Universidad ORT Uruguay
Facultad de Ingeniería

GoData

Arquitectura de Software - Obligatorio 1

Brahian Calo - 170540

Ramiro Gonzalez - 167011

Martín Comesaña - 183579

Grupo N7A

Docentes:

Andrés Calviño López

María Gimena Bernadet

Guillermo Repetto

Junio 2020

ÍNDICE

1. Introducción

El alcance de este documento es lograr mostrar cómo es la arquitectura propuesta para el sistema GoData. Cada una de las secciones de este documento de arquitectura provee información útil para que los usuarios de dicho documento puedan entender la arquitectura del sistema.

2. Propósito

El propósito del presente documento es proveer una especificación completa de la arquitectura del sistema GoData.

3. Propósito del sistema

Brindar un servicio dirigido a desarrolladores de aplicaciones o empresas con el que puedan recibir datos procesados sobre multas por mal estacionamiento en la ciudad de Nueva York con el fin de que puedan agregar valor a los mismos y venderlos a diferentes posibles interesados.

GoData recibe por medio de NYCDataAgregator la información cruda de multas reportadas por los diferentes agentes que trabajan en la ciudad. Ésta información es luego derivada a GoDataSync, quien se encarga de validar los campos provenientes en las multas, almacenarlas en el repositorio IssuesHistory y luego derivarlas a los componentes GoDataSyncReports y GoDataTransformation para comenzar a procesarlas.

GoDataSyncReports se encarga de alimentar de datos al repositorio IssueReports quien permite resolver las consultas que se solicitan a GoDataQueryTool.

GoDataQueryTool es una web api que ofrece a los clientes una interfaz para poder realizar consultas sobre estadísticas de multas.

El propósito de GoDataTransformation es procesar las multas para los distintos clientes que se registran a GoData por medio de la web api Registry. Los criterios de procesamiento varían dependiendo del cliente y se basan en los filtros que especifica cada uno a la hora de registrarse en el sistema.

Registry es una web api que ofrece una interfaz para el registro de clientes, y los registra en el repositorio ClientRegistry.

4. Requerimientos significativos de Arquitectura

4.1. Resumen de Requerimientos Funcionales

ID Requerimiento	Descripción	Actor
RF1 - Recepción de multas	GoData se debe poder registrar a NYCDataAgregator para recibir multas	GoData
RF2 - Registro de clientes	Permite que aplicaciones clientes se puedan registrar para recibir multas de acuerdo a determinados parámetros (ejemplo: cambiar formato de las fechas)	Aplicación Cliente
RF3 - Envío de datos procesados a las aplicaciones registradas	Permite transformar/validar información sobre multas que llegan al sistema (RF1) de acuerdo a los parámetros de las aplicaciones clientes.	GoData
RF4 - Generación de reportes para aplicación de terceros	GoData debe ofrecer una API REST que permita a los clientes realizar consultas complejas sobre los datos históricos de multas procesados.	Aplicación Cliente

4.2. Resumen de Requerimientos No Funcionales

ID Requerimiento	Descripción
RNF1 - Registrar eventos	El sistema debe proveer suficiente información que permita un análisis detallado sobre los eventos del sistema. Sean estos normales o fallas que puedan suceder.
RNF2 - Demostrar identidad ante clientes	El sistema debe poseer un mecanismo que permita a las aplicaciones de clientes verificar que los datos que recibe son enviados por GoData y no de un origen desconocido.
RNF3 - Manejo de carga	El sistema debe lograr la mayor capacidad de procesamiento posible sin pérdida de datos y logrando la mejor latencia posible.
RNF4 - Manejo de carga QueryTool	El promedio de latencia para las consultas complejas sobre GoDataQueryTool debe ser menor a 2 segundos.
RNF5 - Incorporación de nuevos procesamientos de datos	El diseño del sistema deberá permitir incorporar a futuro nuevas validaciones y transformaciones a los datos que envía a las aplicaciones registradas, con el menor costo posible.
RNF6 - Cambio en la interfaz del servicio StateData	Se debe poder cambiar en tiempo de inicialización la URI que se utiliza para invocar el servicio externo StateData
RNF7 - Cambios a parámetros en interfaz de Servicio	El diseño deberá permitir realizar cambios en los parámetros especificados para la interfaz de servicio en tiempo de ejecución.

4.3. Resumen de Requerimientos de Atributos de Calidad

ID Requerimiento	ID Requerimiento de Calidad o restricción	Descripción
RF1 - Recepción de multas	AC1 - Seguridad	Se debe aplicar un mecanismo de seguridad que permita garantizar que solamente GoData sea quien pueda registrar y modificar su registro en NYCDataAgregator
RF1 - Recepción de multas	AC2 - Seguridad	Se debe aplicar un mecanismo de seguridad que permita asegurar que las multas recibidas provengan de NYCDataAgregator y no de desconocidos.
RF1 - Recepción de multas RF3 - Envío de datos procesados a las aplicaciones registradas RNF3 - Manejo de carga	AC3 - Eficiencia	Se debe garantizar que no se pierdan multas por falta de capacidad de procesamiento.
RF1 - Recepción de multas	AC4 - Disponibilidad	La recepción del sistema debe tener alta disponibilidad asegurando que no se pierdan multas enviadas por NYCDataAgregator.

RF1 - Recepción de multas	AC5 - Interoperabilidad	La recepción de multas debe ser genérica de modo que sea posible en tiempo de compilación integrar nuevos sistemas proveedores de multas y debe tener la capacidad de poder interpretar los diferentes formatos de orígen.
RF2 - Registro de clientes	AC6 - Seguridad	Se debe aplicar un mecanismo de seguridad para que GoData pueda comprobar la identidad de los clientes en el momento del registro o modificación del mismo.
RF2 - Registro de clientes	AC7 - Modificabilidad	El diseño de envío de datos a las aplicaciones cliente debe poder ser extendido para permitir la utilización de otros protocolos de comunicación
RF2 - Registro de clientes	AC8 - Usabilidad	El sistema debe permitir a los clientes modificar sus parámetros de registro en tiempo de ejecución
RF2 - Registro de clientes	AC9 - Interoperabilidad	El sistema debe proveer información clara y detallada sobre cómo espera que los clientes registren sus validaciones y transformaciones, así como también cómo deben indicar el formato de datos a esperar y cómo deben registrar su endpoint al que serán enviados los datos.
RF3 - Envío de datos procesados a las aplicaciones registradas RNF2 - Demostrar identidad ante clientes	AC10 - Seguridad	Se debe aplicar un mecanismo de seguridad que permita garantizar a los clientes que la información de multas proviene de GoData
RF3 - Envío de datos procesados a las aplicaciones registradas RNF1 - Registrar	AC11 - Disponibilidad	El sistema debe poder detectar defectos y registrarlos para futuro análisis.

eventos		
RF3 - Envío de datos procesados a las aplicaciones registradas	AC12 - Disponibilidad	El sistema debe poder continuar operando aún ante defectos menores detectadas.
RF3 - Envío de datos procesados a las aplicaciones registradas RNF5 - Incorporación de nuevos procesamientos de datos	AC13 - Modificabilidad	El diseño de validaciones y transformaciones debe ser fácilmente extensible para permitir nuevas implementaciones.
RF4 - Generación de reportes para aplicación de terceros	AC14 - Interoperabilidad	El sistema debe proveer información clara y detallada sobre cómo espera que los clientes realicen la petición de consulta, en qué formato recibirán el resultado y cómo debe ser interpretado.
RF4 - Generación de reportes para aplicación de terceros	AC15 - Modificabilidad	El sistema debe ser extensible para permitir generar nuevos tipos de consultas fácilmente.
RF4 - Generación de reportes para aplicación de terceros	AC16 - Eficiencia	Las consultas complejas deben tener un promedio de latencia menor a 2 segundos
RF4 - Generación de reportes para aplicación de terceros	AC17 - Seguridad	El sistema debe poder identificar y autorizar a los clientes que realicen las consultas.

RNF6 - Cambio en la interfaz del servicio StateData	AC18 - Modificabilidad	El diseño debe permitir cambiar en tiempo de inicialización la URI que se utiliza para invocar el servicio externo StateData
RNF7 - Cambios a parámetros en interfaz de Servicio	AC19 - Modificabilidad	El diseño deberá permitir fácilmente a los clientes realizar cambios en los parámetros especificados para la interfaz de servicio en tiempo de ejecución.
RNF4 - Manejo de carga QueryTool	AC20 - Eficiencia	La latencia promedio para las consultas complejas debe ser menor a 2 segundos.

