ARCHITECTURE DU LOGICIEL

Version :	1.0
Date :	31/03/2017
Rédigé par :	L'équipe SmartLogger
Relu par :	L'équipe SmartLogger
Approuvé par :	---

Objectif : Le but de ce document est de décrire les solutions techniques conçues pour répondre aux exigences définies dans la spécification technique de besoin. Il doit identifier et décrire les différents modules ou constituants du logiciel ainsi que leurs interfaces de telle sorte que chacun d'entre eux puisse être développé de façon autonome par un membre de l'équipe avant d'être intégré.
La lecture de ce document doit également permettre d'appréhender l'ensemble des paramètres techniques pris en considération par les auteurs pour définir et élaborer les stratégies et démarches de développement.

HISTORIQUE DE LA DOCUMENTATION

Version	Date	Modifications réalisées
0.1	17/11/16	Création du document
0.2	01/12/16	Augmentation du contenu de la partie Architecture Statique
0.3	14/12/16	Ajout de l'Architecture Statique détaillée
1.0	11/01/17	Correctifs et Restructuration du document

1. Objet :

Le but du projet est de créer un système, permettant d'alerter l'utilisateur sur des données en provenance d'applicatifs défectueux dans l'optique de faciliter leurs correctifs. Une fois produit, ce système sera utilisé par l'entreprise cliente à leurs propres fins.
Dans ce but, l'entreprise cliente a fait part de ses exigences en termes de fonctionnalités minimales :

- Le système devra être capable d'analyser des flux de données de type Shinken, Logstash et Kafka.
- Il alertera les opérateurs en employant des moyens de communication utilisés par l'entreprise, tels que l'application Slack, l'envoi de mails ou par notification SMS.
- Il pourra s'adapter à des flux de données ou méthodes d'alerte non prédéfinies qui seront implantées dans le système par de futurs utilisateurs.
- Enfin, il devra fonctionner sur de longues périodes de fonctionnement et, dans l'idéal, être opérationnel indéfiniment.

A ce titre, l'exigence principale en terme de conception réside dans le découpage efficace de l'application à réaliser. Le souhait du client étant de récupérer un système facilement personnalisable. Il faudra s'assurer que certains composants-clés soient aisément interchangeables :

- Le type d'algorithme employé par l'analyseur du système
- Le type de format de données à traiter
- Le type d'alerte et de notifications

2. Documents applicables et de référence

- Le document de Spécification Technique du Besoin : STB.pdf
- Le document de présentation client : SmartLogger.pdf
- Le glossaire associé à la documentation : Glossaire.pdf

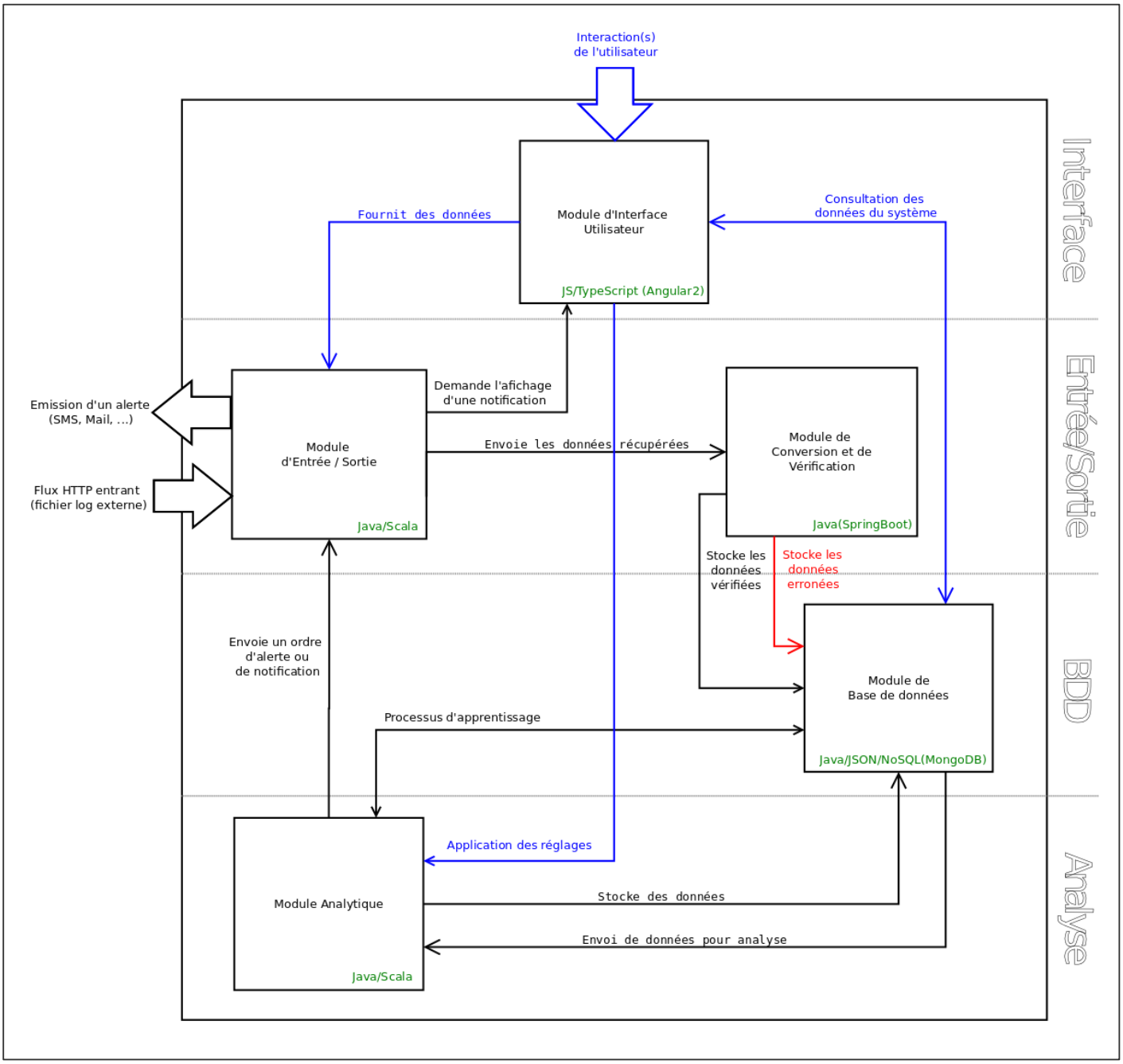
3. Configuration requise

- Périphérique hôte : Serveur local de l'entreprise cliente
- Système d'exploitation : OS de type Linux (version précise non définie)
- Produits et composants logiciels utilisés :

