ARCHITECTURE DU LOGICIEL

Version :	0.3
Date :	14/12/2016
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	17/11/2016	Création du document
0.2	01/12/2016	Augmentation du contenu de la partie Architecture Statique
0.3	14/12/2016	Ajout de l'Architecture Statique détaillée
0.3.1	09/01/17	Correctifs

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

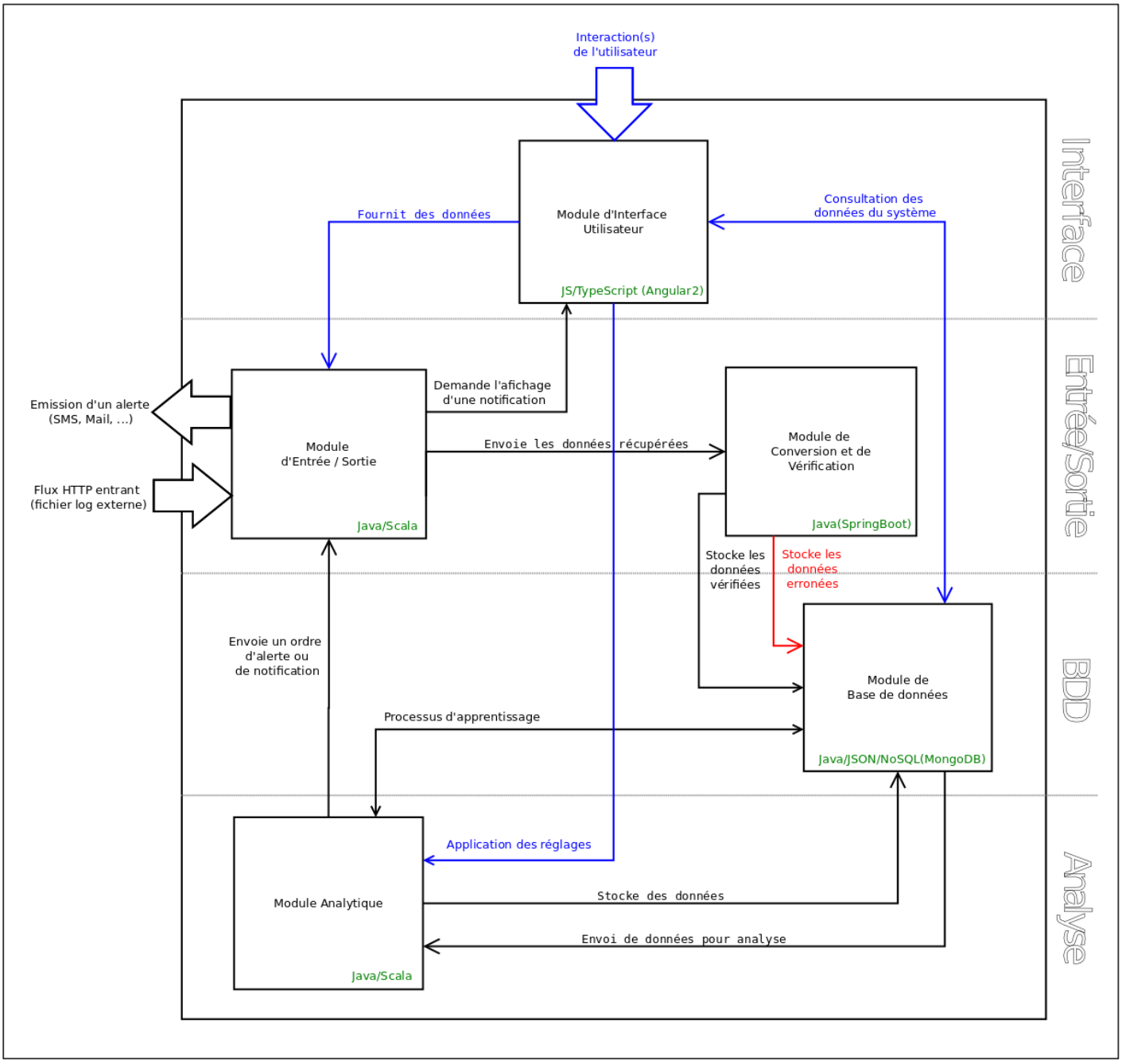
