Universidad ORT Uruguay

Facultad de Ingeniería

Arquitectura de software
Obligatorio

Joselen Cecilia (233552)
Giovanni Ghisellini (205650)
Mateo Gioia (238327)

Entregado como requisito de la materia Arquitectura de software

Link al repositorio de GitHub:
https://github.com/ORTArqSoft/233552_205650_238327

25 de noviembre de 2021

Índice general

1. Introducción

Este documento tiene como objetivo mostrar y explicar la arquitectura propuesta para el desarrollo del sistema de SenTech, en este se justificaran todas las decisiones tomadas y como estas nos llevaron a la solución planteada actualmente para el sistema.

Junto con cada justificación estarán los diagrama correspondientes y los patrones y tácticas que se aplicaron. Para lo que se hará un análisis exhaustivo del sistema y los requerimientos solicitados e identificados por el equipo.

1.0.1. Problema

La D.I.N.A.M.A (Dirección Nacional de Medio Ambiente) y el nuevo Ministerio de Ambiente decidieron intensificar las tareas de monitoreo sobre variables medio ambientales. Para esto, evaluaron que es necesario incorporar nuevos sensores que le permitan capturar distintos tipos de datos variables en función del tiempo. Actualmente, la plataforma que poseen está sumamente limitada a los sensores instalados, y por lo tanto a las variables que estos monitorean. Es sumamente costoso introducir cambios en la plataforma por lo que se decidió contratar a una empresa del medio local para que se encargue del desarrollo de una nueva. La empresa elegida, llamada SenTech, tiene una amplia trayectoria en el campo de IoT (Internet of Things).

El sistema debe ser capaz de recibir información de distintos sensores. Los sensores pueden enviar información muy variada, y a su vez a través de protocolos distintos (HTTP, MQTT, COAP, etc.). Una vez recibida la información, se debe de poder capturar la información relevante en función de los parámetros que se desean monitorear, para luego poder transformar esta información para que sea fácilmente analizable. Por último, la información debe quedar disponible como datos abiertos para que sea fácilmente consumible por terceros que deseen utilizarla.

Los datos relacionados con los sensores (nombre, tipo, ubicación), como así también las propiedades observadas (por ejemplo, temperatura), deben ser configurables por miembros de la D.I.N.A.M.A, garantizando los mecanismos de seguridad correspondientes, para que la información pueda ser fiable, una vez recibida y procesada.

1.0.2. Propósito

El propósito de este documento es proveer una especificación de la arquitectura y su justificación para el sistema de SenTech.

2. Antecedentes

2.1. Propósito del sistema

SenTech busca brindar toda la información relevante capturada por los sensores a través de su producto de software. El mismo permitirá a sus usuarios consultar sobre las observaciones de las lecturas de los sensores, como así también obtener el promedio de los valores registrados de una propiedad observable de un sensor en función del tiempo.

El sistema deberá transformar y validar los datos procesados, para así poder procesarlos y de esta manera permitirá controlar el flujo de las transacciones.

2.2. Requerimientos significativos de Arquitectura

2.2.1. Requerimientos funcionales

A continuación se presenta una tabla con los requerimientos funcionales del sistema, una descripción y los actores.

Tabla 2.1: Requerimientos funcionales.

Requerimiento	Descripción	Actor
REQ 1 - Mantenimiento de sensores	El sistema deberá permitir registrar y validar la información de los sensores autorizados para reportar datos en la plataforma. Se deben considerarse los posibles errores.	Catalog
REQ 2 – Mantenimiento de Propiedades Observables	El sistema deberá permitir registrar la información de las propiedades observables de los distintos sensores. También, permitirá asociar las propiedades observables a los sensores.	Catalog
Continúa en la siguiente página		

Requerimiento	Descripción	Actor
REQ 3 - Validar y registrar una nueva lectura.	El sistema deberá permitir registrar la información enviada por los sensores generando nuevas lecturas, además se debe validar que el sensor que envía este registrado en el sistema.	Gateway
REQ 4 – Registro de una nueva lectura.	El sistema debe permitir extraer las mediciones, realizar la conversión de unidades y persistir los datos, tanto el resultado final como la lectura que vino desde el sensor.	Observations
REQ 5 – Alertar ante valores anormales.	El sistema debe permitir definir un rango de valores normales, validar las propiedades observables de un sensor con dicho rango y notificar de datos anormales a las personas definidas.	Analytics
REQ 6 – Consulta en función del tiempo de los valores registrados de una propiedad observable de un sensor.	El sistema debe permitir analizar los datos climáticos en función del tiempo y calcular el promedio de los valores obtenidos dado un rango de fechas, una propiedad observable y un sensor.	Analytics
REQ 7 – Registro de consumidor.	El sistema debe permitir registrar la información de los consumidores de los datos que se procesan en la plataforma.	Exporter
REQ 8 - Exportación de datos.	El sistema debe permitir obtener los datos de las lecturas de los distintos sensores que se están procesando.	Exporter

2.2.2. Requerimientos de atributos de calidad

A continuación se presenta una tabla donde se especifican los requerimientos no funcionales teniendo en cuenta que atributo de calidad debería de beneficiar

Tabla 2.2: Requerimientos funcionales.

Requerimiento	Descripción	Actor
REQ 9 – Extensión de soporte de protocolos y formatos en la ingesta de datos.	El sistema deberá permitir agregar nuevos formatos o protocolos al Gateway con el menor impacto posible tanto en tiempo de desarrollo como de despliegue.	Modificabilidad
Continúa en la siguiente página		

Requerimiento	Descripción	Actor
REQ 10 – Disponibilidad del punto de entrada de lecturas.	El sistema deberá estar disponible la mayor cantidad de tiempo posible y poder seguir respondiendo a los sensores, aunque la plataforma este operando de manera degradada o ante un eventual spike de la actividad de los sensores.	Disponibilidad
REQ 11 – Usuarios registrados.	El sistema deberá autenticar a los usuarios contra un proveedor externo, en las funcionalidades que impliquen tanto la gestión de entidades como la consulta de datos.	Seguridad
REQ 12 – Procesamiento de una lectura.	El procesamiento de las lecturas debe permitir agregar, quitar o intercambiar pasos con el menor impacto posible, soportar el reintento de los pasos que no lleguen a cumplirse en caso de que el proceso se vea interrumpido.	Modificabilidad
REQ 13 – Gestión de errores y fallas.	El sistema debe permitir proveer información suficiente para poder analizar los fallos o cualquier tipo de error.	Disponibilidad
REQ 14 – Tiempo de respuesta de consultas.	El sistema debe resolver las consultas de datos sobre Analytics en un tiempo promedio de 5 segundos, siendo el máximo aceptable, 15 segundos.	Performance
REQ 15 - Información de auditoría.	El sistema debe permitir y registrar información que permita realizar auditorías de acceso de forma de identificar los accesos autorizados y no autorizados al sistema.	Seguridad
REQ 16 - Configuración de alertas.	El sistema debe permitir obtener los datos de las modificar fácilmente el conjunto de destinatarios para las alertas ante valores anormales.	Modificabilidad

3. Documentación de la Arquitectura

Aquí se detallará la arquitectura del sistema a desarrollar. Cada vista presentada a continuación permitirá entender y visualizar el funcionamiento del sistema a través de los componentes y conectores. También se visualizará la distribución del sistema y sus módulos.

3.1. Vistas de módulos

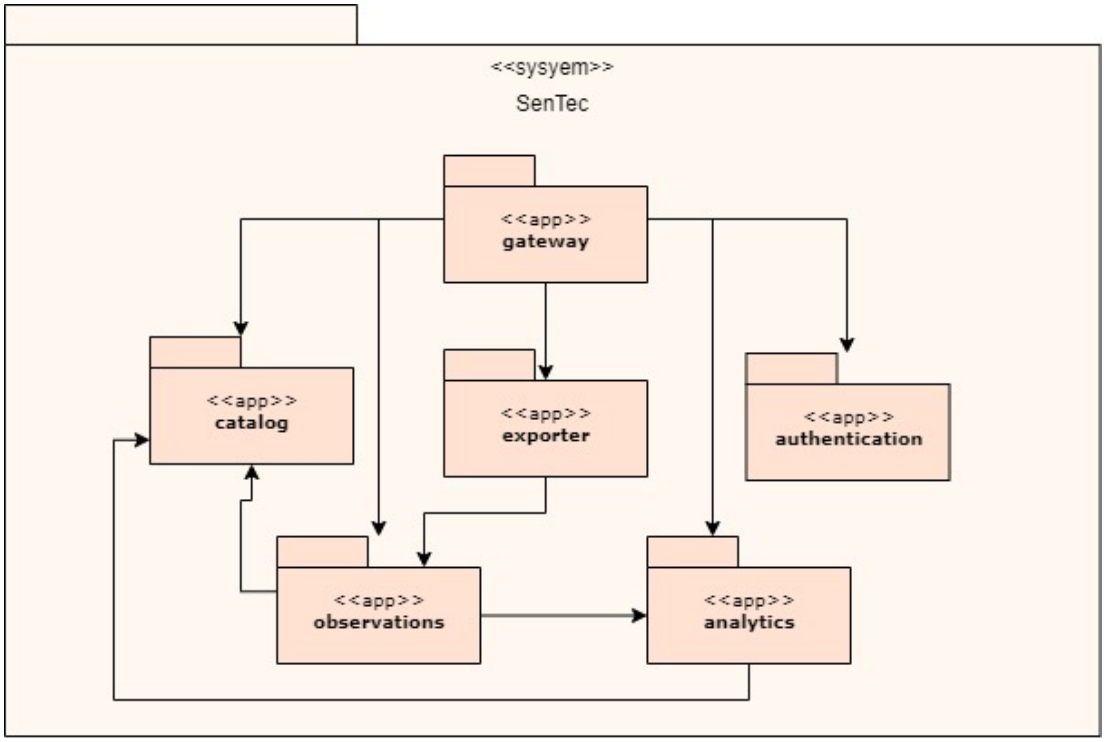
3.1.1. Vista de descomposición general

La vista de descomposición tiene como objetivo describir la estructura jerárquica del sistema.

Para esta vista hicimos una representación de descomposición general, en la que se muestra la estructura de mayor nivel del proyecto y luego fuimos entrando en detalle de la estructura de cada modulo por separado. Queremos aclarar que no dejamos representado en un diagrama la estructura de Catalog, ya que es idéntico a Analytics.

Como vemos es gateway quien conoce a todos los demás módulos ya que es el punto de entrada al sistema, y es quien se encarga de dirigir las request a los demás elementos.

Antes de poder hacer cualquier acción de consulta de datos o de manejo de datos, el gateway hace pasar la llamada por authentication para validar que el consumidor este logueado en el sistema, en caso contrario no podrá acceder a esas funcionalidades.



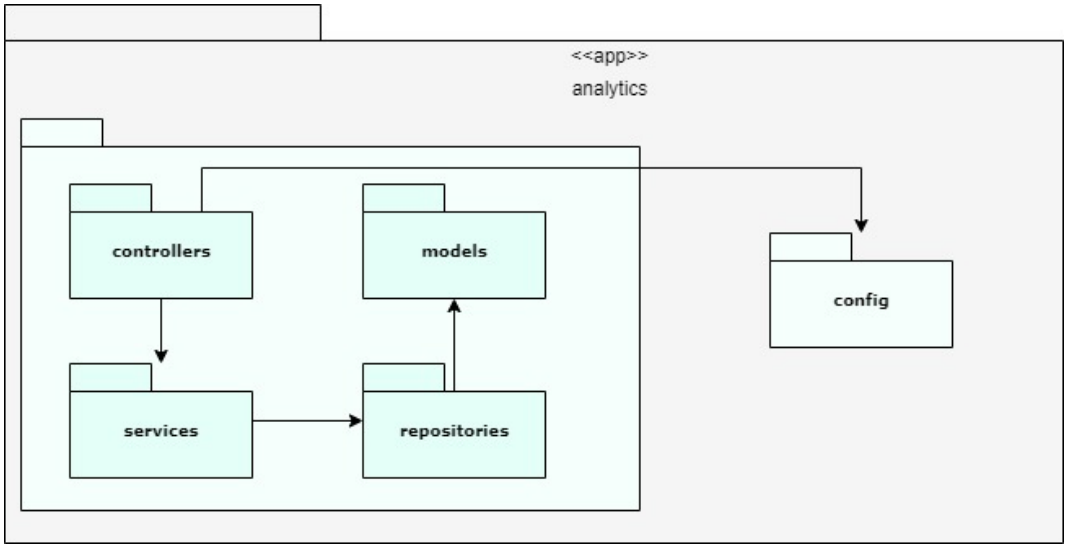
Veremos ahora como es la estructura de cada modulo internamente, se trato de mantener una estructura estándar en todos, con mínimos cambios para facilitar la organización y entendimiento del código.

Debajo de los diagramas, en el catalogo de elementos se explica la funcionalidad de cada elemento.

3.1.2. Vista de descomposición Especifica

Representación primaria Analytics

Como mencionamos anteriormente tienen la misma estructura que catalog.



Catálogo de submódulos analytics

Tabla 3.1: Catálogo de submódulos analytics.

Elemento	Responsabilidades
Analytics.controllers	Se definen aquí los endpoints que exponen desde este modulo y a quien llamar para resolverlos
Analytics.services	Se encarga de recibir las peticiones de los controllers y mandarlas a los repositorios, es un intermediario entre el sistema y la base de datos.
Analytics.models	Es donde se encuentran los diferentes objetos que se manejan y guardan en la base de datos, definiendo acá la estructura de las colecciones de mongo
Analytics.repositories	Aquí se inicializa la base de datos y se establece la conexión con mongoose, también se define aquí el repository encargado de manejar las operaciones de escritura y lectura sobre la base de datos
Analytics.config	Este submódulo se encarga de guardar la configuración general para el funcionamiento del servidor, tales como el connection string y el puerto en el que correrá el servidor.

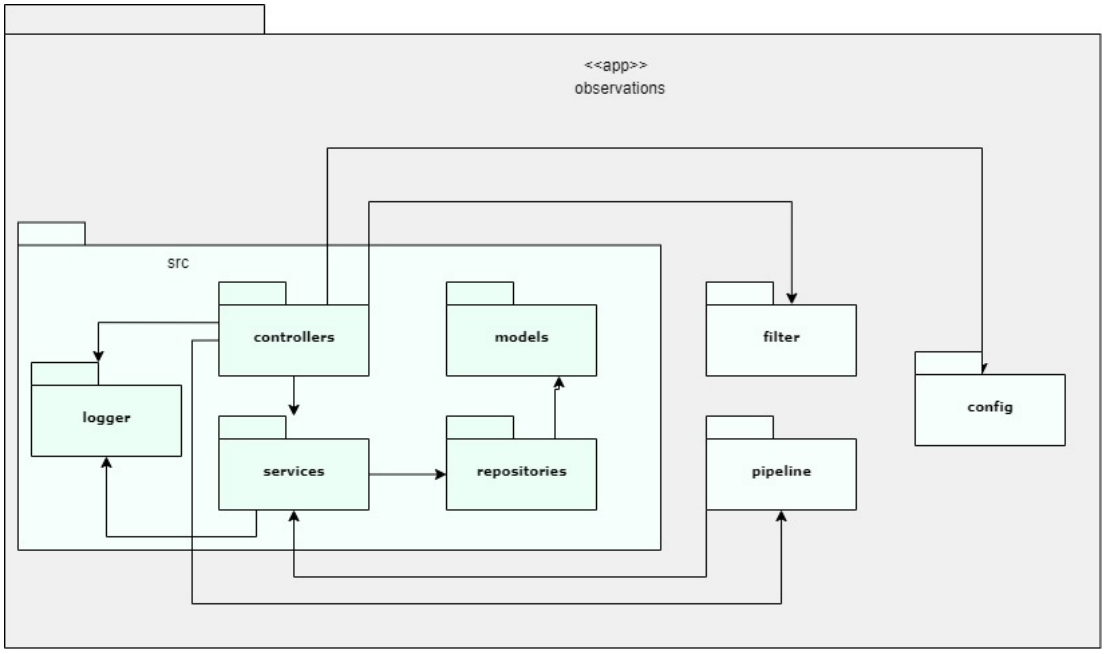
Catálogo de submódulos catalog

Tabla 3.2: Catálogo de submódulos catalog.

Elemento	Responsabilidades
Catalog.controllers	Se definen aquí los endpoints que exponen desde este modulo y a quien llamar para resolverlos
Catalog.services	Se encarga de recibir las peticiones de los controllers y mandarlas a los repositorios, es un intermediario entre el sistema y la base de datos.
Catalog.models	Es donde se encuentran los diferentes objetos que se manejan y guardan en la base de datos, definiendo acá la estructura de las colecciones de mysql
Catalog.repositories	Aquí se inicializa la base de datos y se establece la conexión con sequelize, también se define aquí el Catalog encargado de manejar las operaciones de escritura y lectura sobre la base de datos
Catalog.config	Este submódulo se encarga de guardar la configuración general para el funcionamiento del servidor, tales como el connection string y el puerto en el que correrá el servidor.

Representación primaria Observations

Como mencionamos anteriormente tienen la misma estructura que exporter.



Catálogo de submódulos observations

Tabla 3.3: Catálogo de submódulos observations.

Elemento	Responsabilidades
Observations.controllers	Se definen aquí los endpoints que exponen desde este modulo y a quien llamar para resolverlos
Observations.models	Es donde se encuentran los diferentes objetos que se manejan y guardan en la base de datos, definiendo acá la estructura de las colecciones de mongoose
Observations.services	Se encarga de manejar las acciones a realizar por los controllers de Observations, como la interacción con la base de datos
Observations.repositories	Aquí se inicializa la base de datos y se establece la conexión con sequeleze, también se define aquí el repository encargado de manejar las operaciones de escritura y lectura sobre la base de datos
Observations.logger	Es el encargado de generar un archivo de reporte de errores ocurridos en las funciones de este modulo

Continúa en la siguiente página

Elemento	Responsabilidades
Observations.filter	Este módulo contiene a todos los filtros que procesan y transforman las observaciones.
Observations.pipeline	Provee una interfaz que permite instanciar el pipeline y utilizar operaciones para ejecutar los diversos filtros y así procesar los datos.
Observations.config	Este submódulo se encarga de guardar la configuración general para el funcionamiento del servidor, tales como el connection string y el puerto en el que correrá el servidor.

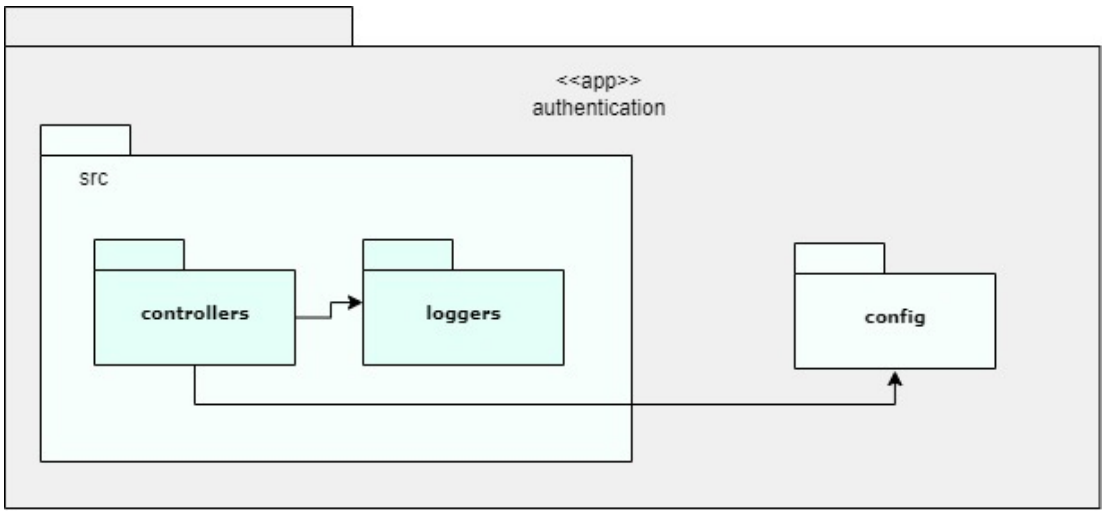
Catálogo de submódulos exporter

Tabla 3.4: Catálogo de submódulos exporter.

Elemento	Responsabilidades
Exporter.controllers	Se definen aquí los endpoints que exponen desde este modulo y a quien llamar para resolverlos
Exporter.services	Se encarga de recibir las peticiones de los controllers y mandarlas a los repositorios, es un intermediario entre el sistema y la base de datos.
Exporter.models	Es donde se encuentran los diferentes objetos que se manejan y guardan en la base de datos, definiendo acá la estructura de las colecciones de mongo
Exporter.repositories	Aquí se inicializa la base de datos y se establece la conexión con mongoose, también se define aquí el repository encargado de manejar las operaciones de escritura y lectura sobre la base de datos
Exporter.logger	Es el encargado de generar un archivo de reporte de errores ocurridos en las funciones de este modulo
Continúa en la siguiente página	

Elemento	Responsabilidades
Exporter.config	Este submódulo se encarga de guardar la configuración general para el funcionamiento del servidor, tales como el connection string y el puerto en el que correrá el servidor.

Representación primaria authentication



Catálogo de submódulos authentication

Tabla 3.5: Catálogo de submódulos authentication.

Elemento	Responsabilidades
Authentication	Realiza las operaciones de autenticación de las peticiones que llegan a gateway por parte de los consumidores
Authentication.controllers	Se definen aquí el endpoint de authenticacion y a quien llamar para resolverlos
Authentication.logger	Es el encargado de generar un archivo de reporte de errores y de auditorías de acceso ocurridos en las funciones de este modulo

Representación primaria Gateway

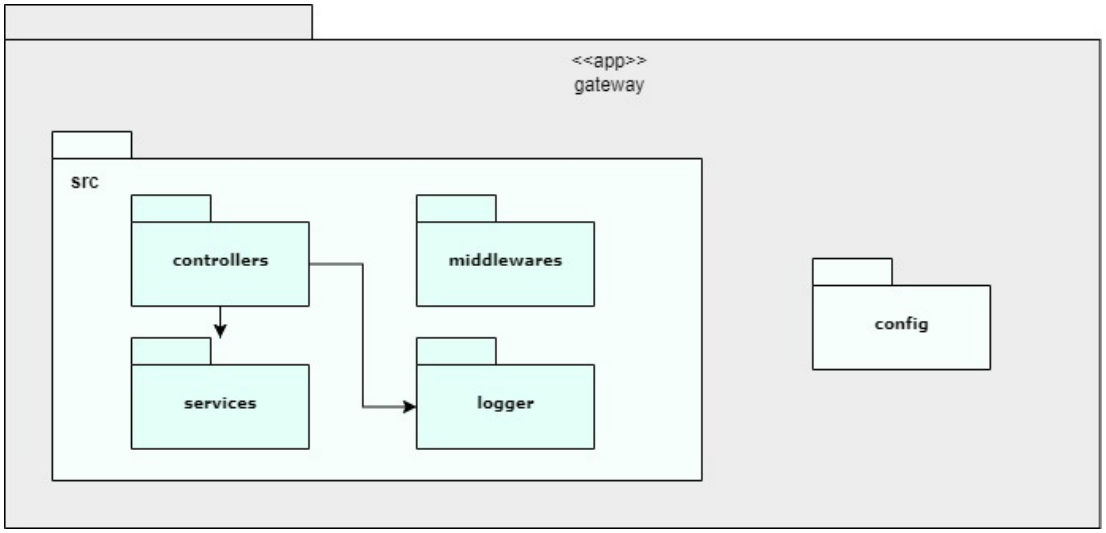
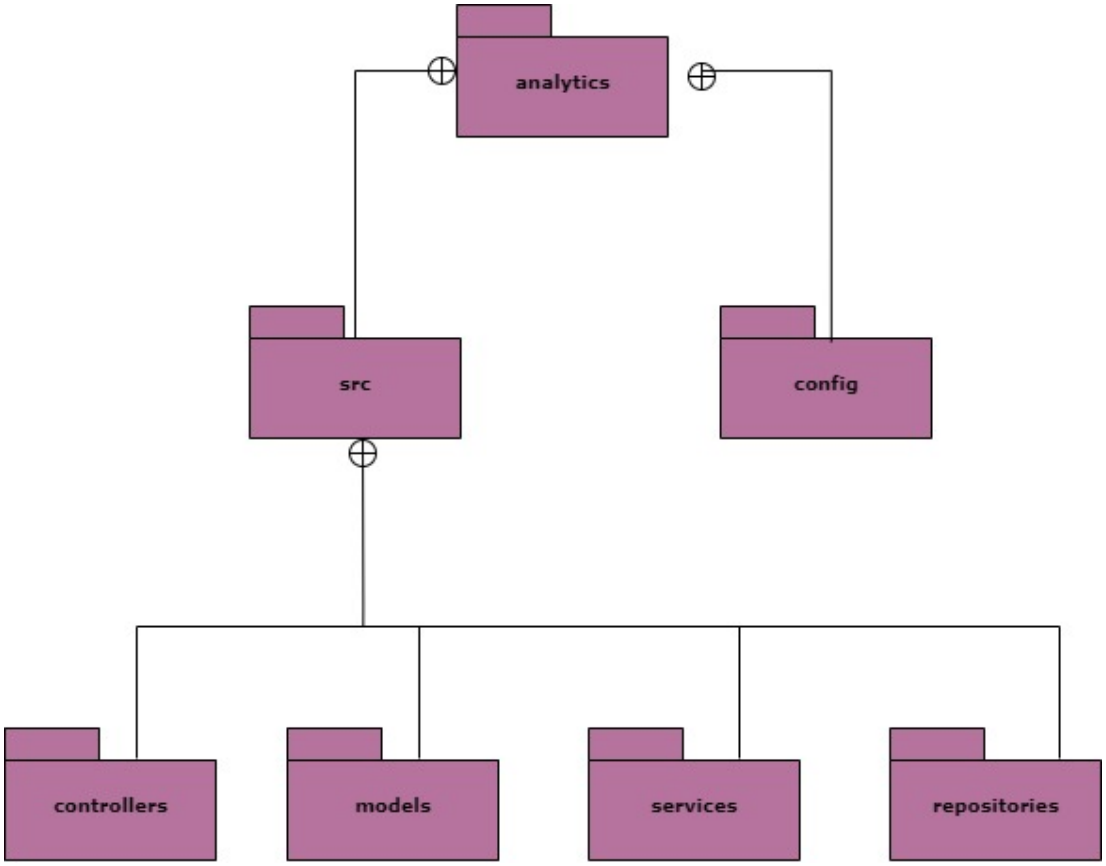


Tabla 3.6: Catálogo de submódulos gateway.

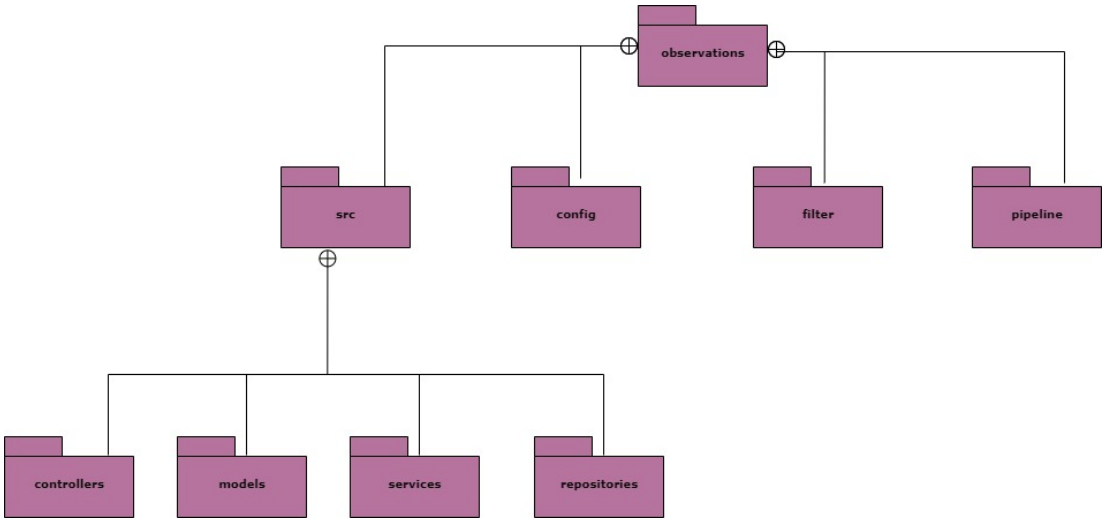
Elemento	Responsabilidades
Gateway.controllers	Se definen aquí los endpoints que exponen desde este modulo y a quien llamar para resolverlos
Gateway.services	Se ecnarga de realizar las peticiones como cliente HTPP mediante axios a los demas servicios.
Gateway.middlewares	Se encarga de definir el tipo en el que pueden llegar las request al sistema y en el que se devolverán las responds
Gateway.logger	Es el encargado de generar un archivo de reporte de errores ocurridos en las funciones de este modulo
Gateway.config	Este submódulo se encarga de guardar la configuración general para el funcionamiento del servidor, tales como el connection string y el puerto en el que correrá el servidor.