Nom	Type	Version
Shinken	Logiciel	2.4
Logstash	Logiciel	2.4
Kafka	Logiciel	0.9
Spark	Bibliothèque	2.0.1
MongoDB	SGBD	3.2.10
Angular 2	Framework	2.0.0
Spring Boot	Framework	1.4.2

4. Architecture statique

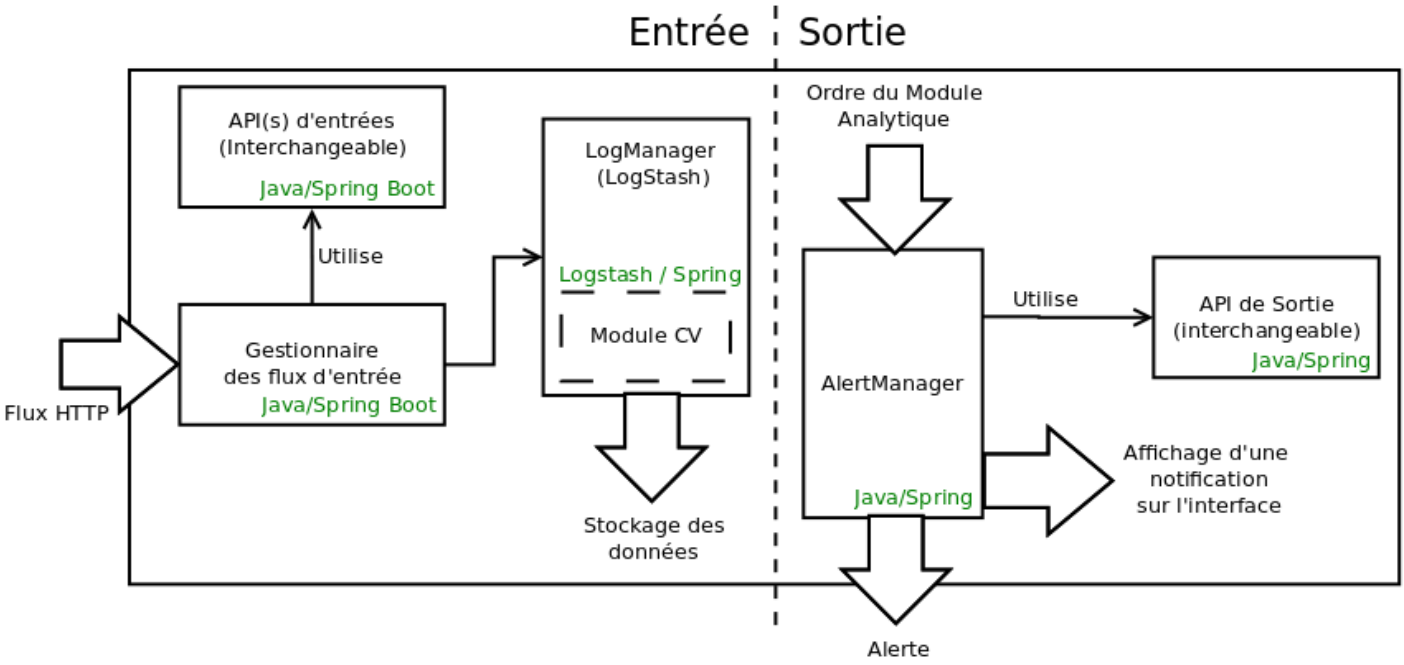
4.1. Structure principale du système



Le système s'organise en 5 modules distincts :

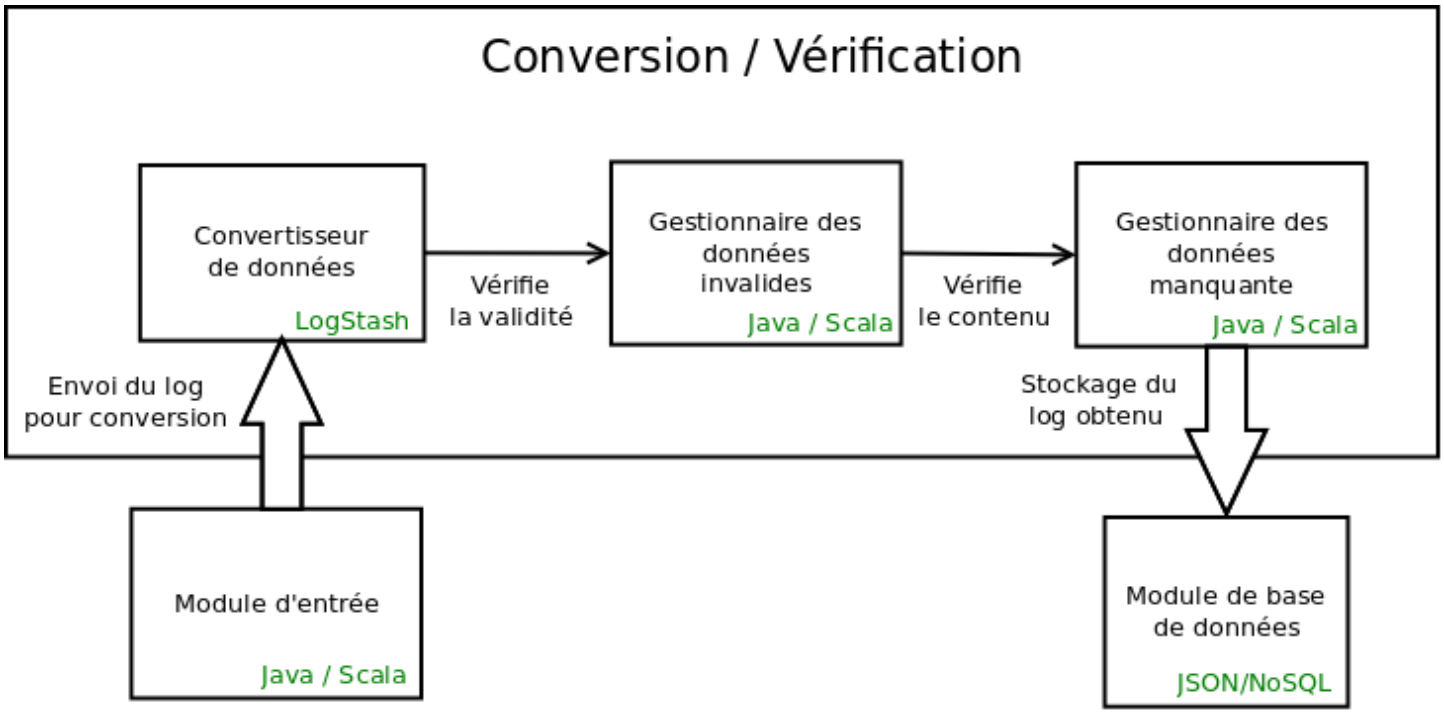
- Le module d'entrée/sortie (IO) : Permet toute interaction entre le système et tout acteur extérieur.
- Le module de conversion/vérification (CV) : Assure la validité des données avant leur exploitation par les autres modules du système.
- Le module d'interface utilisateur (UI) : Permet à un opérateur de communiquer avec le système : soit dans le but de visualiser les actions du système, soit pour émettre des ordres.
- Le module de gestion des données (GD) : Réalise le stockage permanent de toute donnée ayant été manipulée par le logiciel.
- Le module d'analyse (ML) : Constitue le modèle applicatif majeur du système, il réalise l'ensemble de ses analyses et prédictions sur les différents échantillons de données fournis.

