Obligatorio I
Universidad ORT

Arquitectura de Software

Mateo Forestier
198157

Gonzalo Linares
191873

Docentes: Andrés Calviño - Mathias Fonseca

2017

Indice

Introducción

Para este obligatorio, el equipo Gonzalo Linares y Mateo Forestier diseño e implemento la arquitectura necesaria para el desarrollo del backend para la empresa **Ruski Oil International (ROI)**.

Este sistema, consta de varios sub sistemas y es capaz de manejar ordenes de abastecimiento, planes de suministros, así como también registrar datos de sensores y a partir de esto determinar el estado de los mismos.

Importante:

Todas las imágenes se encuentran adjuntas dentro de la carpeta “documentación” en el directorio /img, dentro de este se separaran por vistas.

Objetivos de la arquitectura

Requerimientos Particulares

REQ-1: Registrar Orden de Abastecimiento

Aplicación: **roi.Supplying**
Permitir crear, modificar, eliminar y consultar Órdenes de Abastecimiento.

REQ-2: Comunicar las Órdenes de Abastecimiento a roi-planner

Aplicación: **roi.Supplying**
Al crea, modifica o elimina una Orden de Abastecimiento **roi-supplying** comunicará inmediatamente a **roi-planner** indicando la operación realizada.

REQ-3: Registrar Plan de Suministro

Aplicación: **roi.Planner**
Al recibir una Orden de Abastecimiento desde roi-supplying, el sistema procede a registrar el Plan de Suministro correspondiente según la operación indicada:

- Si la operación informada con la Orden es “creación”, se debe crear un nuevo Plan de Suministro.
- Si la operación informada con la Orden es “modificación”, se deben reflejar los cambios que correspondan en el Plan de Suministro.
- Si la operación informada con la Orden es “eliminación”: se debe marcar el Plan de Suministro como “anulado”.

La creación del Plan de Suministro implica además conectarse a la API de **roi-pipeline-calc** para obtener la ruta del suministro (tramos de red).

REQ-4: Manejo de Secciones y Sensores

Aplicación: **roi.PipelineMonitor**
El sistema debe de ser capaz de crear, modificar, eliminar y consultar secciones y sensores.

REQ-5: Procesar señales de diversos sensores sobre una misma fuente

Aplicación: **roi.PipelineMonitor**
El sistema debe implementar un mecanismo que a partir de las medidas recibidas de los sensores de un mismo punto de suministro sirva para detectar sensores con fallas o medidas incorrectas. En estos casos, se espera poder conocer estas situaciones por algún medio.

REQ-6: Gestión de errores y fallas

Libreria: **roi.Logger**
El sistema debe proveer proveer un log con las tareas que se realizan. En particular, en el caso de ocurrir una falla o cualquier tipo de error, el sistema provea toda la información completa del error, dentro del log.

REQ-7: Autenticación de operaciones

Aplicación: **roi.Authenticator**
El sistema debe autenticar toda llamada realizada entre servidores.

Requerimientos No Funcionales

REQN-1: Performance del procesamiento de señales de sensores

Se desea que el sistema procese los datos de un sensor en un tiempo medio de 300ms o menos.

REQN-2: Portabilidad de logeo de errores

Se espera que el log sea proporcionado en forma de libreria u otra forma para poder ser rehusado.

REQN-3: Modificabilidad de logeo de errores

Se espera que el sistema a futuro permita usar diferentes tecnologías de log.Adicionalmente se espera que el log permita al usuario cambiar de tecnología, incluso en tiempo de ejecución.

REQN-4: Seguridad de llamadas internas entre servidores

Se espera que las llamadas entre servidores estén autenticadas, es decir que no puedan ser ser accedidas por un usuario desde un navegador.

REQN-5: Disponibilidad de llamadas internas entre servidores

Se espera que cada aplicación está sujeta a su propia disponibilidad, es decir que en caso de que una aplicación falle, las otras puedan seguir funcionando.

REQN-6: Disponibilidad recuperación de “fault”

Se espera que cada aplicación registre las “fault” y notificando al cliente del problema. El despues de una de estos el sistema debe ser capaz de recuperarse y seguir su funcionamiento.

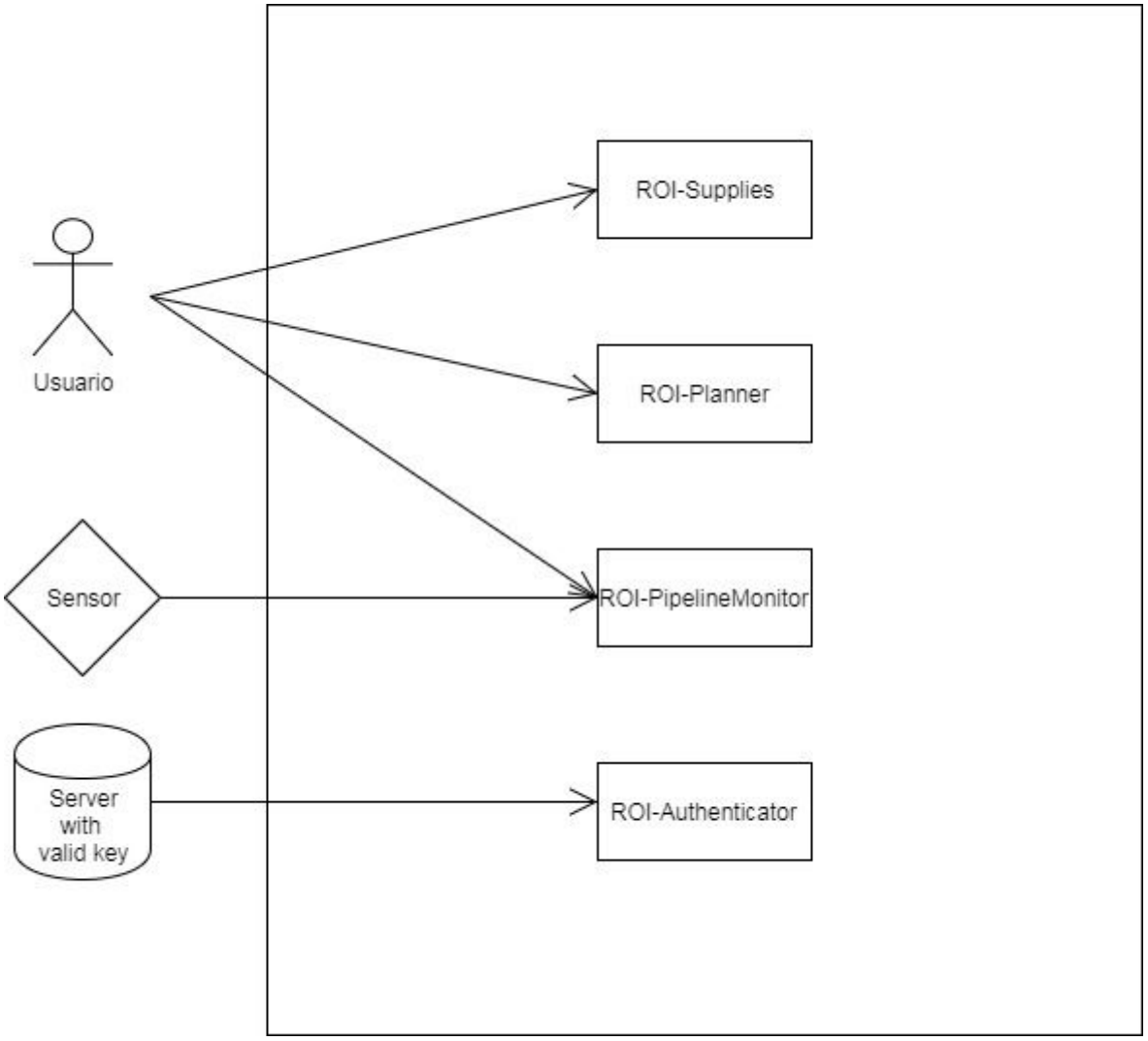
Restricciones

- La implementación del backend debe desarrollarse en JEE utilizando las tecnologías vistas en el curso (web services, EJB, JMS, JPA). De ser necesario, podrán utilizarse otras tecnologías Java (logging, mail, formatos de datos -xml, json-, etc.)

- Las API que exponen los backend de **roi-supplying** y **roi-planner** a sus respectivos front-end **debe ser REST**. Para las comunicaciones entre aplicaciones de backend tiene libertad de seleccionar el mecanismo.

- Todo el código fuente, documentación, archivos de configuración o cualquier otro artefacto referido al desarrollo de los prototipos debe gestionarse en el repositorio Git asignado.

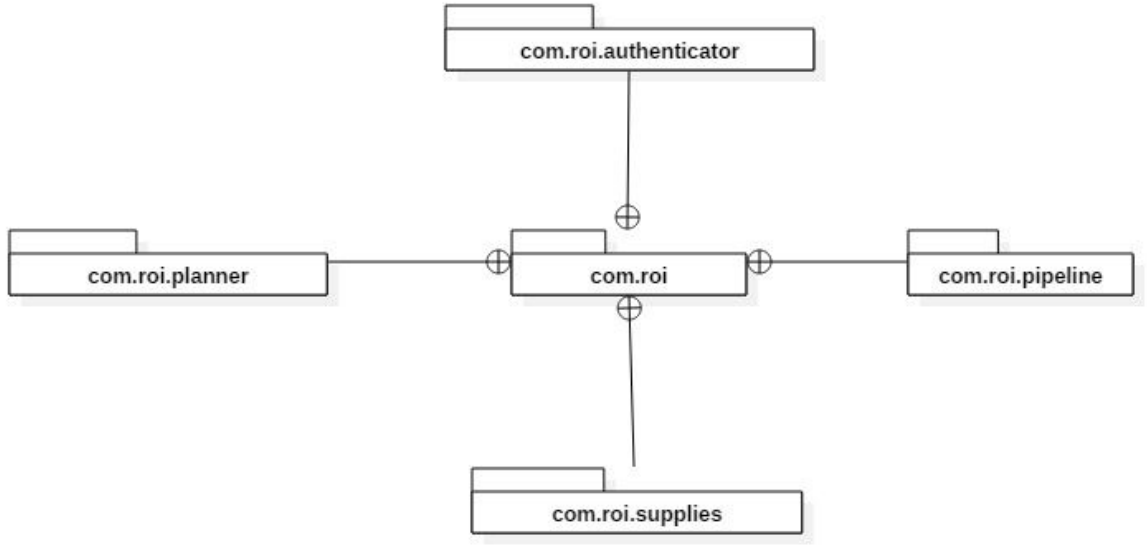
Vista de actores



Vista de módulos

Vista de descomposición

Contexto



Justificación general del diseño

Layered Pattern

Originalmente nuestro plan fue usar un Layered Pattern dentro de cada aplicación, separando a estas en capas. Como se puede ver en los diagramas de la representación primaria, las aplicaciones constan principalmente de las siguientes capas businesslogic, dataaccess y domain. Igualmente, después de una discusión con el profesor del curso nos dimos cuenta que este patrón de diseño estaba mal aplicado, pero ya era demasiado tarde para re plantear el diseño, por lo que decidimos mantener este.