3.1.3. Vista de descomposición Especifica

Representación primaria Analytics
catalog tiene la misma estructura



Representación primaria Observations
exporter tiene la misma estructura



3.1.4. Decisiones de diseño

Como se fue mencionando a lo largo que se iban mostrando los modelos se trato de mantener un estándar en la estructura de los distintos módulos, la cual se basaba en una estructura de básicamente cuatro capas, controller - service - repository - model, mas una capa de configuración. La misma vario en alguno de los modelos

pero mínimamente. Mantener este estándar nos ayudo a organizar la estructura de los micro servicios y entender mejor el problema.

Podemos ver reflejado en los diagramas anteriores el claro manejo de responsabilidades de los módulos del sistemas, siguiendo los principios **SOLID** y el **principio de dependencias estables de paquetes**, logrando así el bajo acoplamiento favoreciendo la mantenibilidad del sistema.

La división en submódulos nos permitió separa las responsabilidades y asi lograr **independencia** entre ellos.

Una de las variantes en los distintos módulos fue la existencia de **loggers**, estos existen en los casos donde se realizan auditorías de acceso y en gateway que accede a todos los modulso y puede llevar un control general de errores.

3.1.5. Vista de Uso

En esta viste se mostraran las dependencias de uso de los submódulos.

Representación primaria Observations

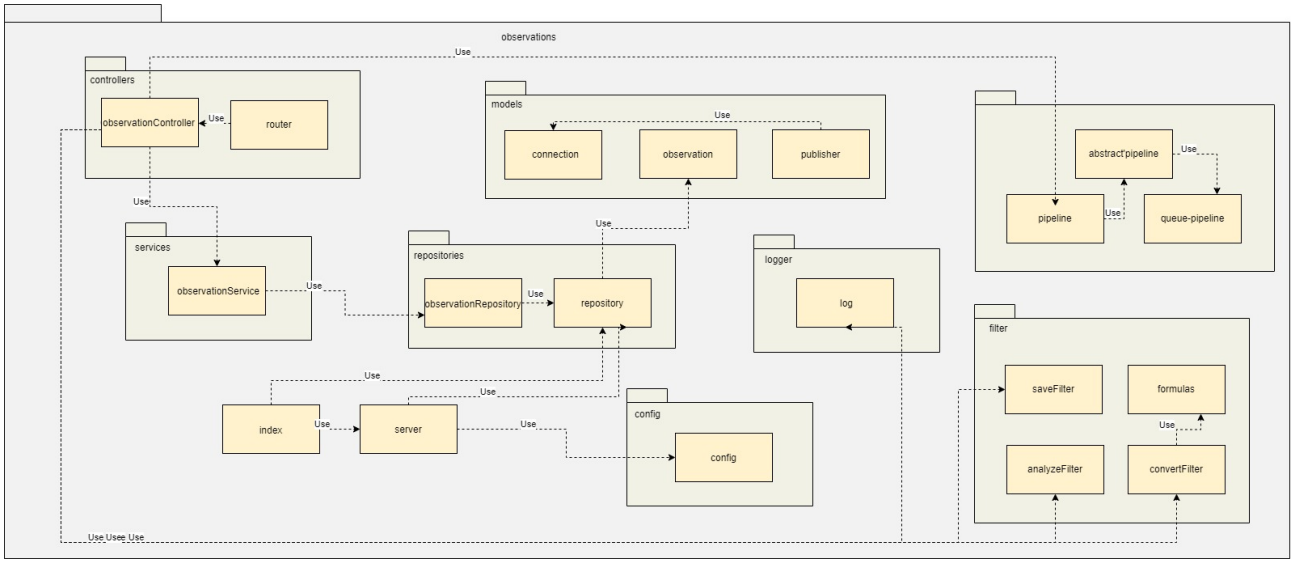


Tabla 3.7: Catálogo de elementos observations.

Elemento	Responsabilidades
index	Es el encargado de iniciar el modulo observations, se inicializa aquí también el repositorio de observations.

Continúa en la siguiente página

Elemento	Responsabilidades
server	Es el encargado de inicializar el servidor de observations, escuchando en el puerto indicado en la configuración del mismo.
router	Es el encargado de definir los endpoints para obtener las observaciones, guardarlas y filtrarlas por date y sensor, para lo que llama a los métodos de observationController para dirigir la request
observationController	Se encarga de aceptar las request dirigidas por router, y armar la request que se le enviaran al service para que ejecute las funciones, lego de esto el controller obtiene la respuesta del service y arma la response que se le muestra al usuario.
observationService	Se encarga de recibir las request del controller y las pasa al repository, es un intermediario entre el controller y el repository
observationRepository	Se encarga de guardar y obtener las observations de la base de datos
repository	Se encarga de inicializar el repositorio donde se guardan las observations
log	Se encarga de registrar los errores ocurridos durante las operaciones sobre observations
connection	Se encarga de inicializar un cliente redis para que el publisher pueda usarlo
observation	Se encarga de definir el modelo para guardar las observaciones en la base de datos de mysql, este contienen las properties necesarias para guardar toda la información de las observaciones.
publisher	Es el que envía mediante una cola la data para que se analizada en analytics
config	Contiene el puerto al que se conecta el modulo analytics, el connection string a la base de datos de mysql, el tipo de pipeline que se va a usar que en este caso solo tenemos la implementación de queue y los datos para la conexión a redis

Continúa en la siguiente página

Elemento	Responsabilidades
analyxeFilter	Es el filtro que al momento de guardar una observación se encarga de enviar la data a analytics mediante el publisher para que la analice.
convertFilter	Es el filtro que al momento de guardar las observations se encarga de transformar la data a la estándar
saveFilter	Es el ultimo de los tres filtros y se usa para pedir al service que guarde la data
formulas	En este archivo se pusieron en un diccionario de clave valor, todas las formulas de las unidades que el sistema es capaz de transformar
pipeline	Es aquí donde se decide que tipo de pipeline se va a usar según lo definido en el archivo config, a partir de este se crea la instancia, es un defer binding
abstract-pipeline	Es una clase abstracta que define los métodos que debe tener un pipeline para que nuestro sistema siga funcionando
queue-pipeline	Es el filtro concreto que se usa actualmente en el sistema, este recibe en una cola los filtros que se van encolando para luego y sacándolos de a uno e ir ejecutándolos en orden, para respetar este se va fijando en el nombre de los filtros.

Decisiones de diseño

REQ 4 – Extracción de observaciones de una lectura dentro de observationsController se encuentra el método saveObservation que se encarga de extraer los valores de la observación, guardar en observation tanto la data que llega como la data que es transformada a la estándar de la propiedad observable a la que corresponde, para lo que usa tres filtros distintos:

convertFilter, en este filtro se busca la property con el mismo nombre, y del sensor al que pertenece la observation, para encontrar la unidad estándar. Luego con esa unidad y la que llego con la observación, buscar una formula para transformar la data a la estándar, en caso de querer extender el filtro para transformar mas unidades, simplemente se agrega la nueva formula al dicccionario de clave valor definido en formulas, ya que es imposible definir todas las transformaciones que existen. Se definieron solo aquellas que se creían mas útiles.
Al extraer en un archivo aparte las formulas, se hace que se mas fácil extender a

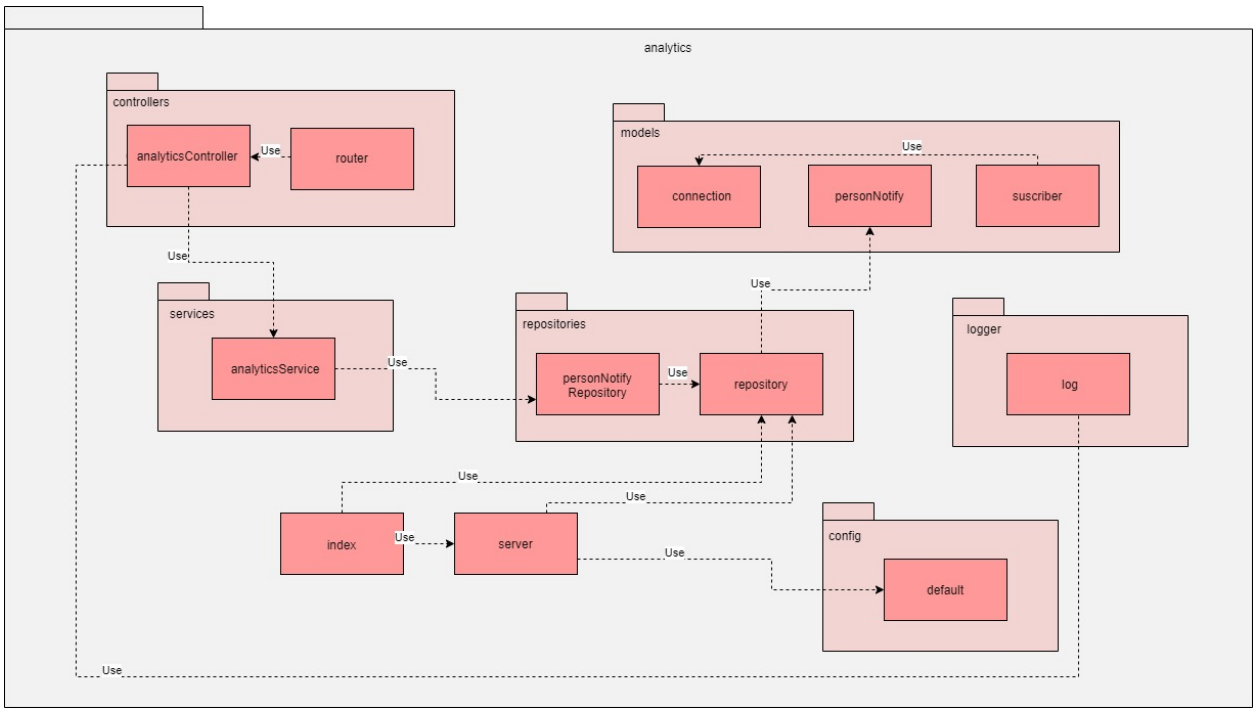
nuevas conversiones, en caso de ser necesario.

analyzeFilter luego de transformada la data, se manda la observación a Analytics para lo que se usa una cola bull que va encolando toda la data para que luego analytics vaya desencolando y analizando, proceso que se describirá en analytics. Para esto se uso una especie de **publish suscribe** pero con un solo suscribe, que es analytics y esta todo el tiempo esperando nueva data para analizar.

saveFilter se guardan todos los datos de la observación que llego junto con la data que se transformo, la unidad estándar, el tiempo que llevo el proceso anterior, el camino de la property, y el esn del sensor al que pertenece esta observación.

REQ 12 – Procesamiento de una lectura para permitir el cambio de pasos o la extensión de estos al procesar una observation, se implementa un **pipe and fillter**, permitiendo así agregar fácilmente nuevos pasos, creando nuevos filtros que luego se deben agregar haciendo un use de ellos, o cambiar el orden de estos cambiando simplemente el orden en el que se usan los filtros. Como se menciono, también quedo abierta la implementación de como procesar los filtros en el pipeline, actualmente se hace con una cola, pero la implementación del **defer binding** juntos con la clase abstracta, permite que se agreguen nuevas implementaciones si los nuevos pasos lo requieren

Representación primaria Analytics



Catálogo de elementos

Tabla 3.8: Catálogo de elementos Analytics.

Elemento	Responsabilidades
index	Es el encargado de iniciar el modulo analytics, iniciar el repositorio de personas a notificar y inicializar a suscriber.
server	Es el encargado de inicializar el servidor de analytics en el puerto indicado en el archivo config.
router	Es el encargado de definir los endpoints para agregar o borrar personas a notificar, y calcular promedios por fecha y parámetros , para lo que llama a los métodos de analyticsController para dirigir la request
analyticsController	Se encarga de aceptar las request dirigidas por router, y armar la request que se le enviaran al service para que ejecute las funciones, lego de esto el controller obtiene la respuesta del service y arma la response que se le muestra al usuario.
analyticsService	Se encarga de recibir las request del controller y las pasa al repository, es un intermediario entre el controller y el repository, es también este el encargado de enviar los mails ante valores anormales a las personas a notificar y el encargado de ver cuales son los parámetros con los cuales se calculara el promedio
personNotifyRepository	Se encarga de guardar, borrar y obtener las personas a notificar de la base de datos
repository	Se encarga de inicializar el repositorio donde se guardan las personas a notificar
connection	Se encarga de inicializar un cliente redis para que el suscriber pueda usarlo
personNotify	Se encarga de definir el modelo para guardar las personas a notificar en la base de datos de mongo, este contienen las properties necesarias para poder enviarle mails a las personas, como su email.

Continúa en la siguiente página

Elemento	Responsabilidades
suscriber	Recibe mediante la cola los datos a analizar, y en caso de que los datos estén fuera de rango mandar mails a las personas antes definidas
config	Contiene el puerto al que se conecta el modulo analytics, el connection string a la base de datos de mongodb, y los datos para la conexión a redis

Decisiones de diseño

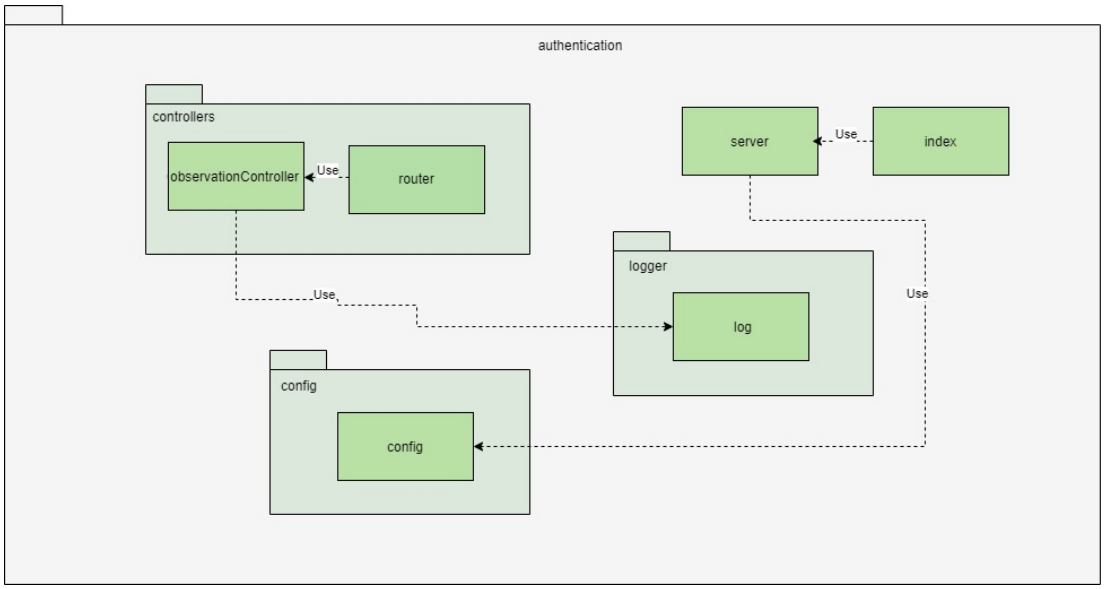
REQ 5 – Alertar ante valores anormales Se define un rango por propiedad observable, si el valor de la observación esta por fuera de ese rango, se envía un mail a todos las personas registradas como personas a notificar. Para esto se utiliza el modulo nodemailer y se crea una cuenta de gmail para el obligatorio, ya que al momento de crear el transporte para enviar el mail se debe especificar un email y una password.
Para guardar las personas a notificar, se uso una base de datos en mongo y se creo una colleccion PersonNotify, a la cual se agregan personas o se borran de esta mediante endpoints en gateway que mandan las request a analytics. Se decidió persistirlas en mongo ya que se busca una alta performance en analytics y queríamos consultas rápidas.
Esta funcionalidad se llama en el filtro analyzeFilter de observation.

REQ 6 – Consulta en función del tiempo de los valores registrados de una propiedad observable de un sensor Se recibe un rango de fechas y un parámetro que puede ser: day, days, month, months, year o years y se calcula el promedio de valores observados dado ese rango de fechas, agrupándolos por el parámetro. Para esto, se manda mediante axios la request a observations, para obtener las observaciones que se dieron durante esa fecha. Luego, a partir de estas, se arma la response según el parámetro por mes, dia o año y el promedio de valores observados.

REQ 14 – Tiempo de respuesta de consultas Se da prioridad a la operación de consultas para reducir el tiempo de respuesta, haciendo asincrónica la función de envío de mails para que hacer frente a ambas request no lleve mas tiempo por tener que esperar a que terminen las otras funciones. Esto se logra utilizando la tactica **Introducir simultaneidad**. Tambien se aplica el **patron CQRS** sobre este modulo junto con observations ya que observations se encarga de la escritura y actualizacionde las observaciones y analytics solo de la lectura lo que beneficia la performance que era el principal requerimiento pero tambione la mantenibilidad y flexibilidad.

REQ 16- Configuración de alertas El conjunto de personas a notificar para las alertas ante valores anormales es fácilmente modificable, ya que se creo una base de datos en mongo y se crearon puntos de entrada para poder agregar y quitar fácilmente personas de la lista. Esto hace que en tiempo de ejecución se puedan modificar estos destinatarios con una simple request al gateway quien dirigirá esta a analytics.

Representación primaria Authentication



Catálogo de elementos authentication

Tabla 3.9: Catálogo de elementos Authentication.

Elemento	Responsabilidades
index	Es el encargado de iniciar el modulo authentication
server	Es el encargado de inicializar el servidor de authentication
router	Es el encargado de definir el endpoint para autenticar a los consumidores que llegan con la consulta, este modulo actúa como un proveedor externo.
	Continúa en la siguiente página

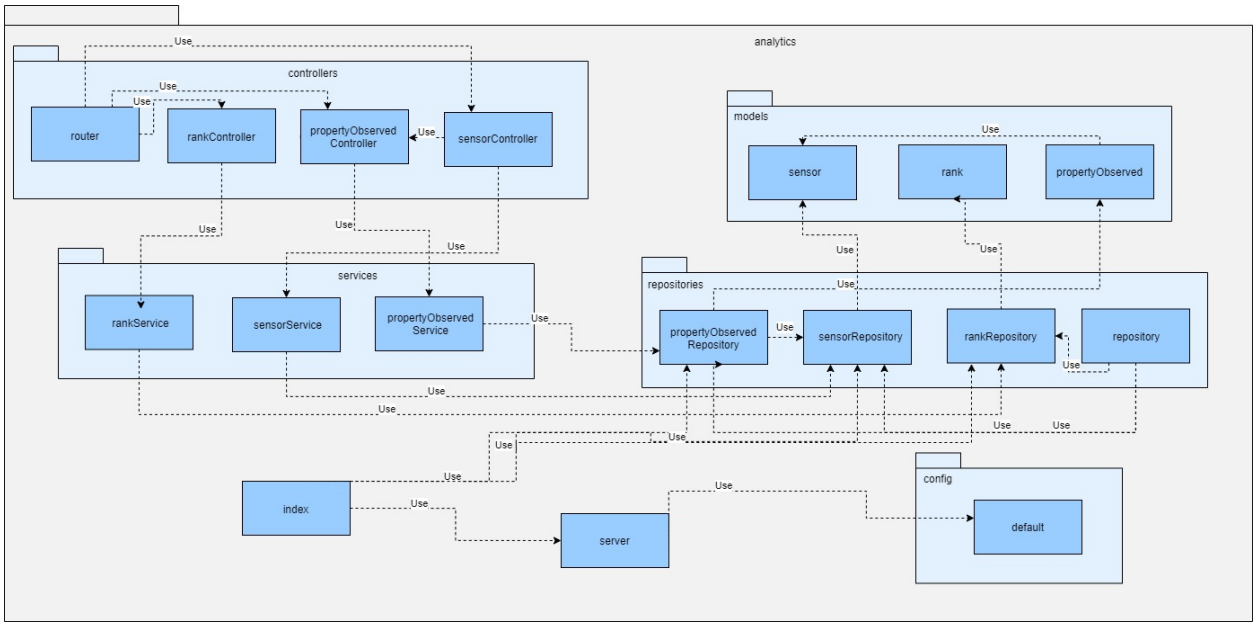
Elemento	Responsabilidades
authenticationController	Se encarga de aceptar la request dirigida por router y con el uso de **JWT** mediante la **JWT SECRET** y el **token** recibido en el header de la request, ver si el usuario esta logueado, verificando con la secret key que la firma es valida y aceptar la request o en caso contrario denegarla, lo que se traduce a si tiene acceso a la función o en caso contrario tirar un error informando al usuario que debe loguearse antes de realizar esa operación. Se utiliza en esta clase **dotenv**, para definir la variable de entorno **JWT SECRET** que pasa a ser la secret key de **JWT** para generar la firma del token
log	Se encarga de registrar las auditorías de acceso, ya sea que se acepte o no el acceso a al funcionalidad.
config	Contiene el puerto al que se conecta el modulo authentication
.env	Se encuentran definidas las private key para usar en la autenticación de JWT

Decisiones de diseño

REQ 11 – Usuarios registrados Para que los usuario puedan hacer uso de las funcionalidades de consulta y de gestión de entidades, primero deben loguearse. Entonces, antes de pasar a las funcionalidades, el gateway en estos casos llama primero al 'proveedor externo' authentication, quien se encarga de validar como explicamos antes. Por ejemplo, si el usuario dado esta logueado en el sistema, se hace uso del patrón de **identidad federada** delegando la autenticación a un proveedor de identidad externo que no es el mismo gateway, se simula con el modulo authentication el proveedor externo.

REQ 15- Información de auditoría Para registrar información que permita realizar auditorías de acceso de forma de identificar los accesos autorizados y no autorizados al sistema, se utilizaron logs. Para esto se se utilizo la biblioteca winston, creando un transporter que aloja los logs en un archivo local llamado logs. Cuando ocurre un error de autenticación porque el usuario no esta logueado, no solo se avisa al usuario, si no que también se crea un log de error indicando por que fallo. En caso contrario se emite un log de info.

Representación primaria catalog



Catálogo de elementos catalog

Tabla 3.10: Catálogo de elementos catalog.

Elemento	Responsabilidades
index	Es el encargado de iniciar el modulo catalog, iniciar el repositorio de catalog, inicializando los repositorios de sensores, propiedades observables y rangos.
server	Es el encargado de inicializar el servidor de catalog en el puerto indicado en el archivo config.
router	Es el encargado de definir los endpoints para agregar y/o obtener las propiedades observable, rangos y sensores, derivando estas request a los controller sensorControlller, propertyOBservedController, y rankController
sensorControlller, propertyOBservedController, y rankController	Se encarga de aceptar las request dirigidas por router, y armar la request que se le enviaran al service para que ejecute las funciones, lego de esto el controller obtiene la respuesta del service y arma la response que se le muestra al usuario.

Continúa en la siguiente página

Elemento	Responsabilidades
sensorService, propertyObservedService, y rankService	Se encarga de recibir las request del controller y las pasa al repository que corresponda, es un intermediario entre el controller y el repository.
sensorRepository, propertyRepository, y rankRepository	Se encarga de obtener y guardar los elementos de la base de datos
repository	Se encarga de inicializar el repositorio donde se guardan los sensores, rangos y propiedades observables
config	Contiene el puerto al que se conecta el modulo catalog y el connection string a la base de datos de mysql.

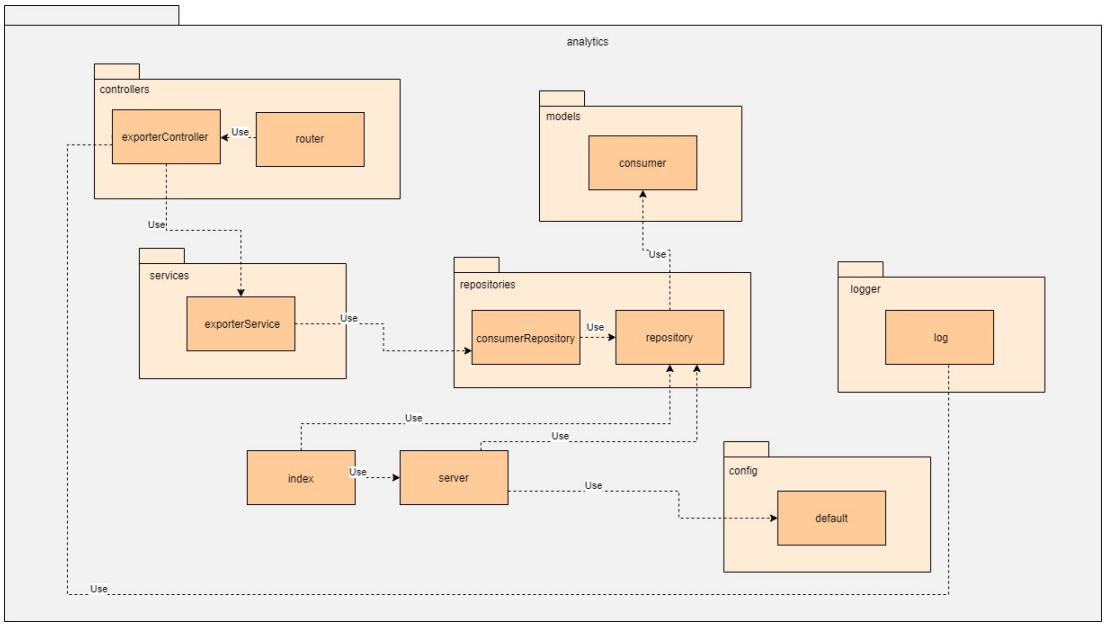
Decisiones de diseño

REQ 1 - Mantenimiento de Sensores Se permite registrar la data de los sensores, validando que los campos no sean nulos, que el ESN del sensor sea único y alfanumérico, sin caracteres especiales y con un largo máximo de 16 caracteres. En caso contrario, se lanza un error al usuario infórmandole que la data es invalida.
Se permite agregar con el sensor una lista de propiedades observables para asociárselas al sensor.

REQ 2 - Mantenimiento de Propiedades Observables Se permite registrar las propiedades observables con un nombre y una unidad, y asignándoles en algún caso un sensor referenciando al id del sensor.

Esta fue una de las razones por la cual elegir para persistir estas entidad mysql ya que debemos mantener varias relaciones y con esta base de datos es mas sencillo.

Representación primaria exporter



Catálogo de elementos exporter

Tabla 3.11: Catálogo de elementos exporter.

Elemento	Responsabilidades
index	Es el encargado de iniciar el modulo exporter e inicializar el repositorio de este
server	Es el encargado de inicializar el servidor de exporter
router	Es el encargado de definir los endpoints para agregar consumidores, hacer el login y exportar la data.
exporterController	Se encarga de aceptar las request dirigidas por router, y armar la request que se le enviaran al service para que ejecute las funciones, luego de esto el controller obtiene la respuesta del service y arma la response que se le muestra al usuario.
exporterService	Se encarga de recibir las request del controller y las pasa al repository, es un intermediario entre el controller y el repository.

Continúa en la siguiente página

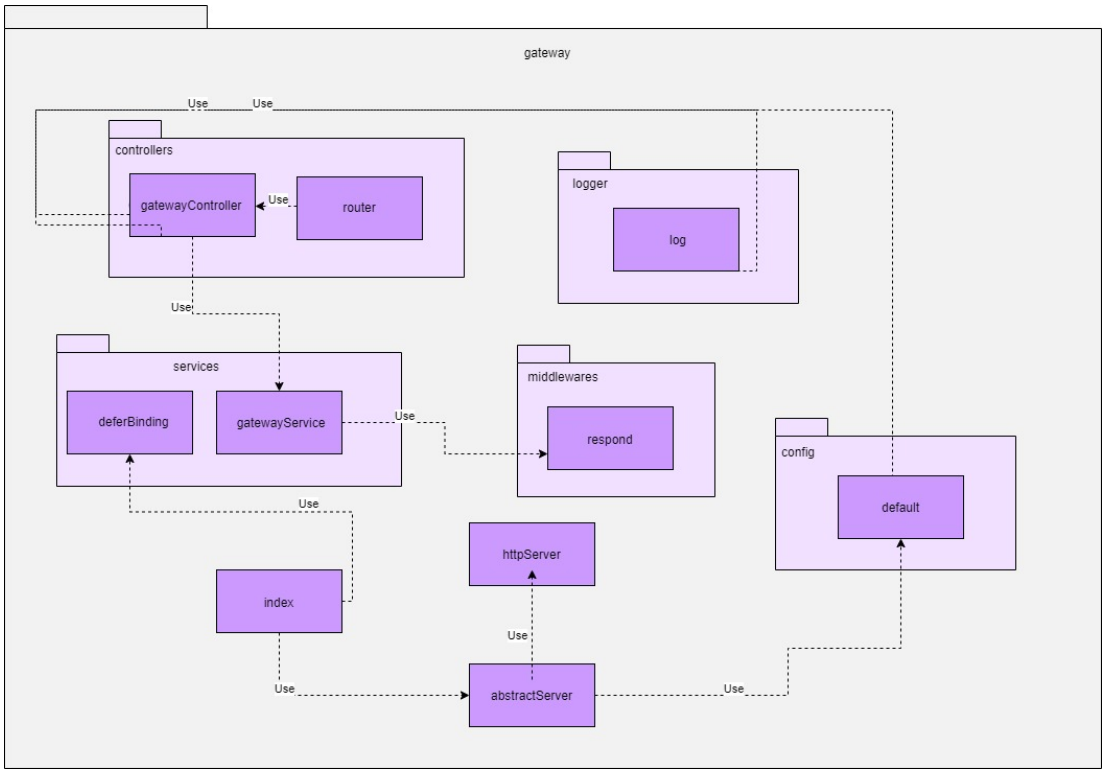
Elemento	Responsabilidades
personNotifyRepository	Se encarga de guardar,y obtener a los consumidores de la base de datos, asignándole antes la URL única con la que el consumer debe acceder a obtener la data, también se encrypta la contraseña del consumidor con el uso de md5 antes de guardar el consumidor. Se hace aquí también la búsqueda del usuario que se quiere loguear para ver si existe en la base de datos con ese email y password que se encrypta antes de la búsqueda y si existe se crea el token con el uso de **JWT** usando la información del consumidor y la JWT SECRET que mencionamos antes.
repository	Se encarga de inicializar el repositorio donde se guardan las personas a notificar
log	Se encarga de registrar los errores ocurridos durante las operaciones sobre exporter
config	Contiene el puerto al que se conecta el modulo exporter, el connection string a la base de datos de mongodb.

Decisiones de diseño

REQ 7 – Registro de consumidor Para poder luego validar los consumidores del sistema, se deben guardar los mismos. Para esto se le piden ciertos datos, además de un email y password, para poder hacer luego el login. Como mencionamos antesm, se usa el algoritmo **md5** para hashear la contraseña antes de guardar al consumidor, ya que se uso una base de datos no relacional.

REQ 8 – Exportación de datos El consumidor accede directamente a exporter para usar este endpoint. Cuando accede por primera vez, se le devuelve la data a partir de la fecha en la que este se registro. Luego, cada vez que se accede al endpoint, se actualiza la fecha desde la que se debe acceder a la data del consumir.

Representación primaria gateway



Catálogo de elementos

Tabla 3.12: Catálogo de elementos Analytics.

Elemento	Responsabilidades
index	Es el encargado de iniciar el modulo gateway en el puerto indicado en el config.
httpServer	Es el encargado de levantar el servidor usando el protocolo http y el tipo e request y respond json.
abstractServer	Es la clase abstracta que define la estructura de los servers para gateway
router	Es el encargado de definir los endpoints de todo el sistema, ya que gateway es el punto de entrada al sistema.
gatewayController	Se encarga de aceptar las request dirigidas por router, y armar la request que se le enviaran al service para que ejecute las funciones, lego de esto el controller obtiene la respuesta del service y arma la response que se le muestra al usuario.

Continúa en la siguiente página

Elemento	Responsabilidades
gatewayService	Se encarga de recibir las request del controller y decide a que modulo debe pasárselas mediante el uso de **axios** par que las resuelvan
deferBinding	Se encarga de instanciar el tipo de server indicado en el config, que por defecto ahora es el http server json
respond	Se encarga de definir el tipo de requests y responds que acepta el modulo que en este caso es json.
log	Se encarga de registrar los errores ocurridos en los distintos endpoints
config	Contiene el puerto al que se conecta el modulo gateway y el tipo de servidor que se quiere usar, en este caso http3

Decisiones de diseño

REQ 3 – Registro de una nueva lectura Gateway es el punto de entrada a la aplicación y es quien expone los endpoint para todas las funcionalidades del sistema, menos la de exportar data que se encarga directamente exporter. Lo que hace es, a través de la peticiones, redirigir las mismas a los distintos servicios mediante el uso de **axios**, para que así los distintos módulos sigan siendo independientes a diferencia si hiciéramos directamente un require. Actualmente el servidor acepta request con protocolo http y de tipo JSON.

REQ 9 – Extensión de soporte de protocolos y formatos en la ingesta de datos Actualmente el server que usa gateway usa protocolo http y formato JSON, pero esto se puede modificar y extender fácilmente, ya que se hace uso de **defer binding** para decidir que server usar. Lo único que se debería hacer al momento de querer cambiar, es agregar un nuevo server con el protocolo y formato que se desea y cambiar el tipo de server en el archivo config.

REQ 10 – Disponibilidad del punto de entrada de lecturas Se hace manejo de errores en toda la aplicación para aumentar la disponibilidad de la misma. Además se cuenta con el modulo logger, que se llevo a cabo por la necesidad de manejar e identificar que esta sucediendo en el sistema, que operaciones se llevan a cabo y tener información útil en caso de que ocurran defectos o fallas. De esta manera se aumenta la disponibilidad del sistema y la testabilidad.

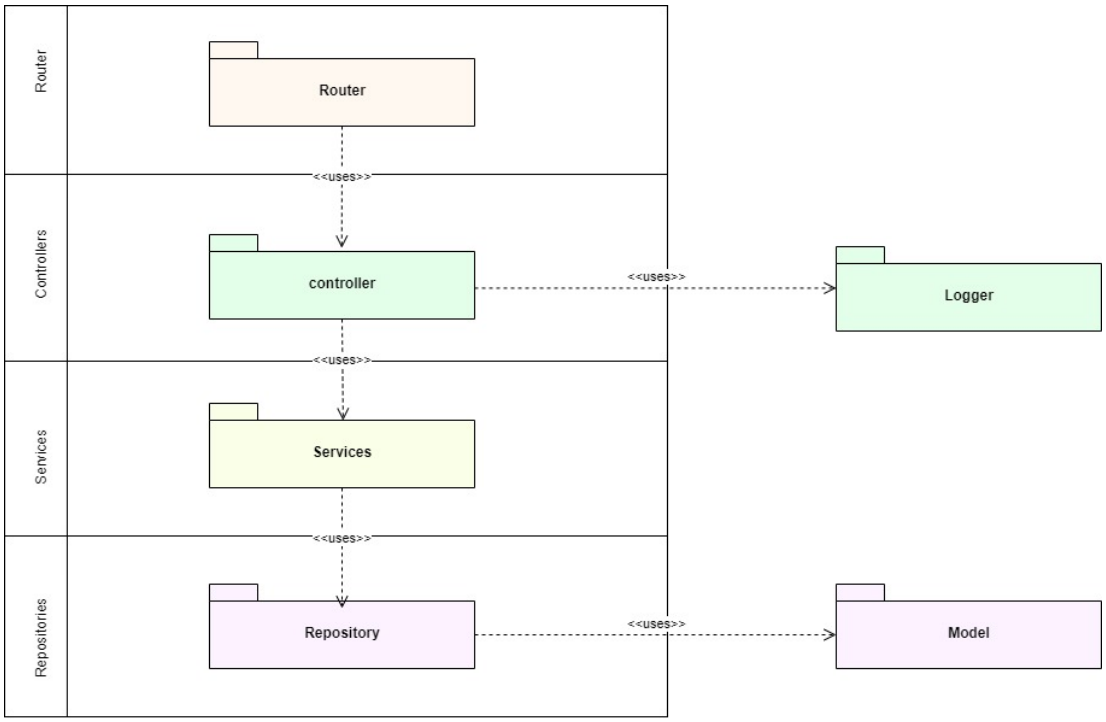
REQ 13 – Gestión de errores y fallas Este requerimiento va de la mano con el anterior. Como mencionamos, dentro de la aplicación se hace control de errores, teniendo en las capas de mas bajo nivel lanzamiento de errores, y en las capas de

mas alto nivel, la captura de esto para luego generar informes de logs y devolver el contexto al cliente con los mensajes de error y el codigo que corresponda.

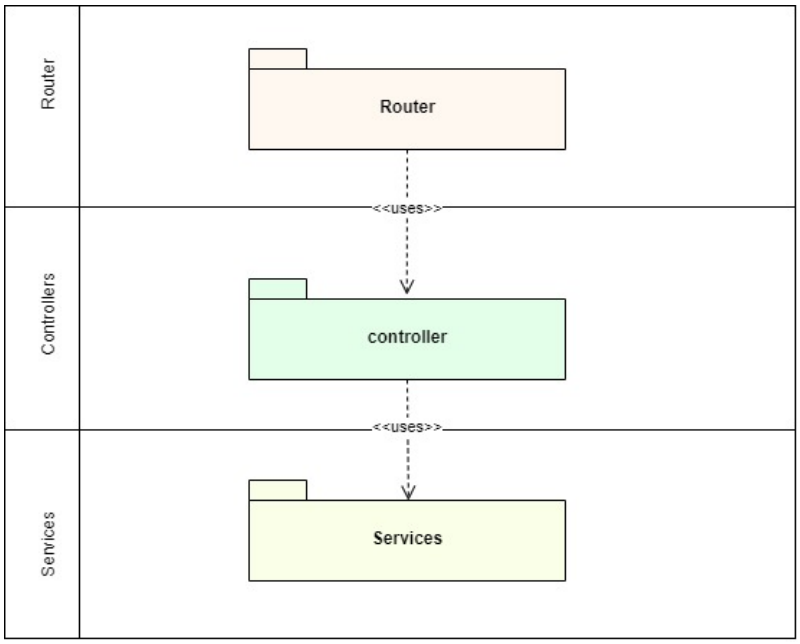
3.1.6. Vista de Layers

Aquí se pueden apreciar los módulos del sistema agrupados conforme a sus responsabilidades. El catalogo de elementos se sigue manteniendo como en los puntos anteriores, por lo que no se vuelve a explicar.

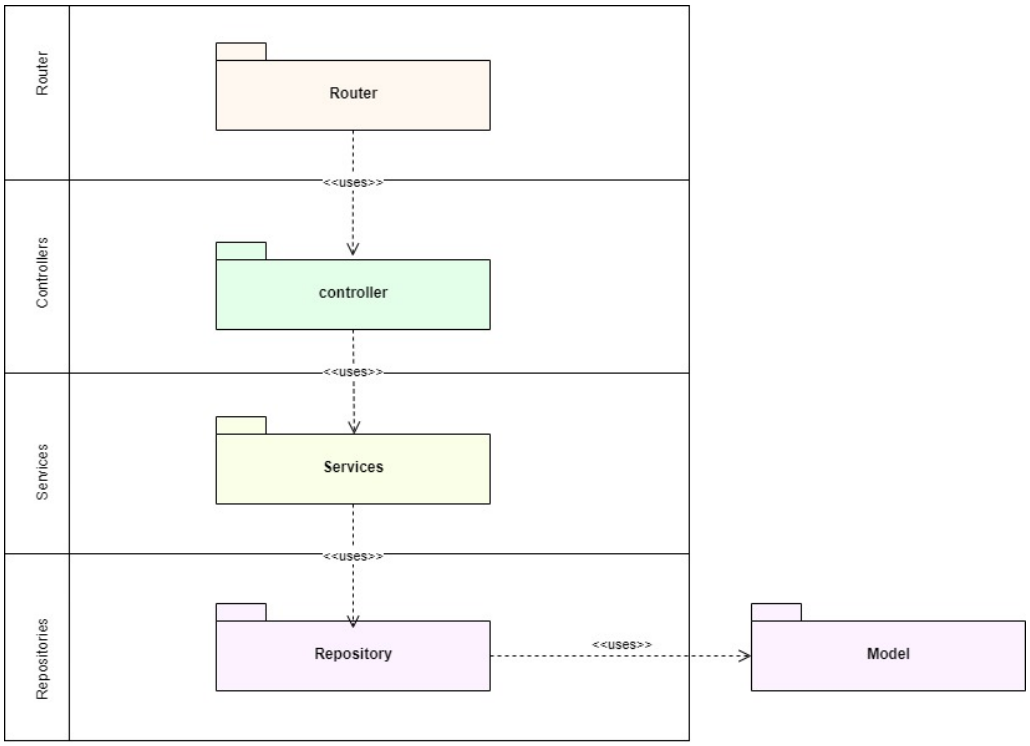
Representación primaria exporter



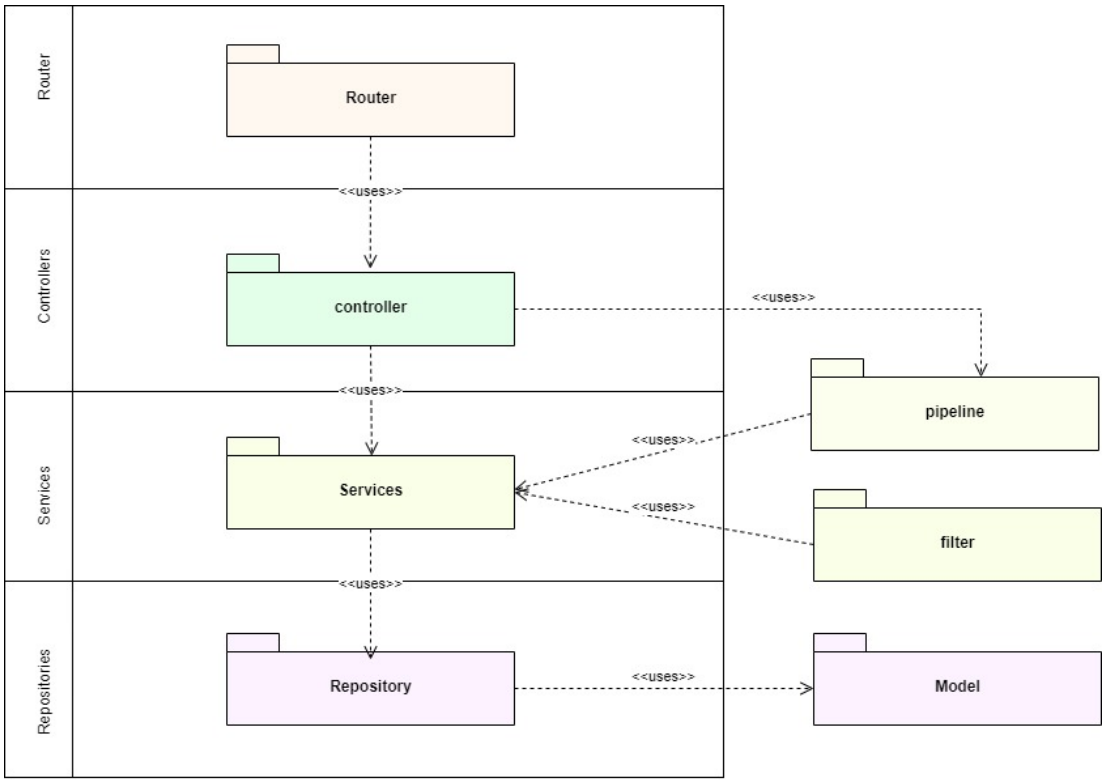
Representación primaria gateway



Representación primaria analytics y catalog



Representación primarias observations



3.1.7. Decisiones de diseño

Como se mencionó anteriormente, con esta vista se puede identificar de forma rápida como se manejan las responsabilidades dentro del sistema. Para lograr un diseño consistente, el equipo buscó siempre priorizar la separación de responsabilidades, ya que esto provoca que el sistema encapsule los módulos con las mismas responsabilidades en un único lugar.
Router no es una capa aparte. pero se diagrama aparte de controller para poder mostrar mejor la idea.

Al cumplir con esto, se cumple también con el **principio de responsabilidad única**, teniendo como consecuencia un sistema más fácil de mantener y además favorece la **alta cohesión y el bajo acoplamiento**.

3.2. Vistas de componentes y conectores

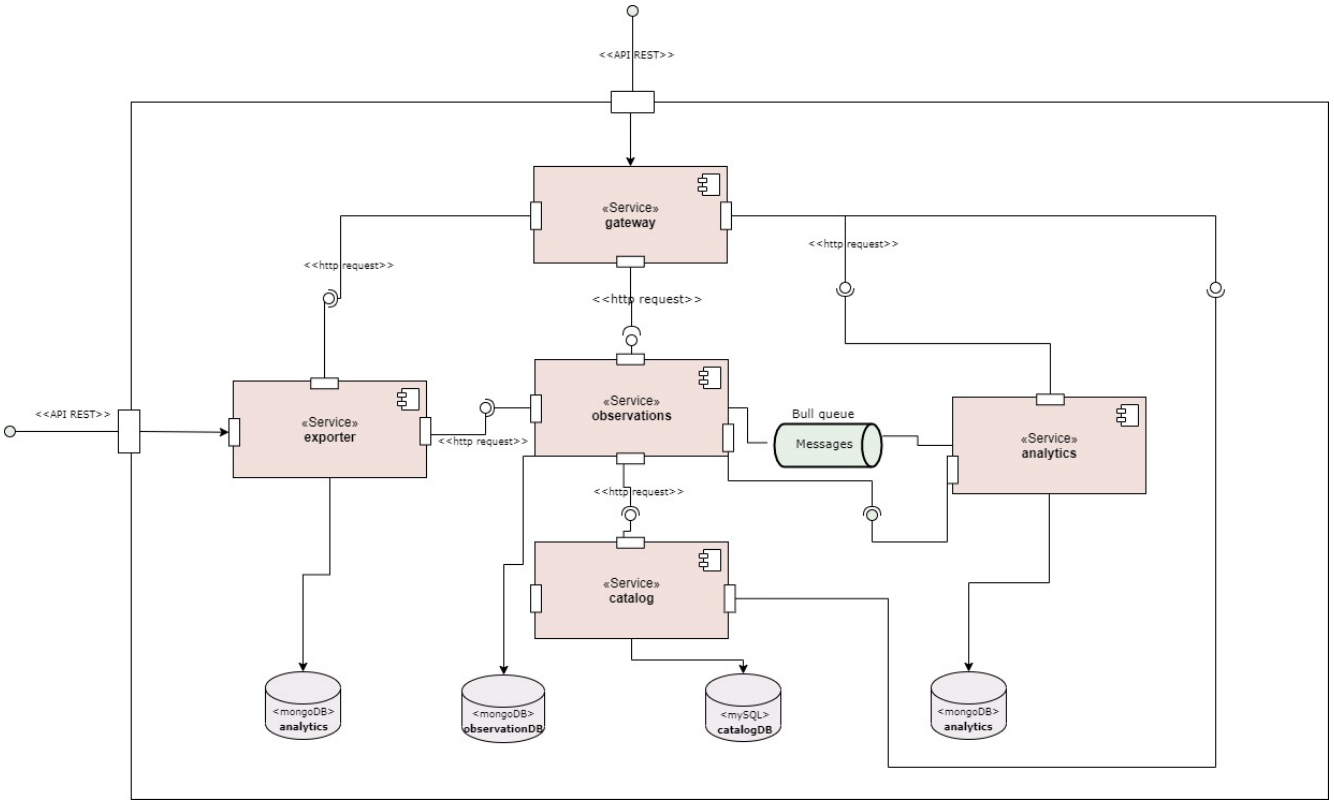
3.2.1. Vistas de componentes y conectores General

Aquí se puede observar la comunicación entre componentes y los procesos generales del sistema, con el objetivo principal de poder observar como funciona el sistema en tiempo de ejecución.

Como fruto se puede ver si son beneficiados los atributos de calidad de disponibilidad, seguridad y performance.

Se excluye authentication ya que se hace un diagrama especifico para este para no hacer mas complejo el siguiente diagrama.

Representación primaria



Catalogo de componentes/conectores

Tabla 3.13: Catalogo de componentes/conectores

Componente/conector	Responsabilidades
Analytics	Aquí se generan los reportes de las distintas observaciones de los sensores en función del tiempo y de ciertos parámetros, también es responsable de alertar a las personas correspondientes ante valores anormales en las observaciones según lo definido como rango normal.

Continúa en la siguiente página

Componente/conecto	Responsabilidades
Catalog	Contiene información relacionada a los sensores autorizados por la plataforma y el conjunto de propiedades observables para el análisis. No se desglosa cada módulo ya que es idéntico a Analytics.
Observations	Realiza las tareas de procesamiento de los datos que ingresan desde el Gateway. Captura la información disponible de cada lectura, valida que los datos sean correctos y transforma la data a la esperada y la persiste.
Exporter	Este modulo es el encargado de registrar los consumidores de la data recabada por los sensores, y de exportar esta.
Gateway	Es el punto de entrada del sistema y está expuesto a Internet. Aquí es donde los sensores, con sus respectivos formatos y protocolos, interactúan con el sistema, para extraer las mediciones, realizar la conversión de unidades, y calcular el tiempo que se demoró en realizar lo anterior. Además, se realiza un conjunto de validaciones mínimas que garanticen que la información que se consuma provenga de dispositivos autorizados.
analyticsDb	Es una base de datos, no relacional donde se persisten los datos de analytics.
observationsDb	Es una base de datos, no relacional donde se persisten los datos de observations.
catalgoDb	Es una base de datos, relacional donde se persisten los datos de catalog
exporterDb	Es una base de datos, no relacional donde se persisten los datos de exporter.
Bull queue	Cola de mensajes implementadas sobre redis y bull Se utiliza para el envío de observations hacia analytics. Nos brinda disponibilidad, para utilizarlo no importa el estado del otro componente

Continúa en la siguiente página

Componente/conecto	Responsabilidades
Http	Para facilitar la comunicación mientras que se cumple con un alto nivel de performance y seguridad.
TCP	Para la conexión con las bases de datos, es el que viene por defecto

Decisiones de diseño

Mecanismos de comunicación

Para la comunicación entre los servidores, se uso el protocolo **HTTP**, para lo que usamos la biblioteca **Axios**. Para acceder a distintas funcionalidades, solo se necesita la URI indicada junto con los parámetros requeridos, lo que favorece la interoperabilidad del sistema.

Si bien los sensores para esta etapa están siendo emulados, en un futuro éstos no serán emulados, sino que existirán de forma física, por lo que la interoperabilidad del sistema debe ser una prioridad, y las request HTTP ayudan en gran medida con esto. Como se hace uso de una API para enviar los datos, por lo que el sistema debe exponer endpoints que sean consumidos por las entidades que lo necesiten.

APIs

Para las APIs usamos la biblioteca **Koa router** para facilitar la creación de éstas. Además, usarlo tiene muchas ventajas, ya que es más fácil de implementar, es más flexible y de fácil integración. Está muy bien documentada, lo que es primordial ya que no teníamos experiencia con esta herramienta y lenguaje.

Bases de datos

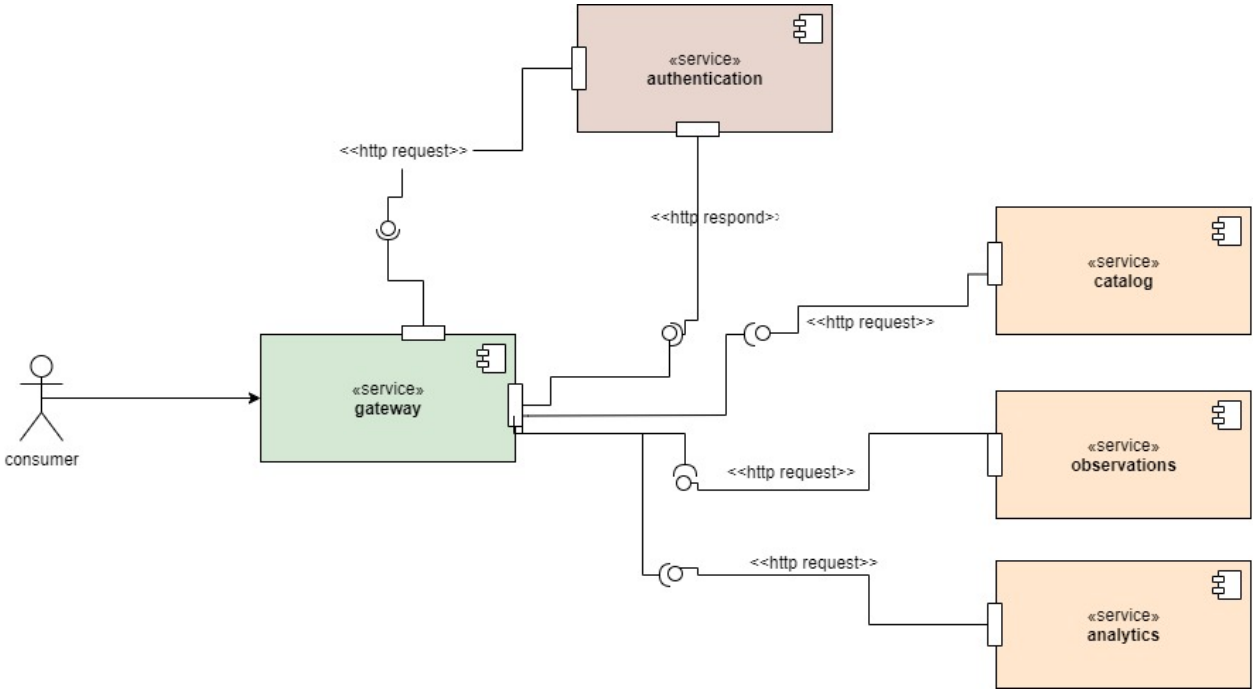
MongoDB: en los módulos que se uso esta base de datos se favoreció la performance, ya que se necesitaban hacer lecturas de forma rápida y sobre grandes cantidades. Se priorizo la performance sobre la seguridad, ya que en este caso, si bien se maneja datos de usuarios, no es información sensible. En el caso que si lo es, como la contraseña para hacer el login, se hizo un encriptado antes para proteger el dato, pero en los demás casos no es información que sea altamente sensible, ya que los usuarios no necesitan de estos para usar el sistema. Teniendo esto en cuenta, y además sabiendo que MongoDB es una base de datos muy utilizada en la actualidad que tiene ciertos niveles de seguridad.

MySQL: se decidió usar MySQL en el modulo catalog simplemente porque se necesitaban mantener relaciones entre los sensores, propiedades observables y rangos y con una base de datos relacional esto sucede naturalmente, lo que nos facilito el

manejo de estas. Además, no es en estos donde va a estar la mayor carga de datos, y se decidió priorizar lo anterior frente a la performance de las consultas.

3.2.2. Vistas de componentes y conectores federated identity

Representación primaria



Catalogo de elemento

Ya se describieron la mayoría, por lo que solo describimos los que no se mencionan antes.

Tabla 3.14: Catalogo de componentes/conectores

Componente/conector	Responsabilidades
Authentication	Aquí se hace la verificación de que el usuario este debidamente logueado para realizar las operaciones que así lo requieren, se reciben request del gateway para que se corrobore el acceso y se responde a este con ok o error según si el consumidor esta logueado o en caso contrario no respectivamente.

Decisiones de diseño

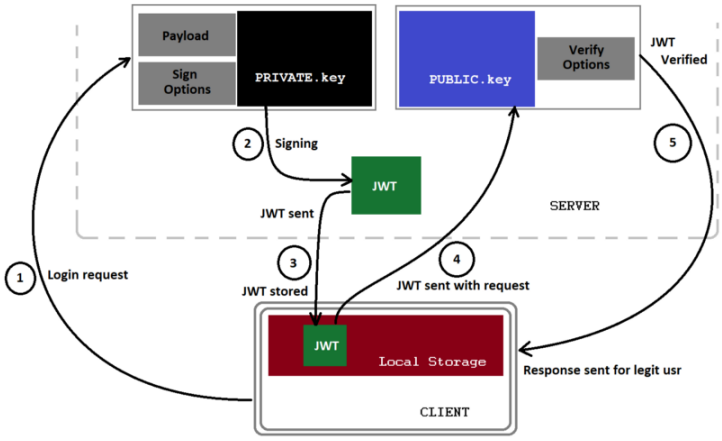
Con la intención de cumplir con el **REQ 11 – Usuarios registrados**, se implemento el modulo authentication, con la intención de seguir con la estructura sugerida por el **patrón identidad federada**.

Dentro de authentication se utilizo, con el fin de favorecer la seguridad del sistema, JWT (JSON Web Token) para la autenticación y la autorización.

En concreto se utilizo JWT para verificar que las acciones que solo puede hacer un usuario logueado así sea. Una vez que el usuario esta logeado, si los datos están correctos, se devuelve un token de acceso. Dentro del mismo se encuentran todos los datos del usuario. Este token luego puede ser utilizado para acceder a las secciones protegidas del sistema, ya que dichas funcionalidades validan nuevamente el token con authentication y conceden o deniegan acceso según corresponda.

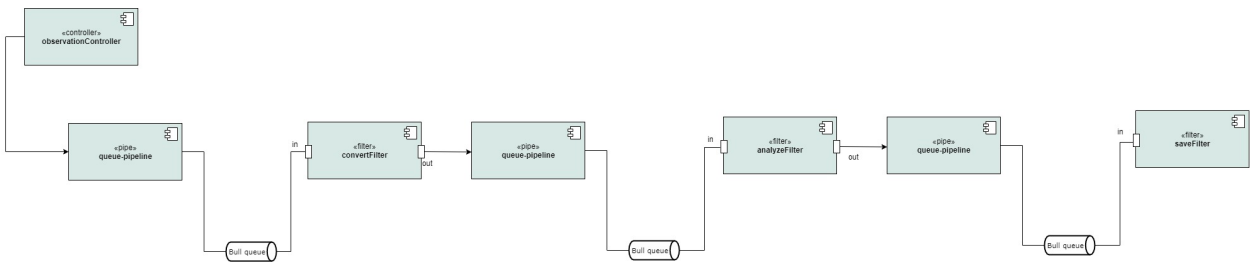
Se implementa como un federated identity para centralizar la lógica de autenticación, y evitar así errores críticos en esta sección de la lógica de negocios si se debe replicar en cada subsistema.

El funcionamiento seria como en el siguiente diagrama que extrajimos de internet para ayudar a la explicación.



3.2.3. Vistas de componentes y conectores pipes and fillters observations

Representación primaria



Catalogo de elemento

Ya se describieron anteriormente en el modulo de observation, no se cree necesario repetir.

Decisiones de diseño

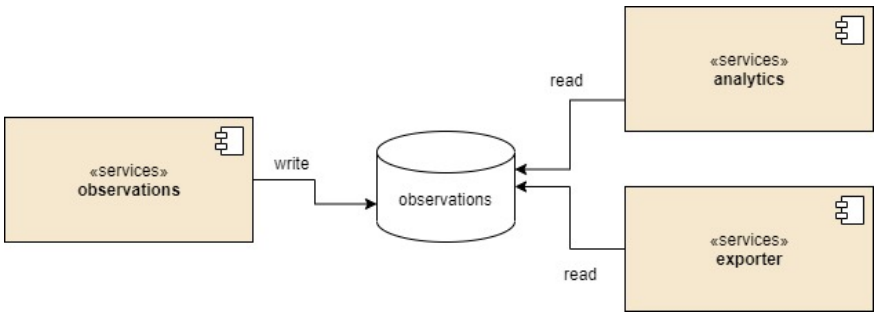
Como se fue mencionando un poco antes en la descripción de observation para el requerimiento **REQ 4 – Extracción de observaciones de una lectura** e intentando cumplir con el requerimiento **REQ 12 – Procesamiento de una lectura**, se implemento el **patrón pipes and filters** sobre los pasos que se hacen al momento de procesar una observación. Estos son transformar la data que llega a la estándar según la propiedad observable a la que corresponde, analizar la data frente a rangos que están predefinidos, si existen, y por ultimo si lo anterior ocurrió correctamente guardar la data.

Esto se hizo de esta forma ya que se deseaba que fuera extensible, modificable a nuevos pasos al momento de procesar una lectura, además de ser claramente pasos bien definidos que eran fácil transportar a la lógica de los **filtros**.

Se decidió además dejar una implementación del **pipeline** que actualmente es con una **bull** queue. De todas maneras, esta puede puede ser cambiada por el defer binding que se usa para decidir el tipo de pipeline. Si se decide que es apropiado hacerlo cuando se extiendan los filtros, y quizás se crea necesaria una nueva implementación.

3.2.4. Vistas de componentes y conectores CQRS

Representación primaria



Catalogo de elemento

Ya se describieron anteriormente no se cree necesario repetir.

Decisiones de diseño

Se decidió implementar una estructura como la propuesta por el **patrón CQRS** sobre las observations, siendo el modulo observation quien escribe directamente sobre

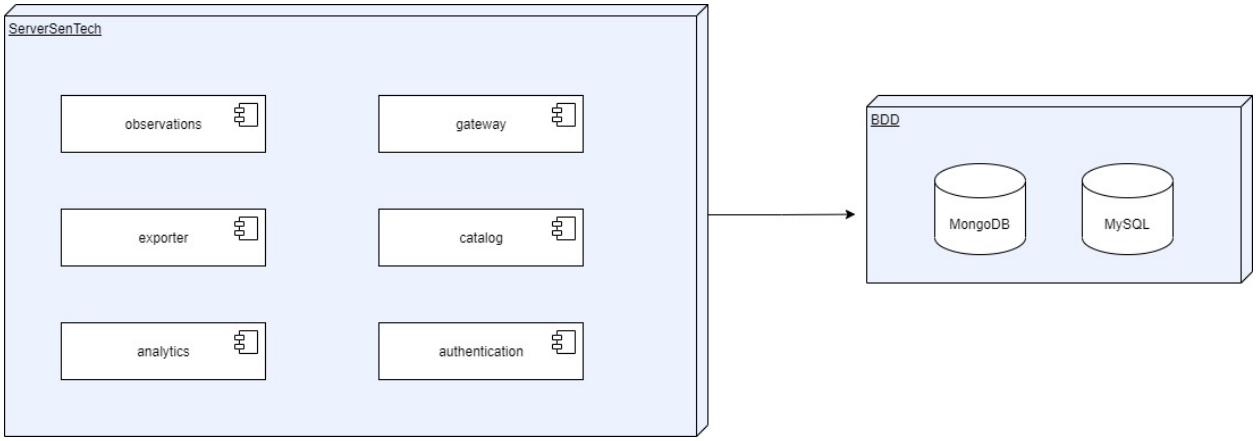
esta base de datos y a quien se le hace esta petición, y los módulos analytics y exporter a quien se les piden las tareas de lectura. De esta manera se puede mejorar el rendimiento, la escalabilidad y la seguridad del sistema.
Lo que puede ayudar también a cumplir con el **REQ 14 – Tiempo de respuesta de consultas de analytics**

3.3. Vistas de Asignación

3.3.1. Vista de despliegue

Vista de despliegue muestra como desplegamos nosotros el sistema, corriendo todo el sistema en una misma PC, por lo tanto va a haber un único server.

Representación primaria



Catalogo de elementos

Tabla 3.15: Catalogo de componentes/conectores

Componente/conector	Responsabilidades
observations	Es el server que contiene la API de observations para el manejo de solicitudes sobre estas
gateway	Es el server que contiene la API de entrada a todo el sistema.
exporter	Es el server que contiene la API de exporter para el manejo de solicitudes sobre consumidores, y exportación de datos.

Continúa en la siguiente página

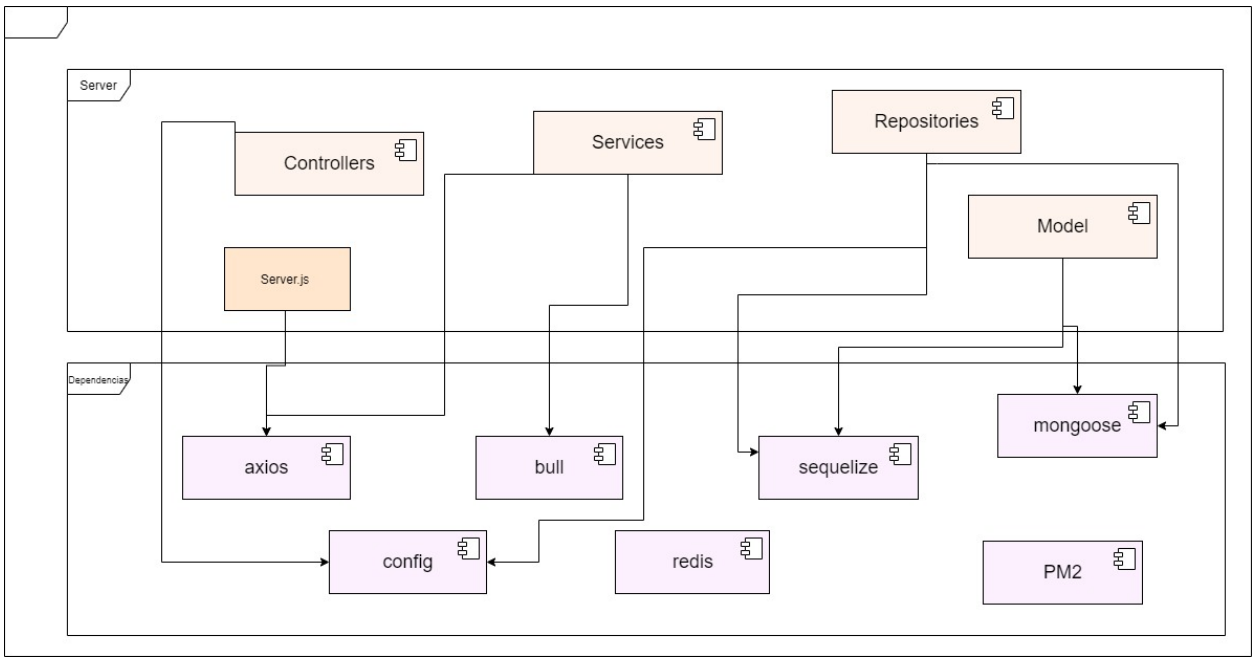
Componente/conecto	Responsabilidades
catalog	Es el server que contiene la API de catalog para el manejo de solicitudes sobre sensores, propiedades observables y rangos.
analytics	Es el server que contiene la API de analytics para el manejo de solicitudes en función del tiempo de observations y el manejo de personas a notificar.
authentication	Servidor encargado de proveer la autenticación de usuarios a los demás servidores.

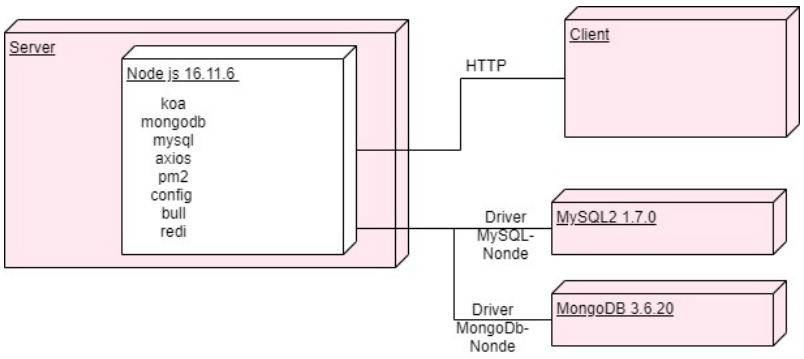
Decisiones de diseño

Al diseñar la arquitectura decidimos que era algo importante poder hacer deploy de los diferentes módulos del sistema en diferentes máquinas, en caso de que sea necesario que cada módulo esté en una máquina por alguna razón de disponibilidad o escalar. Como se puede notar en el diagrama, esto es posible debido a los tipos de conexiones que utilizamos entre los diferentes módulos de nuestro sistema.

3.3.2. Vista de instalación

Representación primaria





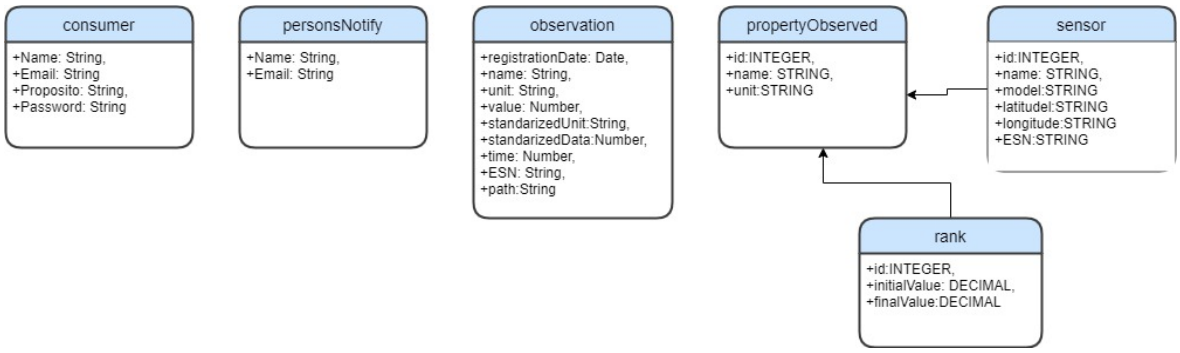
Catalogo de elementos

Tabla 3.16: Catalogo de componentes/conectores

Componente/conector	Responsabilidades
axios	Permite realizar llamadas http desde una aplicación.
config	Librería para simplificar la lectura de archivos de configuración de cualquier lugar del sistema.
bull	Cola de mensajes utilizada para la comunicación entre los sistemas.
redis	almacén de base de datos en memoria con un modelo de datos clave valor
sequelize	Librería para acceso a datos en mySQL con nodejs. Utilizada para acceder al repositorio.
mongoose	Librería para acceso a datos en mongodb con nodejs. Utilizada para acceder al repositorio.
pm2	Librería que permite ejecutar varios clusters de una aplicación simultáneamente y manejar la carga de las requests

3.4. Modelo de Datos

3.4.1. Representación primaria



3.4.2. Catalogo de elementos

Tabla 3.17: Catalogo de componentes/conectores

Componente/conector	Responsabilidades
consumer	En esta tabla se guardan los consumidores que van a utilizar .
personNotify	En esta tabla se guardan las personas a notificar.
observation	En esta tabla se guardan las observaciones de los sensores, tanto lad ata que llega como la transofrmada a estandar.
propertyObserved	En esta tabla se guardan las propiedades que los sensores pueden observar, tienen dentro un id de sensor y un id de rango.
sensor	En esta tabla se guardan los sensores que observan las propiedades observables por lo que cada sensor tiene dentro propiedades observables.
rank	En esta tabla se guardan los rangos de las propiedades observables, por lo que cada rango tiene propiedades observables.

3.4.3. Decisiones de diseño

Consumer:
Este esquema tiene atributos que son requeridos, es decir que no pueden ser null porque se necesitan para el login, para la autorización y autenticación para acceder a las funcionalidades, como lo son Email y Password. Luego tenemos el nombre y el propósito que se hicieron require para cumplir con parte del **REQ 7 – Registro de consumidor**. También tiene una URL que es única y se le asigna cuando se crea el consumidor para que pueda acceder a los datos desde exporter, por lo que esta ruta tiene la forma 'http://localhost:6069/exporter/data/consumer/Email'.

También tenemos dos fechas, una que indica la fecha de registro del consumidor, y otra que se usa para cumplir con parte de lo mencionado en el **REQ 8 – Exportación de datos** y poner una marca desde donde debe observar el consumer ante una subsecuente llamada.

PersonNotify:
Este esquema tiene dos atributos, y ambos son requeridos ya que se necesitan para poder cumplir con la parte de notificación del **REQ 5 – Alertar ante valores anormales**. También se valida que el email sea único y tenga un formato correcto.

observation:
Este esquema tiene varios atributos, ya que en esta se guarda la data que llega al sistema, como la transformada en caso de ser posible. Los atributos que llegan son name, unit y value y son require, ya que sin alguno de ellos no se pueden cumplir con el **REQ 4 – Extracción de observaciones de una lectura**.

También, para cumplir con este requerimiento, se agrego un atributo time. Este se completa luego de terminado el proceso y por ultimo un ESN correspondiente al sensor al que corresponde la observación.

propertyObserved:
Este esquema tiene todos los atributos require, ya que son pedidos por letra para cumplir con el requerimiento **REQ 2 - Mantenimiento de Propiedades Observables**. También cumpliendo con este requerimiento se agrega un atributo path, que indica como llego la propiedad observable.

sensor:
Este esquema tiene todos los atributos require, ya que son pedidos por letra para cumplir con el requerimiento **REQ 1 - Mantenimiento de Sensores**. También cumpliendo con este requerimiento se valida que el ESN sea alfanumérico y único. Además tiene asociado una lista de propiedades observables.

rank:
Este esquema tiene todos los atributos require, ya que son necesario para cumplir con el requerimiento **REQ 5 – Alertar ante valores anormales**. También tienen asociado una lista de propiedades observables.

4. Consideraciones Finales

4.1. Decisiones de diseño no mencionadas

En esta sección se mencionaran decisiones de diseño que no han sido mencionado en otras secciones.

Los rangos pueden tener definido dos veces la misma propiedad observable dentro de su lista pero se tomara solo el ultimo como el mas actualizado.

Los logs se realizan solo en los módulos; gateway quien toca a todos los módulos y puede encargarse de llevar un control mas general de errores, authentication para dejar registrada la auditoría de acceso, y observation para controlar el comportamiento del pipes and filters, en los otros dos módulos no se creyó necesario ya que todos los errores caerán finalmente en gateway.

Se uso un archivo env para declarar las KEY que se usan en el JWT, para administrar estas variables de entorno por separado del código base.

4.2. Consideraciones de atributos de calidad y tácticas aplicadas

A lo largo de todo el proyecto, el equipo tuvo en mente los atributos de calidad para aplicar tácticas para favorecer estos atributos de calidad que el sistema cuenta con.

4.2.1. Performance

Introducir concurrencia

De manera de favorecer la asincronía entre observations y analytics, el equipo decidió implementar el uso de una cola para almacenar y consumir requests HTTP de manera independiente. Así, los recursos son asignados a demanda, ayudando a la performance general del proyecto.
Para implementar las colas, utilizamos la librería Bull, la cual está basada en redis. La misma permite implementar un sistema de colas robusto y rápido, perfecto para ser incluido como parte de la solución del proyecto.

También para favorecer el tiempo de respuesta de analytics y cumplir con el **REQ 14 – Tiempo de respuesta de consultas** se dio prioridad a la consulta del requerimiento **REQ 6 – Consulta en función del tiempo de los valores registrados de una propiedad observable de un sensor**, se hizo asincronica la función de envió de mails para ante una subsecuente llamada de consulta se le de prioridad a esta, para aumentar la performance de respuesta.

También para manejar la carga del sistema se utiliza la herramienta PM2

4.2.2. Interoperabilidad

Se eligió como mecanismo de comunicación entre los componentes HTTP mediante AIs REST.
De esta manera para acceder a distintas funcionalidades solo se necesita la URI indicada junto con los parámetros requeridos, lo que favorece la interoperabilidad del sistema.

4.2.3. Disponibilidad

Manejo de excepciones

El sistema cuenta con el modulo logger para hacer reporte de errores, con objetivo de poder identificar qué está sucediendo en el sistema, que operaciones se llevan a cabo para poder detectar en caso de errores en que modulo o elemento el sistema esta fallando, por esto los logs se les indica la URI de donde se produjo el error.

Recuperación ante fallos

Se hace control de errores a lo largo de todo el sistema, lanzándolos en las capas de abajo como los son el acceso a datos y controlándolos e informando al usuario en las capas superiores como son los controllers.

4.2.4. Testeabilidad

El hecho de tener logs de error favorece y aumenta la testeabilidad del sistema, es mas fácil rastrear los errores.

4.2.5. Redundancia activa

Se utiliza la herramienta PM2, manejando varias instancias realizando cómputos

4.2.6. Seguridad

Autenticar y autorizar actores y identificar actores

Existen ciertas funcionalidades del sistema, que solo deben ser utilizadas por usuarios debidamente autenticados. Esto el equipo lo solucionó haciendo uso de

JWT , para que en caso de realizarse una request con un token inválido, el usuario no pueda hacer uso de esa funcionalidad.

Cambiar la configuración predeterminada

Con el uso de los archivos config para definir los puertos con los que corren los distintos server y los repositorios donde se persisten los datos.

Limitar acceso

Único acceso es mediante la API gateway y exporter, se controla quien accede a que punto del sistema.

4.2.7. Modificabilidad

Encapsular

Con el fin de reducir el acoplamiento, el equipo se centró siempre de encapsular las clases que tienen responsabilidades relacionadas en un mismo módulo. Datos de configuración centralizados y encapsulados en archivos config y env.

Defer binding

El uso de esta táctica se fue explicando a lo largo de la documentación, pero como mencionamos se uso en pipes and fillters y en server de gateway.

5. Manual de instalación

5.1. Pasos

Levantar instancias de mongo, mysql y redis.

Revisar los archivos config, los connections string dentro de estos.

Agregar en la raíz del server de authentication un archivo .env con las variables NODE_ENV y JWT_SECRET

Iniciar sesión en el mail de envíos para poder dar los permisos requeridos.

Levantar con lo que se prefiera pm2 o node directamente los 6 servidores.

5.2. Pasos emulador

Para ejecutar el emulador de datos primero hay que realizar unas requests al sistema que registrarán sensores, propiedades observables, rangos y otros elementos que forman parte de los datos y es necesario que estén creados en la base de datos previo a ejecutar el emulador.Todas las siguientes requests serán métodos POST.

1. URL: http://localhost:8080/gateway/exporter/consumer
   Body: {
   “Name” : “Consumer1”,
   “Email”: “cosnumer@gmail.com”,
   “Proposito”:“Obtener data”,
   “Password”:“Consumer1234”
   }
   Esta request registra un consumidor en el sistema.

2. URL: http://localhost:8080/gateway/login
   Body: {
   “Email”: “cosnumer@gmail.com”,
   “Password”:“Consumer1234”
   }
   Esta request realiza el login al sistema.

NOTA: Hay que copiar el valor del token que devuelve postman y pegarlo en un header llamado “Authentication” para poder realizar las siguientes requests.

3. Aqui registramos dos sensores con propiedades observables asociadas:
   URL: http://localhost:8080/gateway/catalog/sensor
   Body: {
   “model”: “A1re”,
   “latitude”: “-34.2625”,
   “longitude”: “-57.4657”,
   “name”: “Sensor de tiempo”,
   “ESN”: “AAhgS15”,
   “propertiesObserved”:[{
   “name”: “tiempo”,
   “unit”: “s”
   }]
   }

   {
   “model”: “B781zd”,
   “latitude”: “-84.2625”,
   “longitude”: “-97.4657”,
   “name”: “sensor variado”,
   “ESN”: “B781zd”,
   “propertiesObserved”:[{
   “name”:“tiempo”,
   “unit”:“s”
   },
   {
   “name”:“temperatura”,
   “unit”:“C”
   },
   {
   “name”:“distancia”,
   “unit”:“Km”
   }]
   }

4. URL: http://localhost:8080/gateway/analytics/person
   Body: {
   “Name”: “person1”,
   “Email”:“person1@gmail.com”
   }

5. URL: http://localhost:8080/gateway/analytics/person
   Body: {

“Name”: “person2”,
“Email”:“person2@gmail.com”
}
Aqui registramos dos usuarios a notificar en caso de valores anormales.

6. Los siguientes valores son los rangos de valores comunes para las propiedades observables registradas:
URL: http://localhost:8080/gateway/catalog/rank
Body: {
“initialValue”: 0,
“finalValue”: 5000,
“propertiesObserved”:[{
“name”:“tiempo”,
“unit”:“s”
}]
}

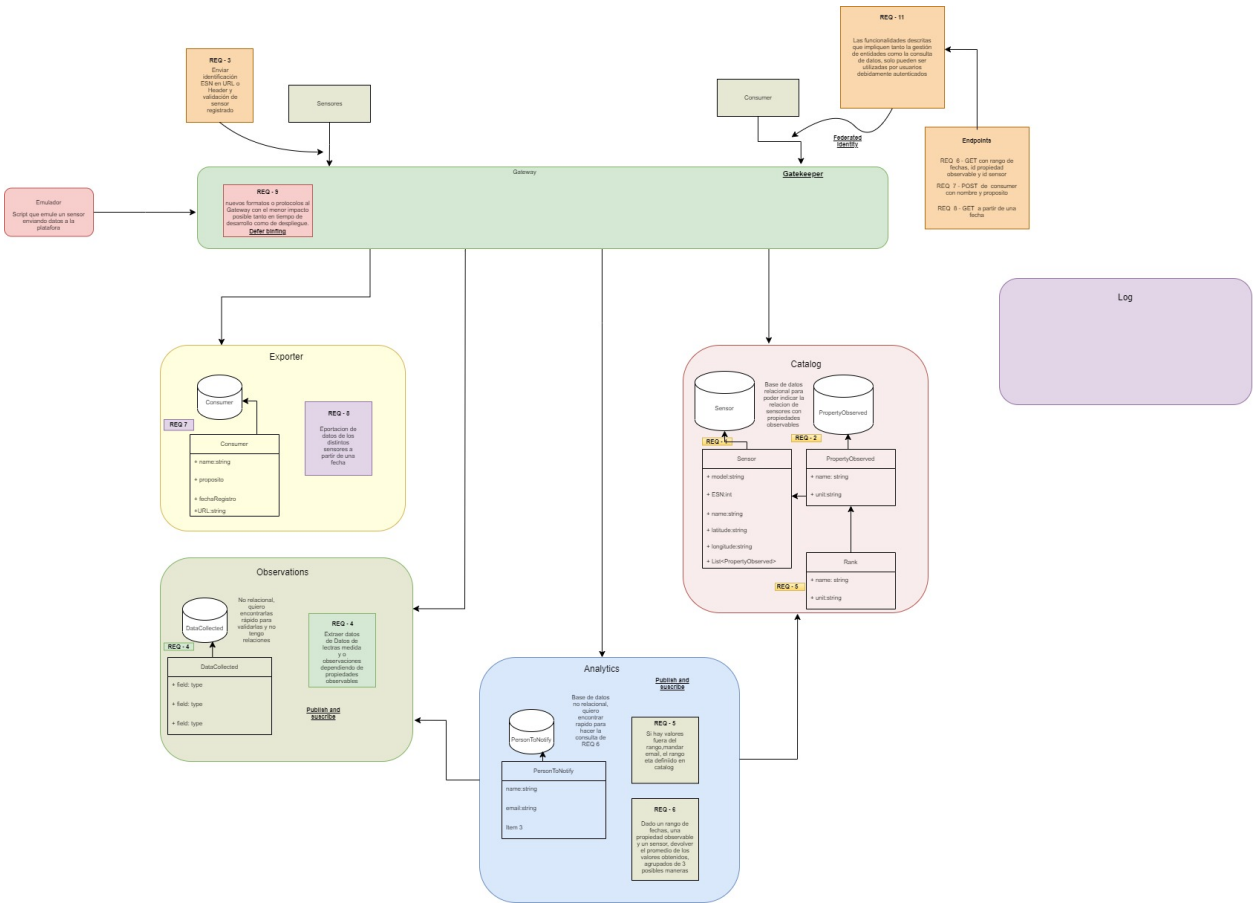
{
“initialValue”: 0,
“finalValue”: 22009,
“propertiesObserved”:[{
“name”:“distancia”,
“unit”:“Km”
}]
}

Luego de crear estos datos, se deben importar los datos del emulador (incluidos en el archivo json llamado observations que se encuentra en la carpeta entregada dentro de documentos emulador) a mongo. Para esto se abre mongo compass y se crea una nueva base de datos llamada “emulator”, con una collection llamada “sensors”. Luego se importan los datos ya mencionados.
Cuando ya se hayan realizado todos estos pasos, se ejecuta sensorEmulator desde nuestro programa y ahi los datos se comenzaran a procesar.

6. Anexo

6.1. Diagrama de primer acercamiento luego de leer la letra

Lo primero que hizo el equipo al momento de arrancar el obligatorio fue leer la letra completa e intentar bajar a tierra con el siguiente diagrama lo que se planeaba hacer, esto fue cambiando a lo largo del desarrollo del obligatorio pero se adjunta para mostrar evidencia del proceso de pienso.



Bibliografía

[1] R. K. Len Bass, Paul Clements, *Software Arquitecture in Practice*, 3rd ed., 2005.

[2] Desing patterns. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/patterns

[3] PPTS del curso.