ARCHITECTURE DU LOGICIEL

Version :	1.2
Date :	13/05/2017
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
1.1	30/03/17	Actualisation du document
1.2	13/05/17	Actualisation du document concernant la 3ème itération.

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 ou l'envoi de mails.
- 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 :

- Les types d'algorithme disponibles, employé par l'analyseur
- Le type de format de données à traiter
- Les mécanismes d'alerte
- etc.

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

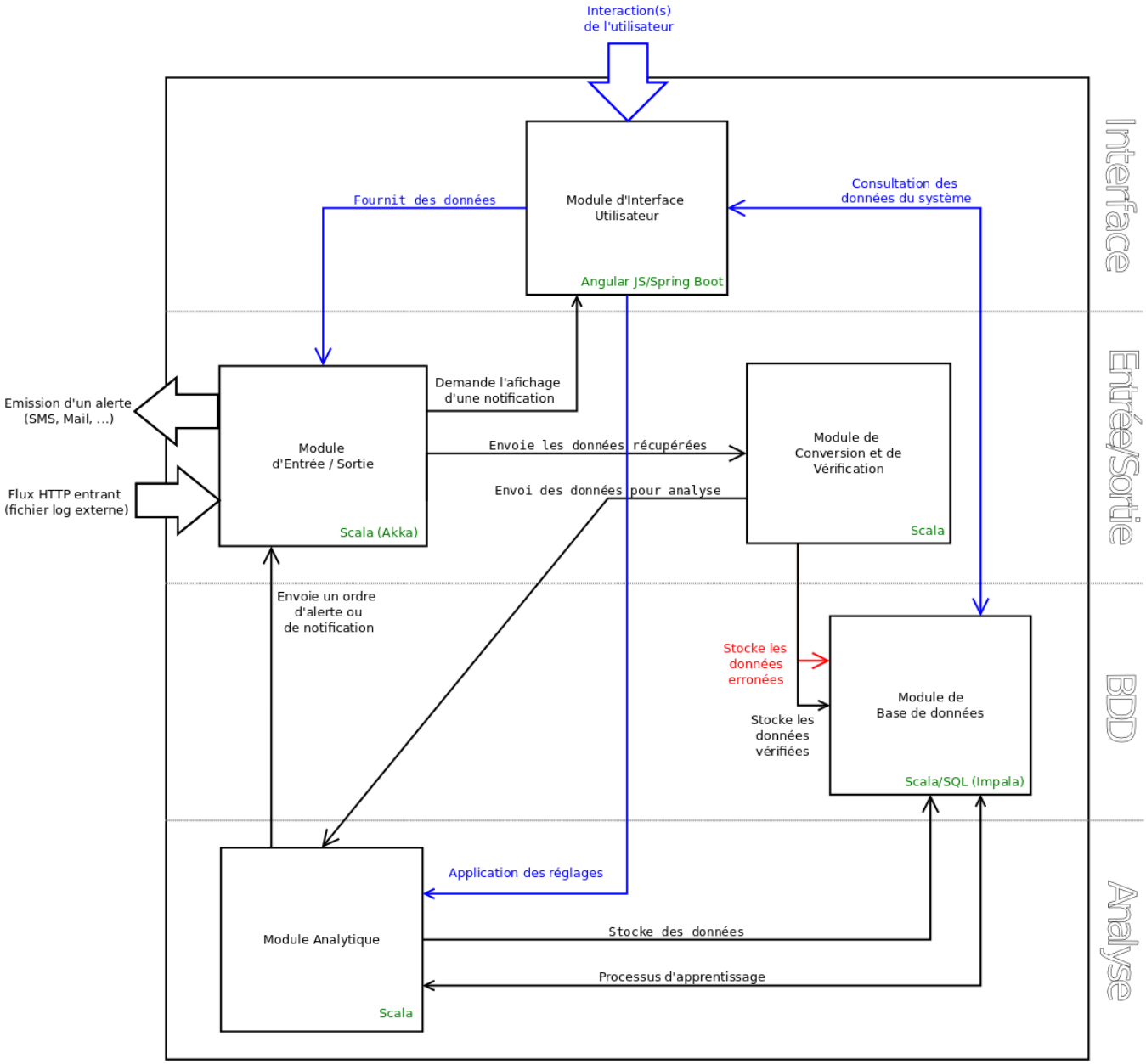
3. Configuration requise

- 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
Spark	Bibliothèque	2.0.1
Javax.mail	Bibliothèque	1.4.7
MongoDB	SGBD	3.2.10
Angular JS	Framework	2.0.0
Spring Boot	Framework	1.4.2
Akka HTTP	Toolkit	10.0.3
Scala-Slack	API	0.4.0

4. Architecture statique

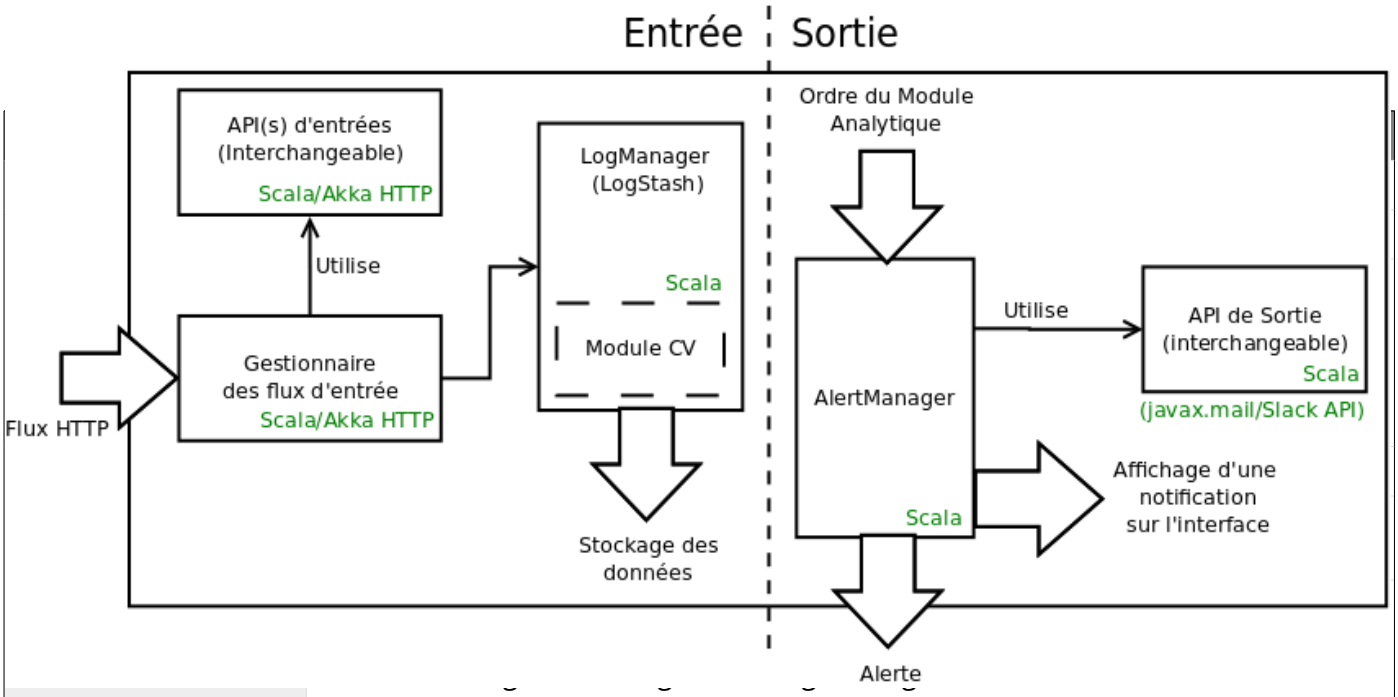
4.1. Structure principale du système



Le système s'organise en 5 modules distincts :

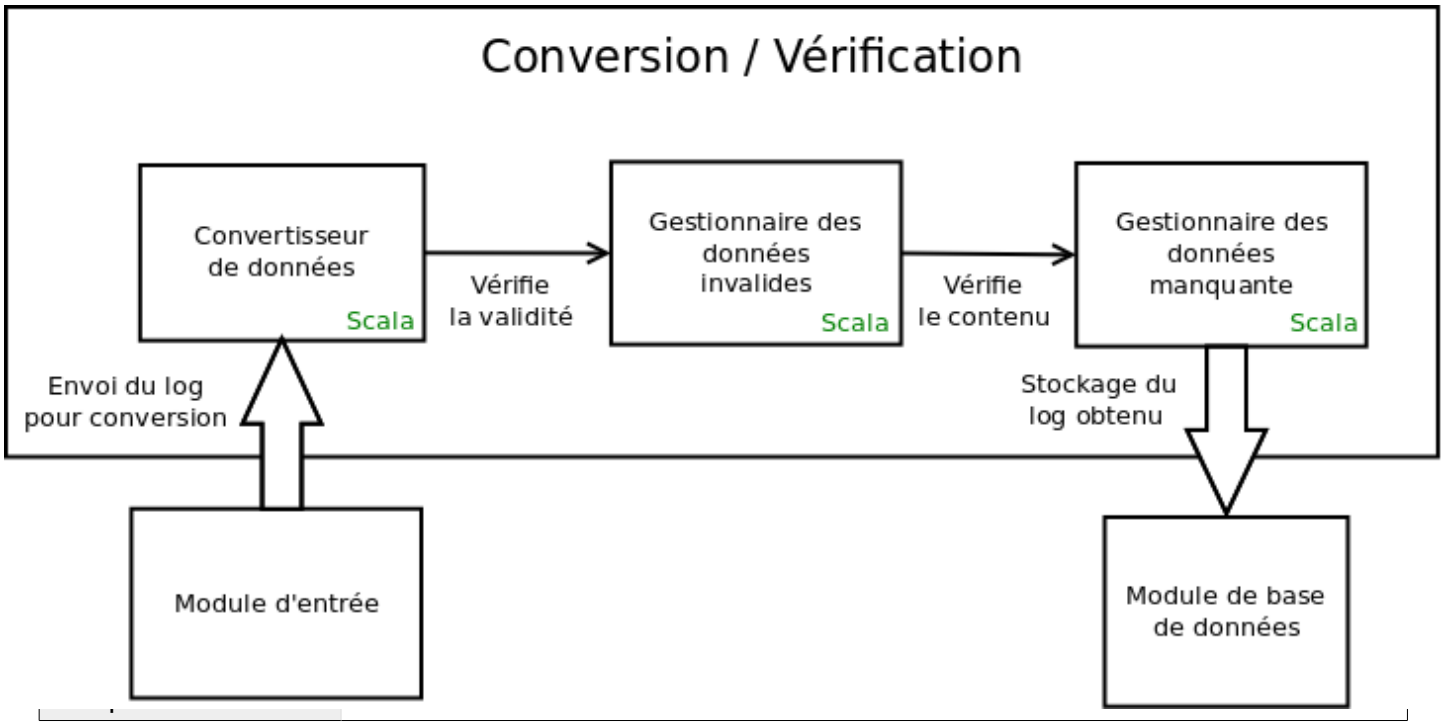
- Le module d'entrée/sortie (IO) : Permet toute interaction entre le système et tout acteur extérieur.
- Le module de conversion/vérification (CV) : Assure la validité des données avant leur exploitation par les autres modules du système.
- Le module d'interface utilisateur (UI) : Permet à un opérateur 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

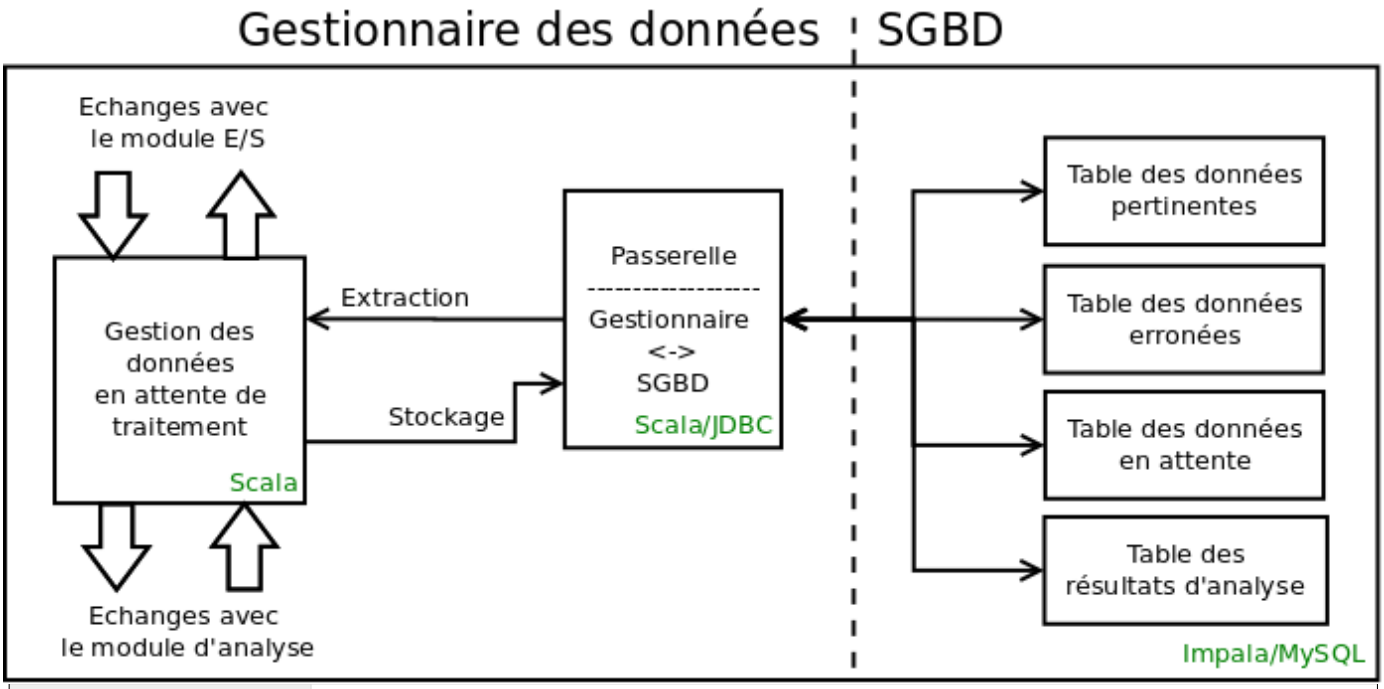


Taille	Faible
Complexité	Moyenne

4.3. Module CV – Conversion et Vérification des données

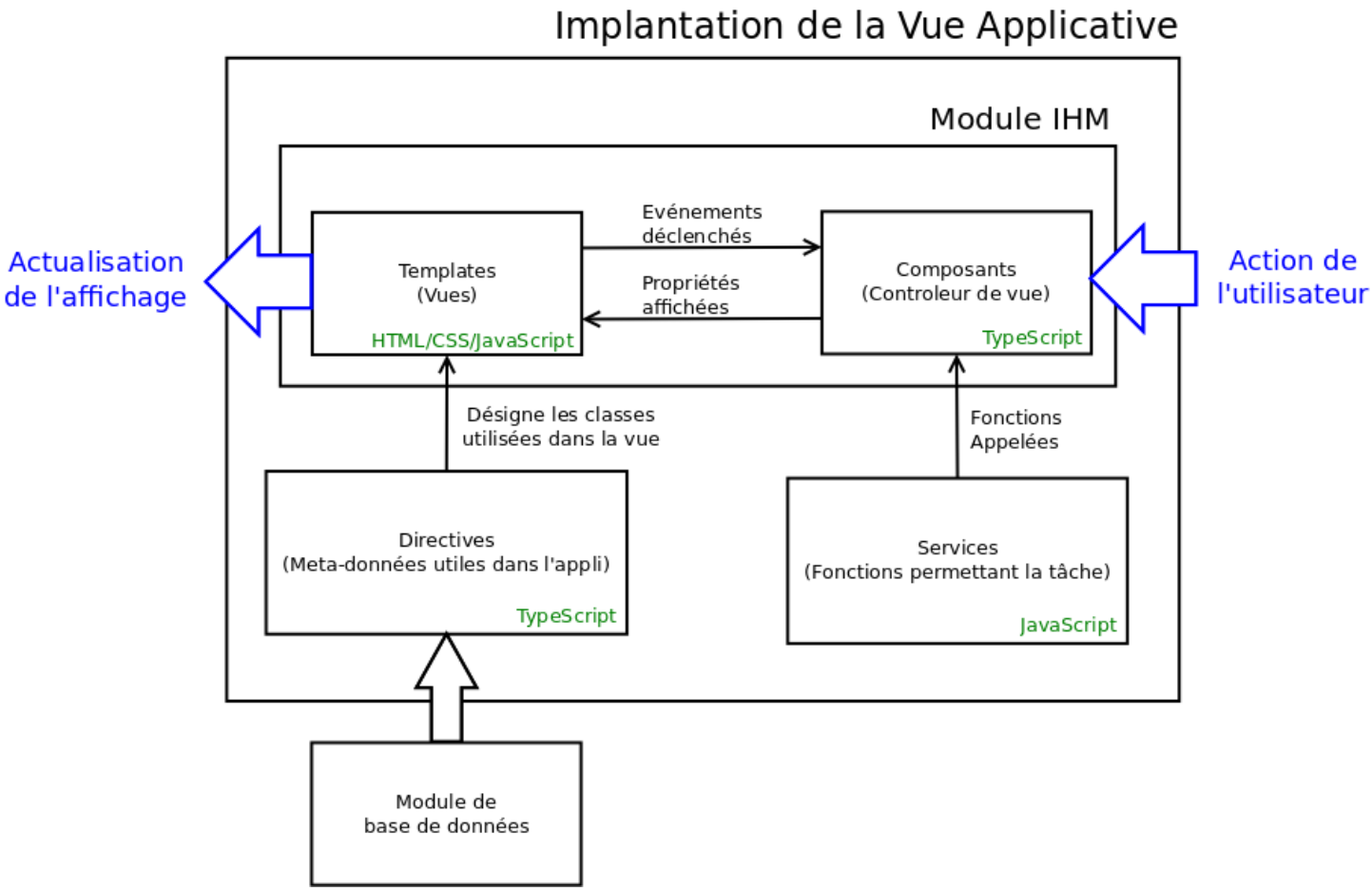


4.4. Module GD – Gestion des Données

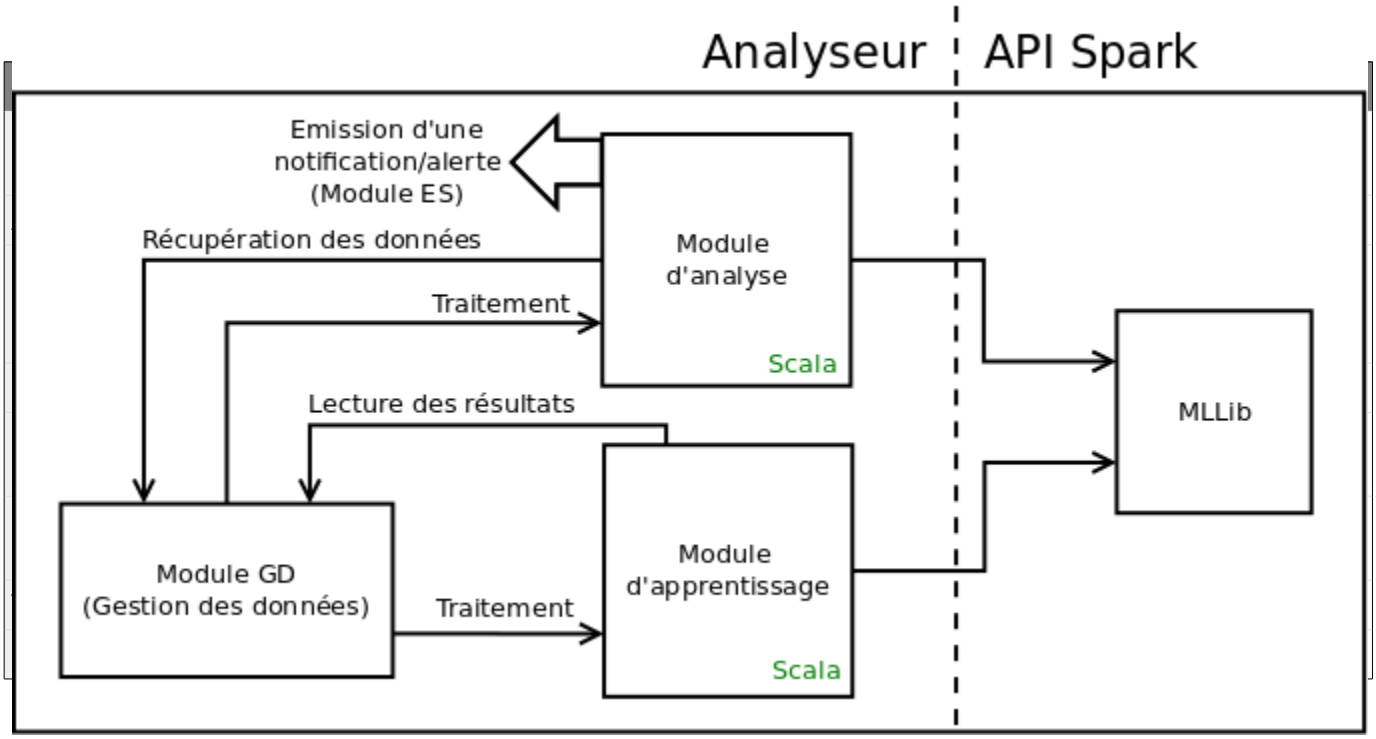


Complexité	Moyenne

4.5. Module IHM – Interface Homme-Machine



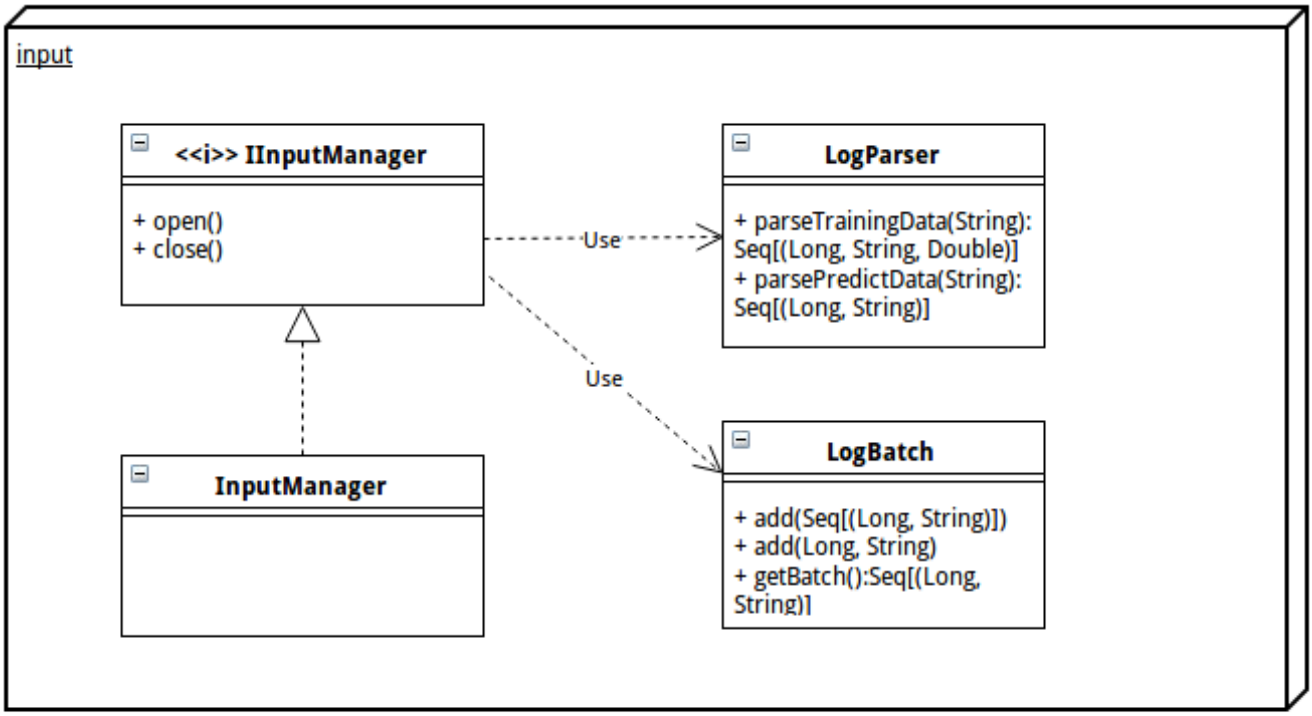
4.6. Module Analytique

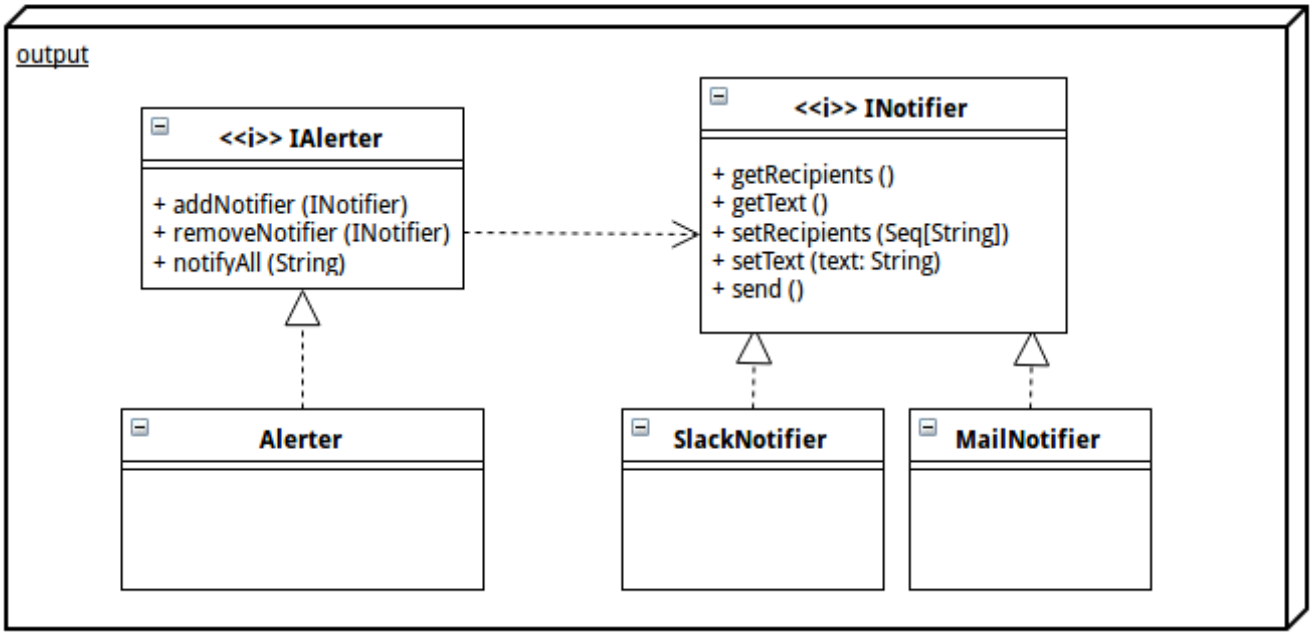


5. Architecture statique détaillée

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.

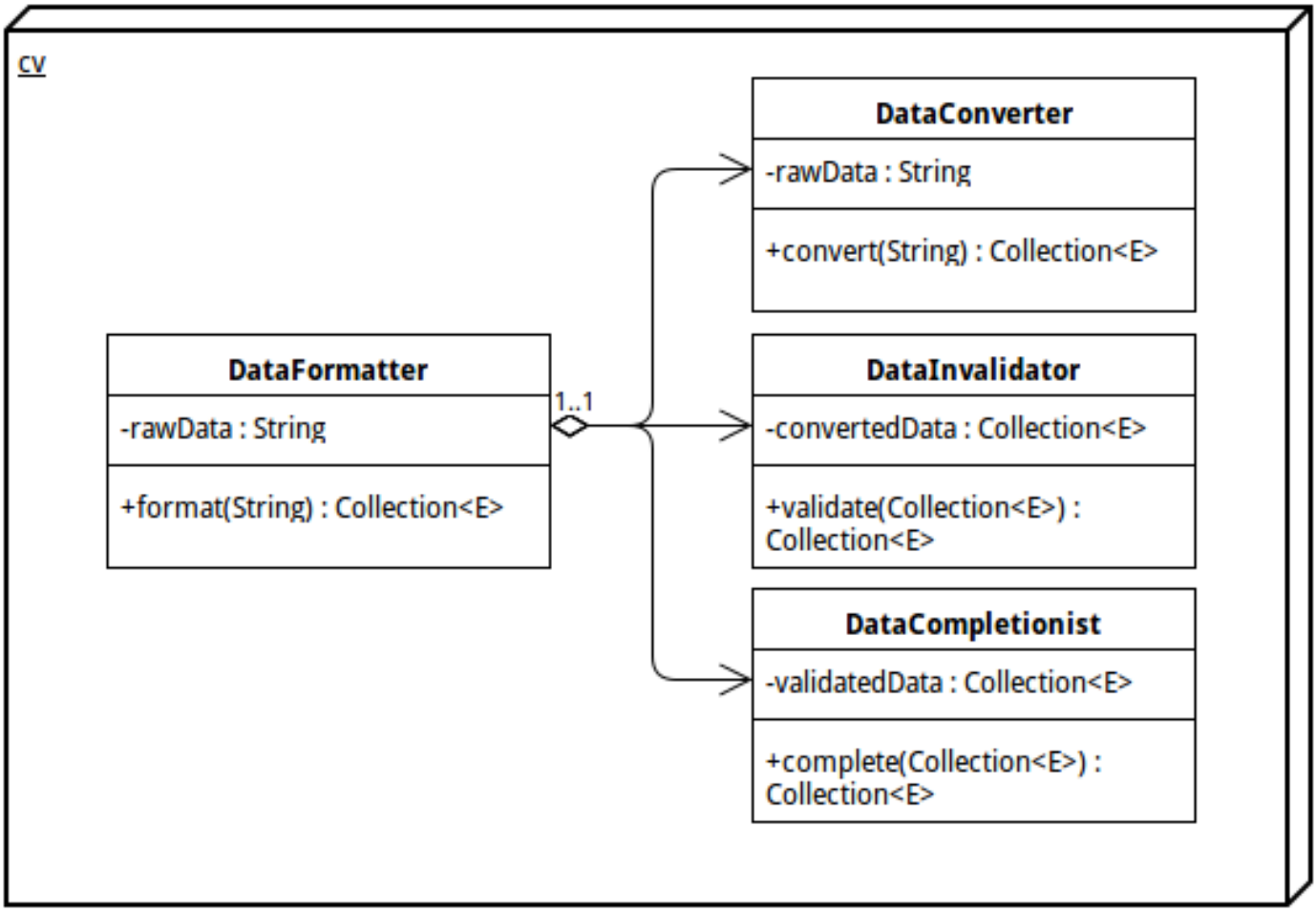
Plusieurs types d'avertisseurs seront alors prédéfinis (Slack, Mail), afin de couvrir le besoin concernant les types minimum d'alerte requis par le système.

Comportements principaux

Fonction	Description
addNotifier	Ajoute un Notifier au gestionnaire d'alertes.
removeNotifier	Enlève un Notifier au gestionnaire d'alertes.
notifyAll	Demande à chacun des Notifier enregistrés d'envoyer un message d'alerte.
getRecipients	Permet de récupérer la liste des différents destinataires d'alerte.
getText	Permet de récupérer le message d'alerte à envoyer.
setRecipients	Permet de définir la liste des différents destinataires de l'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 :

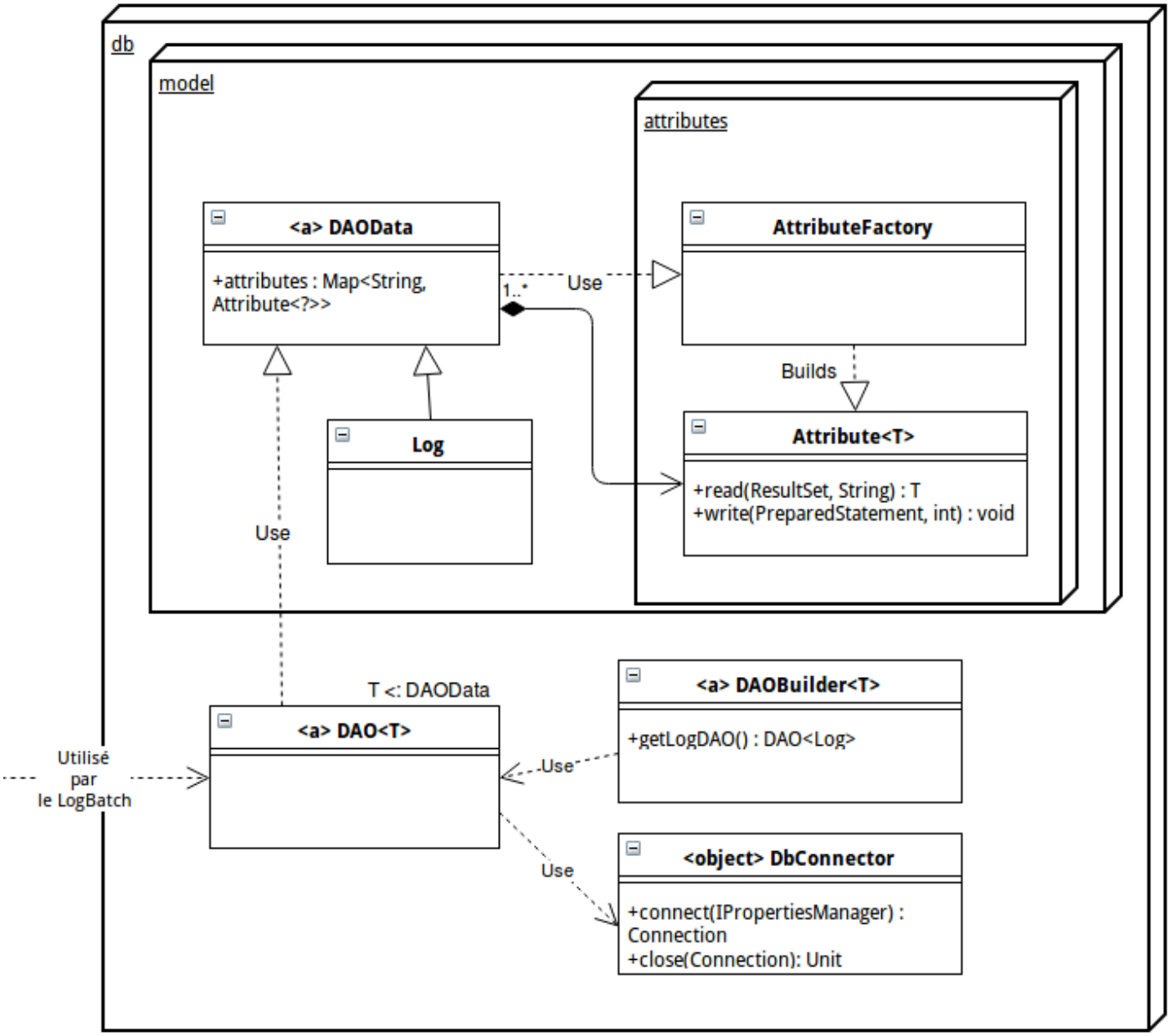
- **DataConverter**, qui réalise la conversion préliminaire. Il s'agit de transformer une chaîne de caractères entrée, en collection de données « utilisables ».
- **DataInvalidator**, qui filtre la collection fraîchement créée, afin d'en supprimer les données non pertinentes pour le système.
- **DataCompletionist**,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



La gestion des interactions avec la base de données se décompose en trois phases :

- La définition d'une entité utilisable
- La gestion des opérations entre le système et le SGBD distant
- La gestion de la connexion permettant d'effectuer ces opérations

Description des interfaces/classes utilisées

5.3.1. Description de DAOData et Attribute

Une instance de DAOData permet de modéliser une entité. Une entité représente un modèle dont les éléments qui le composent sont des valeurs simples, pouvant s'exprimer sous forme de couples clé-valeur.

Ces couples sont alors représentés par des instances de la classe Attribute. Cela permet, au moyen de leurs opérations de lecture/écriture, de gérer implicitement les différences entre le type SQL et le type Java correspondant, lors de leur manipulation par le DAO. La création des attributs des attributs est assurée par une AttributeFactory, qui permet de définir des attributs en fonction du type de SGBD ciblé.

5.3.2. Description du rôle de DAO

Un DAO sert ici à effectuer les opérations standards (méthodes CRUD) permettant de communiquer avec la base de données distante. Le DAO manipule des entités afin de pouvoir généraliser ces opérations à tout type de données. Ainsi chaque implémentation de DAO, permet de réaliser ces opérations pour un type de SGBD donné.

La instanciation d'un DAO est alors réalisée par un DAOBuilder, ce qui permet d'adapter le DAO à l'entité souhaitée.

5.3.3. Description du rôle de DbConnector

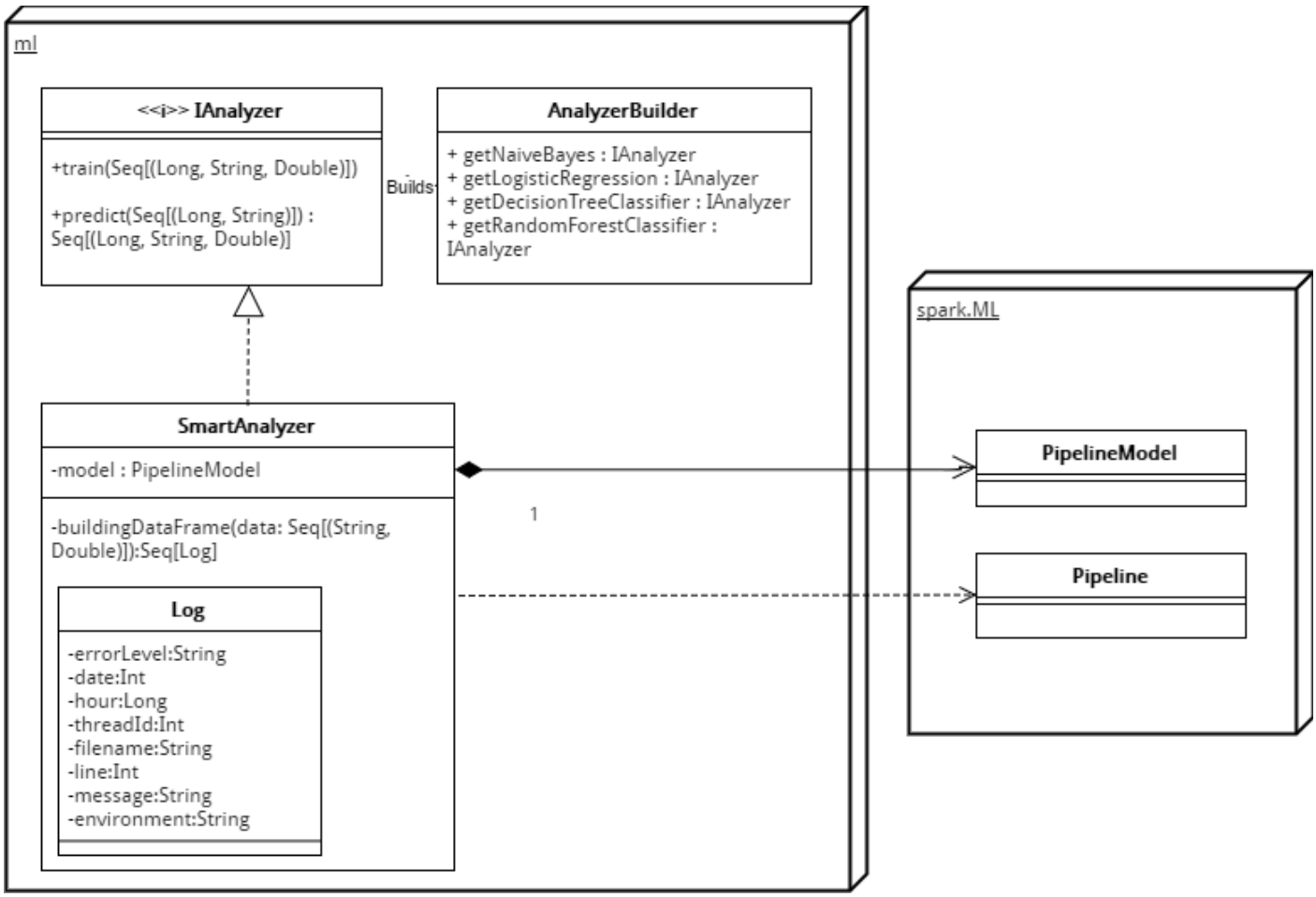
L'objet DbConnector permet d'établir une connexion avec le SGBD distant. Les informations d'authentification utilisées pour réaliser correctement cette tâche sont situées dans un fichier .properties externe.

Comportements principaux

Fonction	Description
connect	Établit une nouvelle connexion avec le SGBD distant.
close	Ferme une connexion établie précédemment

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. Le choix de l'algorithme utilisé est défini ultérieurement grâce à l'emploi de l'objet Pipeline du package spark.ML. La classe SmartAnalyzer sert alors d'implémentation standard du type des analyseurs, prenant en paramètre un algorithme lors de sa création, afin d'en définir son comportement.

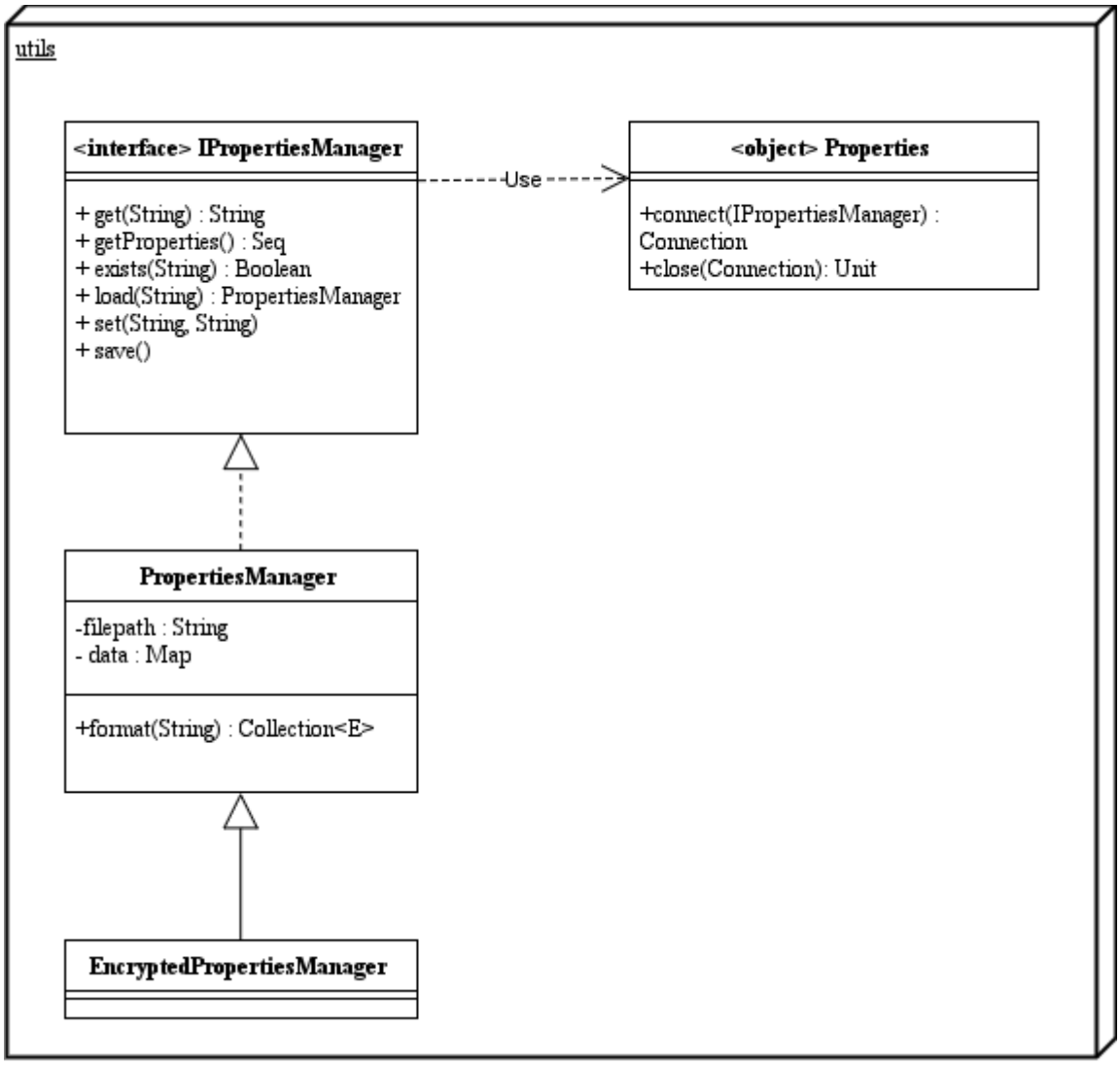
L'objet AnalyzerBuilder permet alors d'offrir des types d'analyseurs pré-construits depuis les différents algorithmes de classification disponibles.

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.
buildingDataFrame	Convertit des données brutes en usant d'une classe intermédiaire permettant de générer un DataFrame (c'est à dire un object simulant un tableau de base de données) qui sera donné à l'algorithme.

5.5. Module Analytique

Schéma de principe



Description des interfaces/classes utilisées

Le but des PropertiesManager est de permettre l'externalisation des configurations du système. On y retrouve entre autre les configurations d'accès à la base de données, aux clés et utilisateurs de Slack, aux adresses mails à contacter, ...

L'EncryptedPropertiesManager permet quant à lui aux données « sensibles » d'être chiffrés par SmartLogger, afin de ne pas afficher en clair le contenu dans le fichier properties. On pensera notamment aux mot de passes d'accès à la base de données, à la clé d'API de Slack, ...

L'objet Properties, quant à lui, permet de rassembler l'ensemble des fichiers properties dans un fichier unique permettant de centraliser l'ensemble des fichiers que l'on peut alors charger de manière aisé.

Comportements principaux

Fonction	Description
get	Retourne la propriété liée à la clé passé en paramètre
getProperties	Retourne l'ensemble des propriétés d'un fichier.
Exists	Vérifie l'existence d'une propriétés.
load	Charge l'ensemble des propriétés issues d'un fichier properties.
set	Change la valeur de la propriété.
save	Sauvegarde le fichier properties dans le dossier dédié.