4.4. Resumen de Restricciones

ID Restriccion	Descripción
R1 - Tecnologia	La solución debe ser implementada usando NodeJS.
R2 - Control de Versiones	Se debe utilizar un repositorio en GitHub dentro de la organización de “ORT-ArqSoft”
R3 - StateData	Se debe obtener la descripción textual de los códigos Registration State y Plate Type por medio de la API REST provista por StateData, que es un servicio externo.

5. Documentación de la arquitectura

5.1. Vistas de Módulos

A continuación se documentaran las vistas de módulo mostrando los elementos de GoData y sus agrupaciones lógicas..

Las vistas que se mostraran son:

- Vista de Descomposición
- Vista de Uso General
- Vista de Uso de Logger
- Vista de uso del módulo Transformaciones

5.1.1. Vista de Descomposición

5.1.1.1. Representación primaria

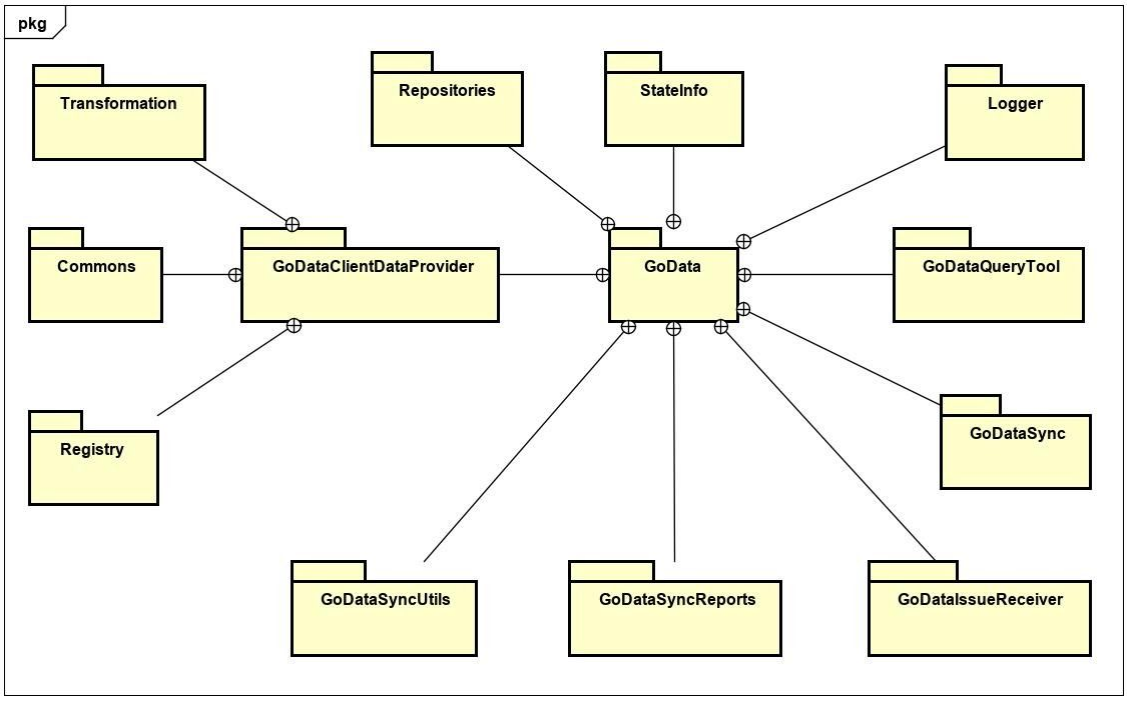

Figura 5.1.1. - Diagrama de descomposición de alto nivel de GoData

5.1.1.2. Catálogo de elementos

GoData

Este módulo agrupa todo los elementos de la solución de GoData.

GoDataClientDataProvider

Este módulo agrupa todo los elementos que interactúan con las aplicaciones clientes de GoData. La api de registro para clientes *Registry*, las transformaciones de los datos *Transformation* y código compartido entre ellos *Commons*.

Registry

Este módulo agrupa las unidades de implementación para la Api de registro de clientes. Las funcionalidades que provee son las siguientes:

- Registrar el cliente
- Modificar registros de clientes
- Consultar clientes registrados

Transformation

Este módulo agrupa las unidades de implementación que transforman y envían datos a los clientes registrados.

Commons

Este módulo agrupa las unidades de implementación que tienen en común *Registry* y *Transformation.*

GoDataQueryTool

Este módulo provee un servicio web que permite consultar reportes de las multas procesadas por GoData en diferentes formatos.

GoDataIssueReceiver

Este módulo agrupa las unidades de implementación para poder recibir multas por parte de NYCDataAgregator.

Repositories

Este módulo agrupa las unidades de implementación para la persistencia en las bases de datos de lectura y escritura.

GoDataSync

Este módulo es la agrupación lógica referente al proceso que se encarga de orquestar las diferentes tareas que implica GoData como validar y persistir multas como histórico y redirigir información a demás procesos.

GoDataSyncReports

Este módulo contiene la agrupación lógica del proceso que se encarga de persistir multas en una base de datos de lectura.

GoDataSyncUtils

Contiene funciones comunes para los procesos de sincronización utilizadas para validar multas.

Logger

Este módulo agrupa las unidades de implementación enfocadas en las tareas de logging del sistema.

StateInfo

Este módulo agrupa las unidades de implementación involucradas en la obtención y caching de información obtenida consultando StateInfo.