3. Configuration requise

- 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

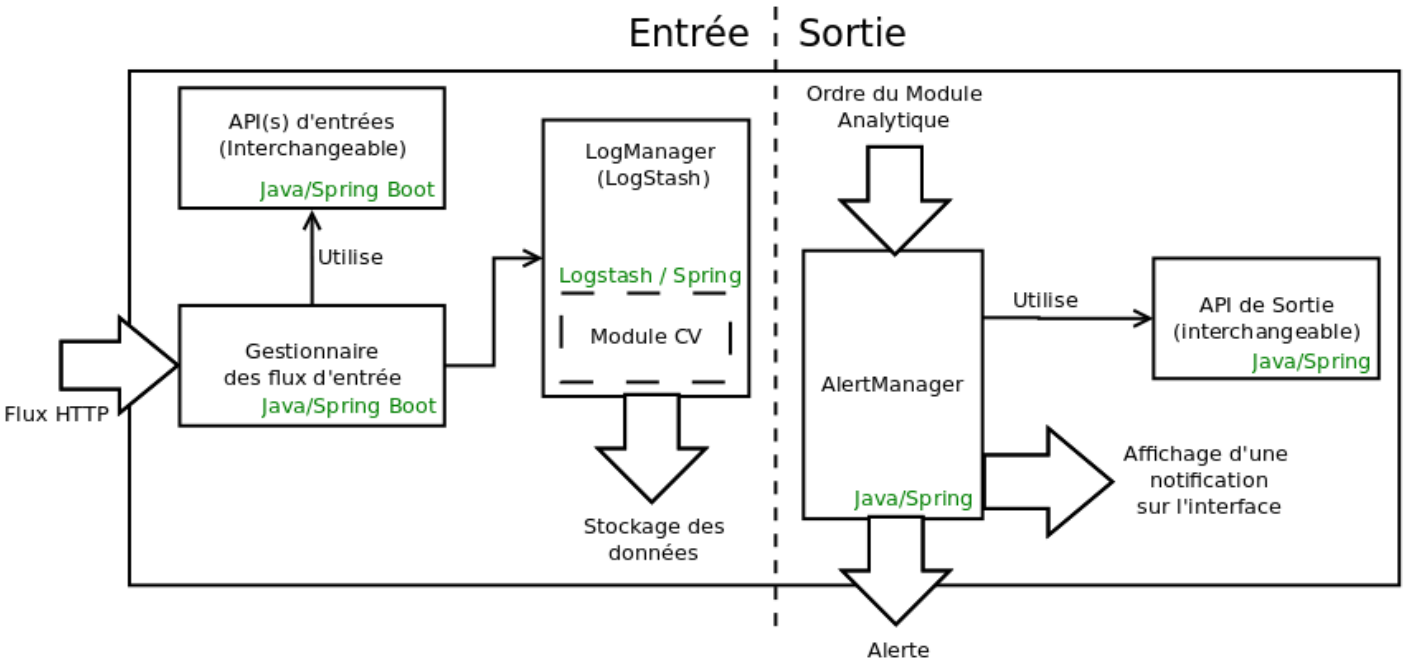


Le système s'organise en 5 modules distincts :

- Le module d'entrée/sortie (ES) : 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 base de données (DB) : 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

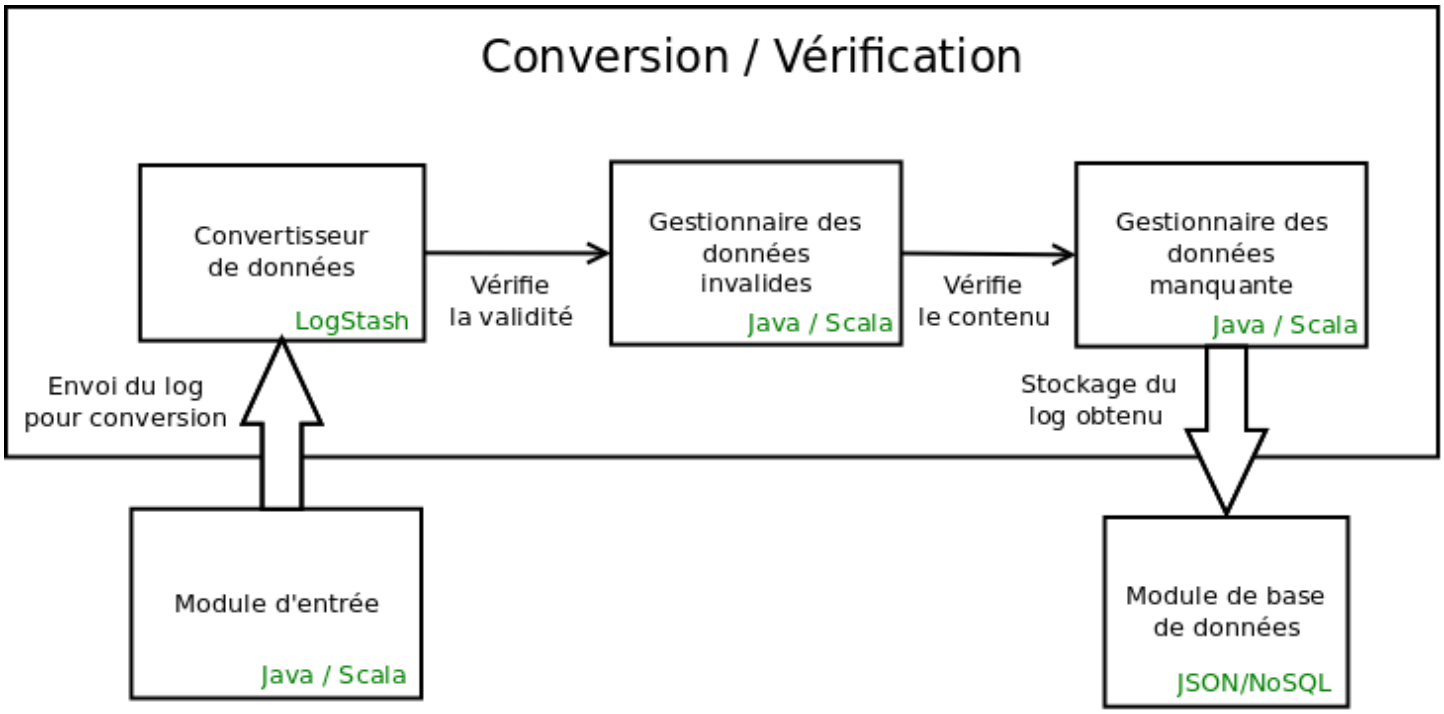
données fournis.

4.2. Module ES – Entrée et Sortie du système



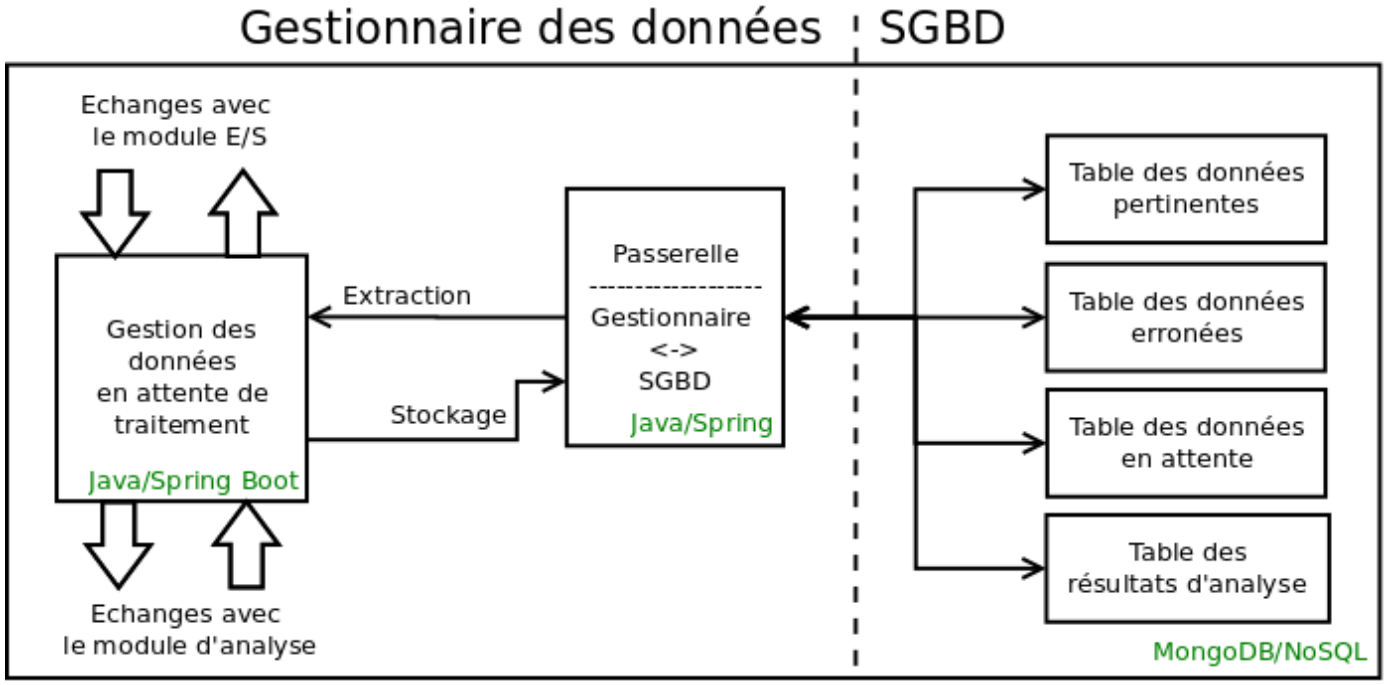
Rôle	Gère les opérations d'entrée / sortie
Propriétés	Sous-modules interchangeables, Adaptabilité à diverses API
Attributs	Unique module capable de communiquer avec un autre système extérieur
Services offerts	Entrée : Permet aux fournisseurs de données de les faire transiter jusqu'à la base de donnée (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