Second Level Cache de JEE

Para mejorar la Performance de las operaciones de ROI-Pipeline se decidió hacer uso de un cache para reducir el Overhead generado por las operaciones de la base de datos.
(REQN-1)

Manejo de Excepciones

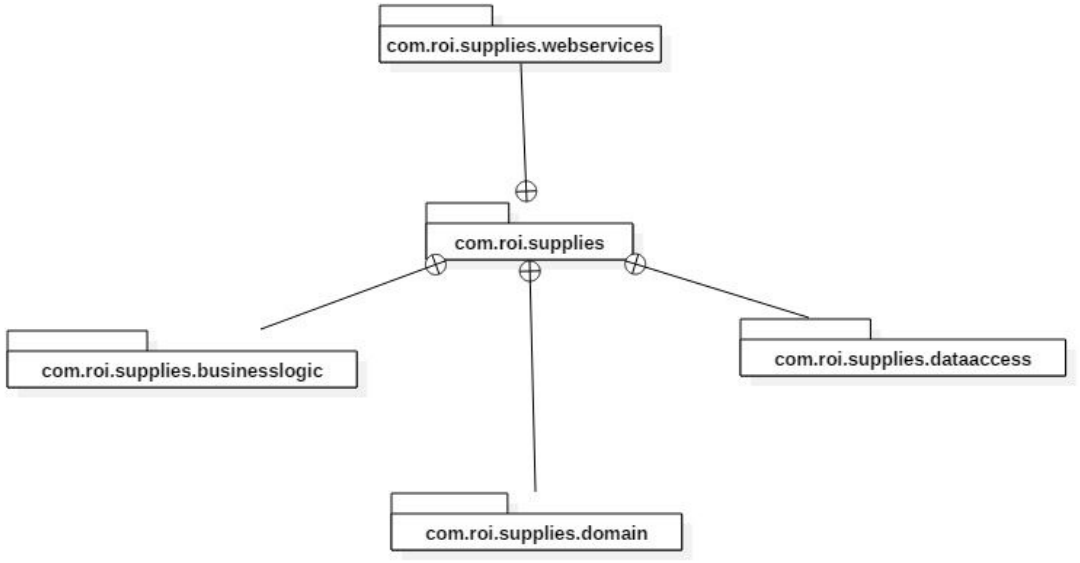
Las excepciones que del proyectos si bien en su mayoría son genericas, todos los mensajes de estas permiten saber exactamente la razón por la cual falla el código. Adicionalmente gracias al uso del log se puede obtener mayor información de las llamadas y el error.
(REQN-6)

Recuperación a estado seguro

Al tener una fall mientras se registran datos en la base de datos, se realizará un rollback para asegurar la integridad de los datos en la base. Estos son hechos de manera automática gracias a la tecnología de JPA.
(REQN-6)

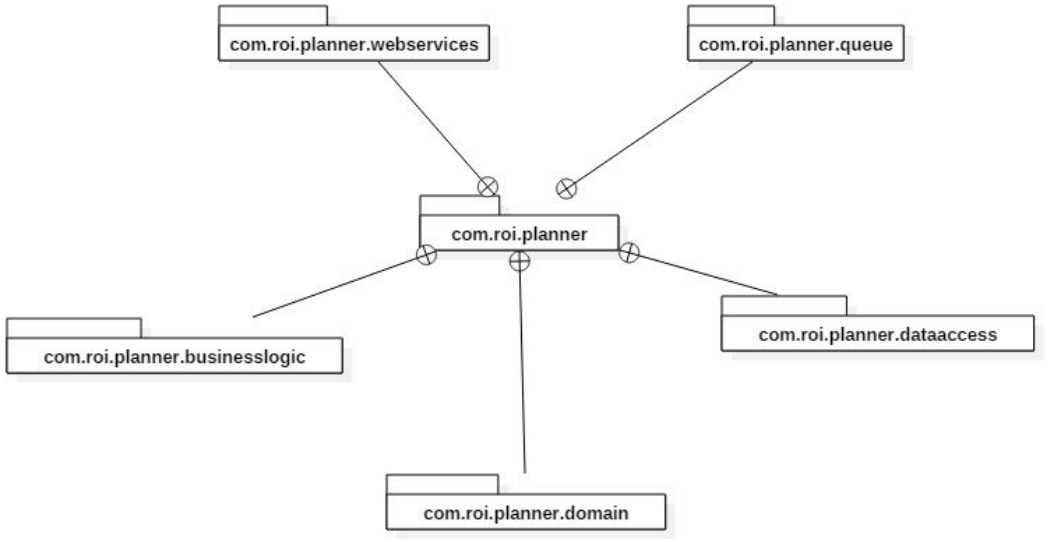
ROI-Supply

Representación primaria



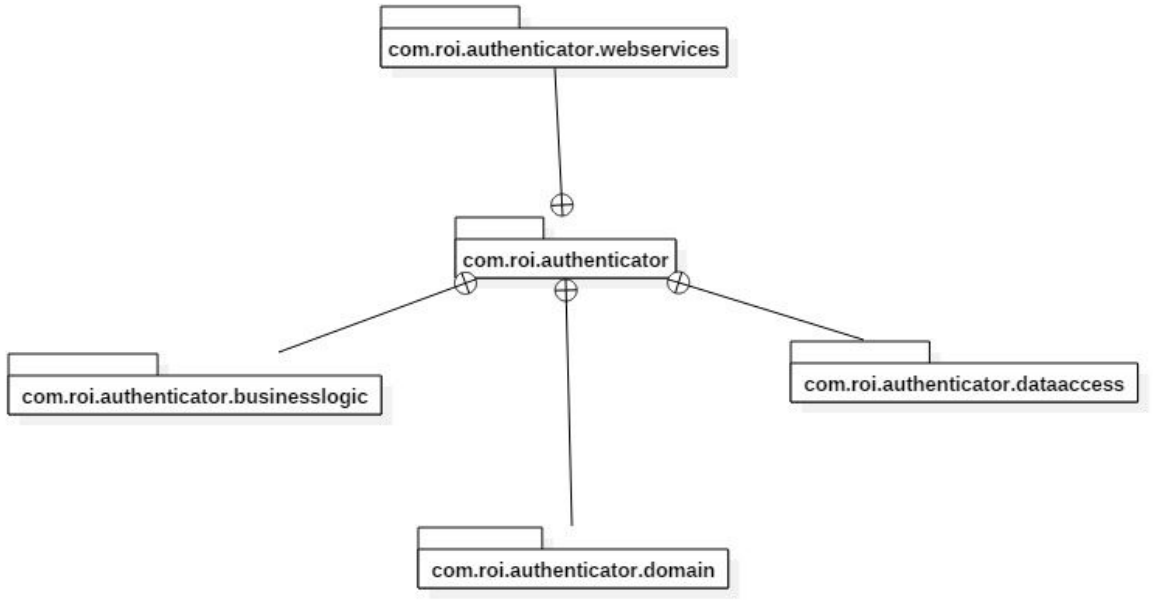
ROI-Planner

Representación primaria



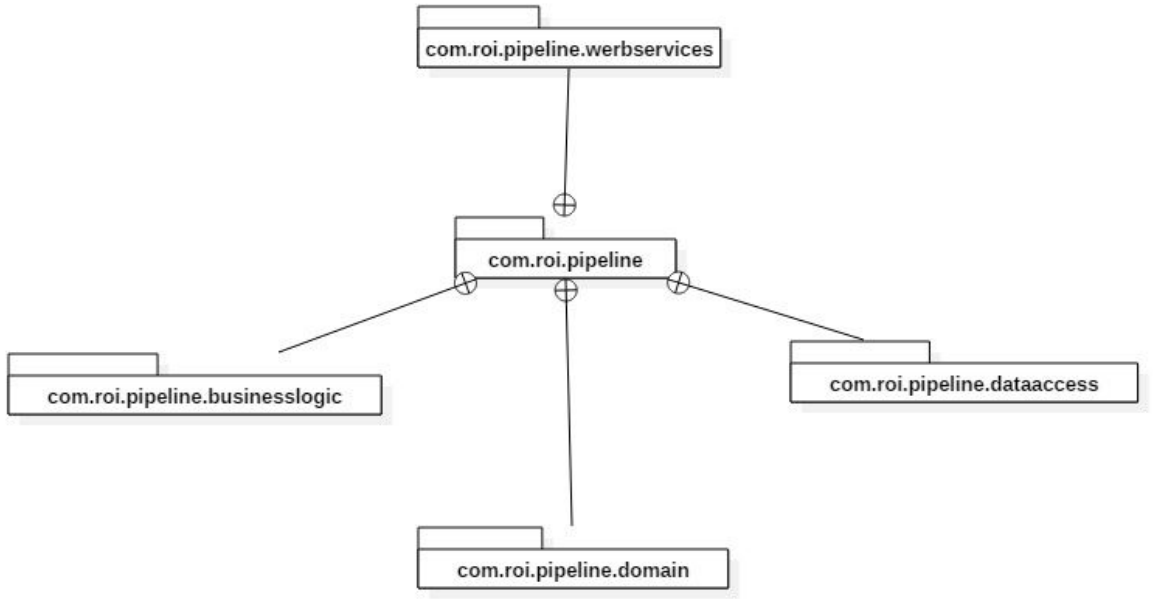
ROI-Authenticator

Representación primaria



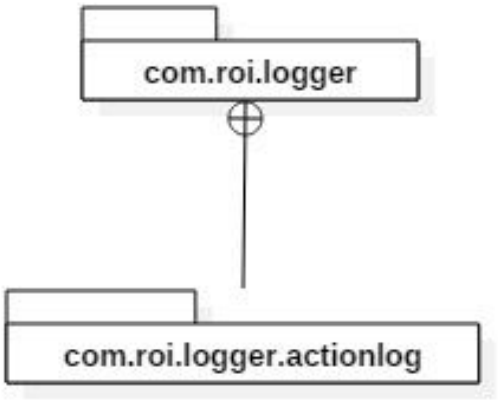
ROI-PipalineMonitor

Representación primaria



ROI-Logger

Representación primaria



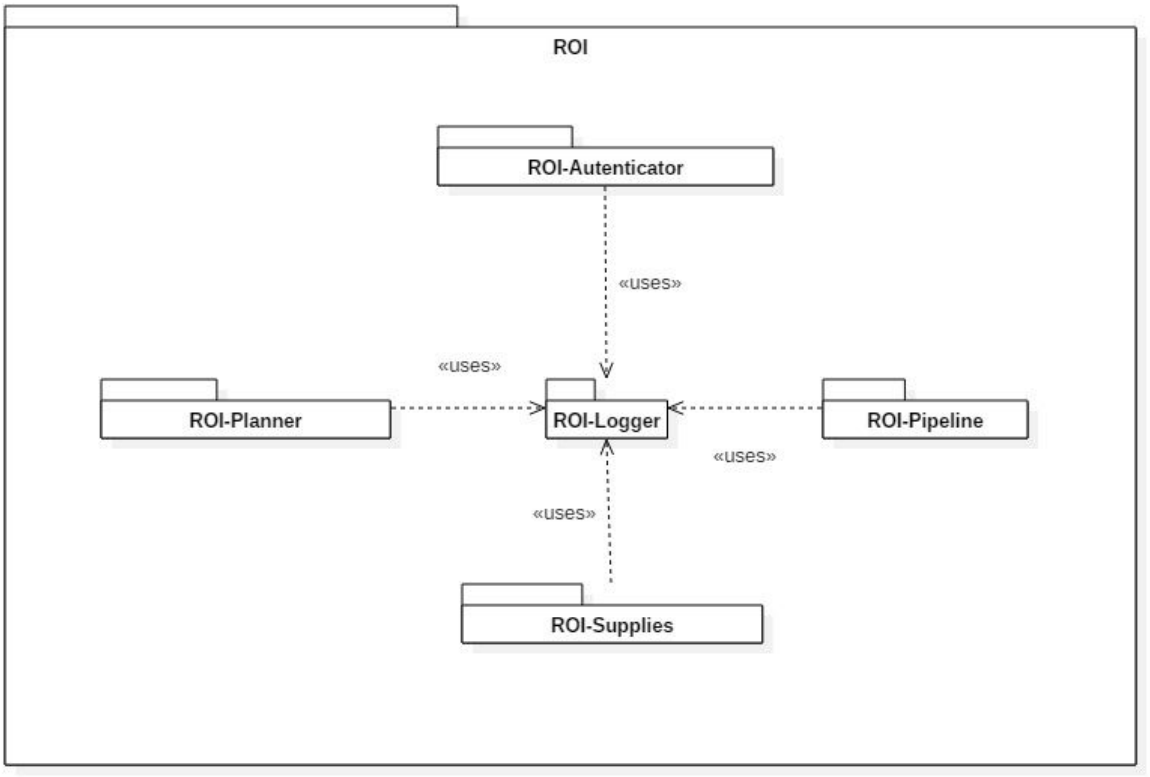
Decisiones de Diseño

Método de notificación de fallas de sensores

Para notificar de los errores decidimos llevar en un log todos los errores de los sensores, para que estos se puedan ver y acceder de manera sencilla con un editor de texto estándar.