4.2. Module ES – Entrée et Sortie du système



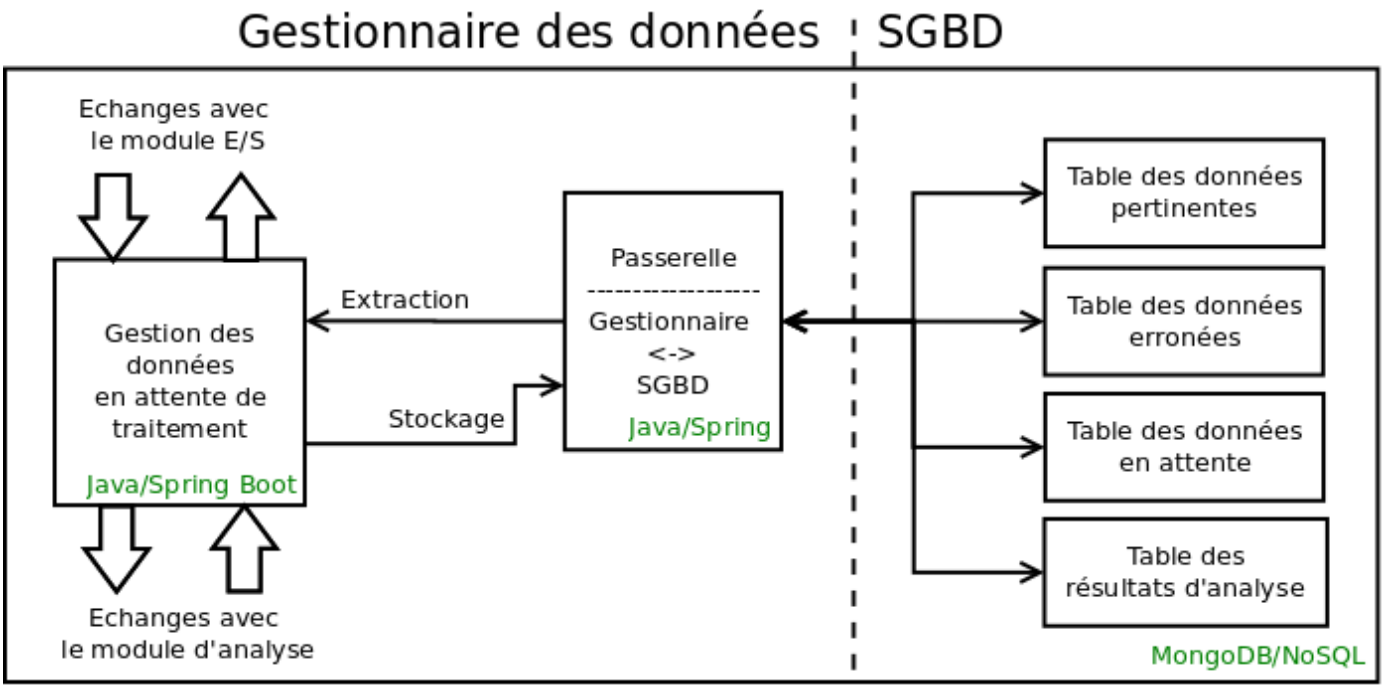
Rôle	Gère les opérations d'entrée / sortie
Propriétés	Sous-modules interchangeables, Adaptabilité à diverses API
Attributs	Unique module capable de communiquer avec un autre système extérieur
Services offerts	Entrée : Permet aux fournisseurs de données de les faire transiter jusqu'à la base de donnée (après une éventuelle vérification) Sortie : Relaye les alertes lancées par le système et affiche sur l'interface graphique les différentes notifications émises par le système
Dépendances	Aucune dépendance
Langage de programmation	Java
Procédé de développement	Utilisation du framework Spring Boot pour les opérations bas niveau d'entrée/sortie. Gestion des logs entrants grâce au logiciel logstash.
Taille	Faible
Complexité	Moyenne