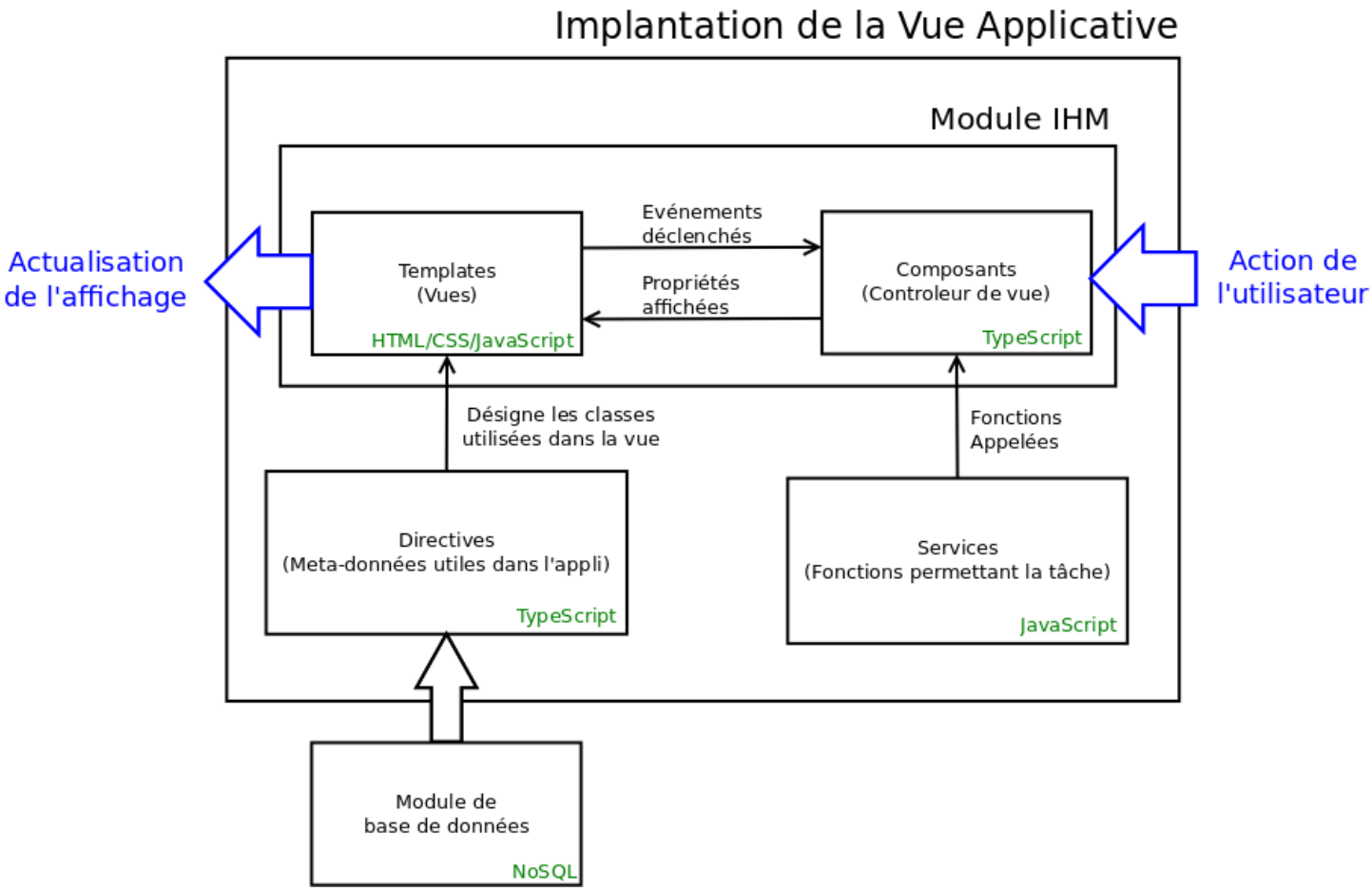
Rôle	Vérifie l'utilisabilité des données pour le module d'analyse
Propriétés	Utilisation optionnelle
Attributs	Permet d'uniformiser l'ensemble des informations issues du fichier en entrée, et d'en vérifier la cohérence avant leur utilisation par le sous-système d'analyse
Services offerts	Conversion des 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



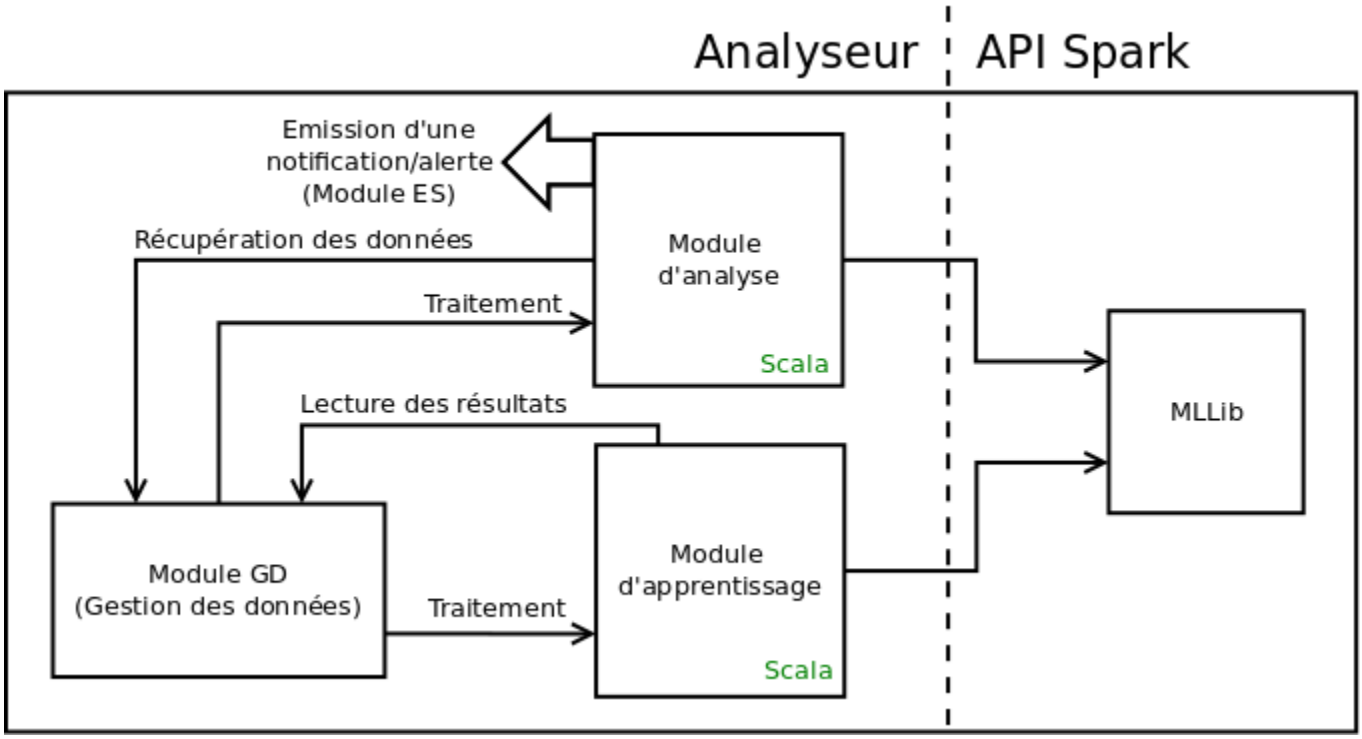
Rôle	Assure l'interaction avec la base de données
Propriétés	Interactions avec un SGBD distant
Attributs	Unique module capable de conserver des données persistantes. Communique avec un unique SGBD extérieur.
Services offerts	Consultation et réutilisation des données. Ordonnancement du traitement des 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

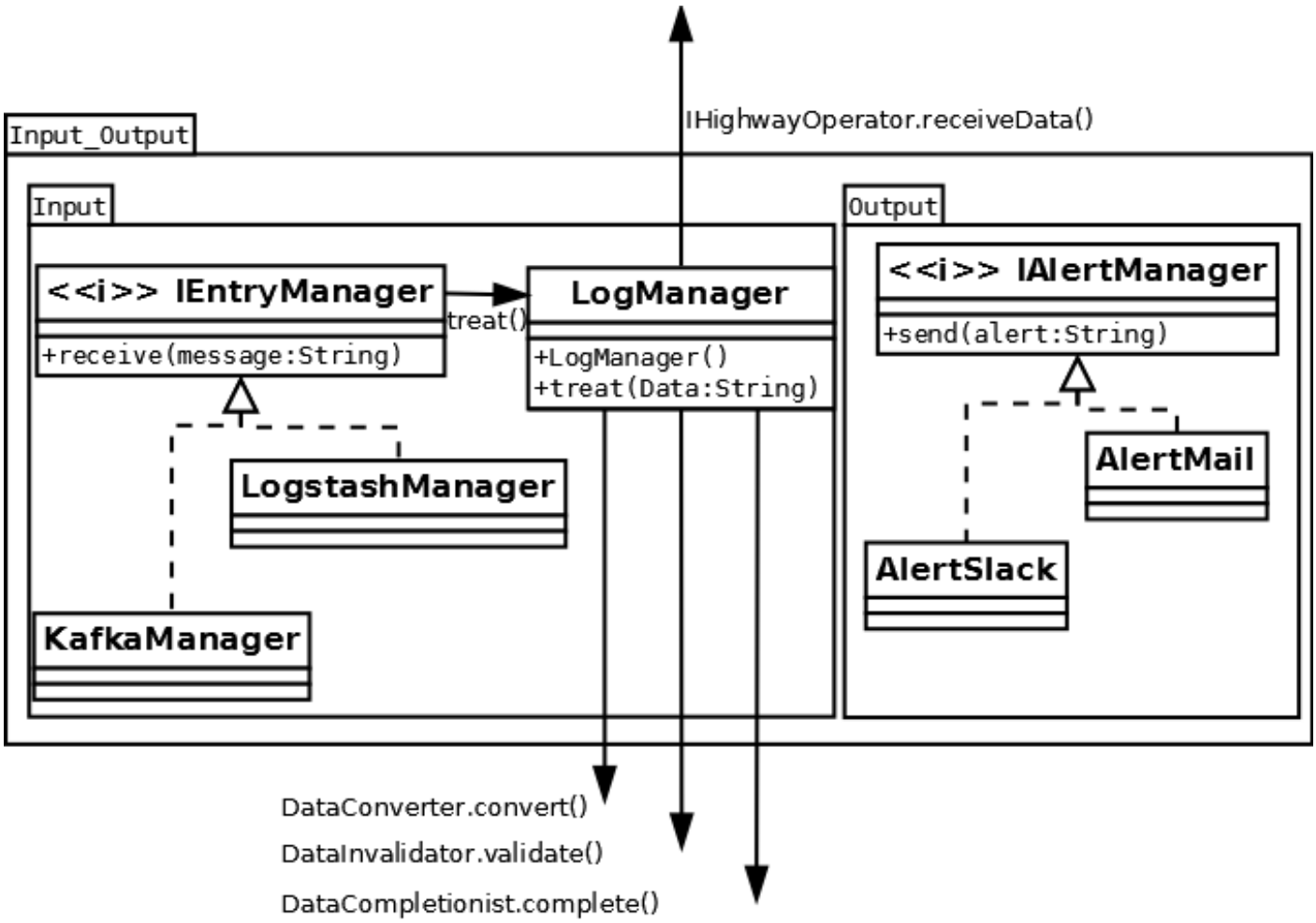
4.6. Module Analytique



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



5.1.1. Description de IEntryManager

Fonction	Description
receive(message:String)	La méthode *receive* est une fonction générique appelé automatiquement lorsque le logiciel reçoit des logs d'un serveur. Selon le type de logiciel envoyant les logs, la classe fille changera et aura un traitement différent puis enverra les données à la classe LogManager.

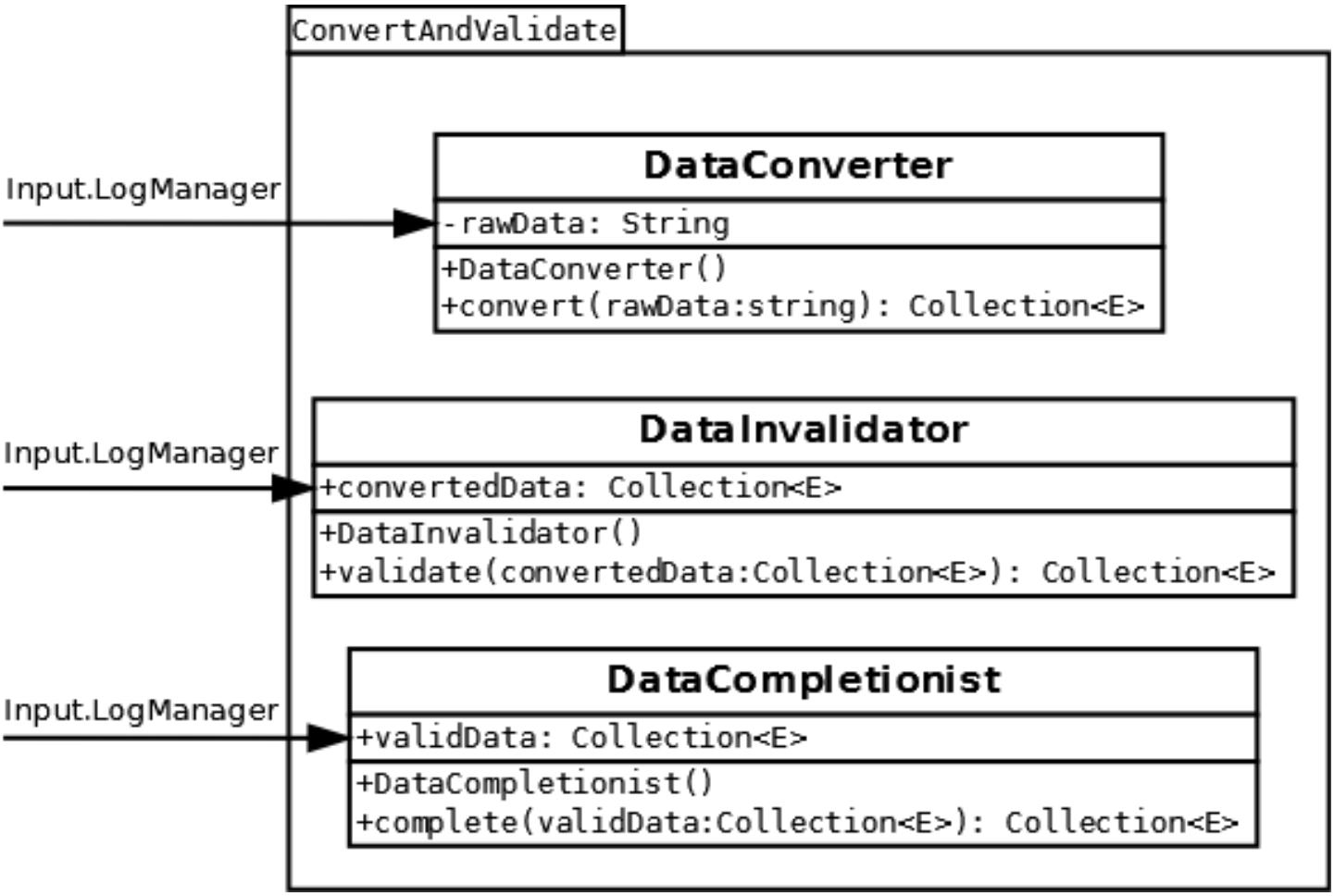
5.1.2. Description de LogManager

Fonction	Description
treat(data:String)	La méthode *treat* est une fonction qui aura pour but d'effectuer un pré-traitement sur les données reçues afin de les normaliser, en effectuant des appels consécutifs aux classes du module ConvertAndValidate avant enregistrement dans la base de données, effectuée ensuite via l'appel à IHighwayOperator.receiveData() .

5.1.3. Description de IAlertManager

Fonction	Description
send(alert :String)	La méthode *send* est une fonction générique déclenchée par un appel du module Analytique, qui utilise une de ses classes filles, qui aura un traitement plus spécialisé à l'aide d'une bibliothèque choisie.

5.2. Module de Conversion et Vérification



5.2.1. Description de DataConverter

Fonction	Description
convert(rawData:String)	La méthode *convert* est une fonction prenant une String en paramètre et qui a pour but de la convertir en donnée traitable et la mettre dans une Collection afin de traiter ensuite les différentes données dans une Collection pour plus de rapidité.

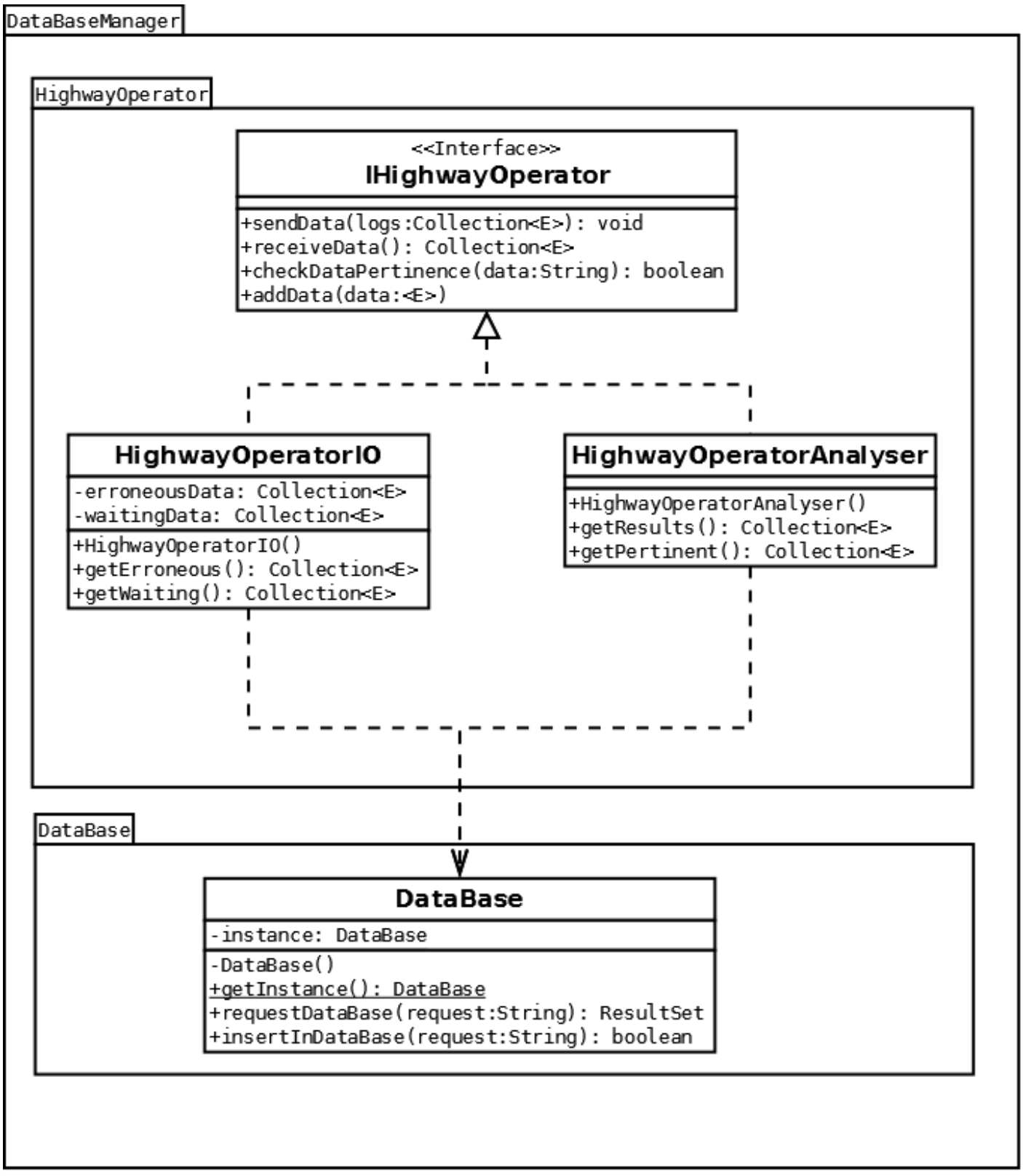
5.2.2. Description de DataInvalidator

Fonction	Description
validate(convertedData:Collection<E>)	La méthode *validate* est une fonction prenant une Collection en paramètre et qui aura pour but de filtrer les éléments correspondants aux normes correspondants aux standards du module Analytique.

5.2.3. Description de DataCompletionist

Fonction	Description
complete(validData:Collection<E>)	La méthode *complete* est une fonction prenant une Collection en paramètre et qui aura pour but de compléter les données incomplètes de la Collection pour poursuivre un traitement plus efficace.

5.3. Module de Gestion de Données



5.3.1. Description de IHighwayOperator

Fonction	Description
sendData(logs:Collection<E>)	La méthode *sendData* est une fonction qui prends une collection de logs en paramètre et qui aura pour but de les envoyer dans la base de données.
receiveData():Collection<E>	La méthode *receiveData* est une fonction sans paramètre qui aura pour but de recevoir des données depuis la base.
checkDataPertinence(data:String) : boolean	La méthode *checkDataPertinence* est une fonction qui prends des données en entrée, sous forme de chaine de caractère, et qui renvoie un booléen renseignant si ces données sont valides.
addData(data:E)	La méthode *addData* est une fonction qui prends des données en paramètre et qui aura pour but de les ajouter à la base de données.

5.3.2. Description de HighwayOperatorIO

Fonction	Description
getErroneous() : Collection<E>	La méthode *getErroneous* est une fonction sans paramètre qui renvoie sous forme de collection les logs qui sont erronés.
getWaiting():Collection<E>	La méthode *getErroneous* est une fonction sans paramètre qui renvoie sous forme de collection les logs qui sont en attente.