5.1.1.3. Vistas relacionadas

- Vista de uso general

5.1.2. Vista de uso general

5.1.2.1. Representación primaria

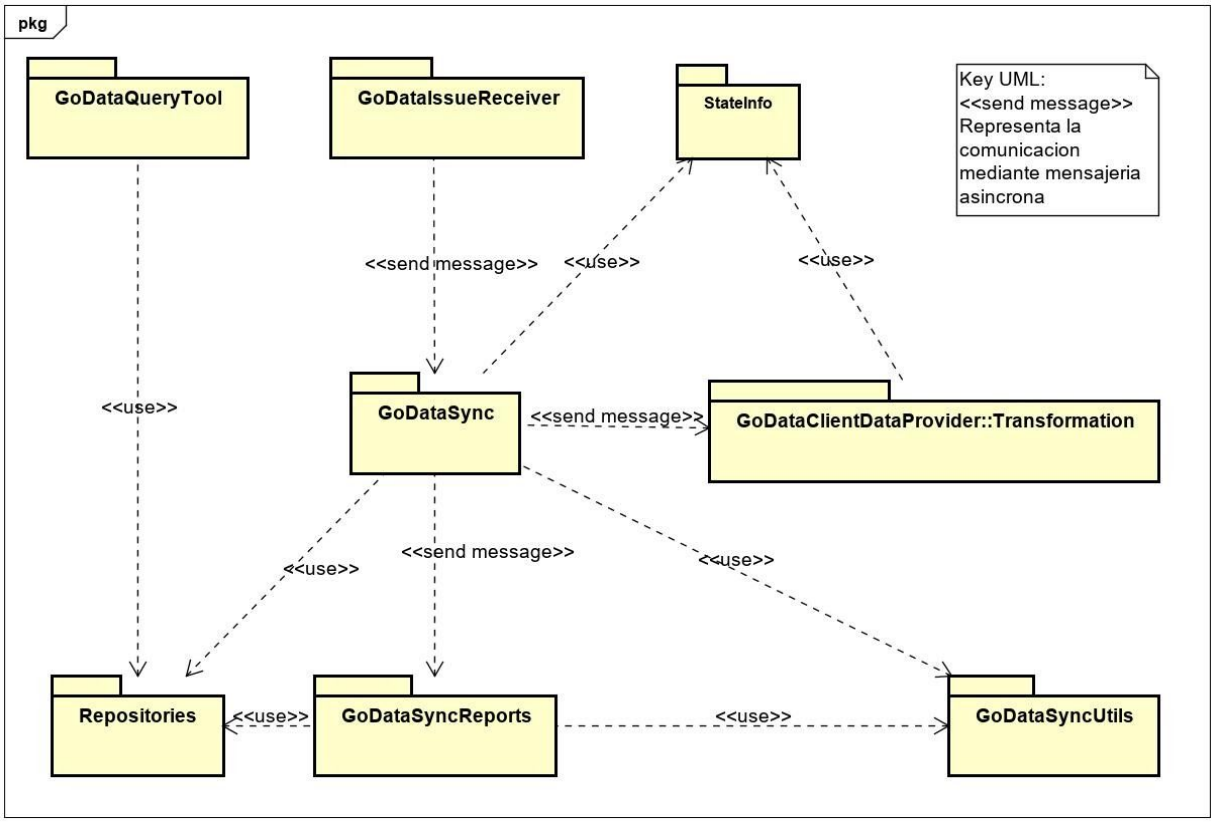

Figura 5.1.2. - Diagrama de uso de alto nivel de GoData

5.1.3. Vista de uso de Logger

5.1.3.1. Representación primaria

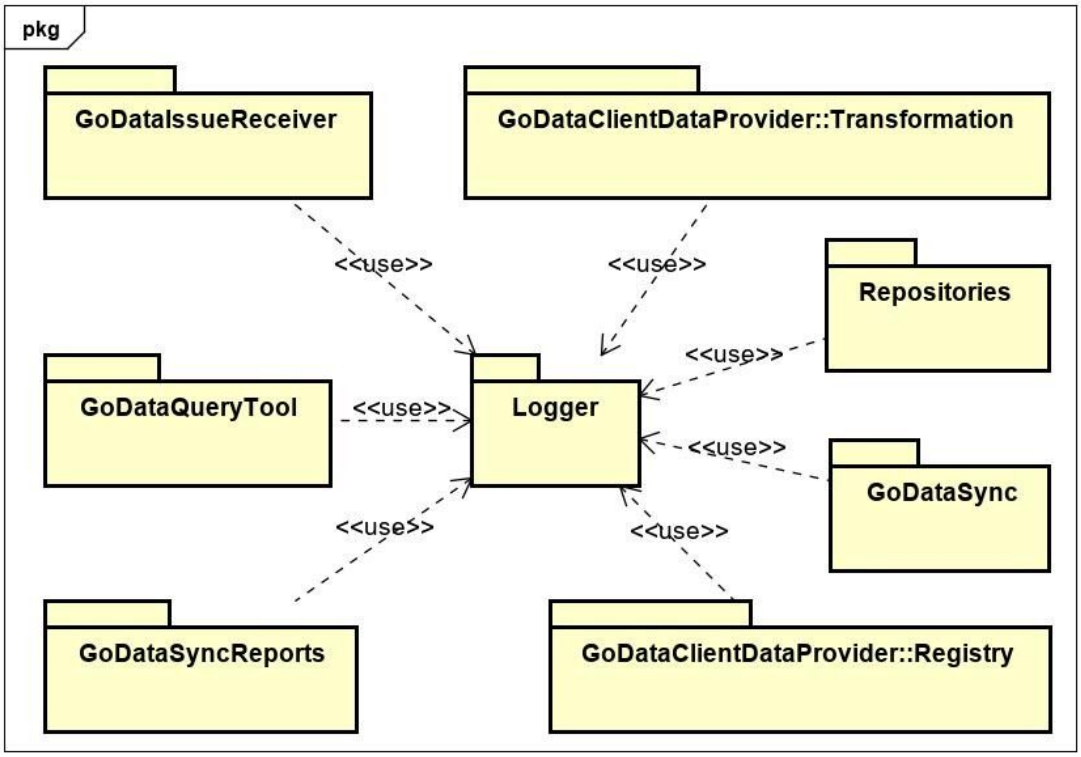

Figura 5.1.3. - Diagrama de uso del módulo de Logger

5.1.3.2. Decisiones de diseño

Implementación

Se implementó una librería con el objetivo de que sea fácil de utilizar. El módulo Logger es un módulo utilizado por todos los módulos de GoData con el fin de imprimir información relevante a la hora de realizar acciones. Se aplicó el patrón strategy con el fin de brindar una solución abierta a la modificación para futuras implementaciones. La abstracción se da a través de una interfaz provista por el módulo Logger.

Mecanismo de importación

Al requerir el módulo Logger, se podrán acceder a sus funcionalidades.

5.1.4. Vista de uso del módulo Transformaciones

5.1.4.1. Representacion Primaria

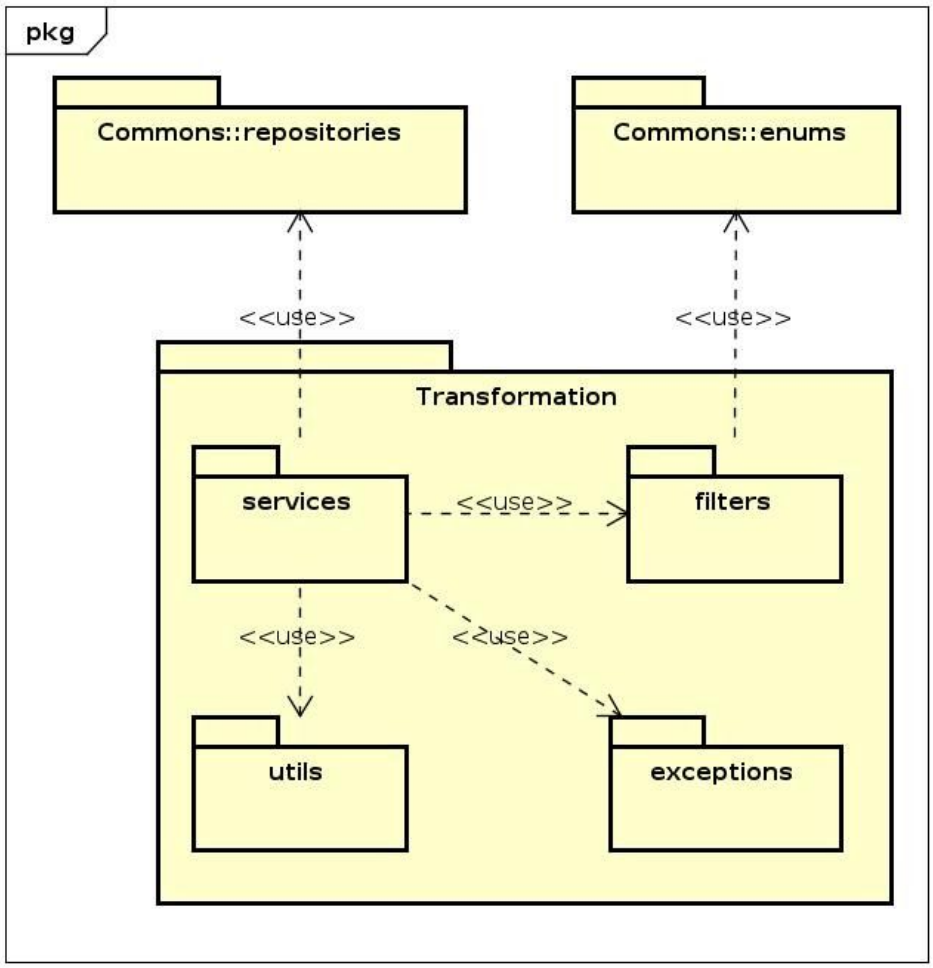

Figura 5.1.4. - Diagrama de uso del módulo Transformation

5.1.4.2. Catalogo de elementos

services

Este módulo agrupa las unidades de implementación de lógica de negocio para la transformación de datos.

filters

Este módulo agrupa las unidades de implementación de los filtros (código que realiza las transformaciones).

exceptions

Este módulo agrupa las clases de excepciones que se manejan dentro de Transformation.

utils

Este módulo agrupa unidades de implementación de uso común dentro de Transformaciones

Commons::repositories

Este módulo agrupa las unidades de implementación relacionadas a la persistencia de las aplicaciones clientes de GoData

Commons::enums

Este módulo agrupa las unidades de implementación de los enums utilizados por los filtros.

5.1.4.3. Decisiones de diseño

Cada transformación descrita en RF2 fueron implementadas como filtros individuales, cada filtro desconoce la existencia de otros filtros, tienen una única responsabilidad bien definida (la tarea que realiza) y además obedecen una interfaz genérica que baja el acoplamiento de los mismos. Se encapsuló la lógica de envío de datos para que los cambios en dicha lógica no afecten a los que la usan.

El costo de incluir un nuevo tipo de transformación es muy bajo porque solamente se debe proveer la implementación en el módulo filters, service no se ve impactado porque se acopla a la interfaz general de los filtros

5.1.4.4. Vistas relacionadas

- Vista de Descomposición
- Vista de estructura interna de GoDataTransformation

5.2. Vistas de Componentes y conectores

A continuación se documentaran las vistas de componentes y conectores evidenciando así la presencia de componentes y de qué forma sucede la comunicación entre ellos en tiempo de ejecución.

