# DOCUMENTO RESUMIDO - DESCRIPCIÓN DE ARQUITECTURA

***Flight Data Middleware***

***26 de junio del 2019***
***Matias Hernández***
***169236***

## 1. Introducción

Se utilizará el formato de descripción de arquitectura correspondiente al estándar Views & Beyond descrito en el libro Documenting Software Architecture 2nd Edition, SEI.

En cada Vista, se hace una descripción general de dicha vista en el sistema (Overview) y luego se procede a hacer otras vistas que atacan directamente a cada Requerimiento del sistema.

En la Vista general, se procede a hablar de estilos de arquitectura utilizados y sus correspondientes justificaciones.

Por otro lado, en las Vistas específicas de los requerimientos, se habla principalmente de las justificaciones arquitectónicas, apoyando las mismas en las tácticas que aparecen en el libro Software Architecture in Practice, 3rd Edition - SEI.

### 1.1. Propósito

El propósito del presente documento es proveer una especificación completa de la arquitectura de Flight Data Middleware.

### 1.2. Repositorio

Repostorio en Github: https://github.com/ORTArqSoft/AS_Obl
Usuario: matiashrnndz
E-mail: matiashrnndz@gmail.com

## 2. Antecedentes

### 2.1. Propósito del sistema

Este sistema provee mecanismos para integrar los servicios de la Autoridad de Aviación y los servicios de las aerolíneas, registrados en tiempo de ejecución. Una de las particularidades de este servicio es que permitirá procesar los datos recibidos (por ejemplo, filtros y transformaciones) previo a enviarlos a los servicios de las aerolíneas mediante las APIs registradas. El servicio también permitirá asociar "disparadores" que permitirán seleccionar qué datos se envían a los servicios registrados por las aerolíneas.

El sistema descrito en este documento de arquitectura es desarrollado en Node Js utilizando el estilo arquitectónico adaptado de SOA para el desarrollo del Middleware. Y el estilo arquitectónico Pipe & Filter para la realización de los filtros y transformaciones que el sistema provee.

A continuación se describen los principales requerimientos del sistema.

## 2.2. Requerimientos significativos de Arquitectura

### 2.2.1. Resumen de Requerimientos Funcionales

| ID Requerimiento | Descripción | Actor |
|---|---|---|
| *Req 1 - Registro de Interfaz de servicio* | *Para cada servicio provisto por las aerolíneas se debe poder:*<br>● *Registrar una "interfaz de servicio" que permita acceder a su funcionalidad.*<br>● *Especificar propiedades sobre la forma de usar cada interfaz (por ejemplo, el formato de las estructuras de datos que se intercambian por la interfaz, o el tipo de comunicación.*<br>● *Especificar ante qué valor o expresión en base a valores de los datos del servicio AvAuthFlightData se debe invocar la interfaz ("valores disparadores"). Por ejemplo, un servicio de aerolíneas dedicado a la planificación de vuelos podrá registrarse para recibir los datos de los vuelos cancelados o cancelados por determinado motivo.*<br>● *Especificar los datos que el servicio desea recibir de los provistos por la fuente de datos del servicio AvAuthFlightData.*<br>● *Especificar las validaciones y transformaciones que se deben realizar previo a enviar los datos.* | *Servicio de aerolínea* |
| *Req 2 - Envío de datos a servicios registrados* | *Ante la recepción de los datos enviados por AvAuthFlightData y para cada servicio de aerolíneas registrado el servicio deberá:*<br>● *Seleccionar los datos a enviar en base a los datos que el servicio espera recibir y a los valores "disparadores" del mensaje.*<br>● *Realizar las validaciones y transformaciones especificadas para la interfaz a invocar. En caso de que se registren valores inválidos se debe registrar el evento de que el registro contiene datos erróneos y el registro no se debe enviar.*<br>● *Enviar los datos mediante la interfaz de servicio registrada. Se debe asegurar que cada aerolínea reciba solamente datos de sus vuelos.* | *Middleware* |

### 2.2.2. Resumen de Requerimientos No Funcionales

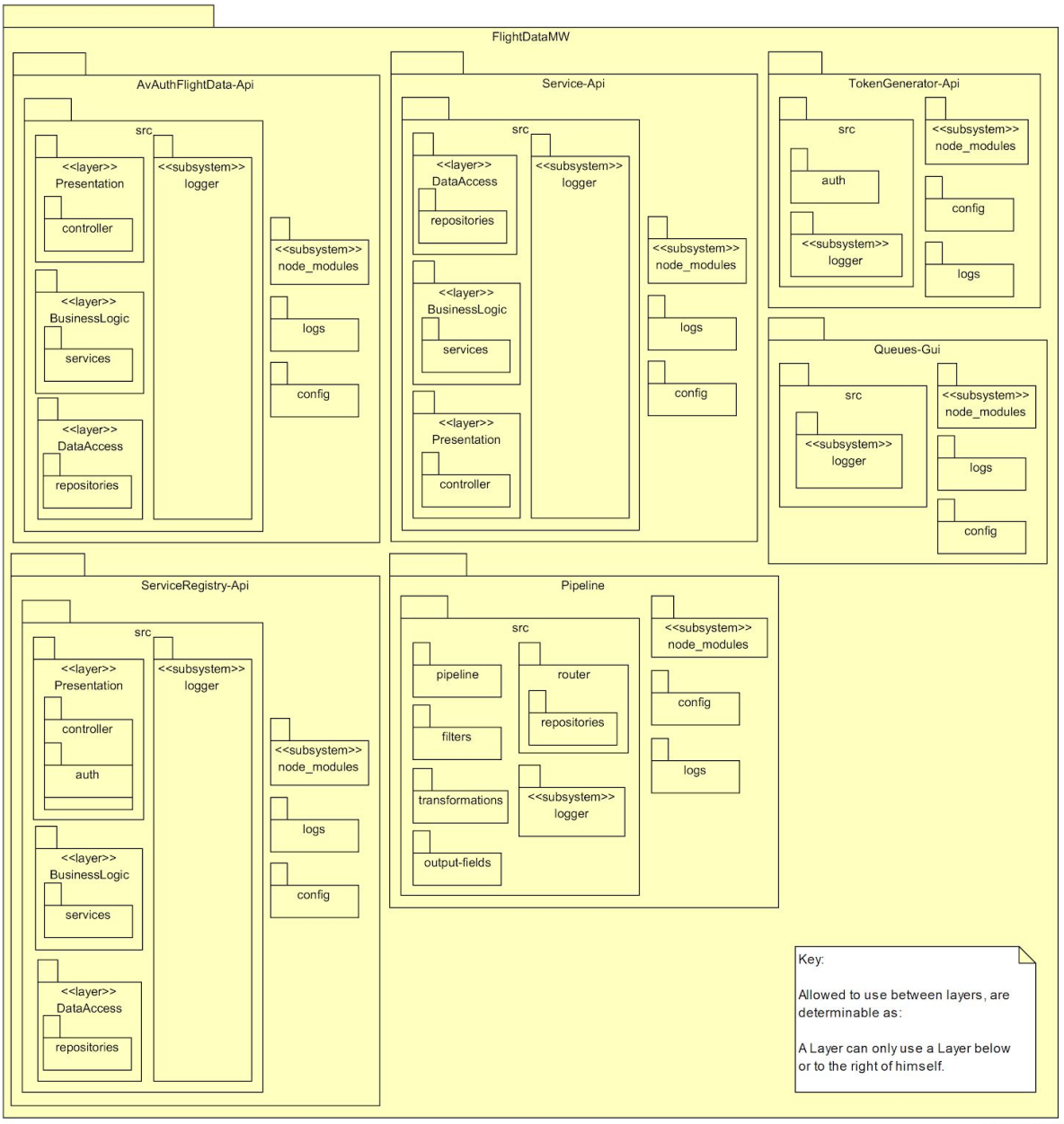
| ID Requerimiento | ID Requerimiento no Funcional | Descripción |
|---|---|---|
| *Req 3 - Gestión de errores y fallas* | *Disponibilidad y Modificabilidad* | *El sistema debe proveer suficiente información que permita conocer el detalle de las tareas que se realizan. En particular, en el caso de ocurrir una falla o cualquier tipo de error, es imprescindible que el sistema provea toda la información necesaria que permita a los administradores hacer un diagnóstico rápido y preciso sobre las causas. Se espera que la solución contemple la posibilidad de poder cambiar las herramientas o librerías concretas que se utilicen para producir esta información, así como reutilizar esta solución en otras aplicaciones, con el menor impacto posible en el código.* |
| *Req 4 - Autenticación de operaciones* | *Seguridad* | *Se requiere que toda interacción desde FlightDataMW hacia los servicios receptores de datos esté autenticada.* |
| *Req 5 - Manejo de carga* | *Performance* | *Debido a que el servicio AvAuthFlightData envía "Rafagas de datos" es importante que FlightDataMW asegure que no se pierdan datos debido a una pobre capacidad de procesamiento. Se deberá lograr la mayor capacidad de procesamiento posible sin pérdida de datos y logrando la mejor latencia posible. Es parte de la prueba de concepto determinar la capacidad máxima de procesamiento que puede manejar FlightDataMW manteniendo una variación de latencia razonable.* |
| *Req 6 - Incorporación de nuevos procesamientos de datos* | *Modificabilidad* | *El diseño deberá permitir incorporar nuevas validaciones y transformaciones a los datos que envía a los servicios de las aerolíneas con el menor costo posible y de forma de reusar las mismas.* |
| *Req 7 - Cambio en las interfaces de los servicios de las aerolíneas* | *Modificabilidad* | *Los cambios en los tipos de interfaces, los datos que se reciben, y las validaciones y transformaciones requeridas por los servicios de las aerolíneas deben de tener el menor impacto posible en el middleware, en lo posible se quiere poder incorporar los cambios sin necesidad de modificar el código del middleware.* |