5.3.3. Description de HighwayOperatorAnalyser

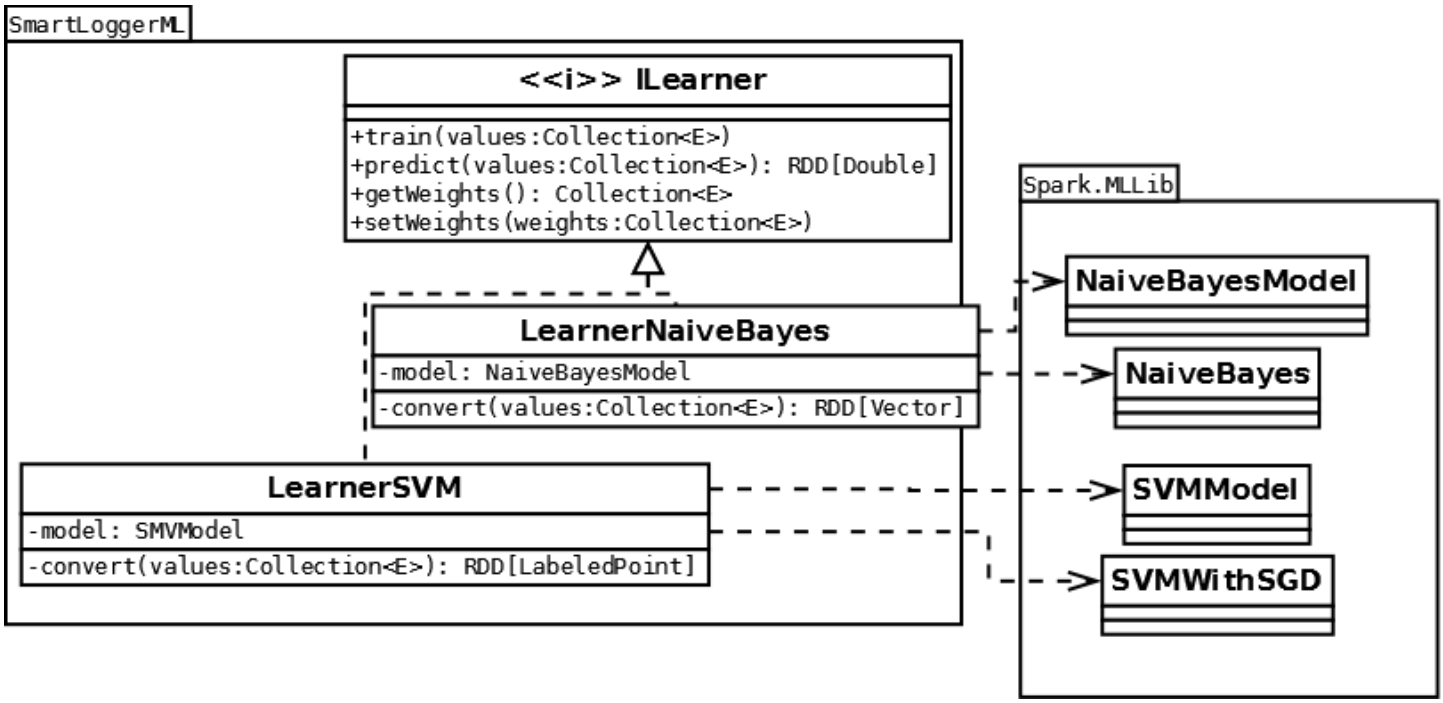
Fonction	Description
getResults():Collection<E>	La méthode *getResults* est une fonction sans paramètre qui retourne l'ensemble des logs qui ont été traités par l'analyseur.
getPertinent():Collection<E>	La méthode *getPertinent* est une fonction sans paramètre qui retourne l'ensemble des logs qui ont été traités par l'analyseur et qui sont

	pertinent.

5.3.1. Description de DataBase

Fonction	Description
getInstance()	La méthode *getInstance* est une fonction statique qui renvoie l'instance de la base de donnée, la créant si besoin.
requestDataBase(request:String):ResultSet	La méthode *requestDataBase* est une fonction qui prends en paramètre une requête sous forme de chaîne de caractère, et qui aura pour but de retourner le résultat de cette requête.
insertInDataBase(request:String):boolean	La méthode *InsertInDataBase* est une fonction qui prends en paramètre une requête d'insertion sous forme de chaîne de caractère, et qui aura pour but de retourner un booléen indiquant la réussite de cette requête.

5.4. Module Analytique



5.4.1. Description de ILearner

Fonction	Description
train(values:Collection<E>)	La méthode *train* a pour but de définir une méthode commune à tous les algorithmes de machine learning afin de créer un modèle correspondant à cet algorithme.
predict(values:Collection):RDD[Double]	La méthode *predict* a pour but de définir une méthode commune à tous les algorithmes de machine learning afin de pouvoir appliquer les données sur n'importe lequel des modèles.
getWeights():Collection<E>	La méthode *train* a pour but de définir une méthode commune à tous les algorithmes de machine learning afin de récupérer les paramètres qui équilibrent le modèle.
setWeights(weights:Collection<E>)	La méthode *train* a pour but de définir une méthode commune à tous les algorithmes de machine learning afin de redéfinir les paramètres qui équilibrent le modèle.

5.4.2. Description de LearnerNaiveBayes et LearnerSVM

Fonction	Description
convert(values:Collection<E>):RDD[?]	Cette méthode *convert* sert à convertir les données brutes en texte en les rendant exploitables par les algorithmes de machine learning dans une structure Resilient Distributed Dataset (RDD) qui permet un accès plus rapide, dans des types de données différents selon l'algorithme.