Las vistas que se mostraran son:

- Vista de contexto de GoData
- Vista de estructura interna de GoData
- Vista de estructura interna de GoDataTransformation
- Vista del patrón CQRS

5.2.1. Vista de contexto de GoData

5.2.1.1. Representación primaria

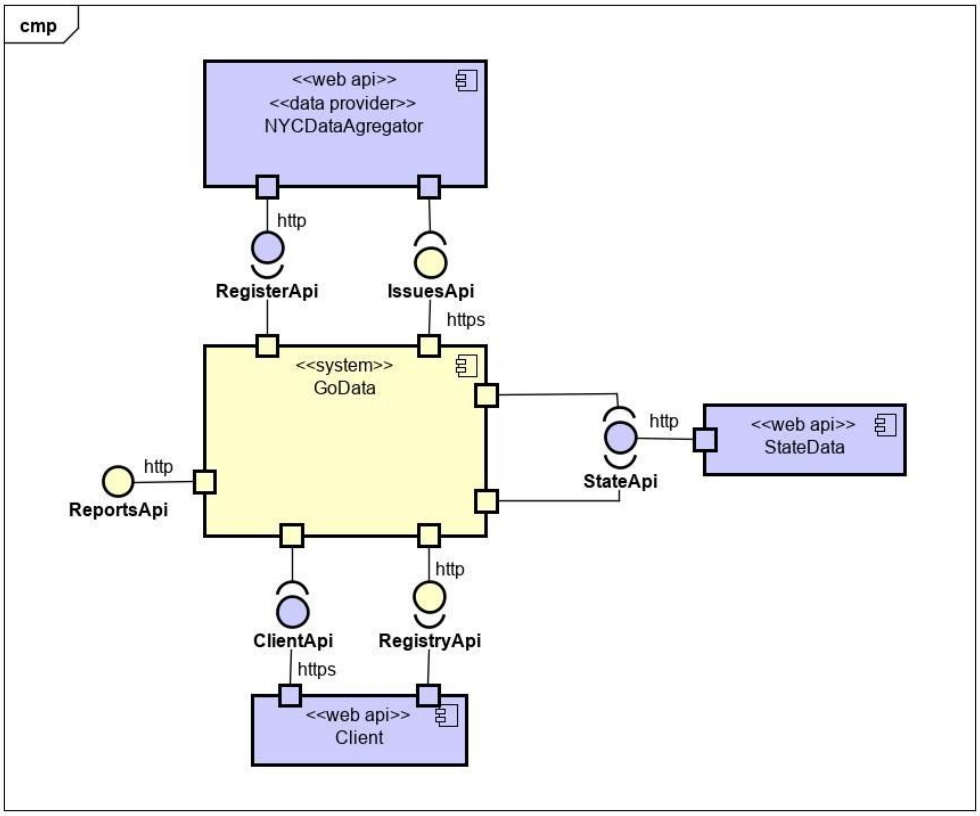

Figura 5.2.1. - Diagrama de componentes de alto nivel de GoData

5.2.1.2. Catálogo de elementos

GoData

Representa al sistema de GoData y todos los componentes que lo forman.

NYCDataAgregator

Este componente es una aplicación web que expone una interfaz para que otros sistemas se puedan registrar (ejemplo. GoData). A su vez envía información sobre las multas a GoData usando la interfaz que el mismo específico en el registro.

State Data

Este componente es una aplicación web hecha por terceros que GoData utiliza para consultar información adicional sobre los datos de las multas que recibe.

Client

Este componente representa aplicaciones web que se registran a GoData para recibir las multas que GoData procesa a través de la interfaz registrada y con las transformaciones y validaciones que la misma especifique.

5.2.1.3. Vistas relacionadas

- Vista de estructura interna de GoData

5.2.2. Vista de estructura interna de GoData

5.2.2.1. Representación primaria

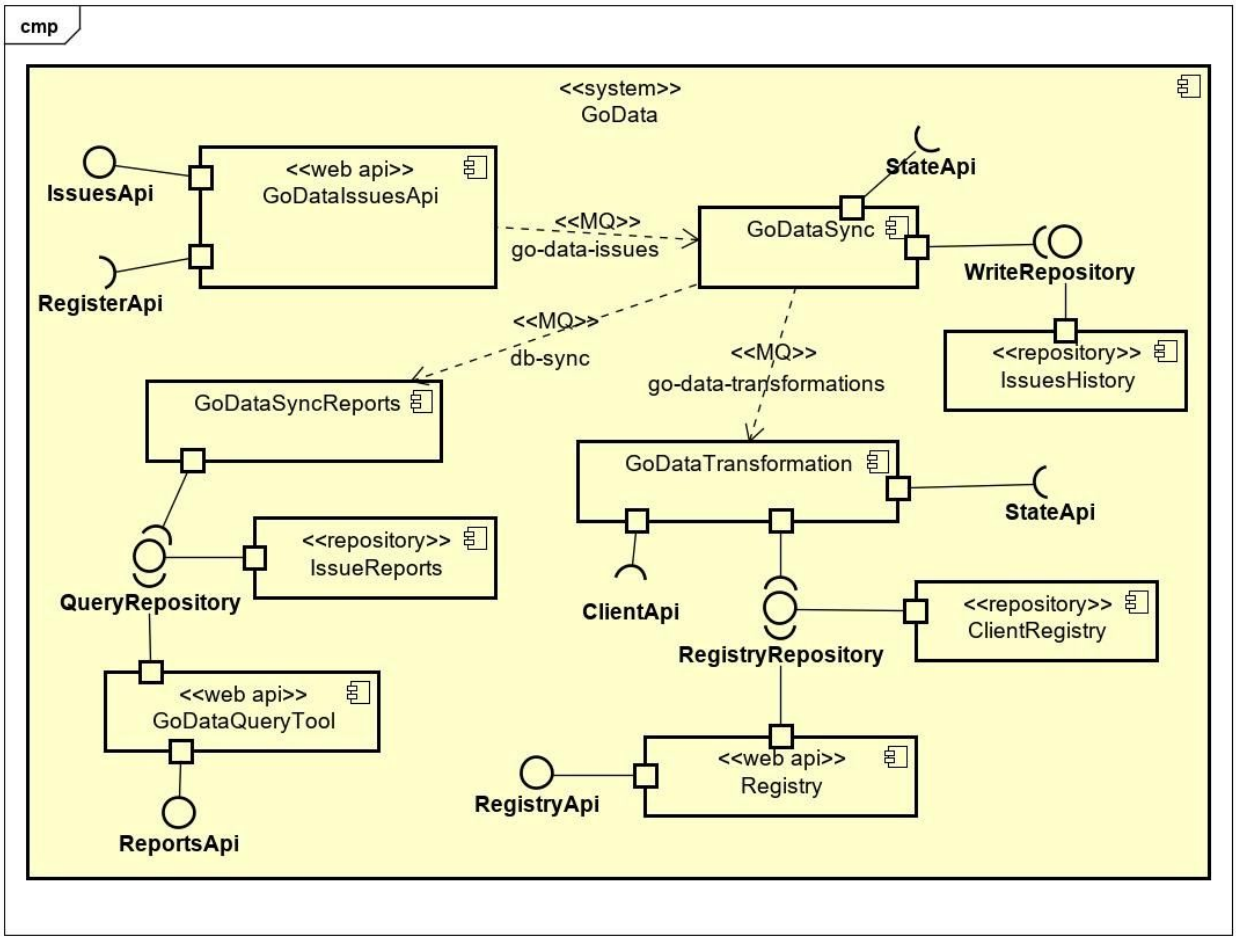

Figura 5.2.2. - Diagrama de componentes de alto nivel de GoData

5.2.2.2. Catálogo de elementos

GoDataIssuesApi

Este componente es una aplicación web hecha con express que se encarga de exponer una API REST para que NYCDataAgregator envie informacion de las multas. Las multas recibidas se mandan a una cola de mensajes llamada go-data-issues.

Registry

Este componente es una aplicación web hecha con express que se encarga de recibir peticiones de registro por parte de aplicaciones clientes para que los mismos puedan recibir información de multas. Cada aplicación cliente puede especificar qué transformaciones desea realizar a la información de las multas que la infraestructura de GoData recibe.

GoDataTransformation

Este componente es un proceso que se encarga de aplicar transformaciones y validaciones a las multas que GoDataSync le provee. Para cada aplicación cliente registrada se aplican las transformaciones y validaciones que solicitaron y al final de del proceso se el resultado a cada cliente.

ClientRegistry

Este componente es un repositorio que se encarga de persistir la información de las aplicaciones clientes que se registran para recibir multas. El servidor de base de datos es Redis.

GoDataSync

Este componente es un proceso encargado de orquestar una serie de funcionalidades con las multas provistas por GoDataIssuesApi mediante la cola de mensajes go-data-issues. Sus funcionalidades son las siguientes:

- Validar la integridad de las multas recibidas.
- Persistir la información procesada en una base de datos relacional de escritura mediante el esquema IssuesHistory.
- Enviar las multas persistidas con éxito a una cola de mensajes llamada go-data-transformations.
- Formatear multas y las envía a una cola de mensajes llamada db-sync.

IssuesHistory

Este componente es un repositorio únicamente de escritura que se encarga de persistir la información de las multas procesadas por el componente GoDataSync poder tener integridad de identidad sobre las multas. El servidor de base de datos MySQL configurado para usar el motor InnoDB.

GoDataSyncReports

Este componente es un proceso que recibe multas con la información necesaria para la correcta generación de reportes. El proceso valida que el formato de las multas recibidas sea el correcto, vuelva a aplicar un formateo a cada multa adaptado a la información de reportes a generar, para luego persistir las multas en una base de datos no relacional mediante el esquema IssuesReports.

GoDataQueryTool

Es una aplicación web utilizando el web framework koa.js que se encarga de exponer una API REST para que clientes puedan consultar tres tipos de reportes diferentes que consumen la información persistida el repositorio IssueReports

IssueReports

Este componente es un repositorio únicamente de lectura que se encarga de persistir la información de las multas procesadas por el componente GoDataSyncReports. Los datos persistidos generan una fuente de datos a ser provista para la generación de reportes. El servidor de base de datos utilizado es MongoDb.

5.2.2.3. Interfaces

IssuesApi

Es el punto de entrada al sistema dónde se reciben las multas enviadas por los proveedores. Consta de una interfaz expuesta por medio de una API REST que dispone del endpoint “/issues” al que son enviadas las multas.

RegisterApi

Es la interfaz de registry que utiliza GoData por medio de HTTP para registrar su endpoint con el fin de que los proveedores le envíen registros de multas.

StateApi

Interfaz requerida por GoDataSync y GoDataTransformation para obtener la descripción textual de los códigos Registration State y Plate Type provenientes en las multas.

WriteRepository

Interfaz provista por IssuesHistroy y utilizada por GoDataSync para almacenar en repositorio las multas que arribaron al sistema y fueron previamente validadas por GoDataSync.

ReadRepository

Interfaz requerida por GoDataQueryTool para realizar consultas al repositorio dedicado a responder consultas complejas sobre estadísticas de multas.

ReportsApi

Interfaz ofrecida a clientes externos por medio de HTTP para realizar consultas estadísticas complejas sobre las multas procesadas por el sistema.

ClientApi

Interfaz requerida por la que GoDataTransformation envía multas procesadas a los clientes registrados en el sistema.

RegistryApi

Interfaz ofrecida por medio de una API REST para que los clientes del sistema se registren o modifiquen su registro para recibir multas procesadas según los criterios indicados al momento del registro o modificación del mismo.

RegistryRegistry

Interfaz requerida por GoDataTransformation para acceder a la lista de clientes registrados en el sistema con el fin de conocer sus preferencias de filtros y validaciones para procesar las multas según estos criterios. Esta interfaz es requerida también por Registry para poder almacenar o modificar los registros de clientes.

5.2.2.4. Guía de Variabilidad

Agregar o modificar aplicaciones clientes

Las aplicaciones clientes pueden tanto registrarse como modificar sus parámetros de registro a través de la interfaz *RegistryApi*. Esto hace que los clientes automáticamente reciban multas procesadas a partir de ese momento.

5.2.2.5. Decisiones de diseño

Para satisfacer el requerimiento no funcional RNF3 y el atributo de calidad AC3 de eficiencia se utilizaron las siguientes tácticas:

Limitar respuesta de eventos

Implementado con la utilización de colas de mensajes como forma de comunicación entre los diferentes componentes y como buffer para permitir que los datos sean procesados al ritmo de cada componente. De ese modo se evita perder información por falta de capacidad de procesamiento.

Concurrencia

Todo componente puede ser lanzado como varios procesos independientes en un mismo servidor con el fin de lograr procesar una mayor cantidad información en el mismo período de tiempo. Esto es posible gracias a la utilización de colas como medio de comunicación entre componentes.

Mantener múltiples copias de datos

En los componentes dónde es necesario realizar consultas con una frecuencia muy alta sobre datos que cambian poco, se utiliza una caché en memoria que permita reducir la contención de recursos que podría ocurrir si se estuviera consultando las fuentes originales.

Los componentes que utilizan caché son:

- ClientRegistry: Mantiene información del registro de clientes y es utilizado por GoDataTransformations para conocer las validaciones y filtros que requiere cada cliente.
- GoDataSync y GoDataTransformations la utilizan por medio de la librería “StateInfo” para obtener información ofrecida por StateData.

Interoperabilidad

Para satisfacer RF2 se decidió implementar una interfaz de registro para que las clientes le especifique a GoData.

Disponibilidad

Para satisfacer el atributo de calidad AC4 que trata sobre la alta disponibilidad del sistema en la recepción de multas se aplicó la táctica de “Aumentar el conjunto de competencias”.

Se implementó en el componente GoDataIssuesApi dotándolo de la capacidad de almacenar multas en el File System para los casos en que suceda que Redis se encuentre caído. Para recuperarlas, se dispone de un script encargado de levantarlas del File System y encolarlas en Redis.

5.2.2.6. Vistas relacionadas

- Vista de estructura interna de GoDataTransformation

5.2.3. Vista de estructura interna de GoDataTransformation

5.2.3.1. Representación primaria

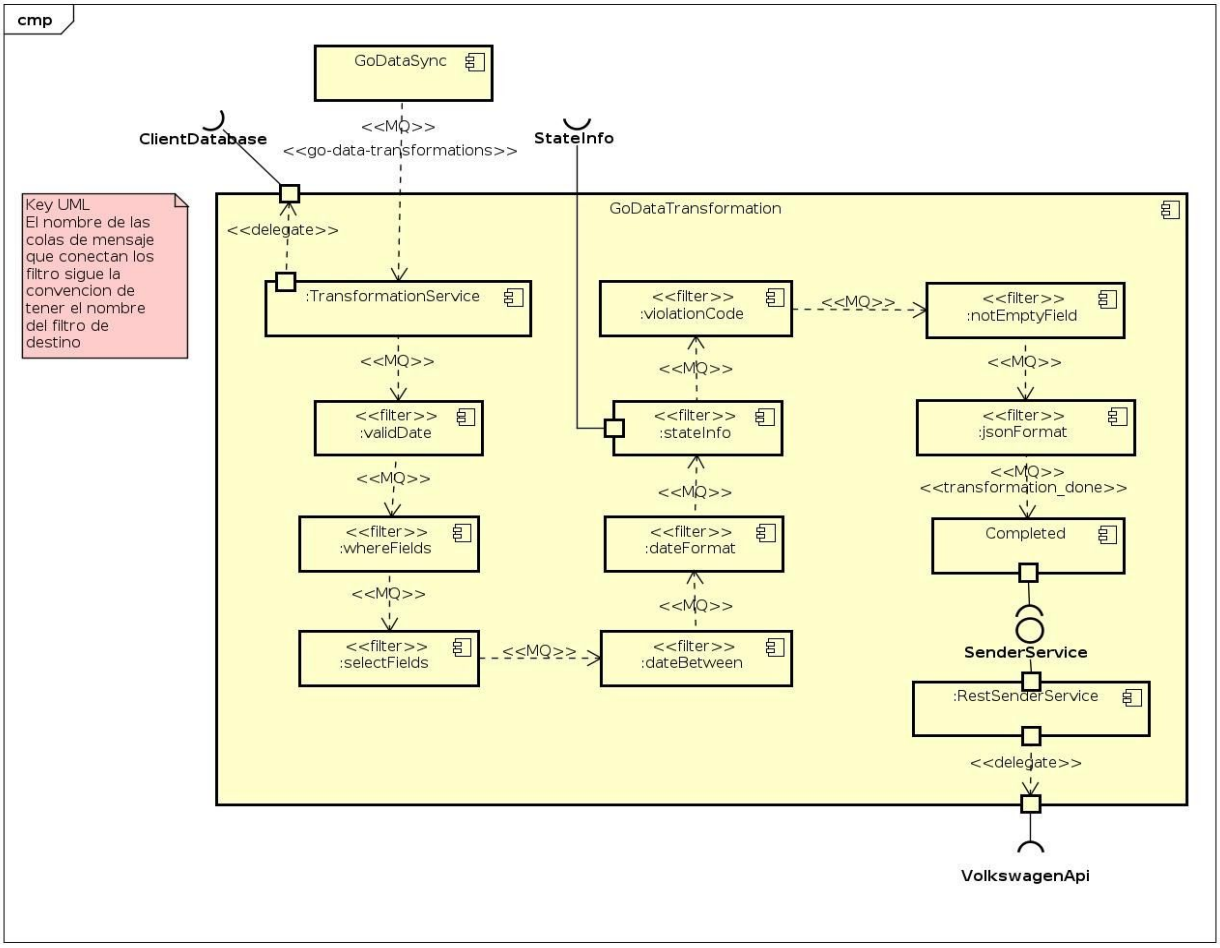

Figura 5.3.1. - Diagrama de ejecución de filtros para Volkswagen

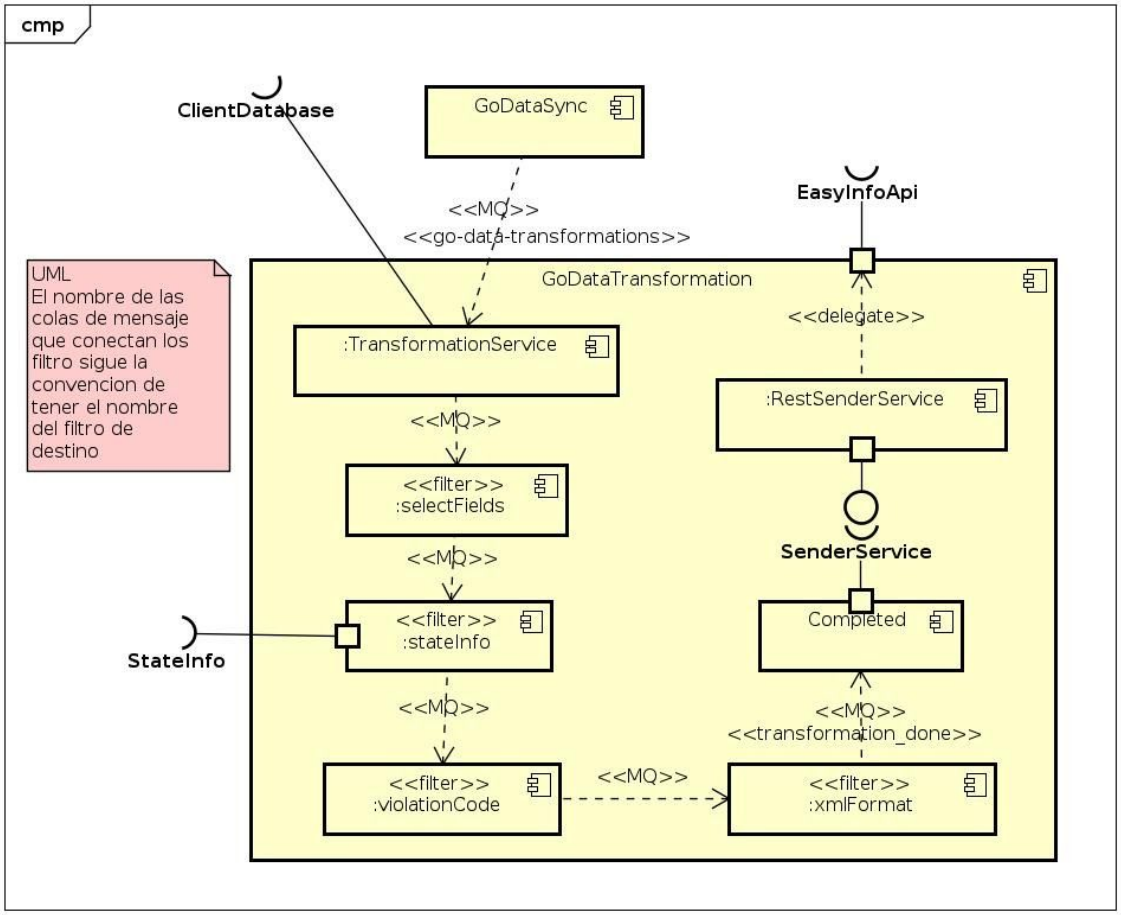

Figura 5.3.1. - Diagrama de ejecución de filtros para EasyInfo

5.2.3.2. Catálogo de elementos

TransformationService

Este componente se encarga de recibir los datos de las multas prontas para transformar y empezar la ejecución de los filtros que se van a aplicar para cada Aplicación Cliente registrada.

validDate

Este componente se encarga de verificar que los datos de las multas a enviar tienen fecha válida.

whereFields

Este componente se encarga de verificar que los datos de las multas a enviar complan con la condicion especificada por el cliente(ejemplo: VEHICLE_MAKE = ‘VOLKS’).

selectFields

Este componente se encarga de seleccionar solamente los campos que la aplicación cliente específico que desea recibir de los datos de las multas a enviar.

dateBetween

Este componente se encarga de verificar que la fecha de las multas a enviar cumplan con un rango de fechas especificado por la aplicación cliente.

dateFormat

Este componente se encarga de transformar el formato de la fecha de las multas a un formato especificado por la aplicación cliente.

stateInfo

Este componente se encarga de agregar la descripción correspondiente a los campos REGISTRATION_STATE y PLATE_TYPE para las multas a enviar a la aplicación cliente que lo especifique.

violationCode

Este componente se encarga de agregar la descripción correspondiente al campo VIOLATION_CODE para las multas a enviar a la aplicación cliente que lo especifique.

notEmptyField

Este componente se encarga de verificar que para las multas a enviar los campos especificados por la aplicación cliente no son vacíos.

jsonFormat

Este componente se encarga de transformar las multas a enviar en formato JSON.

xmlFormat

Este componente se encarga de transformar las multas a enviar en formato XML.

Completed

Este componente se encarga de recibir el resultado final de las transformaciones y validaciones para poder enviarlo a la aplicación cliente a través de RestSenderService.

RestSenderService

Este componente se encarga de realizar la llamada a la interfaz de la aplicación cliente con los datos que el componente Completed le provee.

5.2.3.3. Interfaces

StateInfo

Interfaz requerida que se utiliza para consultar las descripciones para los valores de PLATE_TYPE y REGISTRATION_STATE que vienen en los datos de las multas.

ClientDatabase

Interfaz requerida que se utiliza para consultar las aplicaciones clientes registradas a las que se les debe mandar multas

VolkswagenApi

Interfaz que provee el sistema de Volkswagen para recibir las multas procesadas.

EasyInfoApi

Interfaz que provee el sistema de EasyInfo para recibir las multas procesadas.

5.2.3.4. Decisiones de diseño

Para cumplir con el requerimiento RF2 aplicamos el patrón de arquitectura Pipes and Filters debido a la naturaleza de las transformaciones definidas en los requerimientos. Cada transformación se puede ejecutar de forma aislada e independiente, inclusive pueden ejecutarse en paralelo.

5.2.3.5. Vistas relacionadas

- Vista de estructura interna de GoData
- Vista de uso del módulo Transformaciones