# 3. Documentación de la arquitectura

## 3.1. Vistas de Módulos

### 3.1.1. Vista de Layers

#### 3.1.1.1. Representación primaria




#### 3.1.1.2. Catálogo de elementos

Detalle provisto en la sección 3.1.4.

#### 3.1.1.3. Decisiones de diseño

En las siguientes secciones se pondrá especial hincapié en las decisiones de diseño tomadas para cada uno de los requerimientos del sistema.

En esta sección, se detallarán las principales decisiones de diseño tomadas para realizar cada módulo de la forma en la que está especificado, omitiendo los que se explicarán más adelante en el documento.

En el diagrama, se puede ver que se realizó una arquitectura de Layers, la cual está detallada en el diagrama. Esto fue hecho para poder tener los beneficios de modularidad y separación de responsabilidades que brinda dicha arquitectura.

Se tomaron en cuenta las siguientes tácticas para realizar esta disposición de los módulos:

Modificabilidad:

Reduces Size of a Module:

- Split Module: Con la finalidad de que los cambios que puedan llegar al sistema, tengan el menor impacto posible, se realiza módulos y submódulos con tamaño moderadamente chicos.

Increase Cohesion:

- Increase Semantic Coherence: Se crea módulos con coherencia respecto a las responsabilidades que tienen. Se intenta que no exista un módulo con más de una responsabilidad.

Reduce Coupling:

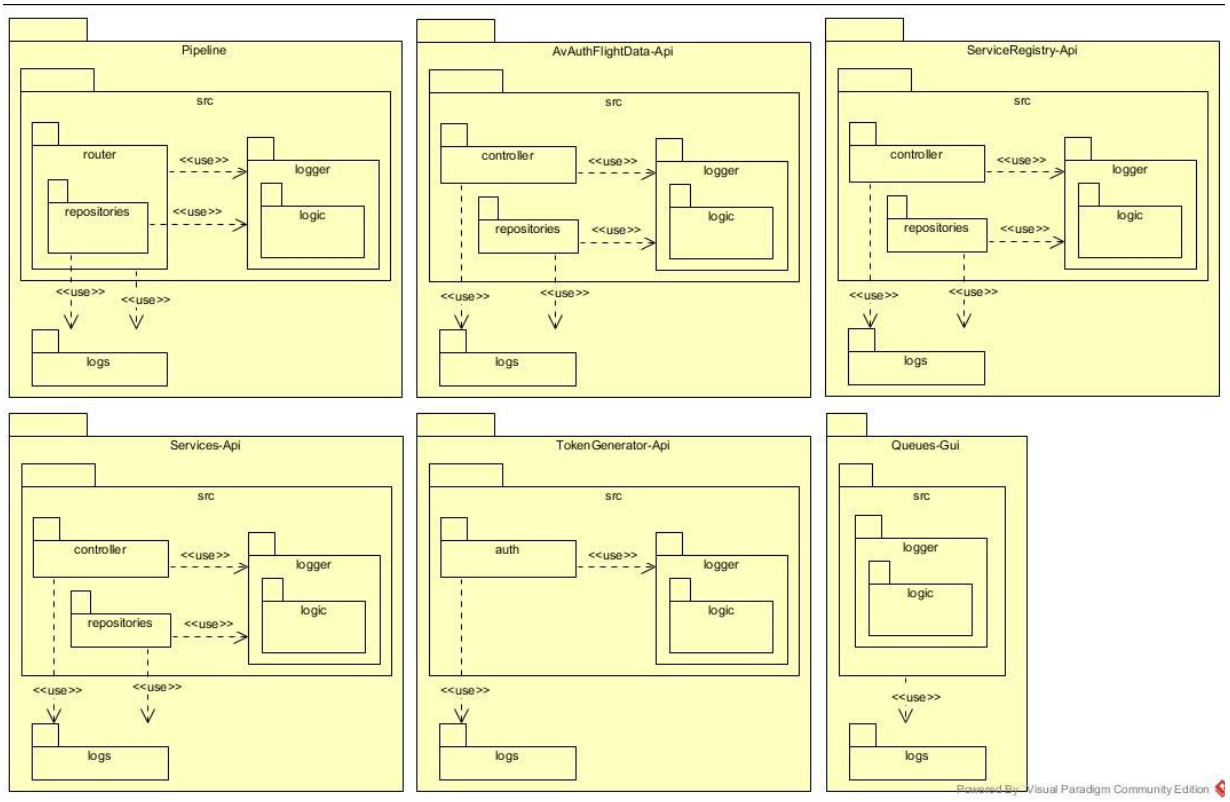
- Encapsulate: Se crean módulos que encapsulan responsabilidades concretas para cada una de las responsabilidades. Se intenta que un posible cambio de una responsabilidad, solo tenga impacto en el módulo que provee dicha responsabilidad.
- Restrict Dependencies: Como se puede notar en el diagrama, se utiliza Allow to use para determinar las dependencias que pueden existir entre los módulos.
- Abstract Common Services: Se abstrajo el servicio de logging y se reutiliza en cada uno de los módulos que lo necesitó. Se abstrajo el módulo de repositorio de tal forma que es posible crear nuevas implementaciones del mismo utilizando la misma interfaz.

Defer Binding:

- Se utilizó la táctica Defer Binding para implementar en tiempo de ejecución el tipo de Pipeline que se utiliza en el módulo Pipeline. También para seleccionar el tipo de Repositorio que se va a utilizar en cada uno de los módulos que cuentan con repositorio.
  También se utilizó con los logs, para poder realizar un cambio de implementación del sistema de Logging de acuerdo a las necesidades de cada módulo.

### 3.1.2. Vista de Uso de Gestión de errores y fallas (Req. 3)

#### 3.1.2.1. Representación primaria




#### 3.1.2.2. Catálogo de elementos

Detalle provisto en la [sección 3.1.4.]

#### 3.1.2.3. Decisiones de diseño

Considerando el Atributo de Calidad Disponibilidad:

Se pueden detectar defectos en el sistema? Sí.

- Se aplica la táctica de Exception Detection a lo largo del código. Se puede ver que se utilizan mecanismos de diseño para realizar la detección de excepciones.
- Se aplica la táctica Timestamp al momento de realizar registros de log en archivo, para poder tener un correcto seguimiento del problema.
- Se aplica la táctica de Monitor, utilizando Queues-Gui, sistema web que sirve para monitorear las colas que están en funcionamiento en Redis.

Se manejan excepciones de tal forma que se puede recuperar de defectos? Sí.

- Se aplica la táctica de Exception Handling, notificando y registrando en archivo de texto la detección de la misma y se recupera del defecto.

Se puede prevenir defectos en el sistema? Sí.

- Se aplica la táctica de Exception Prevention, intentando validar el transcurso del dato hasta última instancia donde se concreta la operación o se realiza una excepción.

Considerando el Atributo de Calidad Modificabilidad:

Para reducir el acoplamiento, se aplican las siguientes tácticas :

- Se aplica la táctica de Encapsulate, ya que se encapsula la implementación dentro del Logger y el que utiliza este módulo no necesita tener conocimiento de la misma.
- Se aplica la táctica Use an Intermediary, ya que el Logger provee una Api que es una fachada para desacoplar la implementación, del módulo que utiliza el Logger.
- Se aplica la táctica Abstract Common Services, realizando una interfaz con las funcionalidades que todo Logger tendría que permitir hacer y sea necesario utilizar en nuestro sistema.

Se puede diferir el vínculo de nuevas librerías de logging? Sí.

