# Rapport Complet : Système de Surveillance des GAB

**Date :** 29 mai 2025

**Version :** 1.0 (FR)

## 1. Introduction

### 1.1. Vision et Objectifs du Projet

La vision principale du projet de Système de Surveillance des GAB (Guichets Automatiques Bancaires) est d’établir une plateforme robuste et en temps réel pour superviser une flotte de GAB. Les objectifs principaux incluent :

* **Disponibilité Améliorée :** Minimiser les temps d’arrêt des GAB en détectant et en traitant les problèmes de manière proactive.
* **Maintenance Rationalisée :** Automatiser la création et la gestion des incidents grâce à des flux de travail définis, améliorant les temps de réponse et l’efficacité.
* **Visibilité en Temps Réel :** Fournir aux opérateurs et aux gestionnaires des informations immédiates sur l’état des GAB, la santé du matériel et les niveaux de liquidités.
* **Décisions Basées sur les Données :** Permettre une meilleure planification opérationnelle grâce à des rapports et des analyses complets sur les performances des GAB et les tendances des incidents.
* **Architecture Moderne :** Exploiter une architecture microservices évolutive et résiliente pour soutenir la croissance future et l’adaptabilité.

### 1.2. Périmètre du Système

Le système couvre le cycle de vie complet de la surveillance et de la gestion des GAB, y compris :

* Réception et traitement des flux de données en temps réel provenant des GAB (état, configuration, compteurs, transactions).
* Gestion des informations statiques des GAB (registre).
* Suivi des états dynamiques des GAB (état opérationnel, santé du matériel, niveaux de liquidités).
* Alertes automatisées basées sur des règles et des seuils prédéfinis.
* Gestion des incidents basée sur des flux de travail (ticketing, affectation, suivi, résolution) en fonction des contrats de maintenance ou d’assurance.
* Capacités de reporting et d’analyse.
* Une couche API sécurisée pour l’intégration et l’accès à l’interface utilisateur.
* Gestion des utilisateurs et authentification.

### 1.3. Objectif du Rapport

Ce document sert d’aperçu complet du projet de Système de Surveillance des GAB. Il consolide les informations clés concernant l’architecture du système, les composants fonctionnels (microservices), les flux de travail opérationnels, les artefacts de planification du projet (backlogs) et les détails initiaux de mise en œuvre. Il est destiné à fournir aux parties prenantes une compréhension approfondie de la conception du système et de l’avancement du développement.

## 2. Architecture Système

### 2.1. Style d’Architecture

Le système emploie une **Architecture Microservices**. Ce style a été choisi pour promouvoir :

* **Évolutivité :** Les services individuels peuvent être mis à l’échelle indépendamment en fonction de la charge.
* **Résilience :** Une défaillance dans un service est moins susceptible d’impacter les autres.
* **Maintenabilité :** Des bases de code plus petites et ciblées sont plus faciles à comprendre, modifier et tester.
* **Diversité Technologique :** Différents services pourraient potentiellement utiliser différentes technologies les mieux adaptées à leur tâche (bien que ce projet standardise sur Java/Spring Boot).
* **Déployabilité Indépendante :** Les services peuvent être mis à jour et déployés sans nécessiter un redéploiement complet du système.

La communication entre les services est principalement événementielle, utilisant **Apache Pulsar** comme courtier de messages pour un flux de données asynchrone. La communication synchrone pour les requêtes directes (par exemple, depuis l’interface utilisateur ou entre des services spécifiques) est gérée via des API REST exposées par une **Passerelle API (API Gateway)**.

### 2.2. Aperçu de l’Architecture de Haut Niveau

Le système se compose de plusieurs couches : sources de données (GAB), une couche d’infrastructure de données (Pulsar, Bases de données), une couche applicative (microservices et API Gateway), et des interfaces externes (UI, Systèmes de Notification).

*(Se référer au diagramme dans system\_architecture.png pour une représentation visuelle)*

Les interactions clés incluent :

1. **Réception des Données :** Les GAB envoient des données brutes à Pulsar. Le Service de Réception des Données consomme, traite et publie des événements standardisés vers des topics Pulsar internes.
2. **Traitement Événementiel :** Des microservices dédiés (Statut, Configuration, Compteurs, Alertes, Reporting) consomment ces événements internes pour mettre à jour l’état, déclencher des alertes ou effectuer des analyses.
3. **Accès API :** Une Interface Utilisateur ou des systèmes externes interagissent avec la plateforme via l’API Gateway, qui route les requêtes vers les microservices appropriés.
4. **Gestion des Incidents :** Le Service d’Alertes peut déclencher le Service de Gestion des Incidents, qui utilise jBPM pour orchestrer des flux de résolution prédéfinis.
5. **Stockage des Données :** Les services persistent leur état dans des bases de données dédiées (BD PostgreSQL Opérationnelle pour le registre, les incidents, les utilisateurs ; BD Séries Temporelles pour le statut, la configuration, les compteurs).

### 2.3. Technologies Clés

* **Framework Backend :** Java 17, Spring Boot 3.x
* **Courtier de Messages :** Apache Pulsar 3.x
* **Moteur de Workflow :** jBPM
* **Bases de Données :** PostgreSQL (Données Opérationnelles), TimescaleDB (Données Séries Temporelles - Recommandé)
* **API Gateway :** Spring Cloud Gateway
* **Conteneurisation :** Docker
* **Orchestration (Recommandé) :** Kubernetes

### 2.4. Bonnes Pratiques Intégrées

L’architecture intègre plusieurs bonnes pratiques pour les systèmes distribués :

* **Base de Données par Service :** Chaque microservice gère ses propres données.
* **Communication Asynchrone :** Le flux événementiel via Pulsar améliore le découplage et la résilience.
* **API Gateway :** Centralise le contrôle d’accès, le routage et les préoccupations transversales.
* **Services Sans État (Stateless) :** Lorsque c’est possible, les services sont conçus sans état pour l’évolutivité.
* **Configuration Externalisée :** Configuration gérée en dehors du code applicatif.
* **Découverte de Services :** Mécanismes pour que les services se trouvent dynamiquement.
* **Modèles de Résilience :** Utilisation de tentatives (retries), délais d’attente (timeouts), et potentiellement de disjoncteurs (circuit breakers).
* **Observabilité :** Accent sur la journalisation, la collecte de métriques et le traçage (détails d’implémentation hors périmètre actuel).
* **Sécurité :** Sécurisation des points d’accès et des canaux de communication.

## 3. Composants Microservices

Cette section détaille les microservices individuels qui constituent la couche applicative du Système de Surveillance des GAB.

*(Se référer au diagramme dans component\_diagram.png pour une représentation visuelle des interactions entre composants)*

### 3.1. Service de Réception des Données GAB

* **Responsabilité :** Consomme les flux de données brutes (état, configuration, compteurs, transactions) publiés par les GAB vers des topics Apache Pulsar. Valide, analyse et enrichit potentiellement les données avant de publier des événements standardisés vers des topics Pulsar internes pour traitement ultérieur.
* **Technologie :** Java Spring Boot, Client Apache Pulsar.
* **Communication :** Consomme depuis des topics Pulsar externes, publie vers des topics Pulsar internes.
* **Stockage de Données :** Traitement principalement sans état ; pas de stockage persistant.

### 3.2. Service Registre GAB

* **Responsabilité :** Gère les données de référence pour chaque GAB, incluant Numéro GAB, Numéro de Série, Marque, Modèle, Libellé, Adresse IP, Région, Agence, et détails du Responsable.
* **Technologie :** Java Spring Boot, Base de Données Relationnelle (PostgreSQL recommandé).
* **Communication :** Fournit des API REST pour les opérations CRUD sur les données GAB.
* **Stockage de Données :** Base de données relationnelle (BD Opérationnelle).

### 3.3. Service État GAB

* **Responsabilité :** Suit l’état opérationnel en temps réel et historique des GAB (hors ligne, hors service, superviseur, alerte, en service), l’état de maintenance, la date de dernière connexion et la date de dernière transaction.
* **Technologie :** Java Spring Boot, Base de Données Séries Temporelles (TimescaleDB recommandé) ou BD Relationnelle.
* **Communication :** Consomme les événements AtmStatusUpdated depuis Pulsar. Fournit des API REST pour interroger l’état actuel/historique. Publie des événements de changement d’état significatifs (ex: AtmOfflineEvent).
* **Stockage de Données :** Base de Données Séries Temporelles (ou BD Opérationnelle).

### 3.4. Service Configuration GAB

* **Responsabilité :** Surveille l’état des périphériques matériels individuels des GAB (Cassettes, Bac de rejet, Coffre, Clavier/EPP, Lecteurs de cartes, UPS, Caméra, Crypteur, Imprimantes, Journal, Distributeur, etc.). Attribue des codes/couleurs d’état de santé (Rouge, Orange, Vert, Gris).
* **Technologie :** Java Spring Boot, Base de Données Séries Temporelles (TimescaleDB recommandé) ou BD Relationnelle.
* **Communication :** Consomme les événements AtmConfigurationChanged depuis Pulsar. Fournit des API REST pour interroger l’état du matériel. Publie HardwareFailureEvent lors de la détection de problèmes.
* **Stockage de Données :** Base de Données Séries Temporelles (ou BD Opérationnelle).

### 3.5. Service Compteurs GAB

* **Responsabilité :** Suit les niveaux de liquidités pour chaque cassette : billets disponibles, montant total, type de billet (dénomination), et état de la cassette (OK, vide, problème, non configurée).
* **Technologie :** Java Spring Boot, Base de Données Séries Temporelles (TimescaleDB recommandé) ou BD Relationnelle.
* **Communication :** Consomme les événements AtmCountersUpdated depuis Pulsar. Fournit des API REST pour interroger les niveaux de liquidités. Publie LowCashEvent, CassetteEmptyEvent, etc.
* **Stockage de Données :** Base de Données Séries Temporelles (ou BD Opérationnelle).

### 3.6. Service de Gestion des Incidents

* **Responsabilité :** Gère le cycle de vie complet des incidents GAB en utilisant les workflows jBPM. Gère la création de tickets (manuelle/automatique), l’affectation, le suivi, la surveillance des SLA et la résolution basée sur les processus définis (Contrat de Maintenance, Assurance, etc.).
* **Technologie :** Java Spring Boot, Moteur jBPM, Base de Données Relationnelle (PostgreSQL recommandé).
* **Communication :** Fournit des API REST pour la gestion des incidents. Consomme les déclencheurs du Service d’Alertes. Interagit avec le moteur jBPM.
* **Stockage de Données :** Base de données relationnelle (BD Opérationnelle) pour les détails des incidents et l’état du workflow.

### 3.7. Service d’Alertes

* **Responsabilité :** Consomme les événements des services État, Configuration et Compteurs. Applique des règles pour détecter les conditions nécessitant des alertes. Envoie des notifications via les canaux configurés (Email, SMS, UI) et déclenche la création d’incidents dans le Service de Gestion des Incidents pour les alertes critiques.
* **Technologie :** Java Spring Boot, potentiellement un Moteur de Règles (ex: Drools).
* **Communication :** Consomme les événements depuis Pulsar. Publie des notifications. Appelle l’API du Service de Gestion des Incidents.
* **Stockage de Données :** Base de données optionnelle pour suivre les alertes générées.

### 3.8. Service de Reporting

* **Responsabilité :** Fournit des rapports agrégés et des analyses sur la disponibilité des GAB, les performances, les statistiques d’incidents, la conformité SLA, etc., en collectant des données auprès d’autres services ou de magasins de données dédiés.
* **Technologie :** Java Spring Boot, potentiellement des outils d’entreposage de données.
* **Communication :** Fournit des API REST pour accéder aux rapports. Interroge d’autres services ou des bases de données de reporting.
* **Stockage de Données :** Peut utiliser sa propre base de données de reporting ou un entrepôt de données.

### 3.9. Passerelle API (API Gateway)

* **Responsabilité :** Point d’entrée unique pour les clients externes. Gère le routage, l’authentification, la limitation de débit, la terminaison SSL et d’autres préoccupations transversales.
* **Technologie :** Spring Cloud Gateway (ou similaire comme Kong, Apigee).
* **Communication :** Route les requêtes externes vers les microservices internes.

### 3.10. Service de Gestion des Utilisateurs / Authentification

* **Responsabilité :** Gère les comptes utilisateurs, les rôles, les permissions. Gère l’authentification des utilisateurs (ex: connexion) et émet des jetons de sécurité (ex: JWT) pour l’autorisation.
* **Technologie :** Java Spring Boot, Spring Security, OAuth2/JWT (potentiellement intégré avec un Fournisseur d’Identité comme Keycloak).
* **Communication :** Fournit des points d’accès pour l’authentification. Interagit avec l’API Gateway et d’autres services pour les vérifications d’autorisation.
* **Stockage de Données :** Base de données relationnelle (BD Opérationnelle) pour les informations d’identification et les rôles des utilisateurs.

## 4. Flux de Travail Clés

Cette section décrit les principaux flux de travail opérationnels au sein du Système de Surveillance des GAB.

### 4.1. Flux de Réception des Données

Ce flux décrit comment les données provenant des GAB sont traitées et mises à disposition dans le système :

1. **Transmission :** Les GAB individuels de la flotte publient leurs données (état, configuration, compteurs, transactions) sous forme de messages vers des topics spécifiques prédéfinis dans le cluster Apache Pulsar.
2. **Consommation :** Le Service de Réception des Données GAB s’abonne à ces topics de données brutes dans Pulsar.
3. **Traitement :** À la réception d’un message brut, le Service de Réception :
   * Valide le format et la structure du message.
   * Analyse le contenu pour extraire les champs de données pertinents.
   * Enrichit potentiellement les données (par exemple, en ajoutant des horodatages standardisés).
4. **Publication Interne :** Le Service de Réception publie des messages d’événements distincts et traités (par exemple, AtmStatusUpdated, AtmConfigurationChanged, AtmCountersUpdated, AtmTransactionRecorded) vers des topics Pulsar internes dédiés. Ces événements suivent un schéma standardisé (par exemple, Avro ou JSON).
5. **Consommation Aval :** Les microservices pertinents s’abonnent à ces topics internes :
   * Service État GAB consomme les événements AtmStatusUpdated.
   * Service Configuration GAB consomme les événements AtmConfigurationChanged.
   * Service Compteurs GAB consomme les événements AtmCountersUpdated.
   * Service d'Alertes et Service de Reporting peuvent consommer plusieurs types d’événements.
6. **Mise à Jour de l’État :** Chaque service consommateur met à jour son état interne (enregistrements de base de données) en fonction des données d’événement reçues.

### 4.2. Flux d’Alerte et de Déclenchement d’Incident

Ce flux décrit comment les problèmes potentiels sont détectés et escaladés :

1. **Détection de Problème :** Un service de traitement d’événements (État GAB, Configuration, ou Compteurs) détecte une condition qui pourrait nécessiter une attention basée sur les données entrantes (par exemple, changement d’état à ‘hors ligne’, un composant matériel critique signale une erreur, une cassette de billets devient vide).
2. **Publication d’Événement :** Le service détecteur publie un message d’événement spécifique et granulaire (par exemple, AtmOfflineEvent, HardwareFailureEvent, LowCashEvent, CassetteEmptyEvent) vers un topic ‘événements’ interne désigné dans Pulsar.
3. **Évaluation de l’Alerte :** Le Service d'Alertes consomme ces événements granulaires.
4. **Application des Règles :** Le Service d’Alertes applique des règles prédéfinies (qui peuvent aller de simples seuils à une logique plus complexe) aux données de l’événement pour déterminer si une alerte doit être générée.
5. **Notification :** Si une condition d’alerte est remplie, le Service d'Alertes envoie des notifications via les canaux configurés (par exemple, Email à la personne responsable, SMS à un groupe de maintenance, mises à jour d’un tableau de bord UI).
6. **Création d’Incident (Optionnel) :** En fonction de la gravité définie dans les règles d’alerte, le Service d'Alertes peut effectuer un appel API synchrone vers le point d’accès REST du Service de Gestion des Incidents pour créer automatiquement un nouveau ticket d’incident, en transmettant les détails pertinents de l’événement déclencheur.

### 4.3. Flux de Gestion des Incidents (jBPM)

Ce flux détaille comment les incidents sont gérés à l’aide du moteur jBPM :

1. **Création d’Incident :** Un ticket d’incident est créé dans le Service de Gestion des Incidents, soit automatiquement via un appel API du Service d'Alertes, soit manuellement via une interface utilisateur interagissant avec l’API de Gestion des Incidents.
2. **Initiation du Workflow :** Lors de la création de l’incident, le Service de Gestion des Incidents identifie la définition de workflow appropriée (basée sur le type d’incident, le statut du contrat du GAB, etc.) et lance une nouvelle instance de processus jBPM, en passant les données nécessaires (comme l’ID du GAB, la description du problème) comme variables de processus.
3. **Exécution du Processus :** Le moteur jBPM prend le contrôle et exécute les étapes définies dans le modèle BPMN 2.0 correspondant. Le chemin spécifique dépend du type de workflow (par exemple, Sous Contrat, Hors Contrat/Assurance).
4. **Affectation des Tâches :** Le workflow attribue des tâches humaines (par exemple, ‘Diagnostiquer le Problème’, ‘Envoyer un Technicien’, ‘Valider la Réparation’) à des rôles ou groupes d’utilisateurs spécifiques (Helpdesk, Centre Suivi GAB, Fournisseur, Validateur).
5. **Interaction Utilisateur :** Les utilisateurs interagissent avec le système (probablement via une UI appelant l’API de Gestion des Incidents) pour voir leurs tâches assignées, réclamer des tâches, fournir les informations nécessaires (commentaires, détails de réparation, coûts) et terminer les tâches.
6. **Tâches de Service :** Le workflow peut inclure des tâches de service automatisées exécutées par le moteur jBPM, appelant potentiellement d’autres API de microservices si nécessaire.
7. **Suivi de l’État :** Le Service de Gestion des Incidents écoute les événements du moteur jBPM ou interroge son état pour mettre à jour l’enregistrement de l’incident dans sa base de données, reflétant l’état actuel du workflow, les affectations de tâches et les horodatages capturés (Date ouverture/fermeture pour différentes étapes).
8. **Surveillance SLA :** Des minuteurs et une logique au sein du workflow (ou du Service de Gestion des Incidents) suivent la progression par rapport aux Accords de Niveau de Service (SLA) ou Accords de Niveau Opérationnel (OLA) définis.
9. **Résolution & Clôture :** Une fois que le workflow atteint un état final (par exemple, ‘Résolu’, ‘Clos’, ‘Rejeté’), le ticket d’incident est mis à jour en conséquence.

### 4.4. Exemple de Diagramme de Séquence : Défaillance Matérielle

La séquence suivante illustre les interactions pour un événement typique de défaillance matérielle :

*(Se référer au diagramme dans sequence\_diagram\_hw\_failure.png pour une représentation visuelle)*

1. Un GAB détecte une défaillance de l’imprimante et envoie un message de configuration à Pulsar.
2. Le Service de Réception des Données GAB consomme le message brut, le traite et publie un ConfigurationEvent vers un topic Pulsar interne.
3. Le Service Configuration GAB consomme cet événement, met à jour l’état de l’imprimante à ‘ERREUR’ dans sa base de données (probablement la BD Séries Temporelles), et publie un HardwareFailureEvent vers un autre topic interne.
4. Le Service d'Alertes consomme le HardwareFailureEvent, évalue ses règles, détermine qu’une alerte et un incident sont nécessaires.
5. Le Service d'Alertes envoie une notification email/SMS via le Service de Notification.
6. Le Service d'Alertes appelle l’API du Service de Gestion des Incidents pour créer un incident.
7. Le Service de Gestion des Incidents crée l’enregistrement de l’incident et démarre l’instance de workflow jBPM appropriée.

## 5. Backlog Projet

Cette section donne un aperçu des artefacts de planification du projet, y compris le Backlog Produit global et un exemple de Backlog de Sprint.

### 5.1. Aperçu du Backlog Produit

Le Backlog Produit capture l’ensemble complet des fonctionnalités, caractéristiques et exigences souhaitées pour le Système de Surveillance des GAB. Il est organisé en Epics, représentant les principaux domaines de capacité. Les epics clés incluent :

* Configuration de l’Infrastructure Centrale et de la Plateforme
* Réception et Traitement des Données GAB
* Gestion du Registre GAB
* Surveillance de l’État GAB
* Surveillance de la Configuration Matérielle GAB
* Surveillance des Compteurs de Liquidités GAB
* Gestion des Incidents & Automatisation des Workflows (jBPM)
* Système d’Alertes & Notifications
* Reporting & Analytique
* Passerelle API & Sécurité
* Gestion des Utilisateurs & Authentification
* Interface Utilisateur (UI)
* Observabilité & Surveillance (de la plateforme elle-même)

### 5.2. Exemple de Backlog Sprint 1

Pour illustrer l’approche de développement itératif, un backlog exemple pour un sprint initial de 3 semaines a été créé.

* **Objectif du Sprint :** Établir le flux principal de réception des données, l’enregistrement de base des GAB et les capacités de suivi de l’état en temps réel. Fournir des API fondamentales pour accéder à ces données de base.
* **Domaines Clés Couverts :**
  + Configuration de l’Infrastructure Centrale (Répertoires, Env Local).
  + Service de Réception des Données GAB (Connexion Pulsar, analyse de base, publication d’événements d’état).
  + Service Registre GAB (API CRUD de base, schéma BD).
  + Service État GAB (Consommation d’événements, stockage d’état, API de requête de base).
  + Passerelle API (Configuration de base et routage pour les services Registre et État).
  + Tests Initiaux et Documentation.

## 6. Diagrammes

Les diagrammes visuels sont essentiels pour comprendre la structure et le flux du système. Les diagrammes suivants ont été créés :

* **Diagramme d’Architecture Système (system\_architecture.png) :** Fournit une vue d’ensemble de haut niveau des principaux composants, couches et de leurs interactions au sein de la plateforme et avec les systèmes externes.
* **Diagramme de Composants (component\_diagram.png) :** Montre les microservices au sein de la couche applicative, les composants d’infrastructure (Pulsar, Bases de données), les acteurs externes et leurs relations principales.
* **Diagramme de Séquence - Défaillance Matérielle (sequence\_diagram\_hw\_failure.png) :** Illustre le flux de messages étape par étape et les interactions entre les composants lorsqu’un événement spécifique (une défaillance matérielle) se produit, depuis le GAB jusqu’à la création de l’incident.

## 7. Conclusion

Ce rapport fournit un aperçu complet du projet de Système de Surveillance des GAB, détaillant sa vision, son architecture, ses composants, ses flux de travail et l’avancement initial de la mise en œuvre. Le système est conçu selon une approche microservices moderne, exploitant des technologies telles que Java Spring Boot, Apache Pulsar et jBPM pour créer une plateforme évolutive, résiliente et maintenable.

L’accent initial a été mis sur l’établissement du flux principal de réception des données en utilisant Pulsar et Spring Boot, démontrant la faisabilité de l’approche choisie.

### 8.1. Résumé de l’État du Projet

* **Architecture Définie :** Une architecture microservices claire a été conçue.
* **Composants Clés Identifiés :** Les microservices clés et leurs responsabilités sont décrits.
* **Flux de Travail Documentés :** Les principaux flux de données et de processus sont décrits.
* **Backlog Créé :** Un backlog produit et un exemple de backlog de sprint fournissent une feuille de route pour le développement.
* **Diagrammes Créés :** Des aides visuelles pour l’architecture, les composants et le flux de séquence sont disponibles.
* **Collaboration :** Pour une équipe de deux, la collaboration sur le code source se fera efficacement via un système de gestion de version comme Git.

### 8.2. Prochaines Étapes Potentielles / Améliorations Futures

En s’appuyant sur les fondations actuelles, les efforts de développement futurs se concentreraient probablement sur :

* Implémenter la logique métier complète au sein de chaque microservice (État, