Deben tener buenos nombres y la mínima cantidad de argumentos, más de 3 ya viola los principios del Clean Code.

Principalmente, se debe eliminar la duplicidad, el código repetido. Si hay dos funciones que realizan lo mismo, entonces debe haber una manera de encapsularlas en una sola.

Si bien en la entrega anterior se respetó esto, en esta segunda entrega no fuimos tan cuidadosos en este aspecto. Más que nada, esto sucedió en la windows form application ya que había que realizar varias validaciones y controlar distintas excepciones por lo que obtuvimos como resultado muchos ifs anidados dentro de estructuras try-catch.

6.3 Comentarios

Si se necesita un comentario para aclarar algo entonces es señal de que se debería rediseñar.

Únicamente son apropiados para remarcar importancia de algo, aspectos legales, advertir consecuencias, por ejemplo. En el resto de los casos se puede generar confusión.

Siguiendo con la entrega anterior, se evitaron los comentarios para cumplir con este ítem que plantea Clean Code.

6.4 Formato

Se debe generar un estándar de codificación. Se puede utilizar uno ya hecho como el de Java por ejemplo, o elegir un conjunto de reglas dentro de un grupo de trabajo. En nuestro caso, se intentaron seguir las convenciones del lenguaje que establece la MSDN, utilizando *properties* en lugar de *setters* y *getters,* utilizando nombres que comenzaran por mayúsculas, etc.

La importancia no reside en qué reglas se toman, si no en que todos los que trabajen en el proyecto se atenga a las mismas.

Igualmente, si bien tuvimos esto en consideración, en esta segunda entrega no fuimos tan estrictos con este aspecto obteniendo como resultado clases que no respetan el formato que habíamos establecido o clases que tienen mezcla de cosas que sí respetan y que no respetan. Esto fue fundamentalmente en los nombres de los atributos de las clases.

6.5 Objetos y estructura de datos

Se debe diferenciar entre objetos y estructuras de datos, viendo cuándo es preferible una que la otra. Los objetos esconden sus datos y exponen funciones para utilizarlos, las estructuras de datos exponen sus datos pero no tienen funciones para manipularlos.

6.6 Procesar errores

Se debe separar el manejo de errores de la lógica del proyecto. Dicho de otra manera, se esconde la verdadera funcionalidad del código.

Se utilizan bloques try-catch que identifican los puntos del programa que pueden producir una excepción. Son menos invasivas a nivel de código que los códigos de retorno.Se deben evitar los retornos *null*.

Tratando de seguir con esto, en nuestro proyecto realizamos las validaciones a nivel de dominio lanzando excepciones, creadas por nosotros, que luego son controladas a nivel de la aplicación API REST.

En esta segunda versión también tuvimos que controlar dichas excepciones desde la windows form application mostrando un mensaje de error cuando alguna de ellas salta, evitando que el programa se caiga y se frene la ejecución del mismo.

```
private static ImportingException ErrorOnParameterInput()
{
    string errorMessage = string.Format(CultureInfo.CurrentCulture,
        ErrorMessages.IncompleteParameters, pathParameterName);
    return new ImportingException(errorMessage);
}
```

6.7 Límites

Se debe definir de forma clara la frontera entre el código y los paquetes de terceros o desarrollados por otros equipos. De esta manera, se minimizan las partes de nuestro código que dependen de elementos externos.

6.8 Pruebas unitarias

Clean Code establece que hay trabajar utilizando la metodología de TDD (Test Driven Development). De esta manera fue que se realizó la mayoría de la implementación de nuestro proyecto.

Sin embargo, sobre el final nuevamente abandonamos dicha forma de trabajo por lo que tenemos codigo sin testear.

6.9 Clases

Primero se deben declarar constantes públicas, luego las estáticas privadas, variables de instancia privadas, y por último los métodos. Además los métodos privados deben estar junto a los públicos que los utilizan.

Se debe mantener la encapsulación de las clases, pequeñas con pocas responsabilidades.

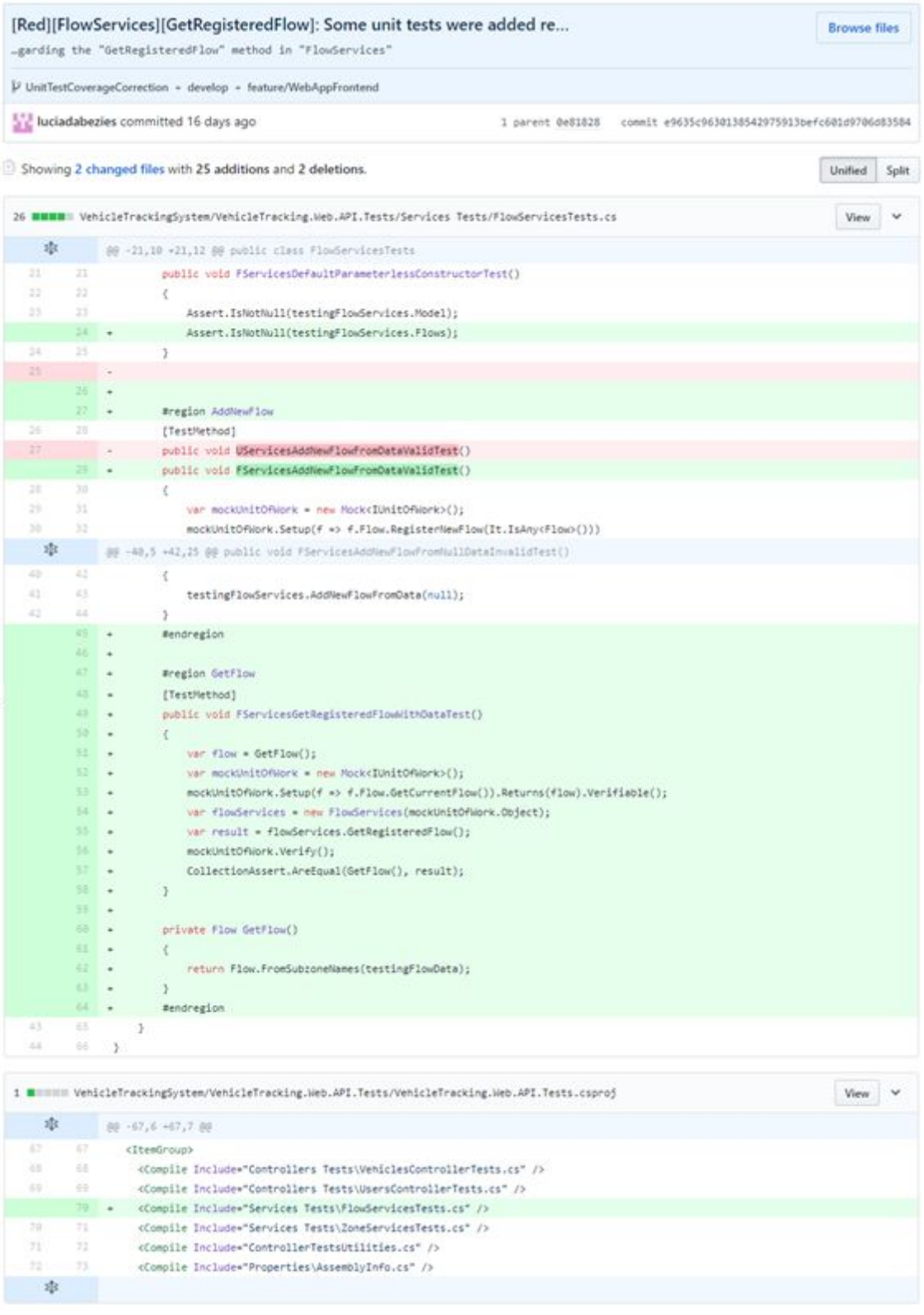
Las nuevas funcionalidades se deben introducir extendiendo el sistema, NO modificando código existente. Las clases deben depender de abstracciones, ya que están expresan conceptos, y luego las concretas que dependen de ellas tienen los detalles de implementación. De otra manera, habría mayor impacto de cambio.

7. TDD (Test Driven Development)

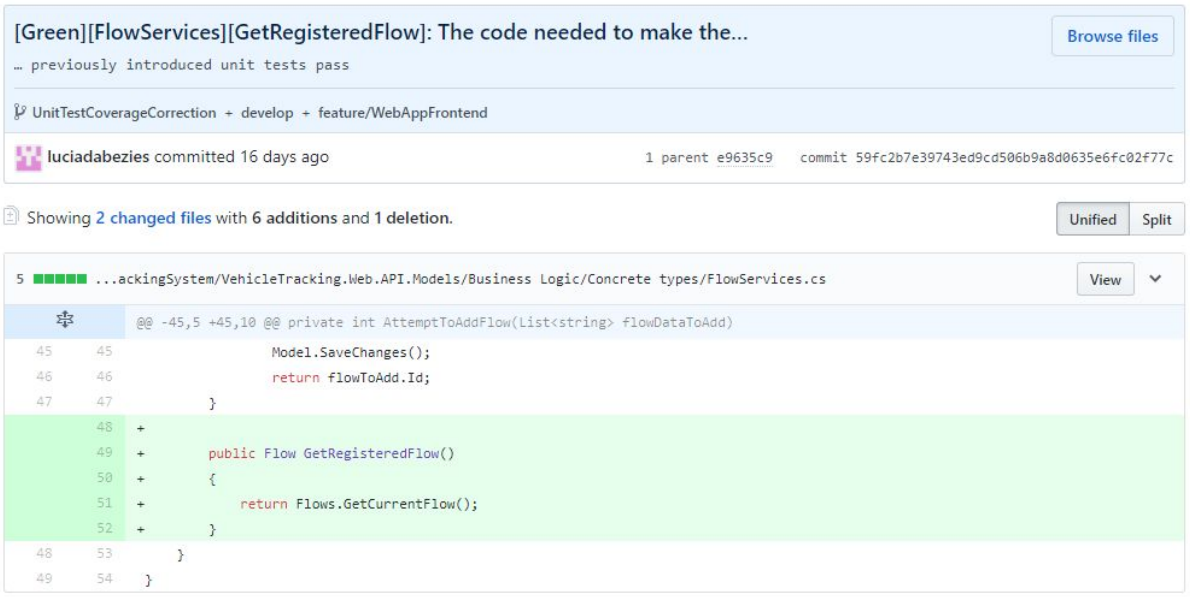
7.1 Evidencia Nº1

La siguiente funcionalidad representa la definición del flujo para la venta.

7.1.1 Etapa Red



7.1.2 Etapa Green



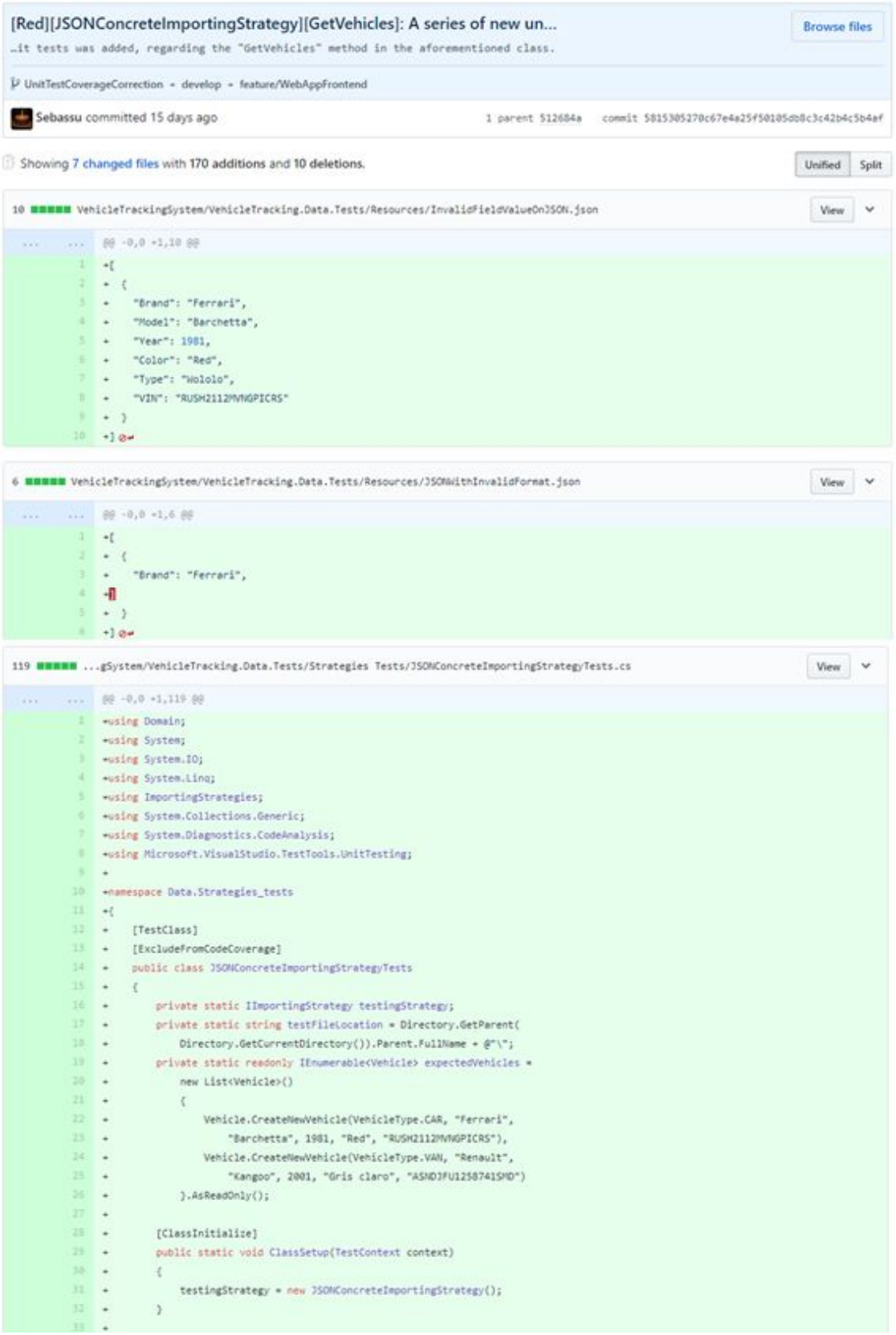
7.1.3 Etapa Refactor

En este caso no fue necesario realizar refactoring.

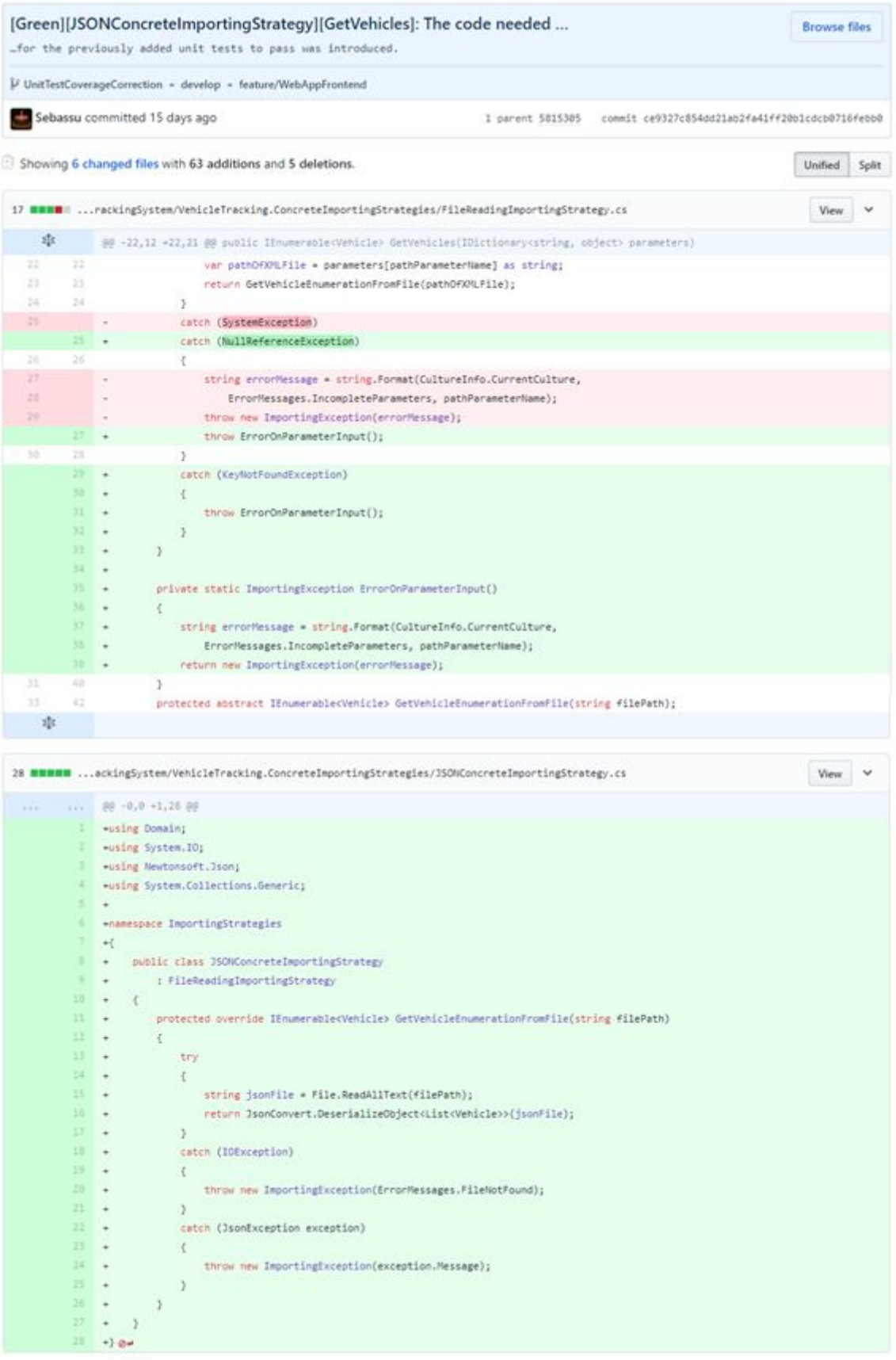
7.2 Evidencia N°2

La segunda funcionalidad representa la estrategia para importar vehículos con archivos de formato JSON.

7.2.1 Etapa Red



7.2.2 Etapa Green



[Green][JSONConcreteImportingStrategy][GetVehicles]: The code needed ...

Browse files

...for the previously added unit tests to pass was introduced.

UnitTestCoverageCorrection + develop + feature/WebAppFrontend

Sebassu committed 15 days ago — 1 parent 5815305 — commit ce9327c854dd21ab2fa41ff20b1cdcb0716fe0b0

Showing 6 changed files with 63 additions and 5 deletions. — Unified | Split

17 ...rackingSystem/VehicleTracking.ConcreteImportingStrategies/FileReadingImportingStrategy.cs — View

```
@@ -22,12 +22,21 @@ public IEnumerable<Vehicle> GetVehicles(IDictionary<string, object> parameters)
                var pathOfXMLFile = parameters[pathParameterName] as string;
                return GetVehicleEnumerationFromFile(pathOfXMLFile);
            }
-           catch (SystemException)
+           catch (NullReferenceException)
            {
-               string errorMessage = string.Format(CultureInfo.CurrentCulture,
-                   ErrorMessages.IncompleteParameters, pathParameterName);
-               throw new ImportingException(errorMessage);
+               throw ErrorOnParameterInput();
            }
+           catch (KeyNotFoundException)
+           {
+               throw ErrorOnParameterInput();
+           }
        }

+       private static ImportingException ErrorOnParameterInput()
+       {
+           string errorMessage = string.Format(CultureInfo.CurrentCulture,
+               ErrorMessages.IncompleteParameters, pathParameterName);
+           return new ImportingException(errorMessage);
+       }
        protected abstract IEnumerable<Vehicle> GetVehicleEnumerationFromFile(string filePath);
```

28 ...ackingSystem/VehicleTracking.ConcreteImportingStrategies/JSONConcreteImportingStrategy.cs — View

```
@@ -0,0 +1,28 @@
+using Domain;
+using System.IO;
+using Newtonsoft.Json;
+using System.Collections.Generic;
+
+namespace ImportingStrategies
+{
+    public class JSONConcreteImportingStrategy
+        : FileReadingImportingStrategy
+    {
+        protected override IEnumerable<Vehicle> GetVehicleEnumerationFromFile(string filePath)
+        {
+            try
+            {
+                string jsonFile = File.ReadAllText(filePath);
+                return JsonConvert.DeserializeObject<List<Vehicle>>(jsonFile);
+            }
+            catch (IOException)
+            {
+                throw new ImportingException(ErrorMessages.FileNotFound);
+            }
+            catch (JsonException exception)
+            {
+                throw new ImportingException(exception.Message);
+            }
+        }
+    }
+}
```

7.2.3 Etapa Refactor

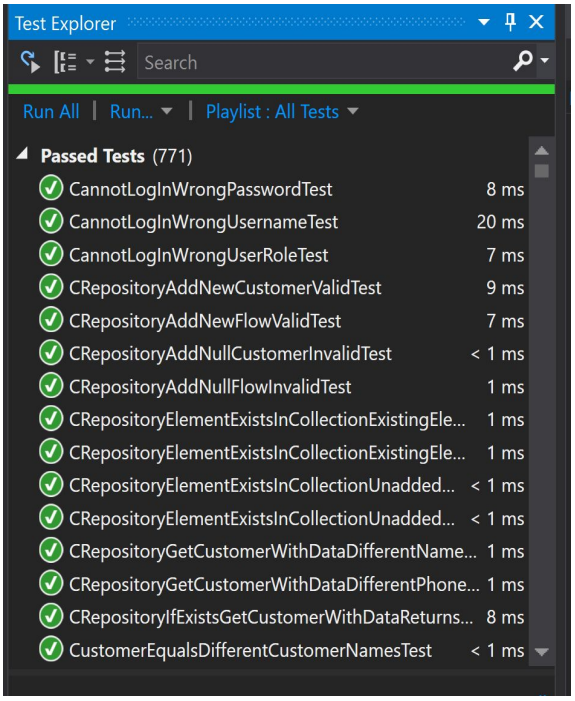
En este caso tampoco fue necesario utilizar refactoring.

8. Cobertura de pruebas

Para esta nueva entrega tratamos de aumentar la cobertura de pruebas del sistema. Lo primero que hicimos antes de trabajar con las nuevas funcionalidades fue agregar pruebas de métodos que antes no habían sido probados.

Luego, como la consigna especificaba que se debía utilizar TDD, probamos varias de las nuevas funcionalidades, haciendo especial énfasis en las pruebas asociadas a entidades del dominio y la persistencia de las mismas, aspectos de la solución considerados claves, y más propensos a errores. Sin embargo, al igual que en la entrega anterior, sobre el final del plazo de entrega abandonamos dicha metodología lo cual, como resultado, hizo que bajara la cobertura. Si bien el porcentaje de código probado es mayor, sigue siendo bajo según el criterio manejado (80% como mínimo aceptable).

A continuación se muestra la cantidad de pruebas realizadas así como el porcentaje de cobertura de código que representan las mismas.



Sebastián_LAPTOP 2017-11-23 19_52_49.co

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
Sebastián_LAPTOP 2017-11-23 19_52_49.coverage	1245	27.74 %	3243	72.26 %
vehicletracking.data.dataaccess.dll	18	2.19 %	803	97.81 %
vehicletracking.data.entities.dll	0	0.00 %	1500	100.00 %
vehicletracking.reflection.dll	0	0.00 %	37	100.00 %
vehicletracking.web.api.services.dll	813	53.38 %	710	46.62 %
vehicletracking_concreteimportingstrategies.dll	0	0.00 %	112	100.00 %
vehicletrackingsystem.web.api.dll	414	83.64 %	81	16.36 %

9. Evaluación del proyecto

En líneas generales, se considera que el proceso de trabajo llevado a cabo como parte de este obligatorio fue correcto, en el cual ambos integrantes se encuentran bastante conformes con el resultado final obtenido.

A diferencia del obligatorio anterior, en el cual el producto a entregarse era único (la solución contenedora de la *Web API*), este contenía varias partes bastante independientes entre ellas; el caso más claro siendo la interfaz *Web* y de escritorio. Esto permitiría en cierto modo una división de trabajo más clara, que además se profundizó debido a ciertos problemas que tuvo uno de los integrantes al lograr ejecutar la *Web API,* obteniendo por alguna razón no descubierta hasta el momento, errores 404 *Not found* al enviarle *requests HTTP.*
A pesar de ello, sin embargo, se intentaría en líneas generales trabajar en conjunto (colaborando cara a cara, factor clave del desarrollo según plantean los principios *agile*).

Afortunadamente no se tendría que realizar mayores modificaciones a lo entregado al momento del primer obligatorio, más aún así el proceso de trabajo fue bastante ajustado en cuanto a tiempo, debido a la extensión del obligatorio en conjunto con lo requerido por las restantes asignaturas. Más allá de esto, se presume (salvando algún posible *bug* no detectado) la funcionalidad fue implementada en su totalidad, incluso realizando *TDD* en etapas iniciales del proyecto, inclusive para temas de *Reflection* y las estrategias de importación, abandonándose al igual que para el primer obligatorio en fases más tardías. Por esto quizás la cobertura de pruebas unitarias no es la mejor, punto que también resulta entonces mejorable a futuro. A pesar de ello, sin embargo, y considerando además los desafíos que planteó este obligatorio en cuanto a la utilización de nuevas tecnologías (*Angular,* por ejemplo), se está bastante satisfecho con el resultado obtenido y el proceso de trabajo llevado a cabo, que en líneas generales se estima correcto.