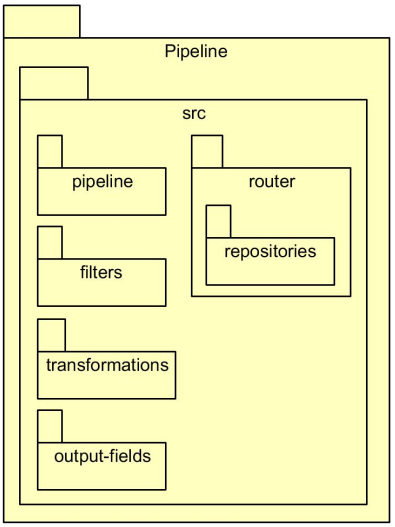
- Existe un archivo de configuración, en el cual se puede escribir cuál librería utilizar para realizar el logging del sistema Logger. Por lo cual, si uno desea cambiar la librería de implementación del sistema de Logger, puede modificar dicho valor, por otro valor válido.

Para incrementar la cohesión, se aplica la siguiente táctica :

- Increase Semantic Coherence : Se mantuvo cada implementación del sistema de Logging por separado, ya que consideramos que de esta forma, se pueden realizar modificaciones y agregaciones de forma menos costosa.

### 3.1.3. Vista de Descomposición de Incorporación de procesamientos de datos (Req. 6)

#### 3.1.3.1. Representación primaria

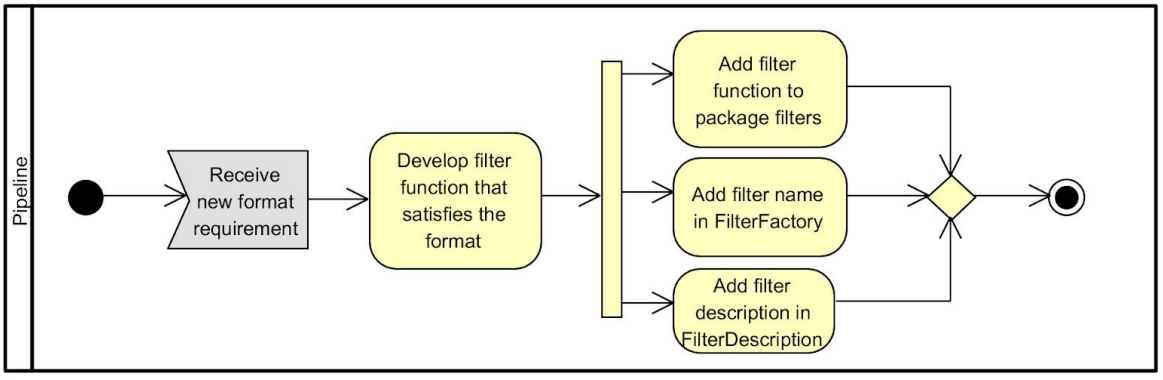



#### 3.1.3.2. Catálogo de elementos

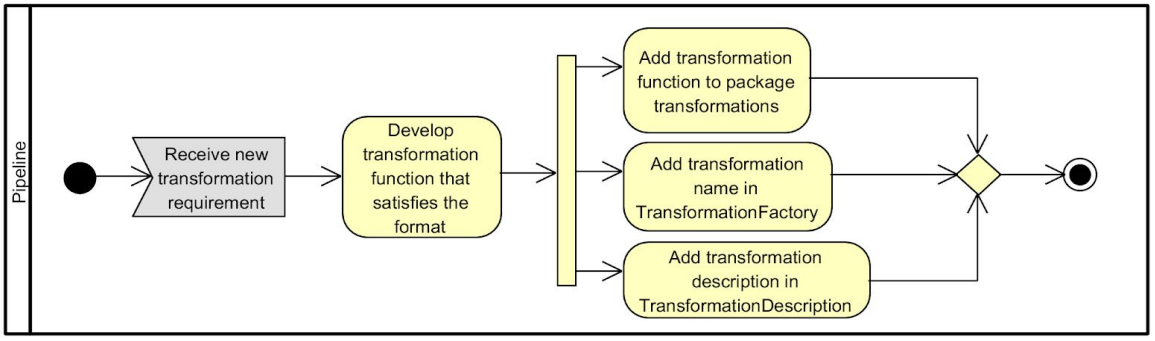
Detalle provisto en la sección 3.1.4.

#### 3.1.3.4. Comportamiento

Ingreso de Filtro nuevo:




Ingreso de Transformación nueva:



#### 3.1.3.5. Decisiones de diseño

Considerando el Atributo de Calidad Modificabilidad:

Se pueden incorporar nuevos filtros y transformaciones nuevas en tiempo de ejecución? No.

- Consideramos que por seguridad, era preferible que no se pudiera realizar este tipo de procedimiento en tiempo de ejecución.

Se pueden incorporar nuevos filtros y transformaciones con bajo costo difiriendo el enlace? Sí.

- Sí, consideramos que al haber separado los filtros, las transformaciones y los outputs, se pueden agregar nuevas funciones que tengan nuevas características a dicho módulo perteneciente. Luego se deberá registrar en el Factory de dicho módulo esta nueva función y se deberá ingresar en Descriptions los detalles de cómo utilizarlo por parte de las aerolíneas. No es necesario realizar ningún otro tipo de modificaciones en el resto del código. El sistema provee un conjunto de filtros y transformaciones ya hechas y este número tenderá a crecer haciendo que cada vez sea menos necesario crear nuevos filtros y transformaciones.

Considerando el factor de Reduce Size of a Module:

- Se aplica la táctica Split Module, donde se vé que los filtros, transformaciones y outputfields están separados en módulos distintos de código, ya que esto favorecerá los cambios que lleguen para cada uno de ellos, sin perjudicar al resto.

Considerando el factor de Increase Cohesión:

- Se aplica la táctica Increase Semantic Coherence, ya que se mantienen los filtros ,transformaciones y outputfields de manera separada, dándole sentido al hecho de que sean diferentes.

Considerando el factor Reduce Coupling:

- Se aplica la táctica Encapsulate, ya que se encapsulan los filtros, las transformaciones, los outputfields, el pipeline y el ruteo del proceso.
- Se aplica la táctica Use an Intermediary, ya que se utiliza una cola de mensajes que desacopla el módulo de Pipeline con el módulo que le envía los datos.
- Se aplica la táctica Abstract Common Services, generando una interfaz única para utilizar el sistema de filtros, transformaciones y outputfields, sin importar qué hacen cada uno de ellos. El sistema siempre aceptará el formato.

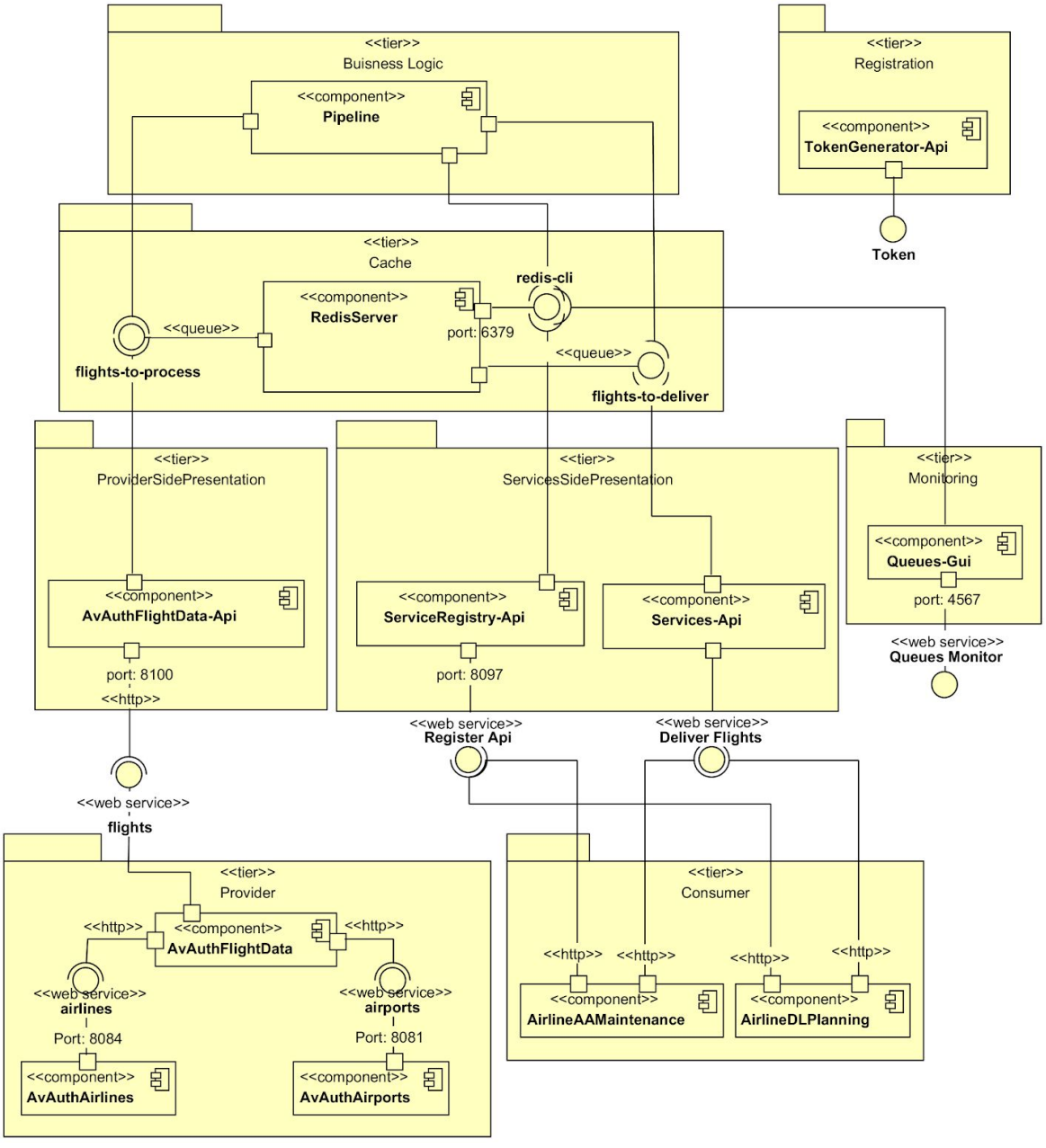
### 3.1.4. Catálogo de elementos

| Elemento | Responsabilidades |
|---|---|
| *AvAuthFlightData-Api* | ● *Exponer una Api REST utilizando el protocolo HTTP para que AvAuthFlightData pueda enviarnos los vuelos en ráfagas.*<br>● *Enviar los vuelos recibidos, a la cola de mensaje en Redis flights-to-process, implementada en otro módulo.* |
| *ServiceRegistry-Api* | ● *Exponer una Api REST utilizando el protocolo HTTP para que los servicios de las aerolíneas puedan registrar interfaces, actualizar registros de interfaces, eliminar interfaces o descubrir funcionalidades que puedan ser utilizadas en dichas interfaces.*<br>● *Ingresar, borrar y obtener desde la base de datos Redis implementada en otro módulo, los datos solicitados a través de su interfaz REST.*<br>● *Autenticar los request para que las operaciones sean realizadas por servicios válidos.* |
| *TokenGenerator-Api* | ● *Dado el IATA_CODE y un SERVICE_ID (autogenerado), generar tokens JWT para que un servicio de una aerolínea luego pueda utilizar ServiceRegistry-Api. Cada aerolínea tendrá un token, para cada servicio que posea.* |
| *Service-Api* | ● *Consumir vuelos desde la cola de mensaje en Redis flights-to-deliver con los vuelos a ser despachados a los servicios de las aerolíneas.*<br>● *Enviar los vuelos consumidos anteriormente, a través de un request HTTP con los datos ya procesados correspondientes.* |
| *Pipeline* | ● *Consumir vuelos desde la cola de mensajes en Redis flights-to-process.*<br>● *Procesar cada vuelo para todos los servicios correspondientes a la aerolínea del vuelo en cuestión, según el criterio de interfaz que registró dicho servicio.*<br>● *Los vuelos que no cumplan correctamente con todo el proceso de filtrado, no prosiguen y se registra en un archivo de texto dónde ocurrió el problema.*<br>● *Ingresar los vuelos que cumplen con todo el proceso de filtrado en una cola de mensaje en Redis flights-to-deliver/* |
| *Queues-Gui* | ● *Brindar la capacidad de monitoreo de colas de mensajes registradas, para que se puedan monitorear desde un cliente web.* |

## 3.2. Vistas de Componentes y conectores

### 3.2.1. Vista de Componentes y conectores

#### 3.2.1.1. Representación primaria




#### 3.2.1.2. Catálogo de elementos

Detalle provisto en la sección 3.2.5.

#### 3.2.1.3. Interfaces

Detalle provisto en la sección 3.2.6.

#### 3.2.1.5. Relación con elementos lógicos

| Componente | Paquetes |
|---|---|
| *Pipeline* | *Ver módulo Pipeline en sección 3.1.1.1.* |
| *TokenGenerator-Api* | *Ver módulo TokenGenerator-Api en sección 3.1.1.1.* |
| *AvAuthFlightData-Api* | *Ver módulo AvAuthFlightData-Api en sección 3.1.1.1.* |
| *ServiceRegistry-Api* | *Ver módulo ServiceRegistry-Api en sección 3.1.1.1.* |
| *Services-Api* | *Ver módulo Services-Api en sección 3.1.1.1.* |
| *Queues-Gui* | *Ver módulo Queues-Gui en sección 3.1.1.1.* |

#### 3.2.1.6. Decisiones de diseño

Se utiliza el Estilo de Arquitectura Service Oriented Architecture (SOA) ya que consideramos que sus cualidades, cumplen de manera significativa con los requerimientos que nuestro sistema tiene que satisfacer. Se le realizarán algunos ajustes para poder utilizarlo en nuestro sistema middleware.

Como una debilidad de los sistemas SOA es que son complejos para construir, se decide realizar especial foco en las tácticas de modificabilidad a lo largo del sistema.
Otra debilidad de los sistemas SOA es que no proveen garantía de performance, nuestro sistema, al utilizar 2 message queues en posiciones fundamentales (uno al recibir los datos del proveedor y otra al haber terminado de procesar los vuelos antes que los mismos sean despachados a la aerolínea correspondiente) y realizar un timestamp en diferentes puntos del sistema, se logra poder regularizar el sistema internamente y poder considerar una medición que garantice cierto nivel aceptable de performance que se debería acordar con el proveedor.

Componentes:

- Service providers: Como proveedor de servicio, está AvAuthFlightData, el cual nos envía los vuelos en ráfagas.
- Service consumers: Como consumidores del servicio, están las aerolíneas que se registren en nuestro sistema para que le sean enviados los vuelos que cumplan ciertas características determinadas por ellos.
- Registry of services: Se cuenta con 2 componentes que permiten registrar servicios. El componente GenerateToken-Api y ServiceRegistry-Api. El primero utilizado internamente por personal autorizado y el segundo, utilizado en runtime por las aerolíneas para descubrir los servicios y para registrar su interfaz de servicio.
- ESB: Se cuenta con el componente Pipeline, el cual tiene la funcionalidad de rutear y de transformar los datos. Este componente sigue un flujo de lógica de negocio preestablecido.

Conectores:

- Se utiliza conector REST para todos los web services establecidos.
- Se utiliza un conector de mensajería asincrónico como producir y consumir en una cola de mensaje para las comunicaciones internas del Middleware.

Se utiliza el Estilo de Arquitectura Pipes & Filters para el proceso que se realiza dentro del componente Pipeline, donde se aplican los filtros, las transformaciones y los outputfields para los vuelos que ingresan en el componente.

Se ve de especial ayuda utilizar este estilo para manejar el requerimiento de procesar filtros y transformaciones de cada vuelo para cada aerolínea. De esta forma, se instancian paralelamente los filtros para que concurrentemente se vayan ejecutando. El proceso sigue el orden de Filtros, luego Transformaciones (ya que si no pasa los filtros, es innecesario transformar los datos) y luego finalmente se filtran los campos solicitados (ya que se pudo haber ingresado un campo nuevo en las transformaciones). Los filtros y transformaciones se auto-validan ya que es algo inherente a su utilidad. Las validaciones son incluidas dentro de los filtros ya que consideramos que una validación de un dato es un filtro en nuestro sistema, ya que determina si se filtra o no se filtra dicho vuelo.

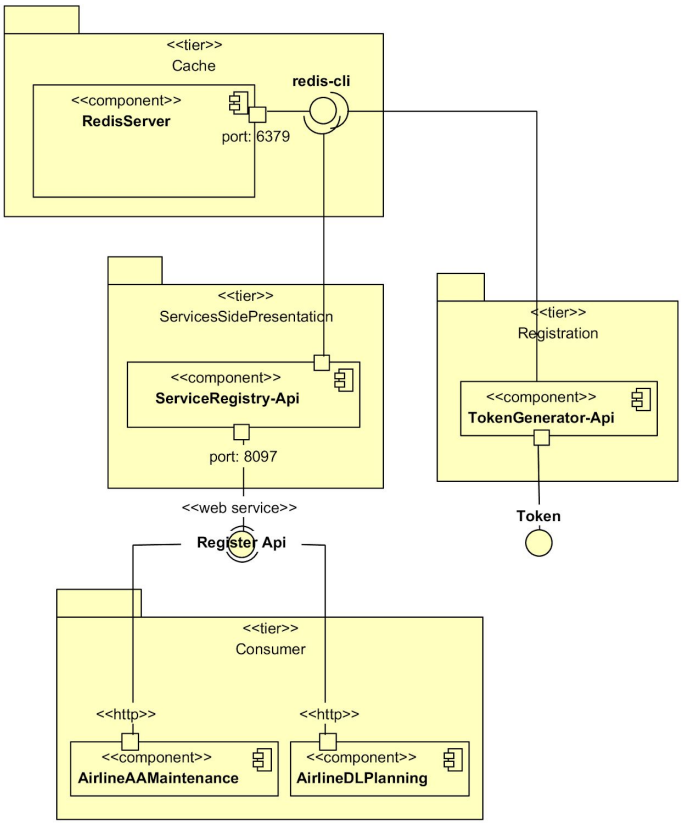
Elementos:

- Filtros: Los filtros, las transformaciones y los outputfields son los ‘filtros’ del estilo Pipe & Filter en nuestro sistema.
- Pipe: Es el módulo Pipeline el cuál es el encargado de tener la lógica del Pipe. El módulo Router dentro del módulo Pipeline, es el encargado de instanciar Pipelines a medida que le van llegando los vuelos. Instancia 1 Pipeline para cada servicio registrado de la aerolínea perteneciente de dicho vuelo.

Se explican las decisiones de diseño más importantes, para cada RF del sistema, en las Vistas siguientes.

### 3.2.2. Vista de Componentes y conectores de Autenticación de operaciones (Req. 4)

#### 3.2.2.1. Representación primaria




#### 3.2.2.2. Catálogo de elementos

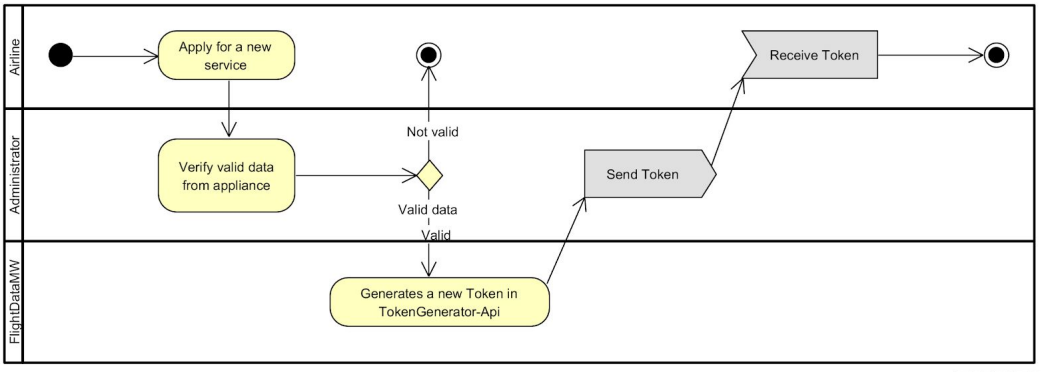
Detalle provisto en la [sección 3.2.5.]

#### 3.2.2.3. Interfaces

Detalle provisto en la [sección 3.2.6.]

#### 3.2.2.4. Comportamiento

Solicitud de Token para dar de alta un servicio de una aerolínea:



Para ver la autenticación en las operaciones de registro, borrar y discovery, ver detalles en sección 3.2.2.4.

#### 3.2.2.5. Decisiones de diseño

Considerando el Atributo de Calidad Seguridad:

La api de registro de servicios puede detectar ataques?

- Se aplica la táctica Verify Message Integrity, ya que se utiliza JWT para encriptar datos sensibles los cuales viajan en formato JSON Web Token en cada request en el encabezado del request HTTP Authorization. En caso de que se modifique algo de dicho mensaje, el mismo no auténtica y quedará registrado.

La api de registro de servicios puede resistir a ataques?

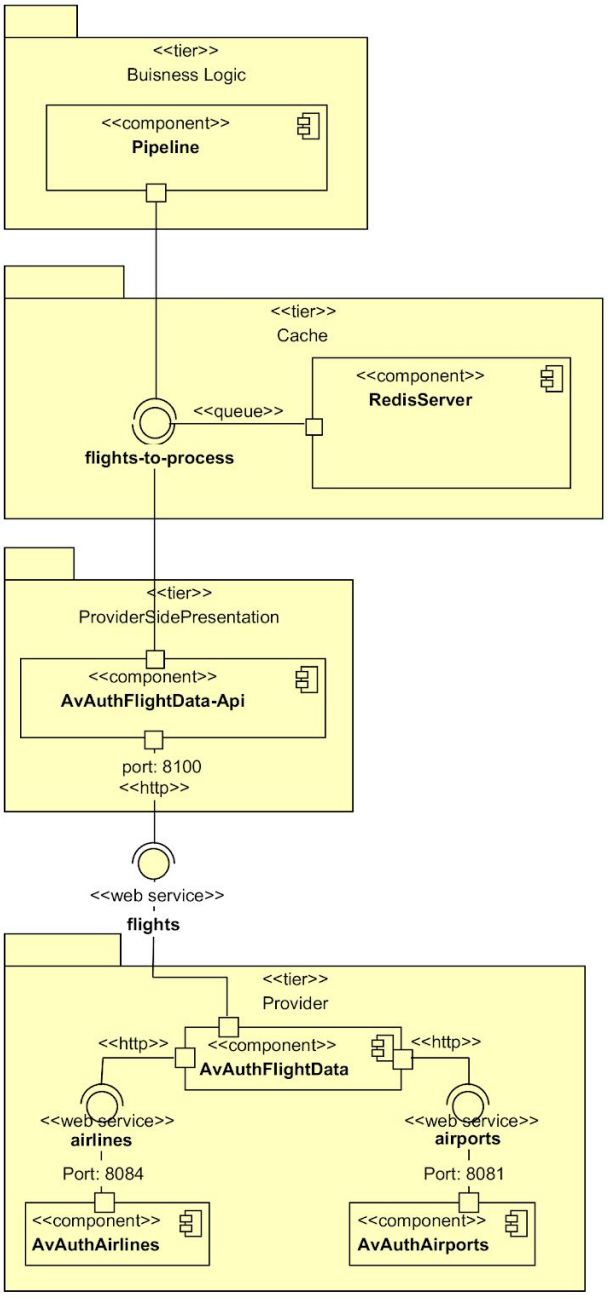
- Se aplica la táctica Authenticate Actors mediante el uso de un token en formato JWT, el cual viaja en cada request, por lo tanto, una persona no autenticada, no podría realizar un ataque sin que el sistema lo detecte y lo rechace. Al menos que se vea vulnerado el mensaje JWT que viaja.
- Se aplica la táctica Authorize Actos, brindándole autorización a las aerolíneas que solicitan nuestro servicio mediante una previa comunicación con ellos y un acuerdo de servicios. Una vez se acuerde, se genera internamente un TOKEN para dicha aerolínea y les es enviado en forma confidencial, ellos tendrán que utilizar dicho TOKEN para ese servicio en particular de ellos. Cada servicio que tenga una aerolínea, tendrá un TOKEN diferente. De esta forma también se puede autorizar a una aerolínea para que sólo pueda acceder a sus vuelos, ya que en el TOKEN viaja la información del IATA_CODE de la aerolínea y el ID del servicio.
- Se aplica la táctica Limit Exposure, ya que para poder acceder al servicio, es necesario que se le haya generado un TOKEN para la aerolínea, por lo tanto, no es posible registrarse sin que lo hayamos hecho nosotros internamente. Esto mejora considerablemente la seguridad de la aplicación.
- Se aplica la táctica Limit Access, ya que se aloja el servidor que tiene el cliente de generación de tokens en una computadora separada con acceso sólo de administradores del MW.
- Se aplica la táctica Encrypt Data, ya que los datos sensibles viajan en forma encriptada.

La api de registro de servicios puede recuperarse de ataques?

- Se utiliza la táctica Maintain Audit Trails para poder visualizar de manera rápida los eventos más importantes del sistema.

### 3.2.3. Vista de Componentes y conectores de Manejo de carga (Req. 5)

#### 3.2.3.1. Representación primaria



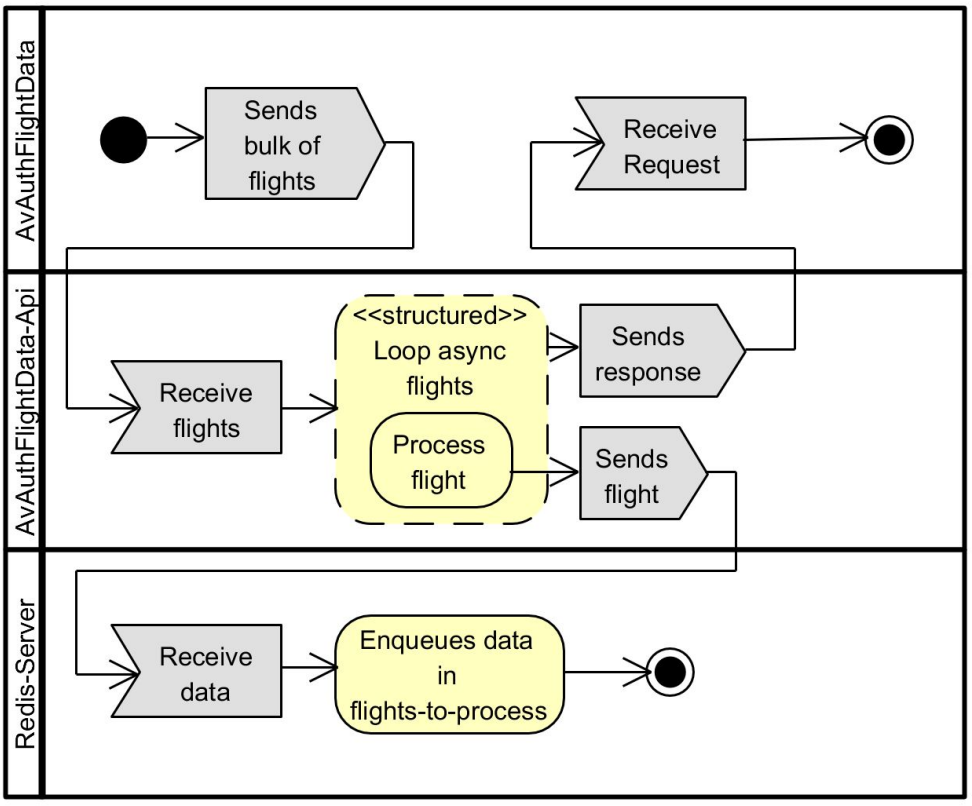

#### 3.2.3.2. Catálogo de elementos

Detalle provisto en la sección 3.2.5.

#### 3.2.3.3. Interfaces

Detalle provisto en la sección 3.2.6.

### 3.2.3.4. Comportamiento




### 3.2.3.5. Decisiones de diseño

Considerando el Atributo de Calidad Performance:

Considerando el objetivo Control Resource Demand:

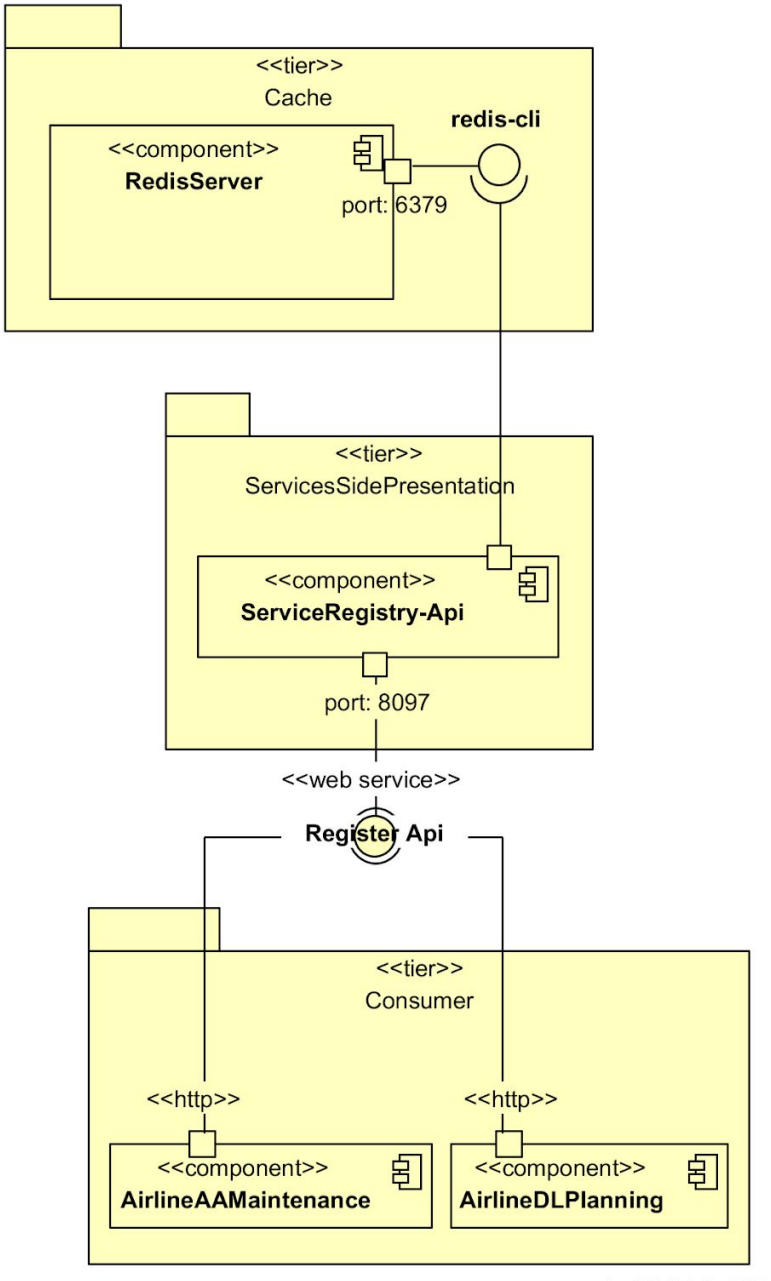
- Se aplica la táctica Manage Sampling Rate, utilizando parámetros para poder determinar las ráfagas de datos que llegan al sistema y ajustándose.
- Se aplica la táctica Prioritize Events, ya que se prioriza recibir los datos provistos por AvAuthFlightData y luego se envían a una cola para que sean procesados posteriormente.
- Se aplica la táctica Reduce Overhead, ya que se dejará un servidor específicamente para recibir los datos enviados por AvAuthFlightData y encolar los vuelos en una Message Queue en Redis con la finalidad de que pueda recibir mayor cantidad de datos y desacoplar el proceso de los mismos.

Considerando el objetivo Manage Resources:

- Se aplica la táctica Introduce Concurrency, ya que el encolamiento de datos se hará de forma asíncrona para no disminuir el performance y generar un overhead innecesario.

### 3.2.4. Vista de Componentes y conectores de Cambios en las interfaces (Req. 7)

#### 3.2.4.1. Representación primaria




#### 3.2.4.2. Catálogo de elementos

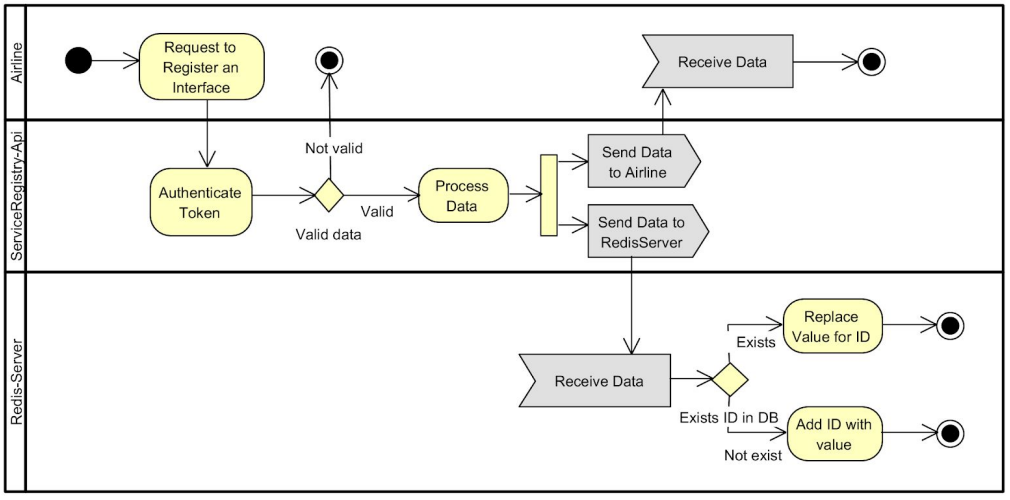
Detalle provisto en la sección 3.2.5.