Vista de Usos

Contexto



Decisiones de Diseño

Uso de GSON

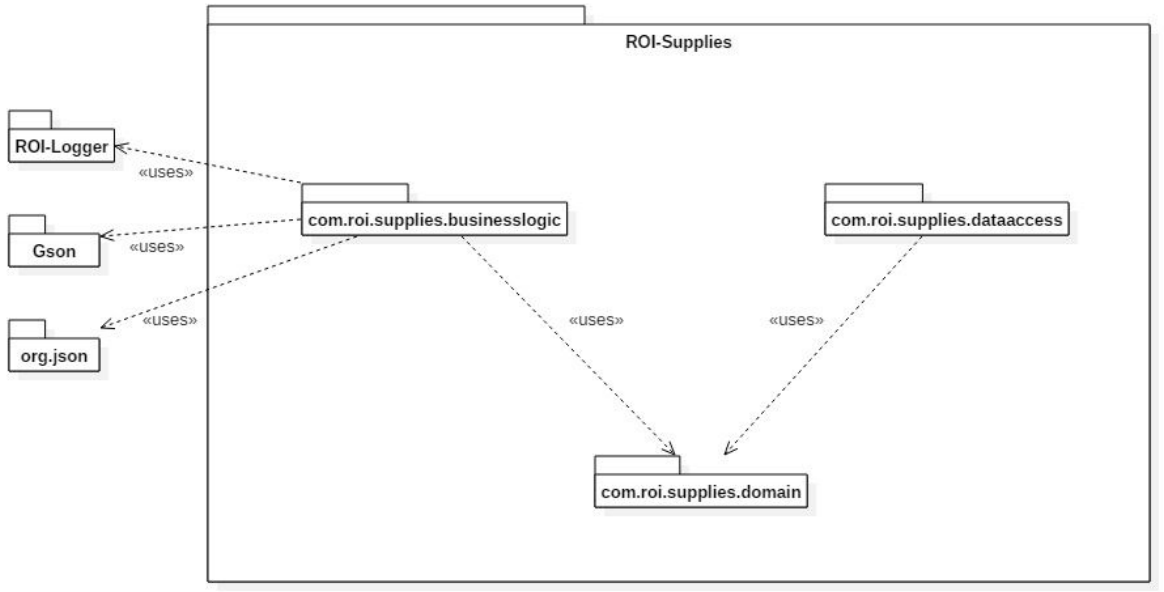
Para la transformación de json string a un objeto decidimos hacer uso de la tecnología de GSON.GSON es una herramienta vista en clase por lo que estamos familiarizados con la misma, además de contar con soporte continuo por Google. Esto nos permitió un manejo de estructuras json con código comprensible y rápido de implementar

Uso de org.json

Como herramienta adicional el manejo de string json usamos org.json, ya que el manejo de array json es realmente práctico y simplifica el código en gran medida.

ROI-Supplies

Representación primaria

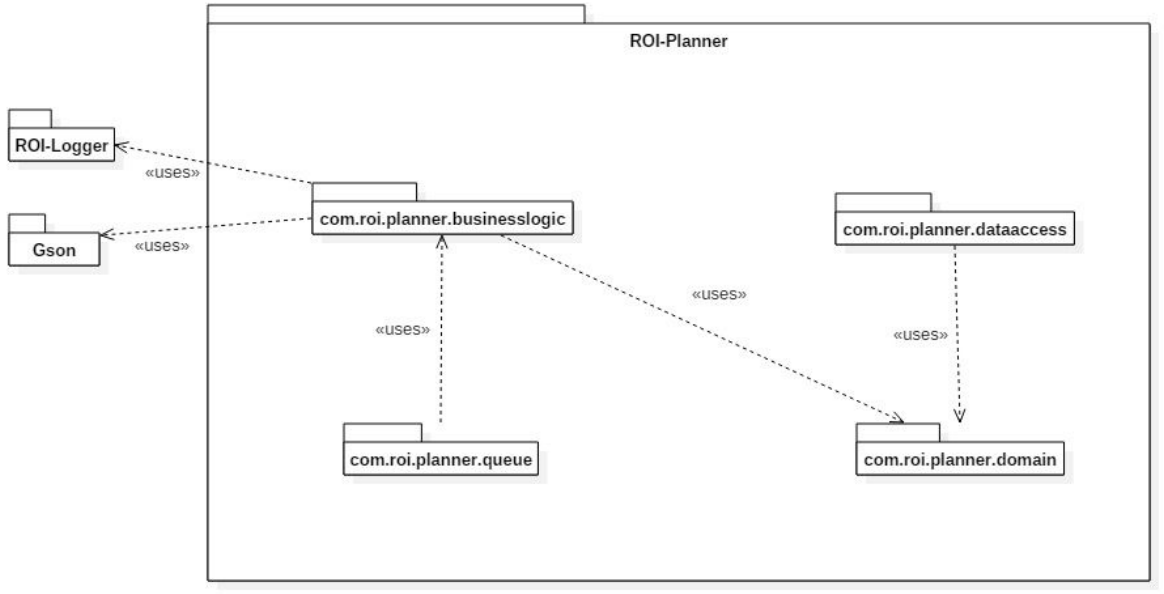


Catálogo de elementos

Elemento	Responsabilidad
com.roi.supplies.webservices	Módulo REST donde se exponen los servicios para poder consultar, agregar, modificar y eliminar SupplyOrder
com.roi.supplies.businesslogic	Módulo donde se encuentra la lógica del negocio que se encarga de conectar los servicios expuestos en los web services con la persistencia de a datos. También contiene SuppyOrderDTO
com.roi.supplies.dataaccess	Módulo que se encarga de la persistencia de los elementos en la base de datos. Se utiliza una interfaz genérica Repository, y luego una implementación específica para cada elemento del dominio (SupplyDA)
com.roi.supplies.domain	Módulo donde se definen la entidad SupplyOrder

ROI-Planner

Representación primaria



Decisiones de Diseño

Uso de JMS Queue

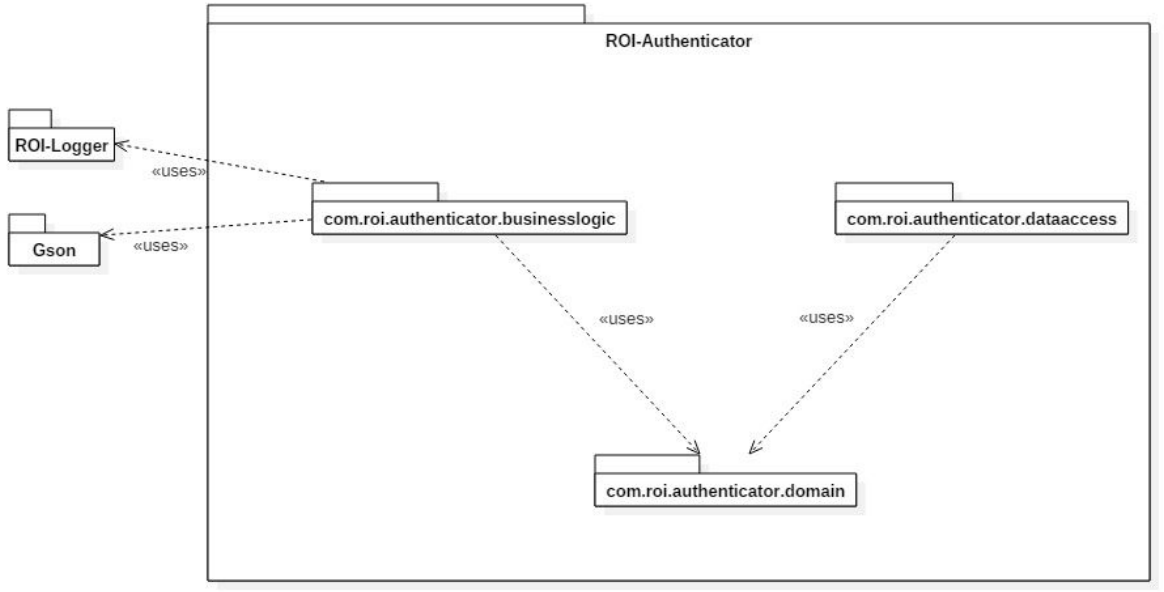
La razón por la cual se decidió usar una queue para la comunicación entre el ROI-Planner y ROI-Supplies fue para mejorar la disponibilidad de Supplies, dado que no necesita de estar conectados simultáneamente para realizar las llamadas. Esta decisión puede llegar a ser cuestionable dado que la cola implementada es local lo cual obligará a ROI-Planner y ROI-Supplies a estar desplegados en el mismo servidor. A pesar de esto nos parece que la disponibilidad del servicio es prioritario ante la portabilidad de este.
(REQN-5)

Catálogo de elementos

Elemento	Responsabilidad
com.roi.planner.webservices	Módulo REST donde se exponen los servicios para poder consultar, agregar, modificar y eliminar SupplyPlan
com.roi.planner.businesslogic	Módulo donde se encuentra la lógica del negocio que se encarga de conectar los servicios expuestos en los web services con la persistencia de datos. También contiene SupplyPlanDTO
com.roi.planner.dataaccess	Módulo que se encarga de la persistencia de los elementos en la base de datos. Se utiliza una interfaz genérica Repository, y luego una implementación específica para cada elemento del dominio(PlanDA)
com.roi.planner.domain	Módulo donde se definen las entidades SupplyPlan y NetworkSection
com.roi.planner.queue	Módulo en donde se separa el servicio de lectura de mensajes de OrderPlanQueue

Authenticator

Representación primaria

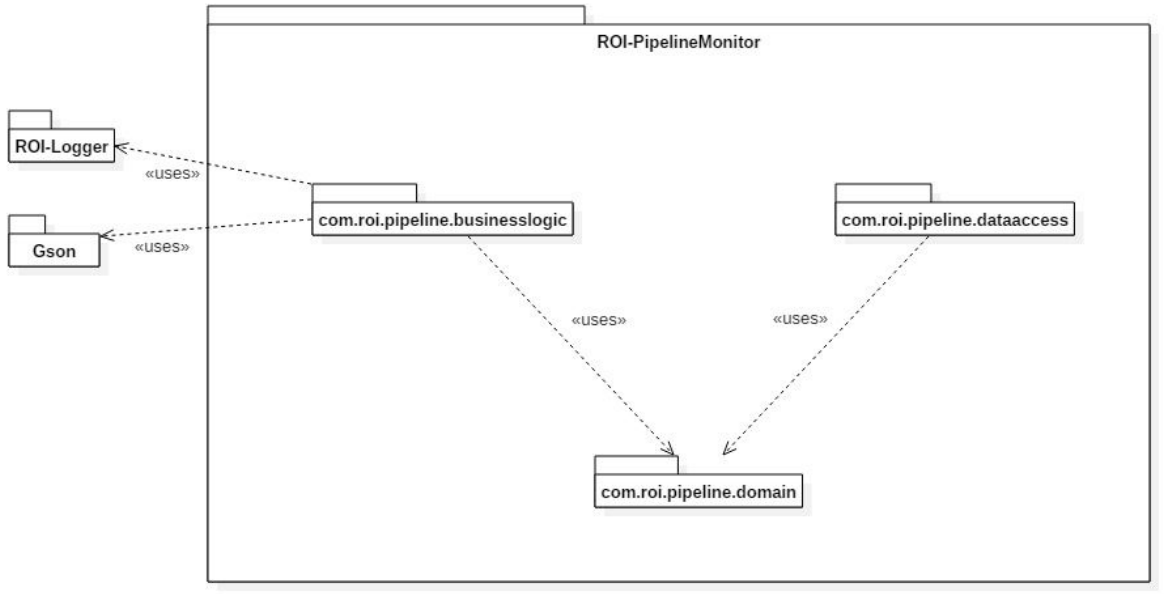


Catálogo de elementos

Elemento	Responsabilidad
com.roi.authenticator.webservices	Módulo donde se exponen los servicios para la autenticación de las operaciones.
com.roi.authenticator.businesslogic	Módulo donde se encuentra la lógica de negocio que se encarga de conectar los servicios en los web services con la persistencia de datos.
com.roi.authenticator.dataaccess	Módulo que se encarga de la persistencia de los elementos en la base de datos. Se utiliza una interfaz genérica Repository y luego una implementación específica para cada elemento del dominio (ValidKey)
com.roi.authenticator.domain	Módulo en donde se define la entidad ValidKey

ROI-PipalineMonitor

Representación Primaria

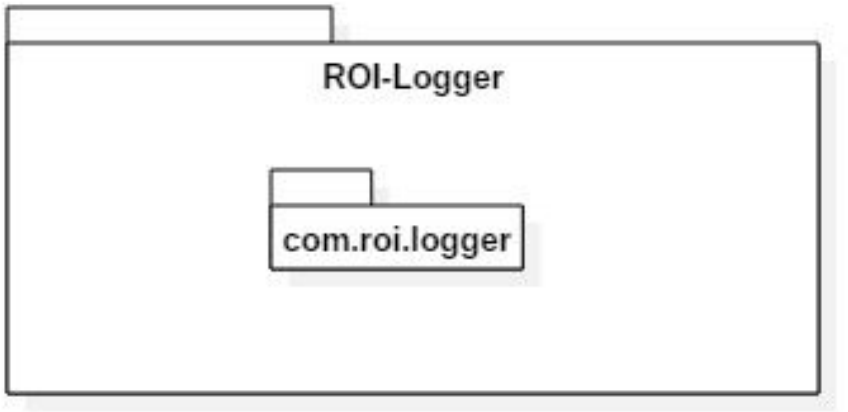


Catálogo de elementos

Elemento	Responsabilidad
com.roi.pipelinemonitor.webservices	Módulo donde se exponen los servicios para agregar y obtener Sensor y Pipeline
com.roi.pipeline.businesslogic	Módulo donde se encuentra la lógica de negocio que se encarga de conectar los servicios expuestos en los web services con la persistencia de datos. También contiene MeasureDTO, que se utiliza para comunicar las medidas reportadas por los Sensors.
com.roi.pipeline.dataaccess	Módulo que se encarga de la persistencia de los elementos en la base de datos. Se utiliza una interfaz genérica Repository y luego una implementación especifica para cada elemento del dominio (PipelineDA y SensorDA)
com.roi.pipeline.domain.	Módulo en donde se definen las entidades Pipeline y Sensor

ROI-Logger

Representación primaria



Decisiones de Diseño

Uso de Strategy en Log

Dado que uno de los principales requerimientos del cliente en cuanto al log era "*Se espera que la solución contemple la posibilidad de poder cambiar las herramientas o librerías concretas que se utilicen para producir esta información*", dado esto decidimos hacer uso de un patrón strategy esto no solo nos permite cambiar de tecnología fácilmente, sino que esto nos permite alternarla en tiempo de ejecución, si así lo desea el cliente. Esto nos permite tener un código más flexible y modificable.
(REQN-3)

Clase factory del Log

Dado que el cliente no solo queria poder cambiar de tecnología, sino que también quería permitir que la solución de log pudiera ser aplicada en otras aplicaciones de manera fácil, "*reutilizar esta solución en otras aplicaciones, con el menor impacto posible en el código*", decidimos hacer uso de una factory la cual se ocupa de instanciar la clase concreta del ActionLogger que se va a usar, permitiendo así desacoplar las aplicaciones de las tecnologías concretas ayudando junto con el patrón strategy usado a tener un código más fácilmente modificado, el cual pueda cambiar de tecnología en tiempo de ejecución.
(REQN-3)

Adapter dentro del Log

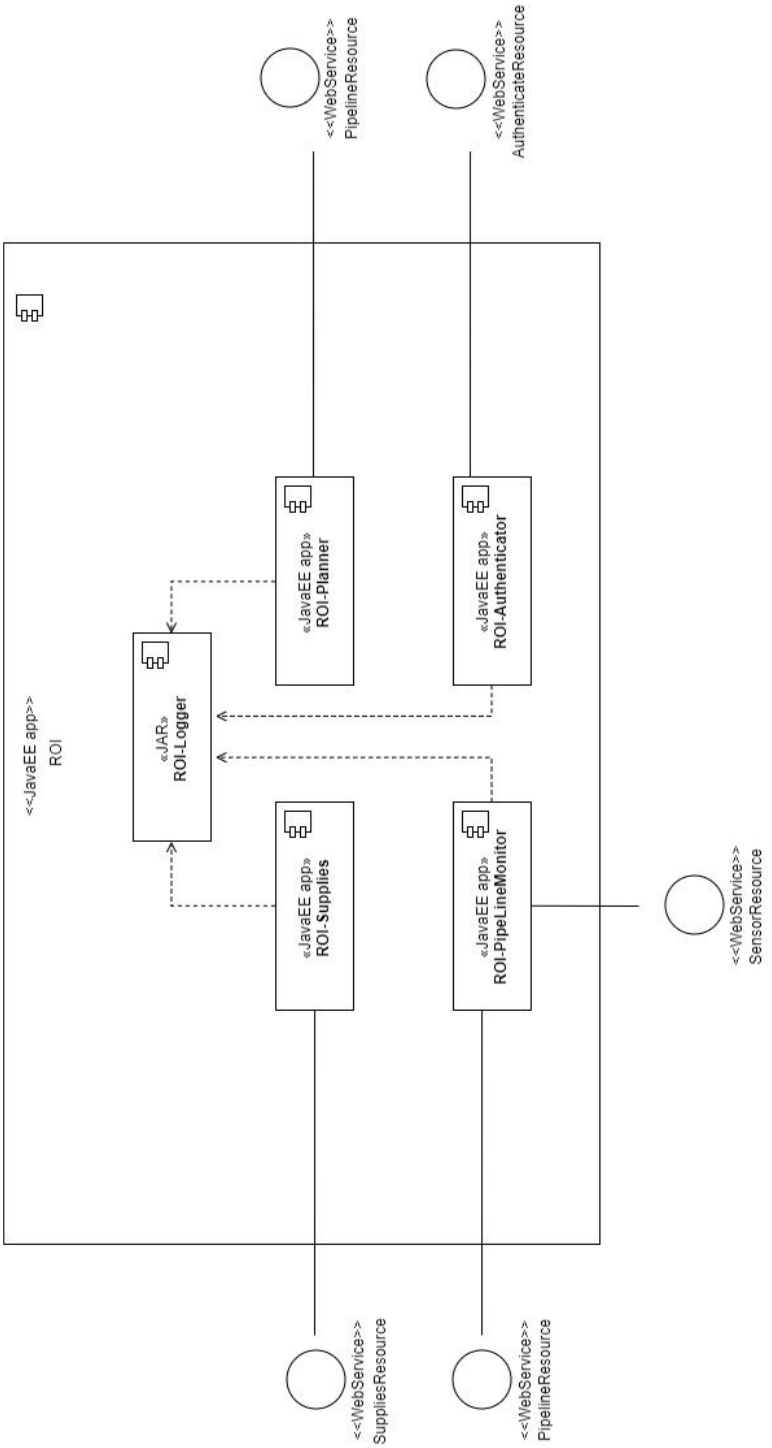
Dado que para implementar el log decidimos usar librerias de terceros, decidimos hacer uso de un patrón adapter para adecuar estas tecnologías a nuestras necesidades particulares del negocio.
(REQN-3)

Catálogo de elementos

Elemento	Responsabilidad
com.roi.logger.actionlog	Módulo donde se define la interfaz ActionLogger y la fabrica LogFactory. También provee una implementación de la interfaz LogAdapter.

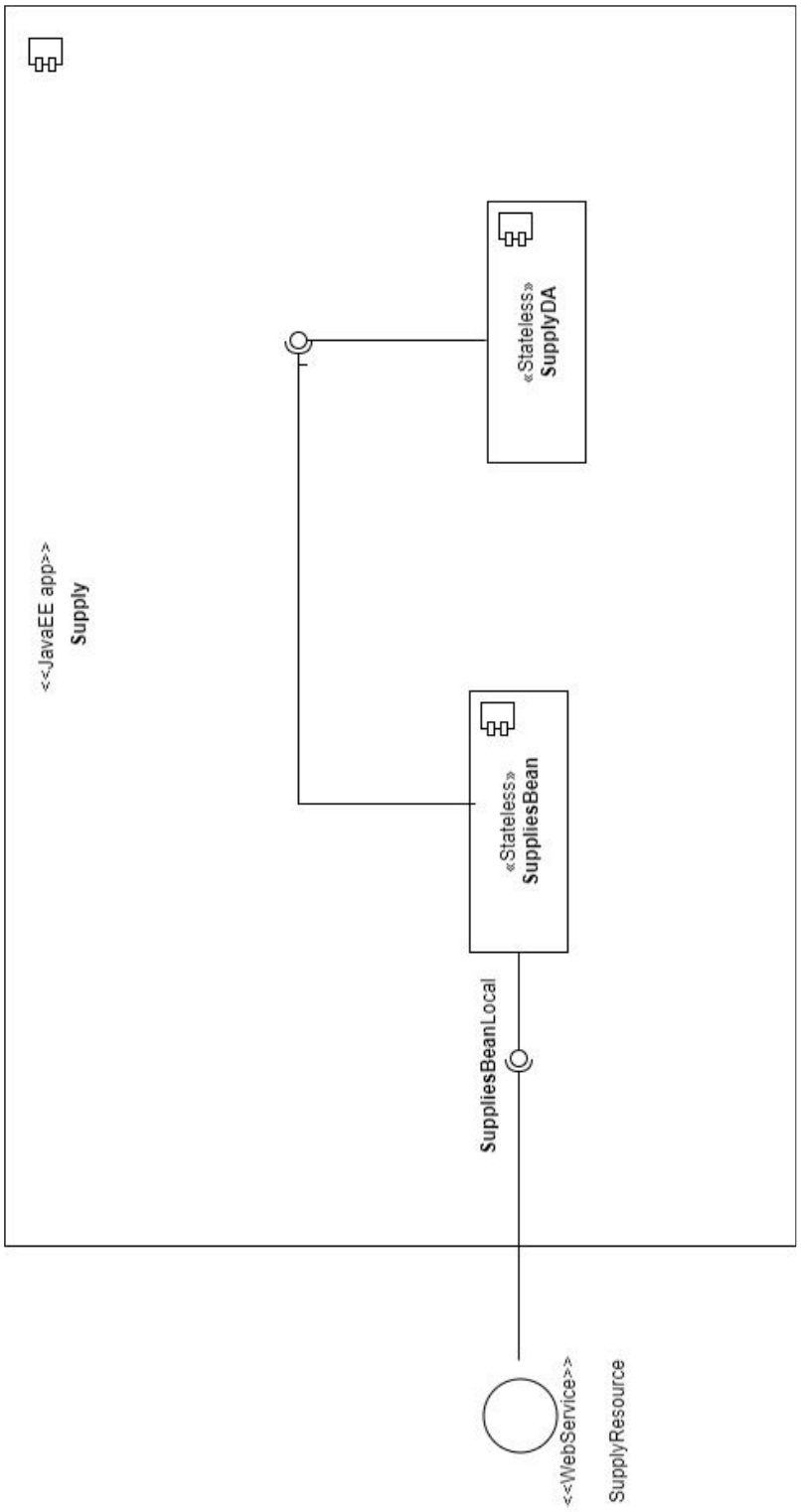
Vista de componentes y conectores

Contexto



ROI-Supply

Representación primaria



Catálogo de elementos

Componente/Conector	Tipo	Descripción
SuppliesBean	Stateless Session Bean	Este componente implementa la interfaz SuppliesBeanLocal y se almacenar y procesar los datos.
SupplyDA	Stateless Session Bean	Este componente se encarga intermediar con la base de datos para poder almacenar una SupplyOrder.
SupplyResource	WebService	Servicio Web REST, el cual permite a lo usuarios acceder a los SupplyOrder.

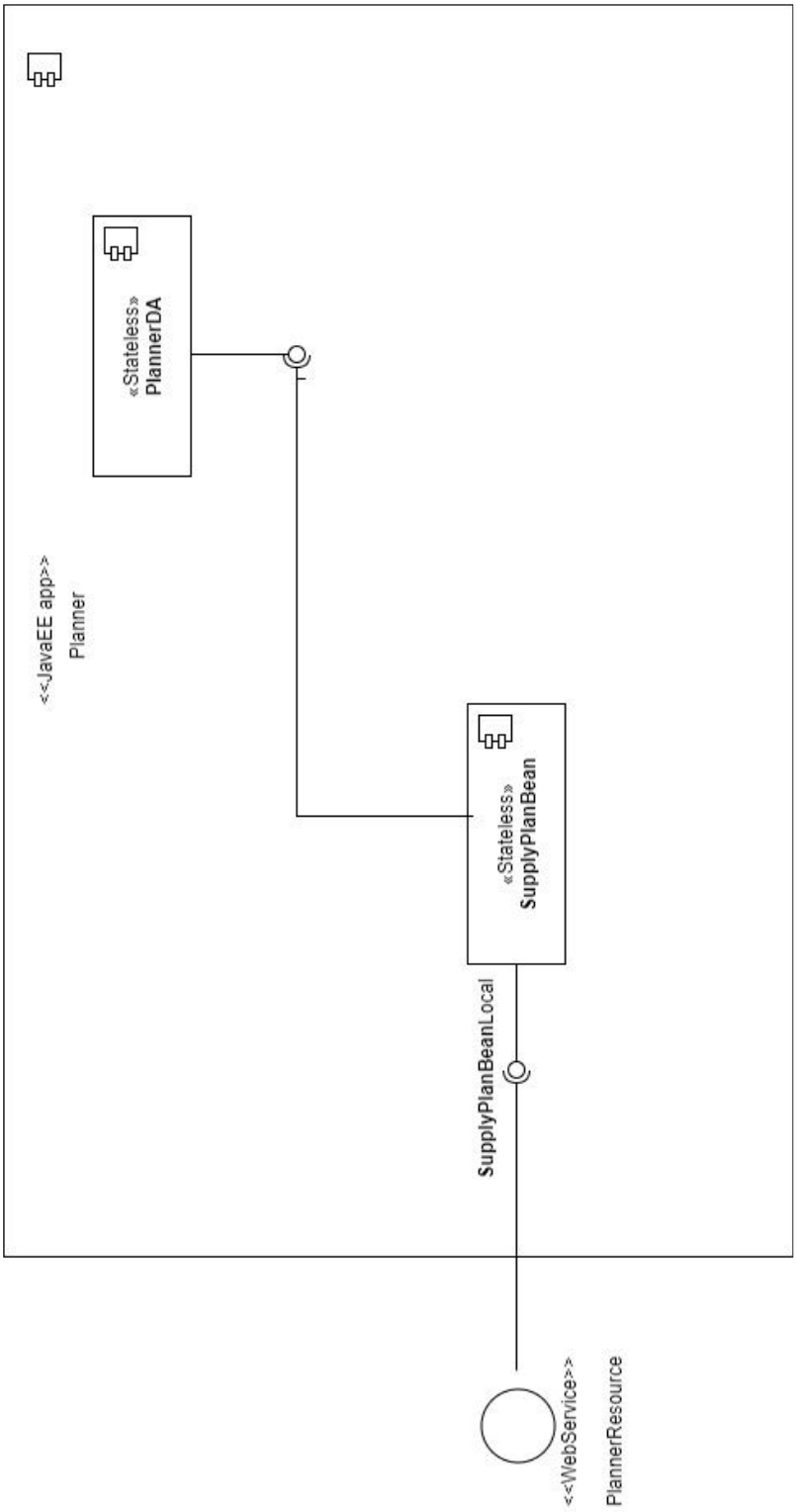
Interfaz	Detalle
SuppliesBeanLocal	CreateSupply **Sintaxis**: void postSupply(String newSupply) **Semántica**: Dado un id de cliente, una fecha de inicio, un volumen contratado, un dia de vencimiento y un id del punto de salida, crea un SupplyOrder **Pre-Condición**: El JSON enviado debería estar en el formato correcto. {"clientId": 8, "supplyingStart": "Nov 20, 2007 11:28:46 PM","volumeContracted": 1523,"serviceEndPointId": 9,"payDay": 25} **Post-Condición**: Crea un SupplyOrder con id autogenerado. **Manejo de Errores**: JsonSyntaxException: Retorna esta excepción si no pudo castear a SupplyOrder correctamente a partir de los parámetros recibidos en el body del request Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
	GetSupplies **Sintaxis**: String getSupplies() **Semántica**: Retorna una lista que contiene todas las SupplyOrder almacenadas en la base de datos del sistema **Pre-Condición**: -

	Post-Condición: Retorna lista de todas las SupplyOrder **Manejo de Errores**: Exception: Retorna esta excepción si ocurre alguna falla inesperada con un mensaje indicando el problema
	GetSupply **Sintaxis**: String getSupply(String getSupply) **Semántica**:Retorna una SupplyOrder especifica, la cual tiene identificador igual al string provisto por parámetro **Pre-Condición**: Debe existir la SupplyOrder que corresponde al identificador provisto por parámetro **Post-Condición**: Retorna una SupplyOrder con identificador {getSupply} **Manejo de Errores**: JsonSyntaxException: Retorna esta excepción si no pudo castear a SupplyOrder correctamente a partir de los parámetros recibido. Esto ocurre si no existe el SupplyOrder Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
	ModifySupply **Sintaxis**: void putSupply(String putSupply) **Semántica**: Dado un id, un id de cliente, una fecha de inicio, un volumen contratado, un dia de vencimiento y un id del punto de salida, modifica al SupplyOrder con identificador igual a {id} para que contenga los otros campos ingresados por parametros **Pre-Condición**: Debe existir la SupplyOrder que corresponde al identificador provisto por parámetro {id} **Post-Condición**: Actualiza en la base de datos segun los parametros ingresados **Manejo de Errores**: JsonSyntaxException: Retorna esta excepción si no pudo castear a SupplyOrder correctamente a partir de los parámetros recibidos en el body del request Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.

	DeleteSupply **Sintaxis**: void deletetSupply(String deleteSupply) **Semántica**: Dado un id por parametro, elimina de la base de datos a la SupplyOrden cuyo id sea igual a {deleteSupply} **Pre-Condición**: Debe existir la SupplyOrder que corresponde al identificador provisto por parámetro {deleteSupply} **Post-Condición**: Elimina de la base de datos al SupplyOrder solicitado **Manejo de Errores**: JsonSyntaxException: Retorna esta excepción si no pudo castear a SupplyOrder correctamente a partir de los parámetros recibidos en el body del request Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
SuppliesResource <<webservice>>	Esta interfaz provee un servicio web REST. Permite a los usuarios del frontend consultar, agregar, modificar y eliminar SupplyOrders

ROI-Planner

Representación primaria



Decisiones de Diseño

Mecanismo de autenticación de operaciones

El mecanismo de autentificación el cual decidimos usar consiste en una nueva aplicación ROI-Authenticator, el cual consta de un servicio el cual genera UUID y revisa si las UUID recibidas son válidas.

Por dar un ejemplo, esta sería la secuencia planeada por el equipo para la comunicacion entre roi.supply y roi.planner:

- Contexto: supply le va a avisar a planner que cree el plan correspondiente a una orden que le llego.
- Condición inicial: tanto supply, como authenticator conocen una clave UUID, única que nunca cambia, a esta le llamaremos CU.
- Paso 1: Supply envía un mensaje, el cual contiene la CU, solicitando una nueva clave a authenticator.
- Paso 2: Authenticator verifica la CU y de ser verdadera genera una nueva clave, esta clave, la guarda en una tabla y la envía como respuesta.
- Paso 3: Supply crea un nuevo mensaje para enviarle a planner. En el incluye la Id de la orden que se creó y la clave la cual recibió de authenticator como respuesta.
- Paso 4: Al recibir el mensaje en el planner este envía un mensaje a authenticator con la clave que recibió, en caso de que esta se encuentre en la lista del authenticator este devolverá “true”.
- Paso 5: Al recibir el mensaje “true”, el planner procederá a procesar la orden con normalidad.

(REQN-4)

(Mirar diagrama de secuencia de ROI-Planner, el cual se encuentra abajo)

Catálogo de elementos

Componente/Conector	Tipo	Descripción
SupplyPlanBean	Stateless Session Bean	Este componente implementa la interfaz SupplyPlanBeanLocaly se almacenar y procesar los datos.
PlannerDA	Stateless Session Bean	Este componente se encarga intermediar con la base de datos para poder almacenar una SupplyPlan.
PlannerResource	WebService	Servicio Web REST, el cual permite a lo usuarios acceder a los SupplyPlan.

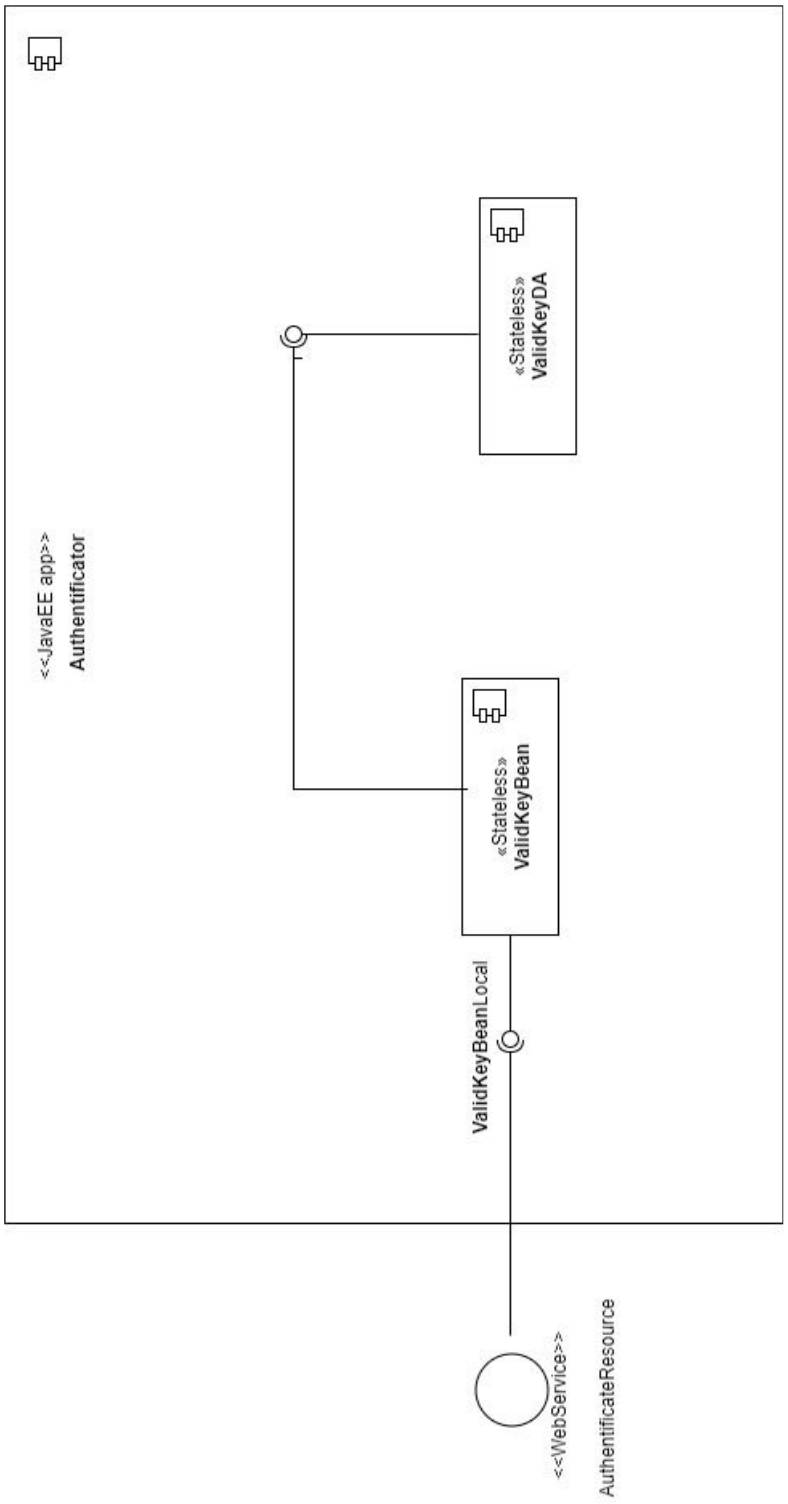
Interfaz	Detalle
SupplyPlanBeanLocal	CreateSupplyPlan **Sintaxis**: void createSupplyPlan(String createPlan) **Semántica**: Dado un identificador de SupplyOrder crea un SupplyPlan **Pre-Condición**: Debe existir una SupplyOrder con identificador igual a {createPlan} **Post-Condición**: Crea un SupplyPlan **Manejo de Errores**: NumberFormatException: Retorna esta excepción si el parámetro recibido en el body del request no es un número; Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
	GetSuppliesPlan **Sintaxis**: String getSuppliesPlan() **Semántica**: Retorna todos los SupplyPlan almacenados en la base de datos **Pre-Condición**: - **Post-Condición**: Retorna una lista con todos los SupplyPlan **Manejo de Errores**: Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
	GetSupplyPlan **Sintaxis**: String getSupplyPlan(String

	supplyPlan) **Semántica**:Retorna el SupplyPlan cuyo identificador es igual a {supplyPlan} provisto por parámetro **Pre-Condición**: Debe existir un SupplyPlan con identificador igual a {supplyPlan} **Post-Condición**: Retorna el SupplyPlan solicitado **Manejo de Errores**: JsonSyntaxException: Retorna esta excepción si no pudo castear a SupplyPlan correctamente a partir de los parámetros recibidos en el body del request. Esto ocurre si no existe el SupplyPlan Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
	ModifySupplyPlan **Sintaxis**: void modifySupplyPlan(String modifySupplyPlan) **Semántica**: Se encarga de modificar al SupplyPlan cuyo identificador sea igual al pasado por parámetro {modifySupplyPlan} **Pre-Condición**: Debe existir el SupplyPlan con identificador igual a {modifySupplyPlan} **Post-Condición**: Modifica al SupplyPlan solicitado. La modificación se realiza mediante una renovacion de las NetworkSections del mismo. **Manejo de Errores**: IndexOutOfBoundsException: Retorna este error si hay problemas al momento de querer obtener las NetworkSections desde la api externa provista. Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
	DeleteSupplyPlan **Sintaxis**: void deleteSupplyPlan(String deleteSupplyPlan) **Semántica**: Se encarga de eliminar logicamente de la base de datos al SupplyPlan cuyo identificador es igual al provisto por parámetro {deleteSupplyPlan} **Pre-Condición**: Debe existir el SupplyPlan con identificador igual a {deleteSupplyPlan} **Post-Condición**: Elimina logicamente al SupplyPlan solicitado. **Manejo de Errores**:

	Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
PlannerResource <<webservice>>	Esta interfaz provee un servicio REST. Permite a los usuarios del frontend consultar, agregar, modificar y eliminar SupplyOrders

ROI-Authenticator

Representación primaria



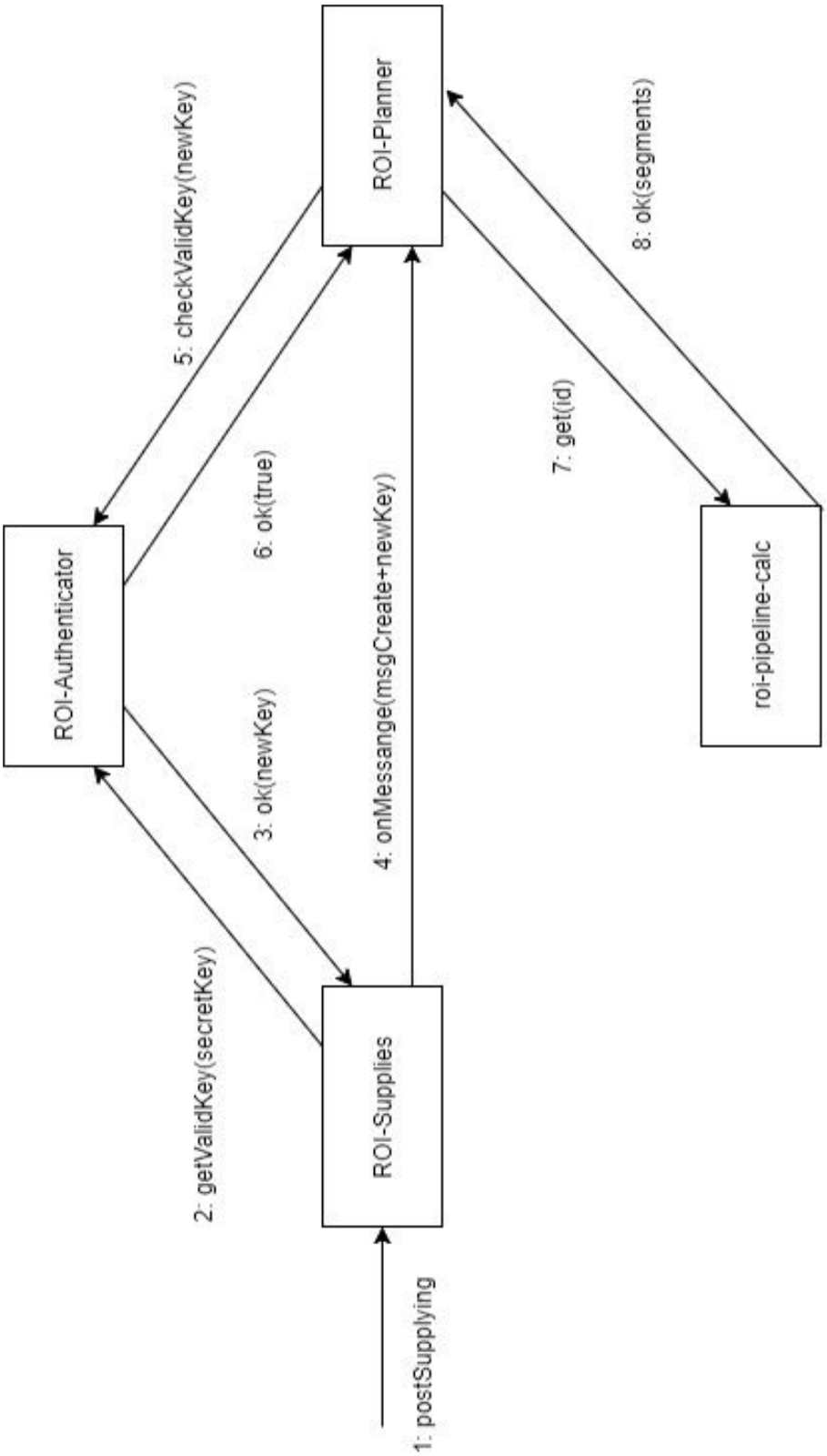
Catálogo de elementos

Componente/Conector	Tipo	Descripción
ValidKeyBean	Stateless Session Bean	Este componente implementa la interfaz ValidKeyBeanLocal y se almacenar y procesar los datos.
ValidKeyDA	Stateless Session Bean	Este componente se encarga intermediar con la base de datos para poder almacenar una ValidKey.
AutentificatorResource	WebService	Servicio Web, que permite obtener claves y validar claves.

Interfaz	Detalle
ValidKeyBeanLocal	GetValidKey **Sintaxis**: String getValidKey(String verifyUuid) **Semántica**: Recibe un UUID como string por parámetro y valida si es válido. En caso afirmativo, retorna un nuevo UUID que permite autenticar la comunicación. **Pre-Condición**: - **Post-Condición**: Retorna un UUID válido para autenticar. **Manejo de Errores**: Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
	CheckValidKey **Sintaxis**: boolean checkValidKey (String key) **Semántica**: Recibe un UUID como string por parámetro y retorna si es válido o no. Para esto verifica si se encuentra en la base de datos. **Pre-Condición**: - **Post-Condición**: Retorna el resultado booleano de si es válido o no la {key} provista **Manejo de Errores**: -

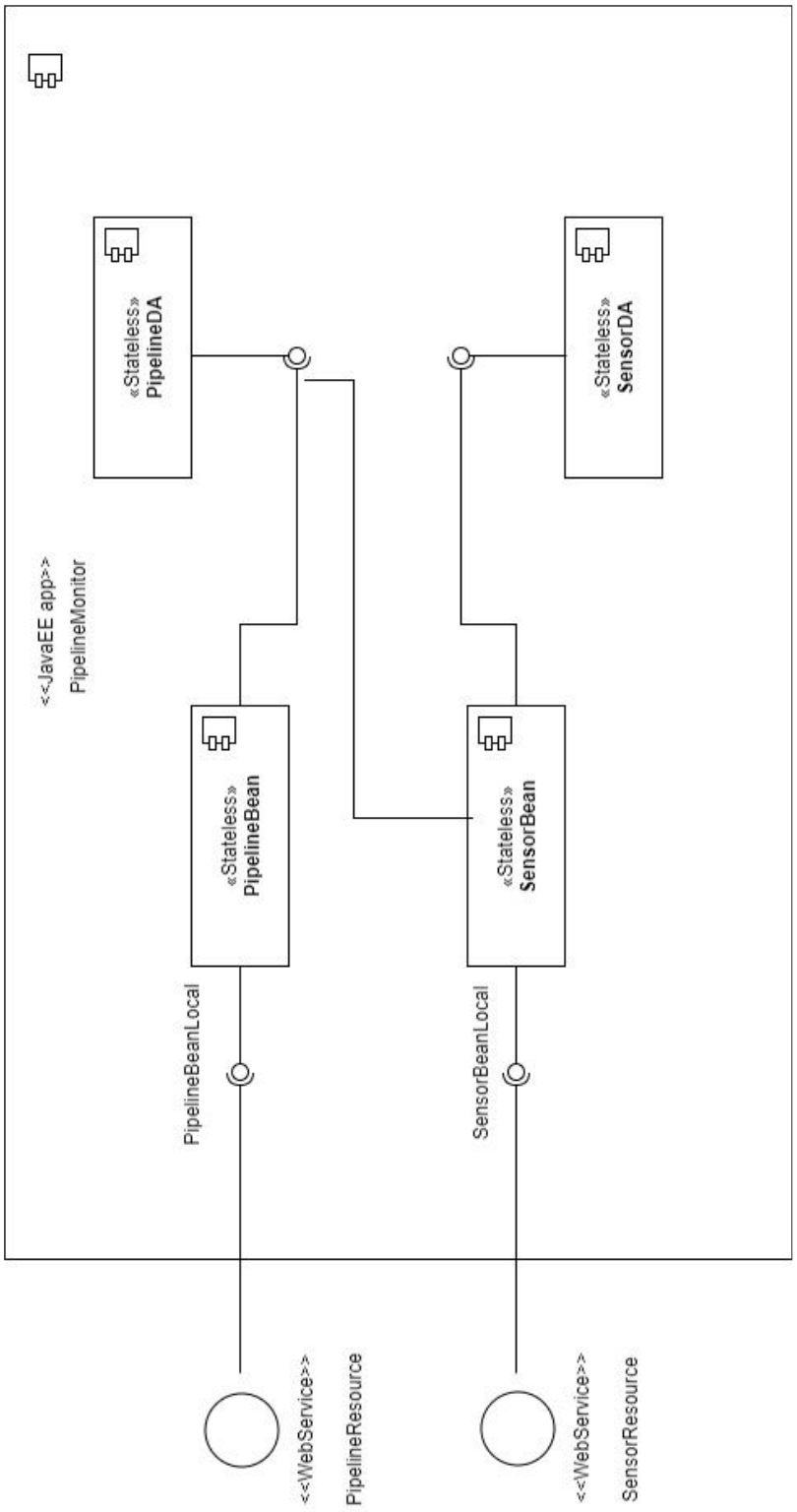
	DeleteKey **Sintaxis**: void deleteKey (String key) **Semántica**: Elimina de la base de datos a la key igual a {key} provisto por parámetro **Pre-Condición**: Existe la key solicitada en la base de datos como una validKey. **Post-Condición**: Elimina de la base la key solicitada **Manejo de Errores**: -
AuthenticateResource <<webservices>>	Esta interfaz provee un servicio con dos operaciones de tipo HTTP POST.. Permite realizar la autenticación de las operaciones de los usuarios mediante dos operaciones: obtener una Key válida y verificar si una Key es válida

Sequence


ROI-Planner
ROI-Authenticator
ROI-Supplies
roi-pipeline-calc
1: postSupplying
2: getValidKey(secretKey)
3: ok(newKey)
4: onMessange(msgCreate+newKey)
5: checkValidKey(newKey)
6: ok(true)
7: get(id)
8: ok(segments)

ROI-PipalineMonitor

Representación primaria



Decisiones de Diseño

Eficiencia de procesamiento de datos de sensores

Dado que el cliente nos advirtió "*Tenga en cuenta que esta operación es de uso masivo e intensivo, por lo tanto se valora una solución óptima en el uso de recursos*", nosotros decidimos tomar las siguientes acciones como medidas para agilizar las consultas de los sensores:

- No guardar los datos enviados por el sensor en una base de datos, si bien mantener un registro de los datos particulares de cada sensor podria ser util para el cliente para poder ver el histórico del estado de las pipelines, guardar estas y buscar los datos en la base enlentece las consultas.
- Minimizar las validaciones, decidimos bajar la cantidad de las validaciones para esta operación para agilitar la consulta, las únicas validación que se tienen son si el sensor existe y si este esta conectado a la pipeline indicada.
- Simplificación de la determinación de un error, si bien el cliente especifica claramente "*Considere que la presión medida en un punto de suministro debiera mantenerse siempre constante tomándose como aceptable una variación del 1% (aunque puntos diferentes pueden tener presiones diferentes)*", no hace referencia que el sensor esté expresamente roto si es que el valor está fuera del margen. Por otro lado el cliente nos dice, "*Esto implica que **roi-pipeline-monitor** recibirá la misma información sobre presión proporcionada por más de un sensor y deberá decidir su validez*" esto podría interpretarse como que si mas de un sensor mide una desviación de más de un 1% esto no quiere decir que todos los sensores esten rotos, sino que indica un posible problema en la válvula y/o actuador. Pero esta validación implicaría un mayor uso de recursos para cada consulta, por lo que decidimos mantenernos simplemente con lo expresamente dicho por el cliente y en caso de que la desviación de la medida está expresamente por arriba del 1% notificar al cliente.

(REQN-1)

(Ver diagrama de secuencia)

Actualización de OrderPlan

Dado que el cliente no especificó exactamente como se actualiza un OrderPlan, decidimos simplemente pedirle nuevos segmentos de red a roi-pipeline-calc.

Catálogo de elementos

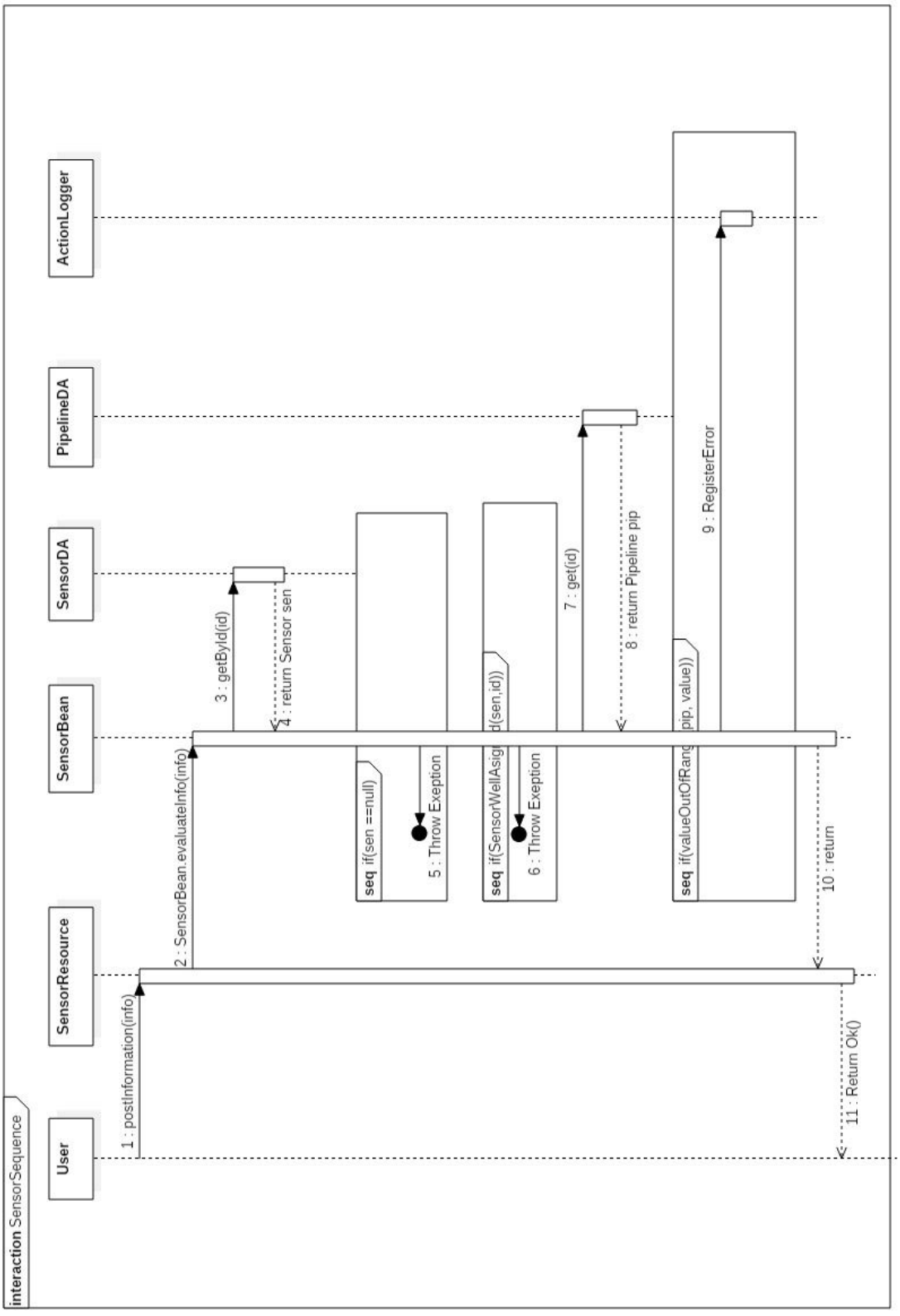
Componente/Conector	Tipo	Descripción
PipelineBean	Stateless Session Bean	Este componente implementa la interfaz PipelineBeanLocal y se almacenar y procesar los datos.
PipelineDA	Stateless Session Bean	Este componente se encarga intermediar con la base de datos para poder almacenar Pipeline.
SensorBean	Stateless Session Bean	Este componente implementa la interfaz SensorBeanLocal y se almacenar y procesar los datos.
SensorDA	Stateless Session Bean	Este componente se encarga intermediar con la base de datos para poder almacenar Sensors.
PipelineResource	WebService	Servicio Web, el cual permite a los usuarios agregar y obtener Pipelines
SensorResource	WebService	Servicio Web, el cual permite a los usuarios agregar y obtener Sensors y registrar datos obtenidos por un sensor.

Interfaz	Detalle
PipelineBeanLocal	CreatePipeline **Sintaxis**: void createPipeline(String averageValue) **Semántica**: Dado un String, que contiene un el valor esperado de la presión en un Pipeline, genera un nuevo Pipeline **Pre-Condición**: El string que se le envía puede ser casteado a integer. **Post-Condición**: Se crea la pipeline con el valor de presión asociado. **Manejo de Errores**: NumberFormatException: Se retorna esta excepción si el parámetro provisto no puede ser casteado a integer. Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
	GetAllPipelines **Sintaxis**: String getAllPipelines() **Semántica**: Retorna un string de formato json con una lista de Pipelines **Pre-Condición**: - **Post-Condición**: Retorna un string de formato json con una lista de Pipelines **Manejo de Errores**: Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
	GetPipelines **Sintaxis**: String getPipelines(String jsonPipeline) **Semántica**: Recibe por parametro un identificador de Pipeline y retorna un string de formato json con el Pipeline cuyo identificador es igual a {jsonPipeline} **Pre-Condición**: Existe el Pipeline con identificador {jsonPipeline} **Post-Condición**: Retorna un string de formato json con el Pipeline solicitado **Manejo de Errores**: IllegalArgumentException: Retorna esta excepción si no encuentra el Pipeline solicitado Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.

PipelineResource <<webservice>>	Esta interfaz provee un servicio con 2 operaciones, GET y POST. Permite a los usuarios del frontend realizar la consulta y creación de Pipelines
SensorBeanLocal	CreateSensor **Sintaxis**: void createSensor(String jsonSensor) **Semántica**: Dado un String, que contiene el id de un pipeline, genera un Sensor **Pre-Condición**: Existe el Pipeline con el id {jsonSensor} provisto por parámetro **Post-Condición**: Se crea la sensor con la id de la pipeline asociada. **Manejo de Errores**: IllegalArgumentException: Se retorna esta excepción si el argumento provisto no corresponde a un Pipeline existente. NumberFormatException: Se retorna esta excepción si el parámetro provisto no puede ser casteado a integer. Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
	GetSensors **Sintaxis**: String getSensors() **Semántica**: Retorna un string de formato json con todos los Sensors **Pre-Condición**: - **Post-Condición**: Retorna un string de formato json con la lista de todos los Sensors **Manejo de Errores**: Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
	GetSensor **Sintaxis**: String getSensor(String jsonSensor) **Semántica**: Retorna un string de formato json con una lista de Pipelines **Pre-Condición**: - **Post-Condición**: Retorna un string de formato json con el pipeline solicitado **Manejo de Errores**: IllegalArgumentException: Se retorna esta excepción si el argumento provisto no corresponde a un Pipeline existente.

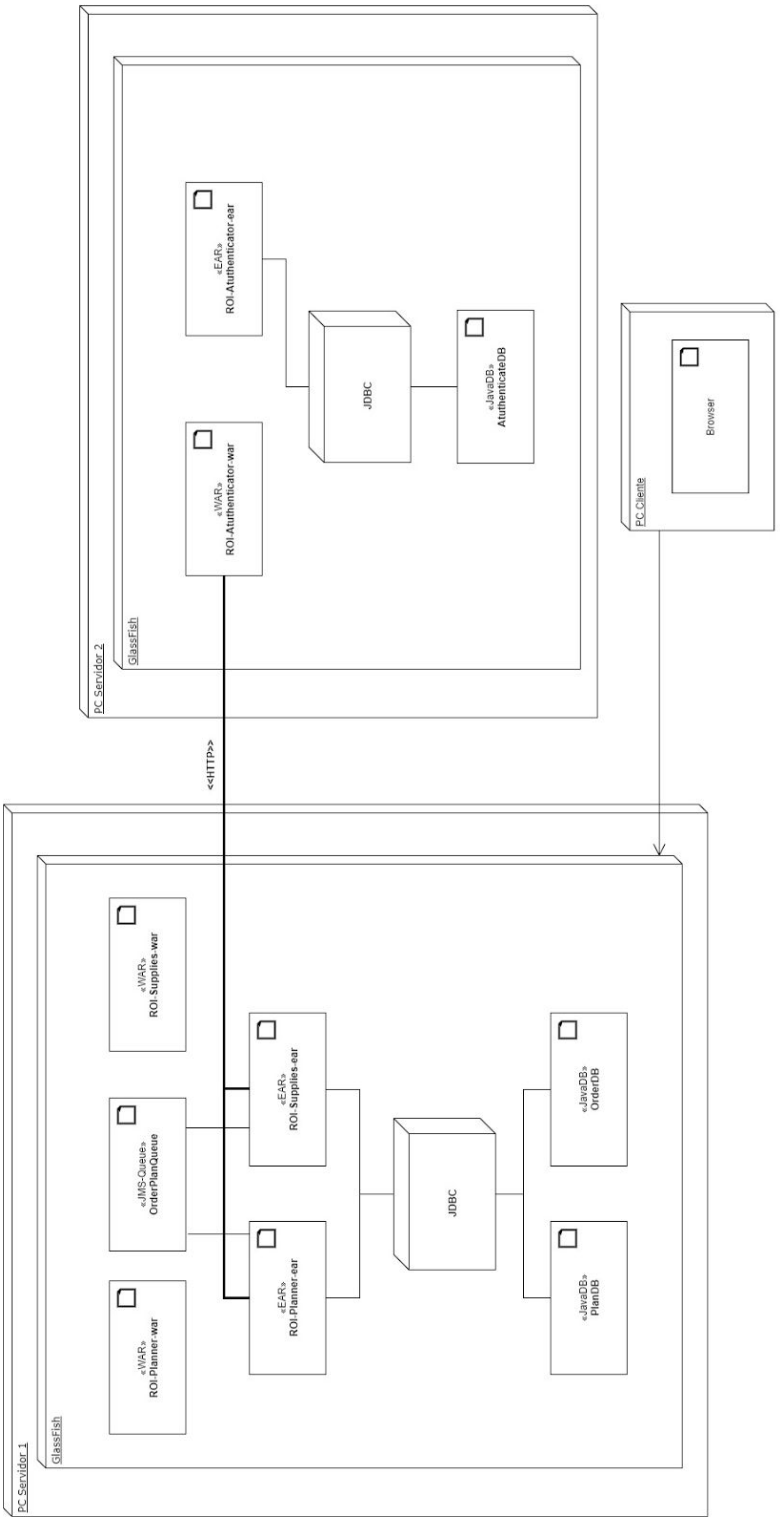
	NumberFormatException: Se retorna esta excepción si el parámetro provisto no puede ser casteado a integer. Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
	EvaluateInfo **Sintaxis**: void evaluateInfo(String jsonInfo) **Semántica**: Recibe por parámetro un Json con el formato {“sensorId":"2", "pipelineId":"2",”value":"322", "date":"Nov 25, 2017 11:28:32 PM"}. Procesa y evalúa el estado del sensor en base a los datos recibidos **Pre-Condición**: El json debería estar bien formado y debe existir la Pipeline que indica **Post-Condición**: Procesa y evalúa el estado del sensor en base a los datos **Manejo de Errores**: JsonSyntaxException: Retorna esta excepción si no pudo castear a MeasureDTO a partir de lo obtenido por parámetro IllegalArgumentException: Se retorna esta excepción si el argumento provisto no corresponde a un Pipeline existente. NumberFormatException: Se retorna esta excepción si el parámetro provisto no puede ser casteado a integer. Exception: Retorna esta excepción si ocurre si alguna falla inesperada con un mensaje indicando el problema.
SensorResource <<webservice>>	Esta interfaz provee un servicio con 2 operaciones, GET y POST. Permite a los usuarios del frontend realizar la consulta y creación de Sensors

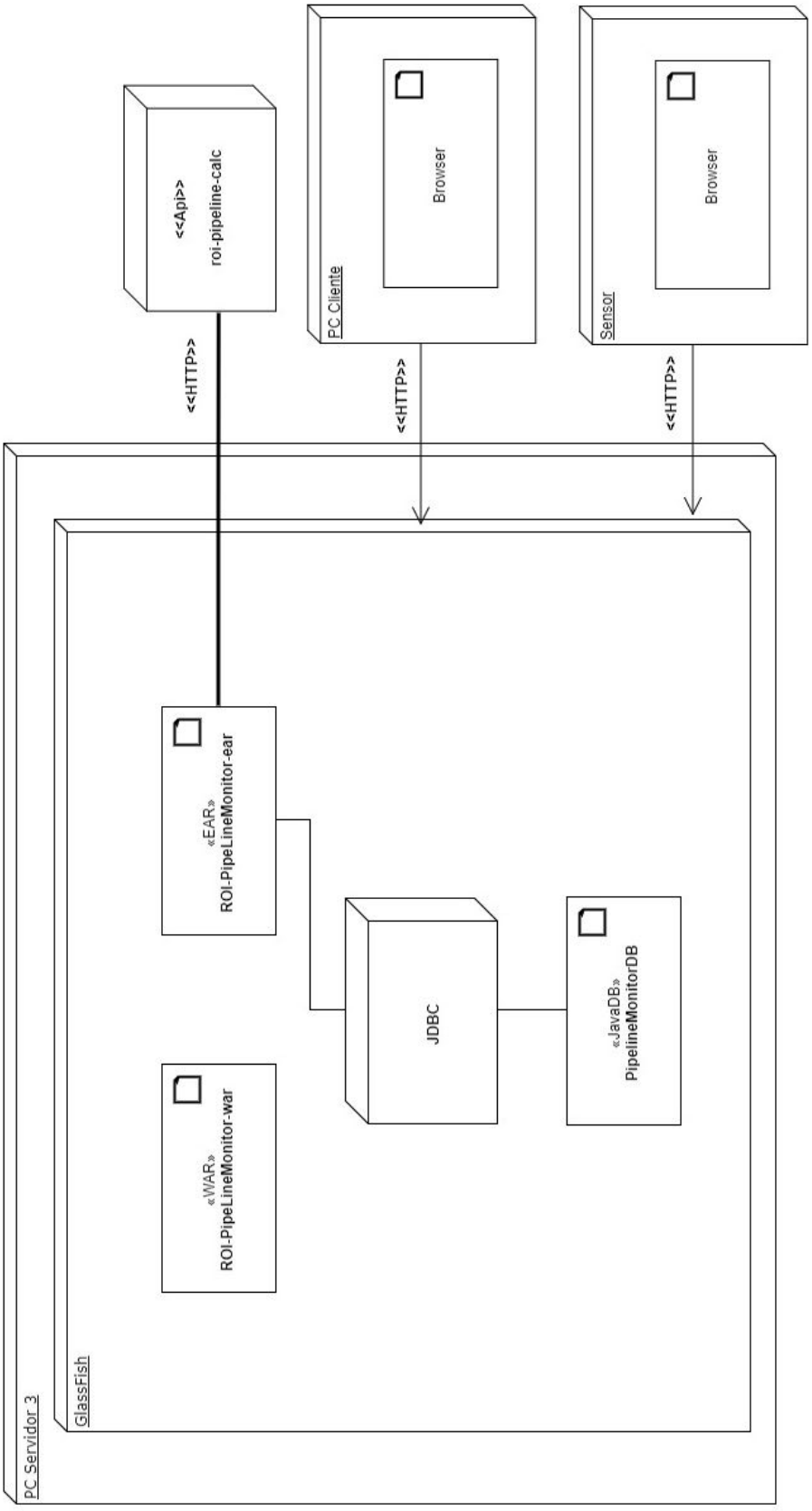
Sequence


interaction SensorSequence
User
SensorResource
SensorBean
SensorDA
PipelineDA
ActionLogger
1 : postInformation(info)
2 : SensorBean.evaluateInfo(info)
3 : getById(id)
4 : return Sensor sen
seq if(sen ==null)
5 : Throw Exeption
seq if(SensorWellAsig
6 : Throw Exeption
7 : get(id)
8 : return Pipeline pip
seq if(valueOutOfRang
9 : RegisterError
10 : return
11 : Return Ok()

Vista de Deploy

Representación Primaria




PC Servidor 3
GlassFish
«EAR»
ROI-PipeLineMonitor-ear
«WAR»
ROI-PipeLineMonitor-war
JDBC
«JavaDB»
PipelineMonitorDB
<<HTTP>>
<<HTTP>>
<<HTTP>>
<<Api>>
roi-pipeline-calc
PC Cliente
Browser
Sensor
Browser

Catálogo de elementos

Nodo	Tipo	Descripción
ROI-Planner-war	WAR	Este nodo recibe las consultas HTTP (Sobre los OrderPlan) enviadas por un Browser utilizando web services de tipo Rest y delega las operaciones al ear.
ROI-Supplies-war	WAR	Análogo a ROI-Planner-war
ROI-Atuthenticator-war	WAR	Este nodo recibe las consultas HTTP enviadas por servidores internos y genera/valida claves para la comunicación entre servidores.
ROI-PipeLineMonitor-war	WAR	Análogo a ROI-Planner-war
ROI-Planner-ear	EAR	Para ver la funcionalidad de este nodo, debe ir a la vista de componentes y conectores correspondiente.
ROI-Supplies-ear	EAR	Para ver la funcionalidad de este nodo, debe ir a la vista de componentes y conectores correspondiente.
ROI-Atuthenticator-ear	EAR	Para ver la funcionalidad de este nodo, debe ir a la vista de componentes y conectores correspondiente.
ROI-PipeLineMonitor-ear	EAR	Para ver la funcionalidad de este nodo, debe ir a la vista de componentes y conectores correspondiente.
PlanDB	JabaDB	Este nodo almacena los datos de las OrderPlan
OrderDB	JabaDB	Análogo a PlanDB
AtuthenticateDB	JabaDB	Análogo a PlanDB
PipelineMonitorDB	JabaDB	Análogo a PlanDB
roi-pipeline-calc	API	Esta es una API la cual nos fue dado por ROI la cual nos permite a partir de un OrderPlan obtener una serie de NetworkSements
JDBC	API	JDBC provee una API que permite mapear,

		manejar entidades y persistir en Java
OrderPlanQueue	JMS-Queue	Queue la cual se usa para comunicar a ROI-Planner, las acciones ejecutadas sobre ROI-Supplies
Glassfish	Server	Glassfish Server se encarga de desplegar todos los artefactos de nuestra aplicación
PC Server	Server	Este nodo nos permite exponer nuestro sistema a clientes y servidores.
PC Clients	Terminal	Estas son terminales las cuales se comunicaran por medio HTTP con las APIs expuestas
Sensor	Terminal	Estas son terminales las cuales se comunicaran por medio HTTP con API de PpipelineMonitor para enviar sus datos captados
Browser	Browser	Medio por el cual las terminales se comunicaran con las APIs expuestas

*Asumimos que como la librería ROI-Logger y debe de estar instalada dentro de los proyectos que la usan, no es necesario mostrarla en el deploy.

Endpoints provistos

Endpoint	Método HTTP	Descripción
http://localhost:8080/ROI-Supplies-war/supplies	GET	Permite obtener todas las SupplyOrders almacenadas en el sistema
http://localhost:8080/ROI-Supplies-war/supplies	POST	Permite generar una nueva SupplyOrder. Esta SupplyOrder es recibida en el body del request en formato JSON. La ID es autogenerada, no se incluye en el JSON.
http://localhost:8080/ROI-Supplies-war/supplies/{id}	DELETE	Permite eliminar físicamente una SupplyOrder con ID igual a {id} provista en la URI
http://localhost:8080/ROI-Supplies-war/supplies	PUT	Permite modificar una SupplyOrder ya existente. Esta SupplyOrder es recibida en el body del request en formato JSON. Se provee también la ID de la orden a cambiar.
http://localhost:8080/ROI-Planner-war/planner	GET	Permite obtener todos los SupplyPlan almacenados en el sistema
http://localhost:8080/ROI-Planner-war/planner	POST	Permite generar un nuevo SupplyPlan. Para generar el SupplyPlan recibe en formato JSON el ID correspondiente a la SupplyOrder que se le desea asignar.
http://localhost:8080/ROI-Planner-war/planner/{id}	DELETE	Permite eliminar lógicamente un SupplyPlan con ID igual a {id} provista en la URI
http://localhost:8080/ROI-Planner-war/planner	PUT	Permite modificar un SupplyPlan ya existente. Recibe en el body del

		request en formato JSON el ID de la SupplyOrder que identifica a dicho plan. Se le asignan nuevas tramas automáticamente.
http://localhost:8080/ROI-PipeLineMonitor-war/sensor	GET	Permite obtener todos los Sensors almacenados en el sistema
http://localhost:8080/ROI-PipeLineMonitor-war/sensor	POST	Permite generar un nuevo sensor. Para generar el sensor se recibe en el body del request en formato JSON el ID correspondiente a la Pipelines a la que pertenece dicho sensor
http://localhost:8080/ROI-PipeLineMonitor-war/sensor/report	POST	Permite recibir al sistema el reporte de un sensor en particular. Para generar este reporte se recibe en el body del request en formato JSON el Id del sensor, el Id de la Pipeline, el valor medido por el sensor y la fecha/hora de la medida.
http://localhost:8080/ROI-PipeLineMonitor-war/pipeline	GET	Permite obtener todas las Pipelines almacenadas en el sistema
http://localhost:8080/ROI-PipeLineMonitor-war/pipeline	POST	Permite generar una nueva Pipeline . Para generar la Pipeline se recibe en el body del request en formato JSON el número correspondiente al valor esperado promedio que deberían registrar los Sensors en dicha Pipeline.

Observaciones

- Si bien el programa puede ser puesto en múltiples computadores, este necesita de ser re-compilado para cambiar las IP de los Endpoints. Por esto esta razón preferimos y recomendamos usar solo 1 dispositivo, como servidor.
- En la actualidad el sistema de ROI-Logger genera archivos .log basura al escribir sobre el verdadero log, si bien esto puede ser molesto esteticamente, no afecta el funcionamiento del mismo. Una posible solución a este problema sería sacar el log de adentro de los stateless bean y llevarlo a una nueva clase estática la cual contenga un instancia singleton del log. Esto solucionaría parte de los archivos residuales.
- En la actualidad la comunicación realizada hacia ROI-Authenticator no es lo suficientemente segura para uso profesional, ya que se simplificaron algunas funciones. Para resolver esto se debería cambiar el mecanismo de comunicación a HTTPS y se mandar las claves y los datos dentro de headers.
- Si bien en la letra parecen haber varios tipos de usuarios, no autentificar ya que no estaba expresamente requerido