4.3. Module CV – Conversion et Vérification des données



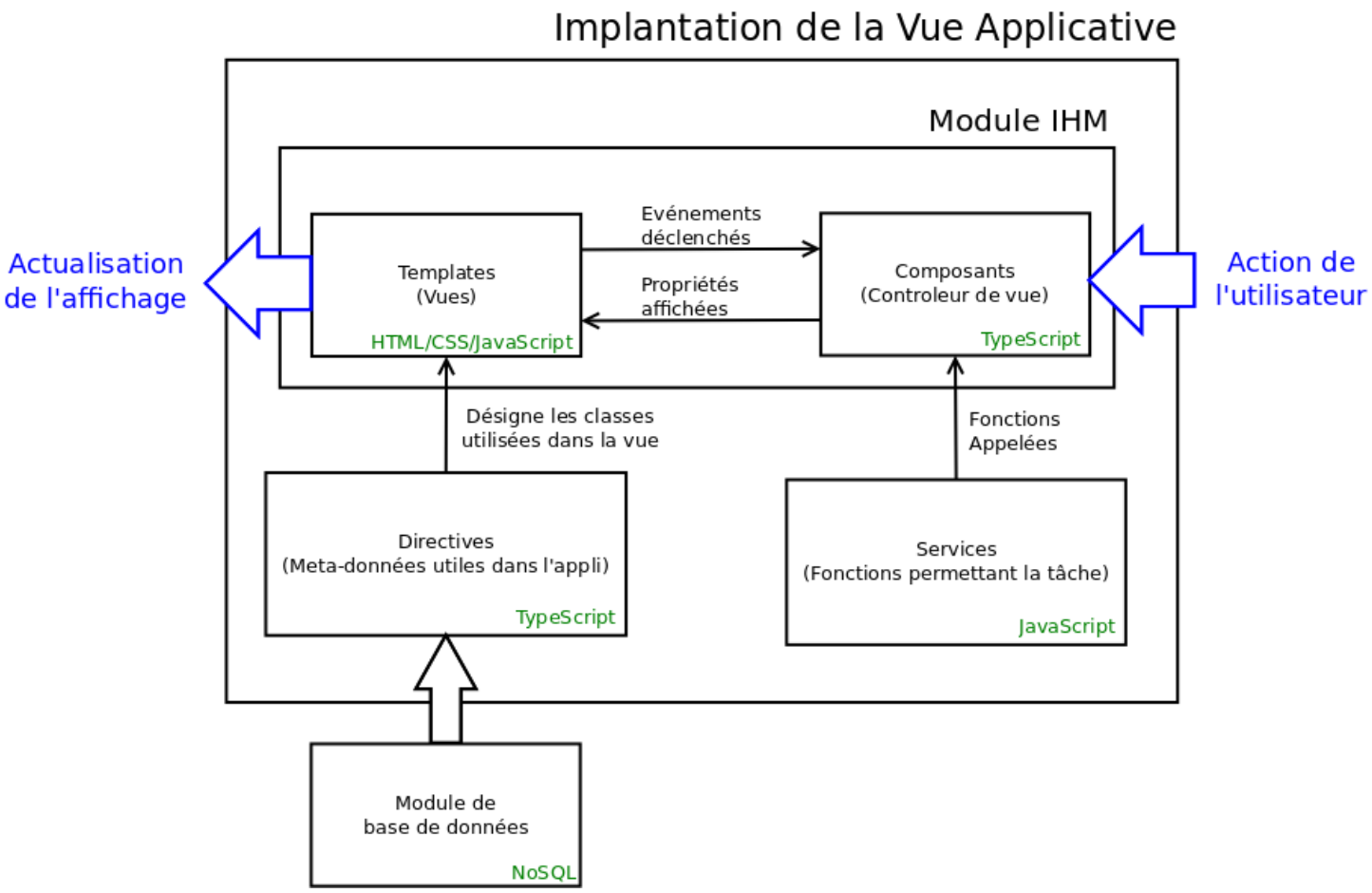
Rôle	Vérifie l'utilisabilité des données pour le module d'analyse
Propriétés	Utilisation optionnelle
Attributs	Permet d'uniformiser l'ensemble des informations issues du fichier en entrée, et d'en vérifier la cohérence avant leur utilisation par le sous-système d'analyse
Services offerts	Conversion des données dans un format unique Vérifie la validité des données (format et données corrompues) Gère l'absence de certaines données majeures
Dépendances	Aucune dépendance
Langage de programmation	Java
Procédé de développement	Conversion des logs en employant Logstash. Utilisation du framework Spring Boot combiné au langage Java, pour les divers modules à programmer.
Taille	Faible
Complexité	Moyenne

4.4. Module GD – Gestion des Données



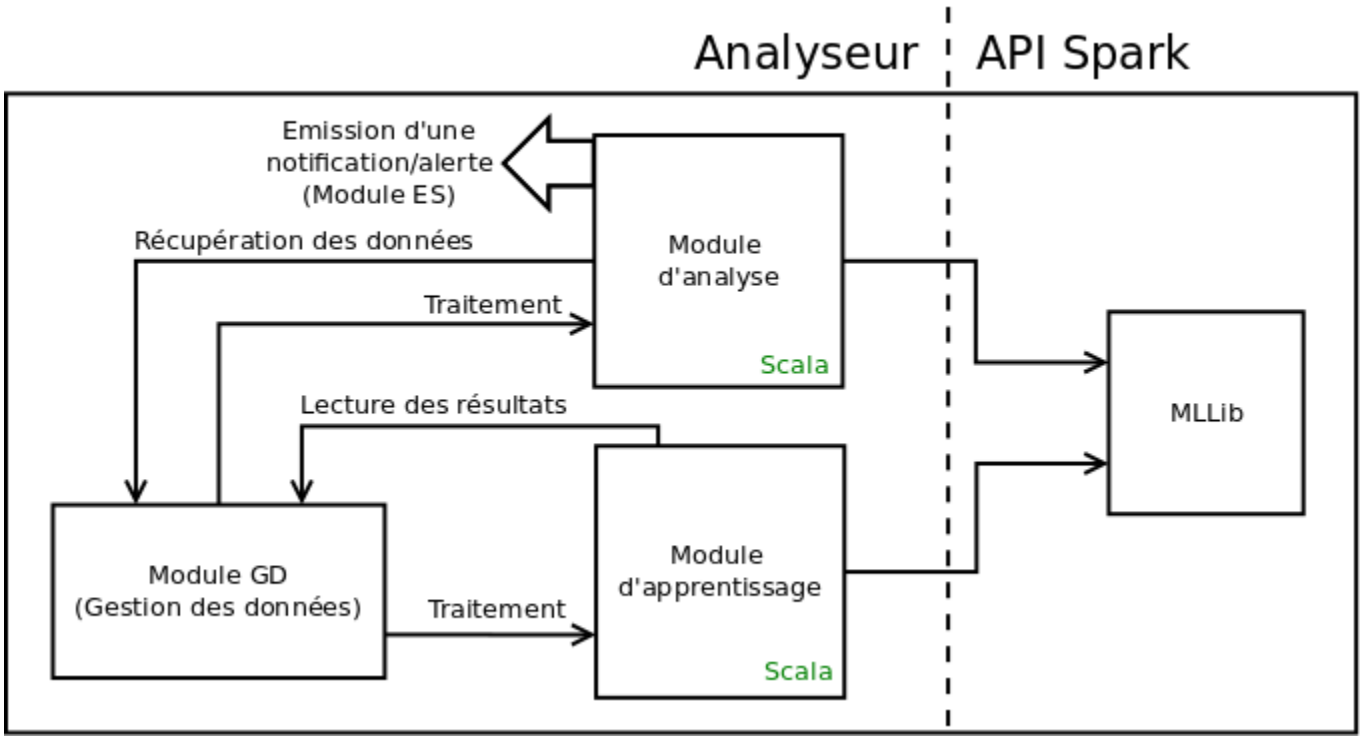
Rôle	Assure l'interaction avec la base de données
Propriétés	Interactions avec un SGBD distant
Attributs	Unique module capable de conserver des données persistantes. Communique avec un unique SGBD extérieur.
Services offerts	Consultation et réutilisation des données. Ordonnancement du traitement des données d'entrée. Stockage des données selon leur pertinence
Dépendances	Module Entrée/Sortie, module Analytique.
Langages de programmation	Java, NoSQL, JSON
Procédé de développement	Utilisation du SGBD MongoDB pour le stockage persistant des données. Emploi du framework Spring Boot pour les autres fonctionnalités
Taille	Grande
Complexité	Faible