#### 3.2.4.3. Interfaces

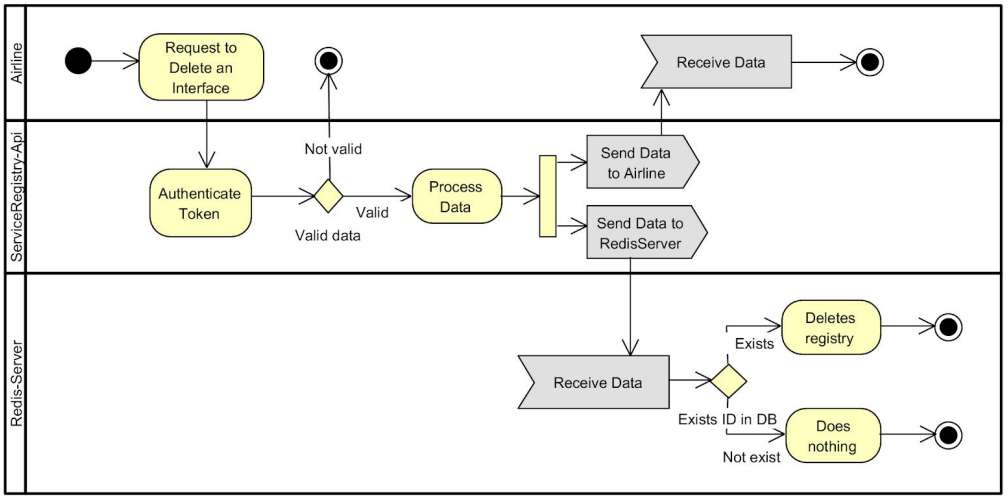
Detalle provisto en la sección 3.2.6.

### 3.2.4.4. Comportamiento

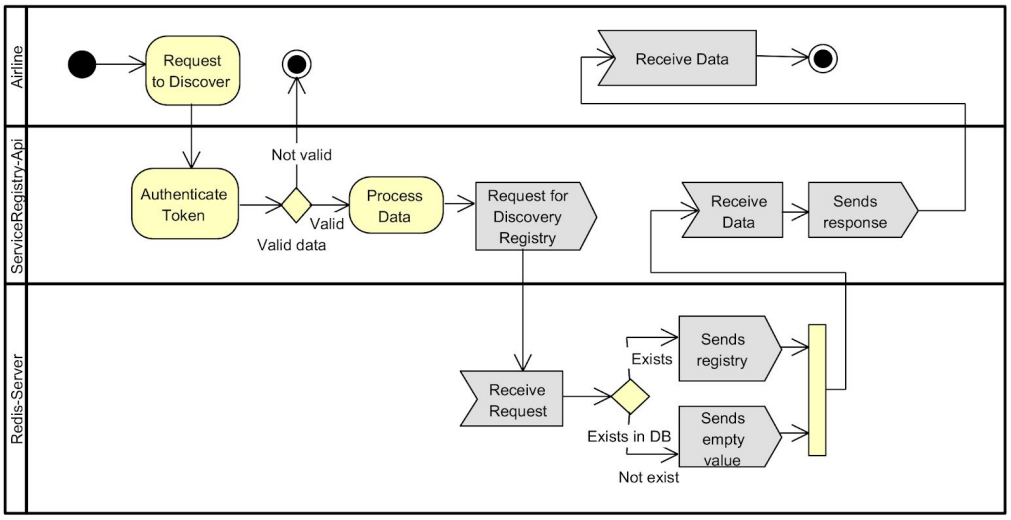
Registro de servicio de una aerolínea cuando ya tiene Token:




Borrar la interfaz de servicio para el servicio de la aerolínea ya tiene Token:




Discovery de funciones cuando el servicio de la aerolínea ya tiene Token:



#### 3.2.4.5. Decisiones de diseño

Considerando el Atributo de Calidad Modificabilidad:

Considerando el objetivo Defer Binding:

- El sistema difiere en enlace lo más posible (runtime) para poder establecer interfaces de servicios, con filtros, transformaciones y outputfields. El servicio registrado, puede modificar sus filtros, transformaciones y outputfields en tiempo de ejecución, utilizando el set de funciones que las puede ver a través del servicio Discovery que ofrece RegistryService-Api.
  En dicho discovery, le aparecerá todas las posibles funciones que puede utilizar y la modalidad de interfaz que se debe respetar.

Considerando el objetivo de Reducir el acoplamiento:

- Se utiliza la táctica Encapsulate, ya que se encapsula el servicio que contactan las aerolíneas con respecto al que los procesa. Y cada petición, se hace de manera independiente, siendo ninguna de estas dependientes de otros servicios paralelos.
- Se utiliza la táctica Use an Intermediary, ya que se utiliza un caché en Memoria RAM a través de Redis que intermedia el proceso entre el que envía los datos a las aerolíneas y el que los procesa. Esto desacopla ambos componentes.

Considerando el objetivo de Incrementar la cohesión:

- Se utiliza la táctica de Increase Semantic Coherence, ya que se utiliza un sistema para comunicarse con el proveedor de la aerolínea y luego el sistema sólo encola estos datos para que luego sean procesados por otro componente.

Considerando el el objetivo de Reducir el tamaño de un módulo:

- Se separan los módulos de recibir los datos de registro y de discovery de las aerolíneas; del componente que procesa la información y produce los datos de discovery. Esto logra que cada módulo sea más pequeño y desacoplado.

### 3.2.5. Catálogo de elementos

| Componente/conector | Tipo | Descripción |
|---|---|---|
| *Pipeline* | *Application* | *Consume de la cola de mensajes flights-to-process de RedisServer para ingresar los datos al pipeline.*<br>*El pipeline implementa el patrón Pipes & Filters para poder realizar los filtros y transformaciones y conoce la lógica de negocio para el proceso de datos.*<br>*Una vez ya procesados por el Pipeline, los exitosos, se envían a la cola de mensajes flights-to-deliver de RedisServer.* |
| *TokenGenerator-Api* | *Application* | *Aplicación utilizada para generar tokens para los servicios de las aerolíneas. Es utilizada por un administrador en una locación segura, sin acceso WAN.* |
| *RedisServer* | *Redis Database* | *Base de datos del formato clave-valor, alojada en memoria RAM, capaz de brindar la utilización de colas de mensajes. También brinda la posibilidad de persistir datos a disco y levantar los datos en cada inicio de la base de datos.* |
| *Queues-Gui* | *Queues monitor web client* | *Brinda conexión desde un cliente web para poder monitorear las colas de mensajes del servidor de base de datos Redis.* |
| *AvAuthFlightData-Api* | *Web server api* | *Servidor web que implementa una interfaz REST para recibir vuelos por parte de AvAuthFlightData y encolar dichos vuelos en la cola de mensajes flights-to-process de RedisServer.* |
| *ServiceRegistry-Api* | *Web server api* | *Servidor web que implementa una interfaz REST para recibir peticiones de registro, borrado y descubrimiento de funciones por parte de las aerolíneas.* |
| *Services-Api* | *Web server api* | *Servidor web que consume de la cola de mensajes flights-to-deliver de RedisServer y envía a través de HTTP los datos de los vuelos que cumplan el contrato con dicho servicio de la aerolínea en cuestión al final de todo el proceso.* |

### 3.2.6. Interfaces

| **Interfaz:** | **flights-to-process** |
|---|---|
| **Paquete que la implementa:** | **RedisServer** |
| ***Servicio*** | ***Descripción*** |
| *Cliente para Message Queue - flights-to-process* | *RedisServer expone una Api para poder hacer CRUD de mensajes a Message Queue flights-to-process.* |

| **Interfaz:** | **flights-to-deliver** |
|---|---|
| **Paquete que la implementa:** | **RedisServer** |
| ***Servicio*** | ***Descripción*** |
| *Cliente para Message Queue - flights-to-deliver* | *RedisServer expone una Api para poder hacer CRUD de mensajes a Message Queue flights-to-deliver.* |

| **Interfaz:** | **redis-cli** |
|---|---|
| **Paquete que la implementa:** | **RedisServer** |
| ***Servicio*** | ***Descripción*** |
| *Cliente para manejar DB Redis.* | *Expone cliente para poder manejar la DB clave-valor de Redis.* |

| **Interfaz:** | **Register Api** |
|---|---|
| **Paquete que la implementa:** | **ServiceRegistry-Api** |
| ***Servicio*** | ***Descripción*** |
| *Register an airline service* | *Pre-requisito: La aerolínea debe tener un token válido.*<br>*Una aerolínea puede registrar una interfaz de servicio. Si la aerolínea ya tenía una registrada para dicho ID, entonces el sistema sobrescribe la interfaz anterior.* |
| *Delete an airline service* | *Pre-requisito: La aerolínea debe tener un token válido.*<br>*Una aerolínea puede borrar una interfaz de servicio.* |

| | |
|---|---|
| *Discovery of functions* | *Pre-requisito: La aerolínea debe tener un token válido.*<br>*Una aerolínea puede hacer discover de las funciones habilitadas.* |

| **Interfaz:** | **Deliver Flights** |
|---|---|
| **Paquete que la implementa:** | **Services-Api** |
| ***Servicio*** | ***Descripción*** |
| *Deliver flights to registered airlines* | *Api encargada de enviar vuelos a las aerolíneas que tengan una interfaz registrada y haya cumplido con los requisitos de dicha interfaz.* |