5.2.4. Vista del patrón CQRS

5.2.4.1. Representación primaria

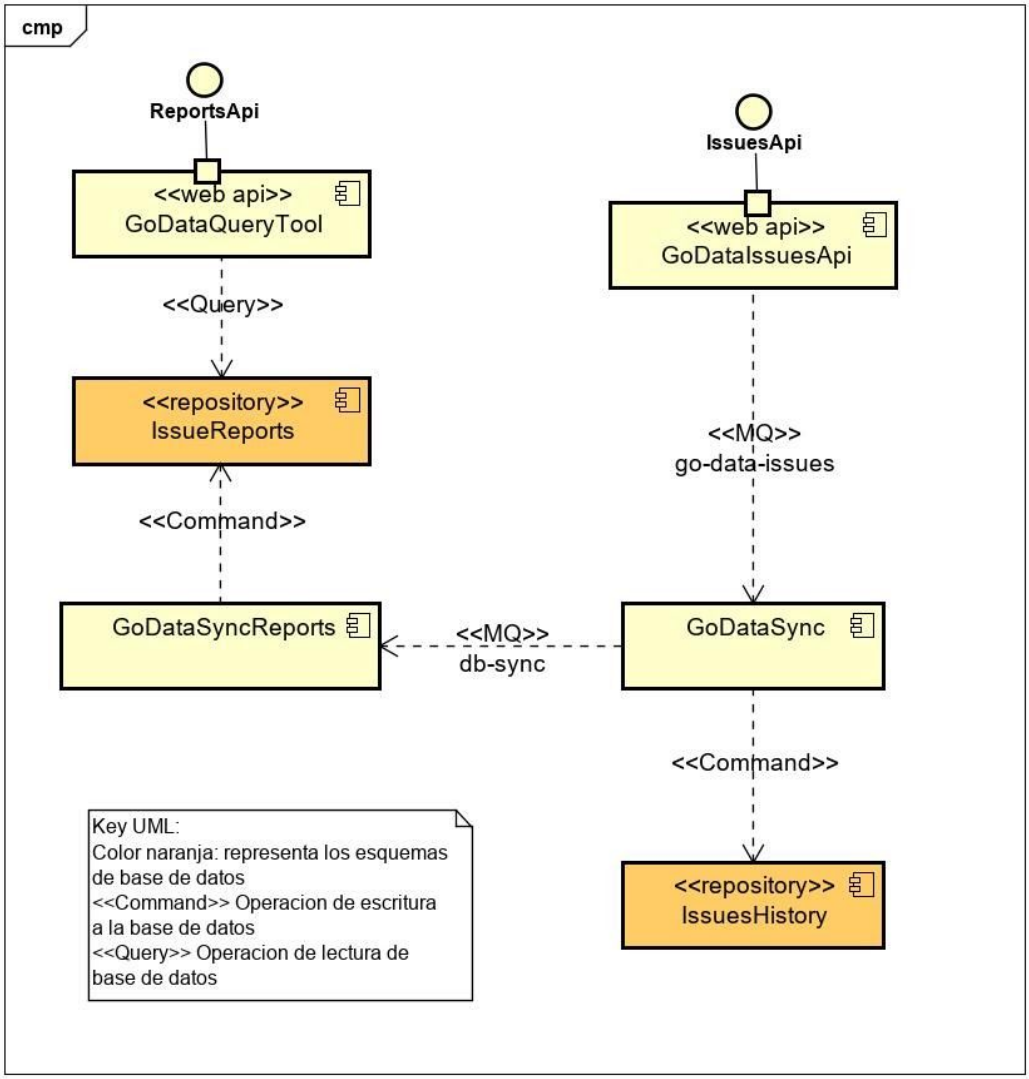

Figura 5.2.4. - Diagrama de C&C de implementación de CQRS

5.2.4.2. Decisiones de diseño

Para satisfacer el RNF4 y atributo de calidad AC20 implementamos el patrón CQRS. El objetivo es separar el modelo de escritura de multas del modelo de lectura.

La razón es que, por un lado tenemos clientes que requieren un uso especial del modelo de base de datos para realizar consultas estadísticas complejas sobre las multas procesadas en el correr del tiempo en el sistema, y por otro lado necesitamos un modelo que permita una inserción veloz con controles de integridad para asegurar la calidad de las multas almacenadas.

El repositorio IssueReports mantiene un modelo de datos no relacional optimizado para las consultas estadísticas requeridas con el fin de responder velozmente a las solicitudes aplicando así la táctica de incrementar la eficiencia de los recursos favoreciendo el atributo de calidad Eficiencia (AC20). Para ello también se encuentra representado en una instancia del motor MongoDb.

El repositorio IssuesHistory mantiene un modelo de datos relacional que permite realizar inserciones veloces asegurando a su vez la integridad de los datos. Este repositorio se encuentra representado en una instancia del motor MySql.

El componente GoDataSync se encarga controlar la integridad de las multas y de orquestar sus inserciones en ambos repositorios.

5.3. Vistas de Asignación

A continuación se documentaran las vistas de asignación mostrando los elementos de go data en sus ambientes de ejecución.

5.3.1. Vista de Despliegue

5.3.1.1. Representación primaria

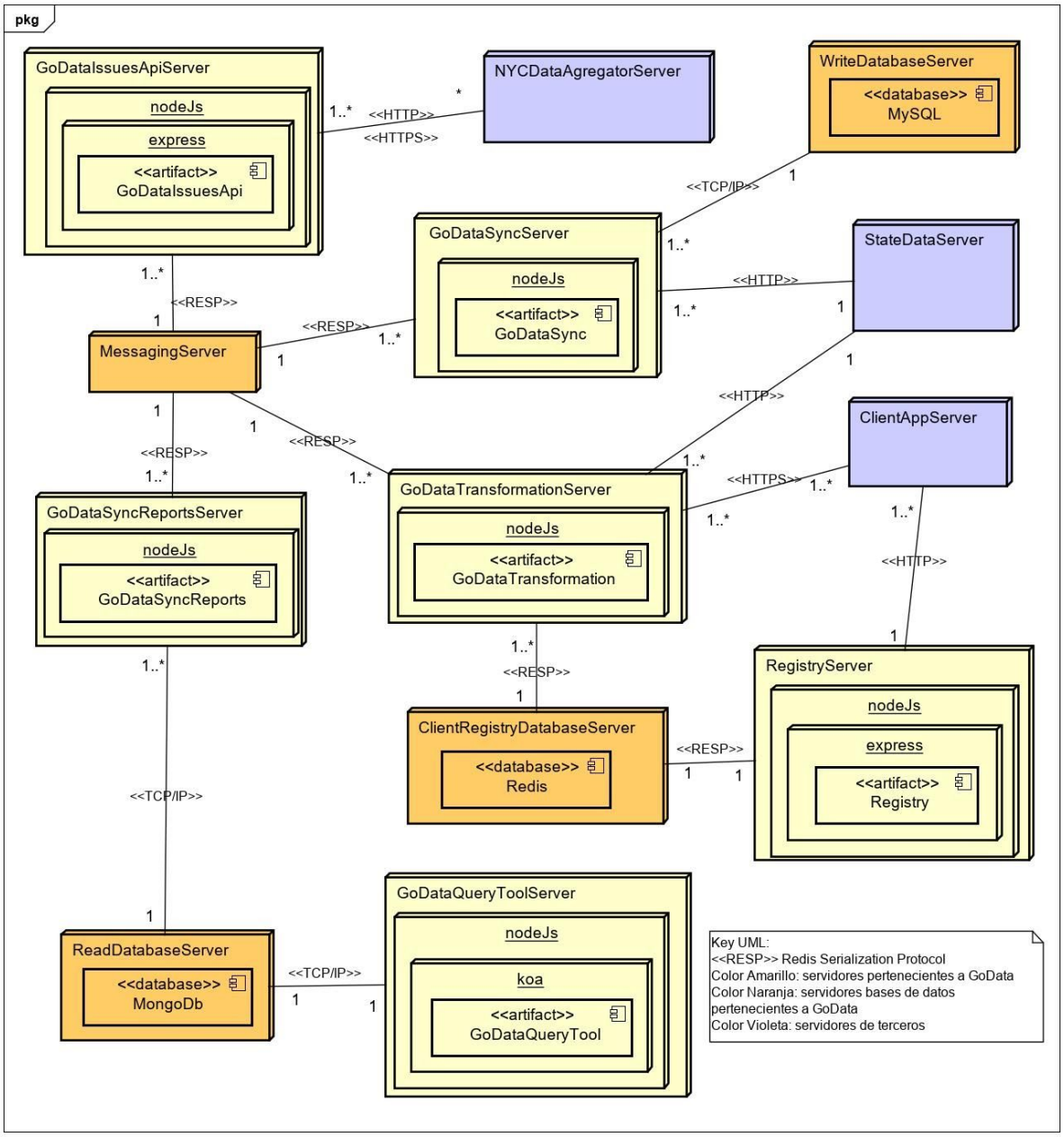

Figura 5.3.1 - Diagrama de despliegue de la plataforma GoData

5.3.1.2. Catálogo de elementos

GoDataIssuesApiServer

Servidor perteneciente a GoData encargado de mantener una instancia de *GoDataIssuesApi* que corre sobre express en un entorno de NodeJS. Pueden llegar a ser varias instancias se servidores conformando un cluster.

GoDataSyncServer

Servidor perteneciente a GoData encargado de mantener una instancia de *GoDataSync* corriendo en un entorno de NodeJS. Pueden llegar a ser varias instancias se servidores.

GoDataTransformationServer

Servidor perteneciente a GoData encargado de mantener una instancia de *GoDataTransformationServer* corriendo en un entorno de NodeJS. Pueden llegar a ser varias instancias se servidores.

GoDataSyncReportsServer

Servidor perteneciente a GoData encargado de mantener una instancia de *GoDataSyncReports* corriendo en un entorno de NodeJS. Pueden llegar a ser varias instancias se servidores.

RegistryServer

Servidor perteneciente a GoData encargado de mantener una instancia de *Registry* corriendo sobre express un entorno de NodeJS.

GoDataQueryToolServer

Servidor perteneciente a GoData encargado de mantener una instancia de *GoDataQueryTool* corriendo sobre koa en un entorno de NodeJS.

ReadDatabaseServer

Servidor perteneciente a GoData encargado de correr una instancia del motor de base de datos MongoDb a la que acceden *GoDataQueryTool* y *GoDataSyncReports* por TCP/IP.

ClientRegistryDatabaseServer

Servidor perteneciente a GoData encargado de correr una instancia de Redis a la que acceden *RegistryServer* y GoDataTransformationServer empleando el protocolo RESP.

ClientAppServer

Representa a los servidores propiedad de los clientes de GoData. Ellos se comunican con GoData por protocolo HTTP y HTTPS.

StateDataServer

Representa al servidor de terceros que consulta GoData para obtener datos detallados sobre multas.

MessagingServer

Servidor perteneciente a GoData encargado de correr el MessageBroker utilizado por *GoDataSyncServer*, GoDataSyncReportsServer y GoDataTransformationServer. Pueden llegar a ser varias instancias conformando un cluster. Se puede observar en mayor detalle el uso de colas de mensajes en la Vista de estructura interna de GoData.

NYCDataAgregator

Representa a los servidores de terceros que proveen a GoData con información de multas.

5.3.1.3. Decisiones de diseño

Despliegue de la aplicación

Utilizamos el patrón de arquitectura Multi-tier con la intención de dividir el sistema en estructuras de ejecución independientes con el fin de lograr cumplir con la alta demanda de recursos que exigen las operaciones.

Cada parte de GoData es independiente y está agrupada de acuerdo a su responsabilidad y éstas colaboran entre sí usando mecanismos de comunicación como son mensajería y HTTP.

Cumpliendo con el requerimiento no funcional RNF3 - Manejo de carga, el atributo de calidad principal que se desea atacar es AC3 - Eficiencia y para ello aplicamos la táctica de Mantener múltiples copias de computación para evitar contención de recursos y utilizamos mensajería como modo de balancear la carga.

Seguridad

Para cumplir con el atributo de calidad Seguridad en los puntos de intercambio de mensajes con clientes externos utilizamos las tacticas de Encriptación y Autenticación.

Éstas fueron implementadas con la utilización de conexión por medio de HTTPS para la encriptación y JWT para la autenticación de la fuente de información.

Los servidores involucrados son: GoDataIssuesApiServer, GoDataTransformationsSever, NYCDataAgregatorServer, ClientAppServer

Disponibilidad

5.3.1.4. Vistas relacionadas

- Vista de estructura interna de GoData.

5.3.2. Vista de Instalación del Logger

5.3.2.1. Representación primaria

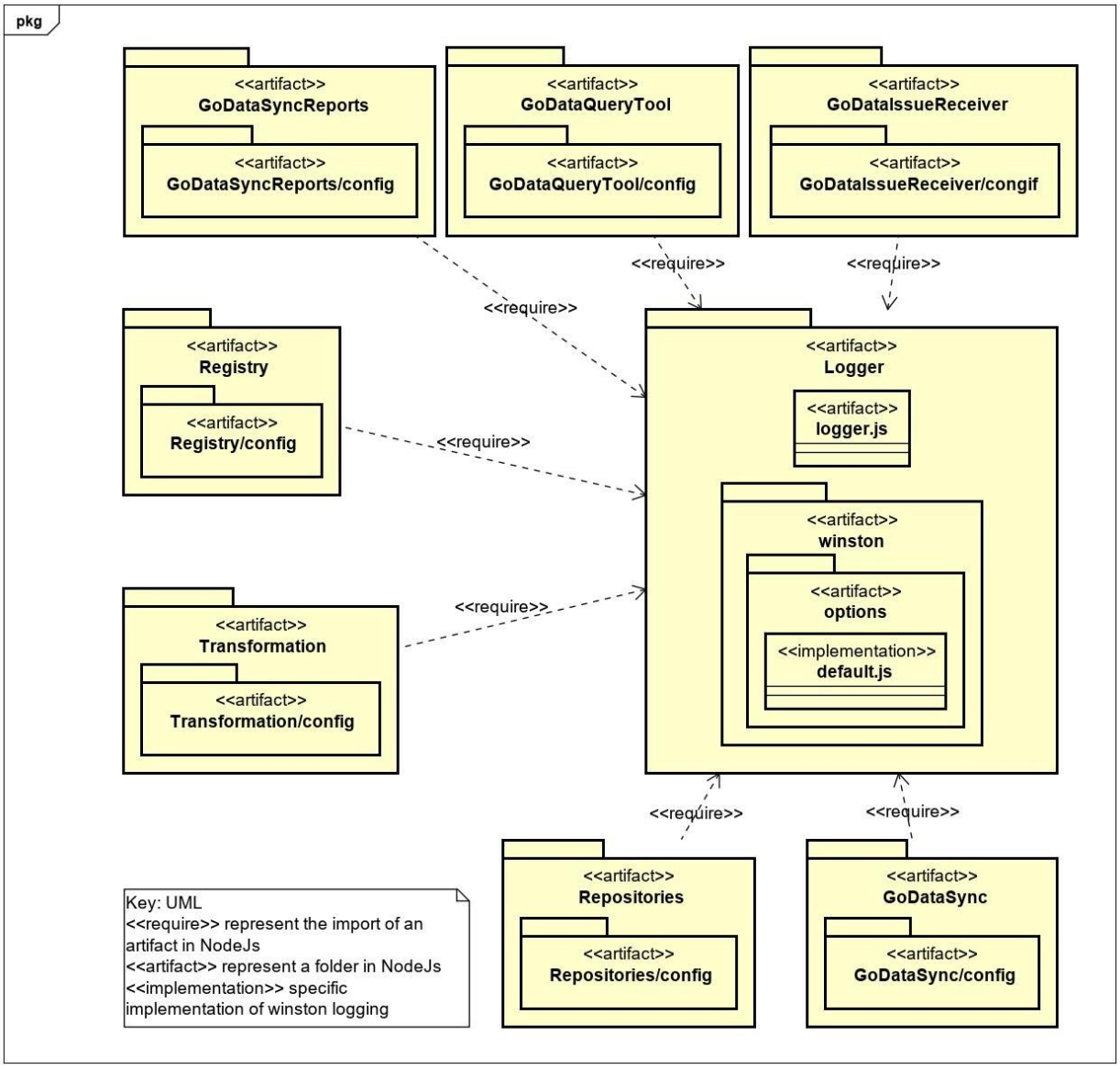

Figura 8.2 - Diagrama de despliegue

5.3.2.2. Catálogo de elementos

****/config**
El siguiente listado de artefactos comprende el archivo de configuración donde se indicará las configuraciones de implementación del módulo de Logger:

- GoDataIssueReceiver/config
- GoDataSync/config
- GoDataSyncReports/config
- GoDataQueryTool/config
- Repositories/config
- Transformation/config
- Registry/config

La configuración se da en el archivo development.json que allí se encuentra. En primer lugar se indicará la implementación del log con el siguiente formato:

```
"logger":{"implementation": "winston" }
```

En el caso particular de la implementación utilizando la librería *winston,* se detalla información de la implementación a utilizar. Existe una implementación por defecto utilizando winston, así como también una configuración por defecto para la misma. En caso que se desee agregar una nueva configuración para loggear con la librería winston, basta definir una nueva configuración e indicar la ruta en la propiedad *configuration.* La configuración necesaria es la siguiente:

```
"winston":{"path": "./winston/winstonImpl.js","configuration":
"./options/default.js"}
```

Adicionalmente se puede agregar a la configuración de winston el nombre de la aplicación que loggea la información así como el nivel de log con el que se imprime pudiendo ser este nivel: *debug, info, warn, error*. La configuración es la siguiente:

```
"meta":{"app_name": "<nombre-app>","log_level": "info"}
```

En caso que un desarrollador desee utilizar una nueva implementación de log utilizando una nueva librería, se deberá especificar una implementación la en la propiedad *implementation* indicando cuál será la implementación a utilizar. Está implementación se corresponderá con una propiedad que aloja la información referente a esta nueva implementación en caso que la tenga.

En caso que no exista la propiedad *logger,* el módulo logger procederá a loggear la información con el objeto *consol*e nativo de javascript.

Logger
Es el módulo de que se necesita requerir desde cualquier aplicación que desea loggear información.

logger.js

Es la interfaz por la cual se encapsulan las operaciones a utilizar.

winston
Es el módulo donde encuentra la implementación por defecto utilizando la librería winston.

options
Es el módulo donde se encuentra la configuración por defecto para la librería winston, indicando detalles de implementación para la construcción del log.

default.js
La implementación concreta de la configuración del log.

5.3.2.3. Decisiones de Diseño

Las siguientes son tácticas que utilizamos para satisfacer el atributo de calidad de Disponibilidad:

Manejo de excepciones
Se implementó por medio del manejo de Errores definidos por el equipo así como también por la captura de errores en todos los proyectos.

Exception Prevention

Lo aplicamos en los puntos donde el sistema puede fallar por problemas externos como ser pérdida de conexión de red, caída de servidor de Redis, MySql o MongoDb.

La siguiente táctica se utilizó para cumplir con el atributo de calidad Modificabilidad:

Defer Binding
Se implementó permitir seleccionar en tiempo de inicialización el tipo de implementación de loger a utilizar.

6. Código Fuente

El código fuente de la aplicación GoData se encuentra versionada en el siguente repositorio: https://github.com/ORTArqSoft/GoData