4.5. Module IHM – Interface Homme-Machine



Rôle	Permet d'interagir avec un utilisateur humain
Propriétés	Seul moyen de consultation des données, Permet à un opérateur de fournir des données
Attributs	Informe sur l'état du système à intervalles réguliers
Services offerts	Consultation de la base de données. Interaction avec le module Analytique. Notification des informations systèmes mineures
Dépendances	Module Entrée et du module Analytique
Langages de programmation	JavaScript, TypeScript, JQuery
Procédé de développement	Utilisation du framework Angular2. Utilisation de la bibliothèque JQuery.
Taille	Moyenne
Complexité	Moyenne

4.6. Module Analytique

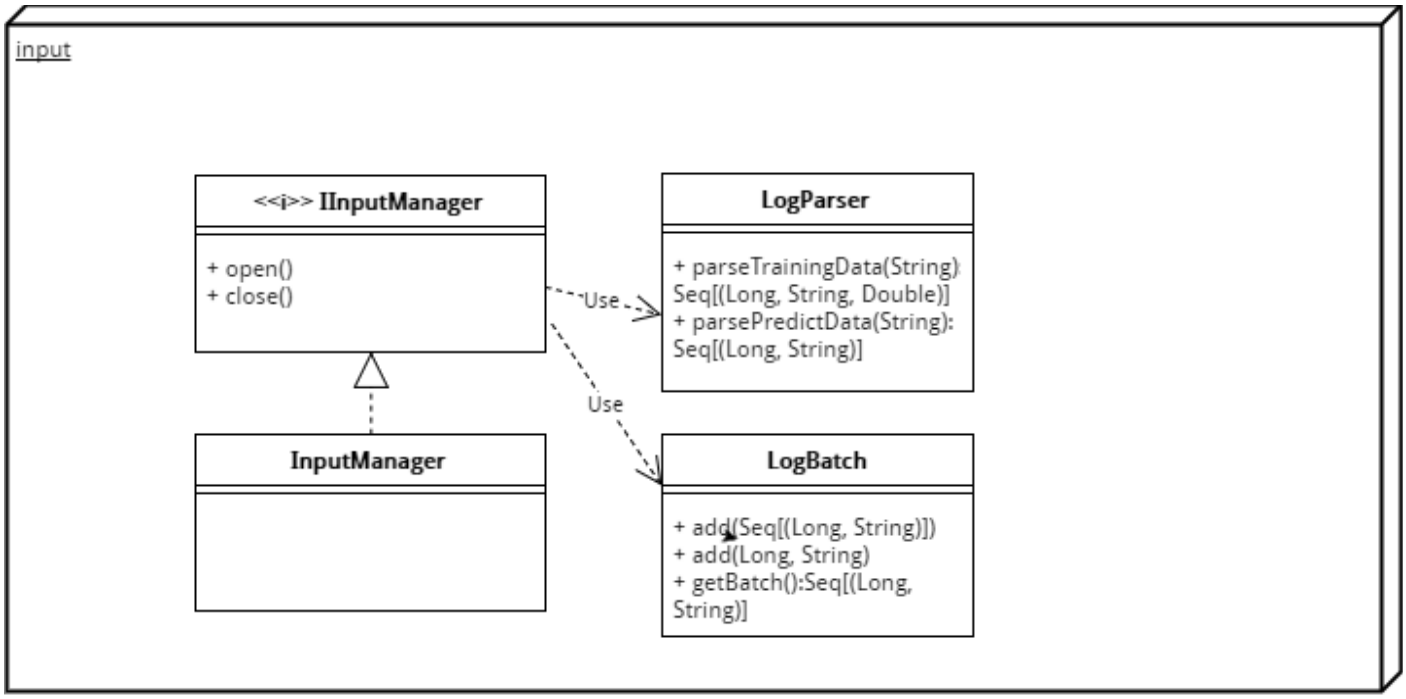


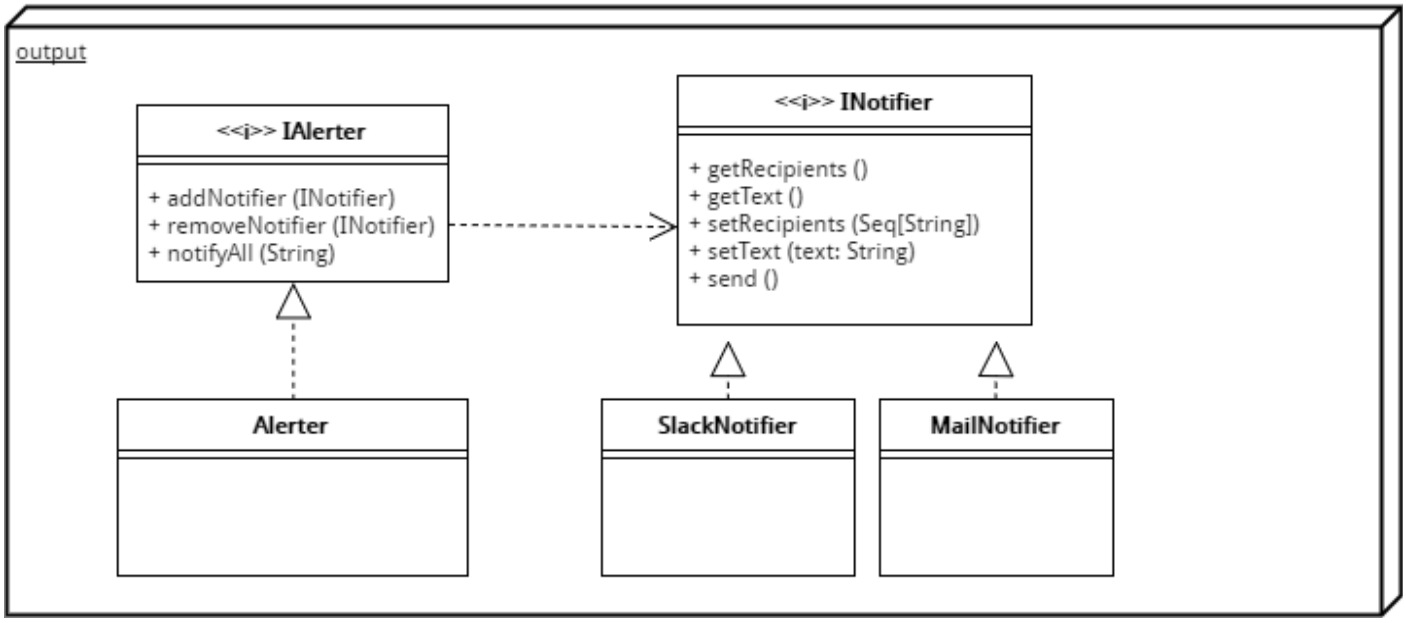
Rôle	Analyse les logs pour déceler d'éventuels dysfonctionnements
Propriétés	Peut apprendre et améliorer sa capacité d'analyse, gère la classification des logs entrants.
Attributs	Implémente des algorithmes de Machine Learning.
Services offerts	Classification des logs selon leur niveau de criticité. Envoi d'alerte via le module de Sortie, si une criticité est prédite. Envoi de notification sur l'IHM, en cas de détection de failles mineures.
Dépendances	Module de Base de Données
Langage de programmation	Scala
Procédé de développement	Framework Spark, API Spark
Taille	Faible
Complexité	Grande

5. Architecture statique détaillée

5.1. Module d'Entrée/Sortie

Schéma de principe





Description des interfaces/classes utilisées

5.1.1. Description du sous-module d'entrée

Le sous-module d'entrée se décompose en 2 parties :

- La première partie s'occupe de la récupération des données sous format texte, depuis une requête HTTP envoyée par un système externe, ou par un entraîneur du système. Cet tâche est assurée par le gestionnaire d'entrée **InputManager**.
- La seconde partie permet de lancer le traitement des données reçues, par le système. Cette tâche est supervisée par un objet de type **LogParser**, qui s'assure du traitement des données avant l'ordre d'exécution.

Comportements principaux

Fonction	Description
open	Cette fonction ouvre le serveur HTTP afin de recevoir les requêtes entrantes permettant de récupérer les logs. A l'intérieur, une fonction de manipulation appelée automatiquement, envoie les données au LogParser.
close	Ferme le serveur HTTP, libérant ainsi le port.

5.1.2. Description du sous-module de sortie

Le sous-module de sortie sera géré à l'aide d'un gestionnaire particulier (**Alerter**) qui aura pour but de récupérer les alertes émises par le module d'analyse, et d'utiliser l'avertisseur adapté à la criticité de l'alerte (implémenté par **Notifier**). Le gestionnaire se mettra alors en attente d'alertes à traiter, et déléguera les tâches d'alerte à son avertisseur.

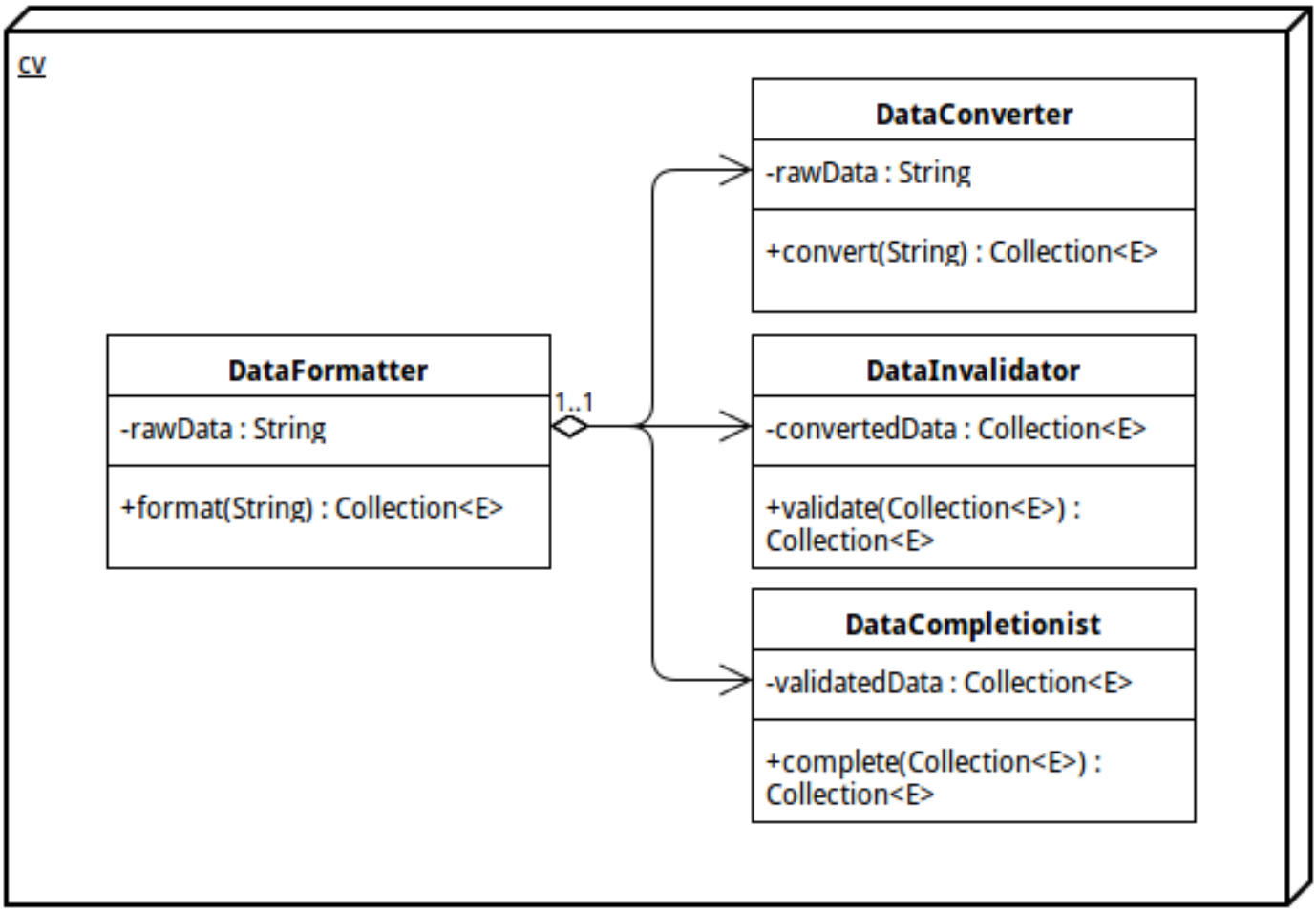
Plusieurs types d'avertisseurs seront alors prédéfinis (Slack, Mail), afin de couvrir le besoin concernant les types minimum d'alerte requis par le système.

Comportements principaux

Fonction	Description
addNotifier	Ajoute un Notifier au gestionnaire d'alertes.
removeNotifier	Enlève un Notifier au gestionnaire d'alertes.
notifyAll	Demande à chacun de ses Notifier d'envoyer un message d'alerte.
getRecipients	Permet de récuperer la liste des différents destinaires d'alerte.
getText	Permet de récuperer le message d'alerte à envoyer.
setRecipients	Permet de définir la liste des différents destinaires d'alerte.
setText	Permet de définir le message d'alerte à envoyer.
send	Envoie le message selon le type de Notifier.

5.2. Module de Conversion et Vérification

Schéma de principe



Description des interfaces/classes utilisées

Le mécanisme du module, qui est de réaliser la conversion et la vérification de données entrantes brutes, est assuré par la classe **DataFormatter**, qui réalise ces opérations à l'aide de 3 sous-composants :

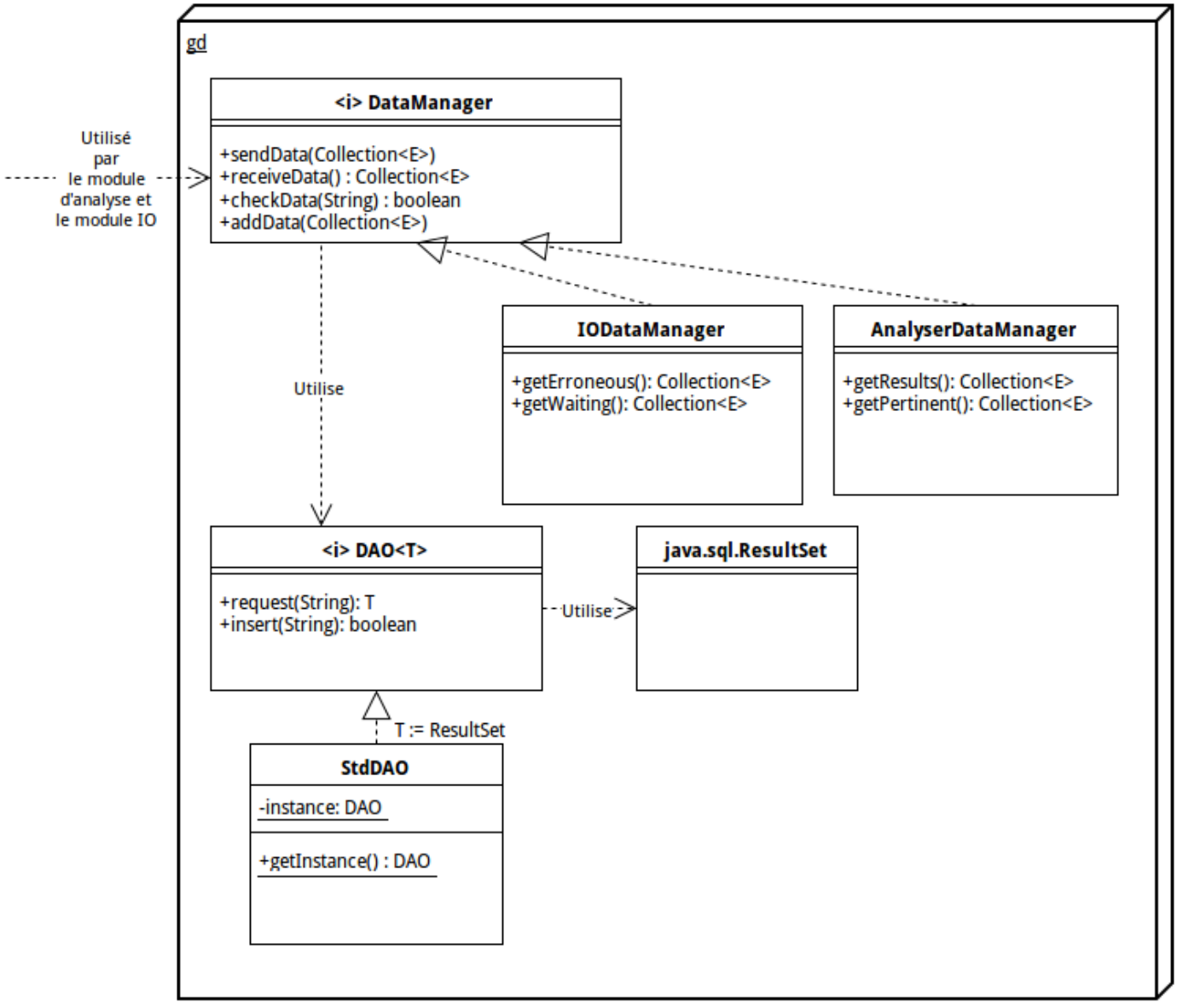
- **DataConverter**, qui réalise la conversion préliminaire. Il s'agit de transformer une chaîne de caractères entrée, en collection de données « utilisables ».
- **DataInvalidator**, qui filtre la collection fraîchement créée, afin d'en supprimer les données non pertinentes pour le système.
- **DataCompletionist**, qui permet de compléter des échantillons de données avec des informations supplémentaires, afin d'augmenter leur pertinence.

Comportement principaux

Fonction	Description
convert	Convertit la chaîne de caractère fournie en collection de données utilisable.
validate	Filtre la collection de données, afin de la faire correspondre aux normes requises par le processus d'analyse.
complete	Complète les données manquantes de la collection pour améliorer l'efficacité du futur traitement.
format	Lance l'opération de conversion/vérification complète, en faisant appel séquentiellement aux 3 méthodes précédentes.

5.3. Module de Gestion de Données

Schéma de principe



Description des interfaces/classes utilisées

5.3.1. Description du rôle de DataManager

DataManager définit le comportement du gestionnaire des interactions entre le système et la base de données distantes. Il définit ainsi les opérations disponibles permettant de manipuler le stockage des données. Cet objet est implémenté en 2 variantes différentes permettant au gestionnaire de s'adapter selon le contexte d'utilisation :

- IODataManager : Gestionnaire des opérations utilisables pour le module IO.
- AnalyserDataManager : Gestionnaire des opérations utilisables pour le module d'analyse.

Comportements principaux

Fonction	Description
sendData	Envoie une collection de données fournie, en tant que contenu à ajouter, au sein de la base de données distante.
receiveData	Renvoie le prochain échantillon de données (sous forme de collection), présent dans la base de données.
checkData	Indique si la représentation textuelle (sous forme de chaîne de caractère) des données entrées est valide pour une utilisation éventuelle, dans la base de données.
addData	Ajoute une nouvelle collection de données, dans le contenu de la base existante

Comportements spécialisés

Fonction	Description
getErroneous	Renvoie sous forme de collection, l'intégralité des logs erronés.
getWaiting	Renvoie sous forme de collection, l'intégralité des logs en attente de traitement.
getResults	Retourne l'ensemble des logs qui ont déjà été traités par l'analyseur.
getPertinent	Renvoie l'ensemble des logs qualifiés comme pertinent suite au processus de validation, qui ont déjà été traités par l'analyseur.

5.3.2. Description du rôle de DAO

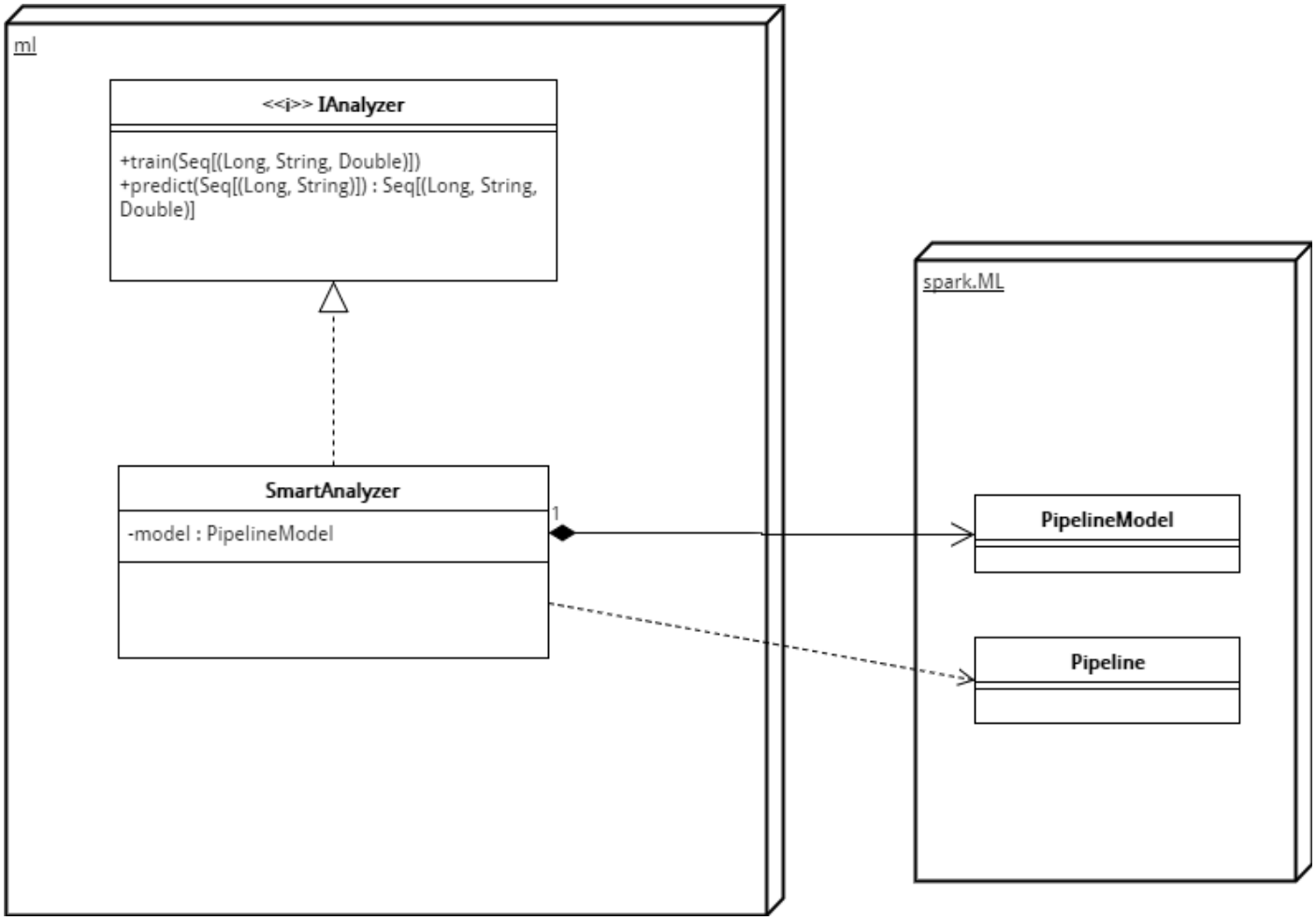
L'objet DAO sert ici à effectuer les opérations de bas-niveau permettant de communiquer avec la base de données distante (création de requêtes, évaluation de requêtes, connexion à la base distante, etc.)

Son implémentation StdDAO, permet ici de communiquer avec le type de base de données sélectionné pour être utilisée ici : une base de type NoSQL.

Fonction	Description
request	Renvoie, sous forme d'ensemble de lignes, le résultat de l'évaluation de la requête fournie en paramètre.
insert	Exécute la requête d'insertion fournie en paramètre et renvoie une indication de succès de la procédure.

5.4. Module Analytique

Schéma de principe



Description des interfaces/classes utilisées

Un objet de type Analyzer permet d'implanter intégralement le mécanisme d'analyse et d'apprentissage de traitement des données. Ce procédé est réalisé grâce à l'exploitation des fonctionnalités offertes par l'API Spark, via une délégation des opérations de manipulation et de calcul sur les données.

Chaque Analyseur peut donc employer un algorithme précis, défini au sein du package spark.ML.

Nous envisageons ici, de créer un seul type d'analyseur, qui sera capable de s'adapter à tous les algorithmes accessibles depuis le package spark.ML grâce à l'utilisation d'un Pipeline.

Comportements principaux

Fonction	Description
train	Permet d'établir un nouveau modèle de prédication/analyse, correspondant au type d'algorithme employé par cet objet.
predict	Applique l'algorithme de prédiction inhérent à l'objet sur l'échantillon de données fourni.