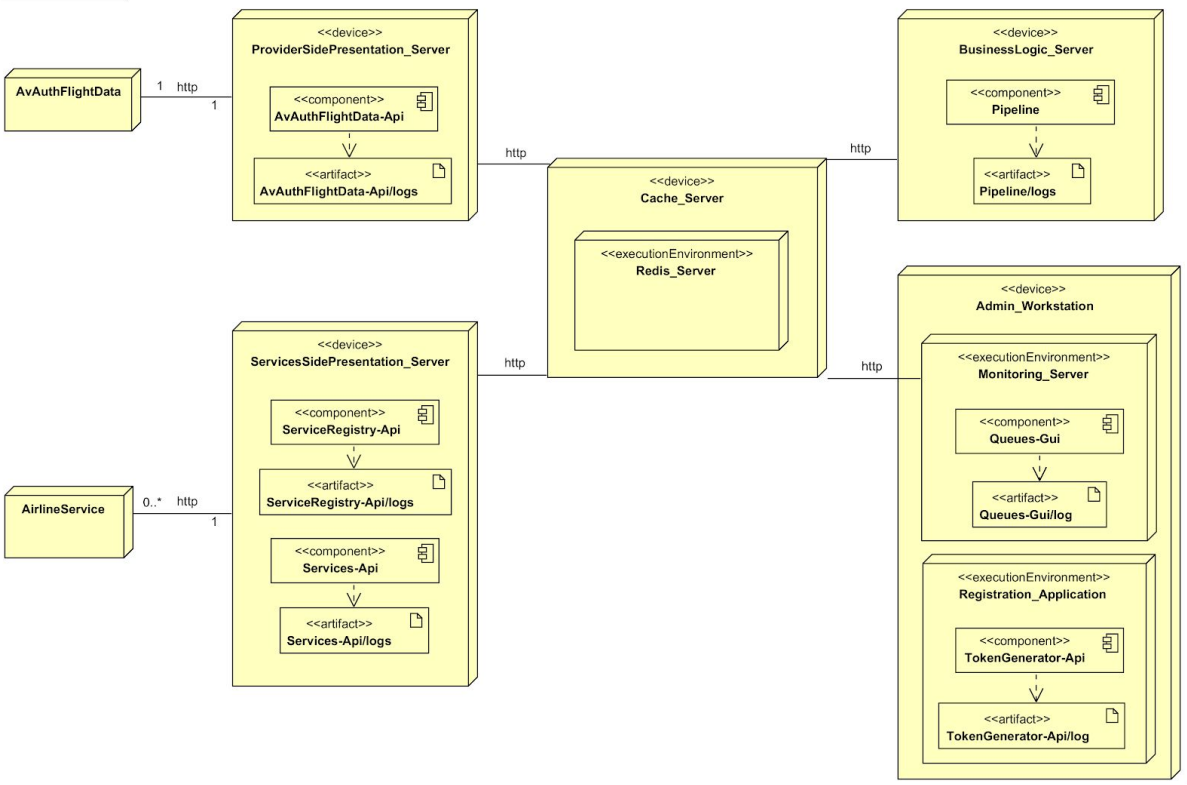
| **Interfaz:** | **Queues Monitor** |
|---|---|
| **Paquete que la implementa:** | **Queues-Gui** |
| ***Servicio*** | ***Descripción*** |
| *Cliente web para monitorear Messages Queues en Redis.* | *Se abre un cliente web en puerto 4567 de la máquina local, donde uno puede monitorear las colas de mensajes activas que existen en Redis y que previamente hayan sido ingresadas en Queues-Gui.* |

| **Interfaz:** | **Token** |
|---|---|
| **Paquete que la implementa:** | **TokenGenerator-Api** |
| ***Servicio*** | ***Descripción*** |
| *Generar token para aerolínea.* | *Dado un IATA_CODE de una aerolínea y un ID de servicio, se genera un token con dichos datos encriptados.* |

## 3.3. Vistas de Asignación

### 3.3.1. Vista de Despliegue

#### 3.3.1.1. Representación primaria




#### 3.3.1.2. Catálogo de elementos

| Nodo | Características (velocidad, memoria, etc.) | Descripción |
|---|---|---|
| *AvAuthFlightData* | *No aplica* | *Servicio proveedor de los vuelos.* |
| *ProviderSidePresentation_Server* | *Alto poder de procesamiento, tener nodejs v10.15.3* | *Servidor encargado de recibir request HTTP a través de una REST Api y enviar los datos recibidos a Cache_Server a través de HTTP.* |
| *Cache_Server* | *Alto poder de procesamiento y alta capacidad de memoria RAM* | *Servidor encargado de alojar el servidor de Redis, con el cual se interactúa con él a través de requests HTTP.* |
| *Redis_Server* | *Redis 4.0.9 64 bit* | *Servidor de Redis con una instalación de la base de datos Redis.* |

| | | |
|---|---|---|
| *BuisinessLogic_Server* | *Alto poder de procesamiento, Tener nodejs v10.15.3* | *Servidor con la lógica de negocio, que necesita alto poder de procesamiento ya que se centra la mayoría de los procesos intensos del programa en dicho elemento. Realiza requests HTTP a Cache_Server para obtener e ingresar datos.* |
| *Admin_Workstation* | *Tener nodejs v10.15.3* | *Workstation de un administrador del sistema, protegido por medidas de seguridad.* |
| *Monitoring_Server* | *Tener nodejs v10.15.3* | *Servidor de monitoreo que corre una Api para poder ver desde un cliente web los elementos de las colas del servidor de Redis.* |
| *Registration_Application* | *Tener nodejs v10.15.3* | *Aplicación con el mecanismo de generación de tokens.* |
| *ServicesSidePresentation_Server* | *Alto poder de procesamiento y tener nodejs v10.15.3* | *Servidor encargado de recibir requests HTTP a través de una api REST, enviar datos a Cache_Server a través de HTTP y enviar datos a AirlineService a través de HTTP.* |
| *AirlineService* | *No aplica* | *Servicios de las aerolíneas que requieren el servicio* |

| **Conector** | **Características (velocidad, etc.)** | **Descripción** |
|---|---|---|
| *http* | *2.0* | *Se utiliza el conector HTTP por defecto para conectar Redis_Server y en las Api de presentación se utiliza REST con mensajería HTTP.* |

### 3.3.1.3. Relación con componentes

| **Nodo** | **Componente** |
|---|---|
| *ProviderSidePresentation_Server* | *AvAuthFlightData-Api* |
| *ServicesSidePresentation_Server* | *ServiceRegistry-Api*<br>*Services-Api* |
| *Cache_Server* | *Redis-Server* |
| *BusinessLogic_Server* | *Pipeline* |
| *Admin_Workstation* | *Queues-Gui*<br>*TokenGenerator-Api* |
| *Monitoring_Server* | *Queues-Gui* |
| *Registration_Application* | *TokenGenerator-Api* |

#### 3.3.1.4. Decisiones de diseño

La separación física que se vé en el Diagrama, si bien representa un despliegue que consideramos ideal para nuestro proyecto, no es un requerimiento infalible ya que se modularizó de tal forma, que permitiría estar desplegado también en un sólo computador, en caso de que sea necesario para abaratar costos.

Más allá de las tácticas nombradas en las vistas anteriores y para no repetir, se omiten en esta sección, se mostrarán criterios directamente relacionados con la disposición física de los nodos.

Performance:

Control Resource Demand:

- Reduce Overhead: Se utilizan dispositivos específicos para cada servidor, con la finalidad de que no exista un overhead propio de otro servidor corriendo sobre el mismo dispositivo.
- Increase Resource Efficiency: Se deja abierta la posibilidad de personalizar cada servidor con diferente hardware propio a los requerimientos de los mismos.

Manage Resources:

- Increase Resources: Se utilizan múltiples nodos con la finalidad de incrementar los recursos y no utilizar sólo uno el cual tendría un overhead inmenso.

Seguridad:

Resist Attacks:

- Limit Exposure: Se puede ver que los nodos que se comunican con los clientes y proveedores son nodos que están totalmente desacoplados del resto del sistema, y por lo tanto, el resto de la interoperabilidad del middle es totalmente interno al mismo.
- Limit Access/Authorize Actors: Se limita acceso al nodo Admin_Workstation ya que tiene datos sensibles que sólo un administrador autorizado debería poder ingresar.

Disponibilidad:

Detect Faults:

- Monitor: Se puede ver cómo cada componente dentro de los nodos, registra eventos en el log perteneciente a su directorio. Esto con la finalidad de detectar defectos.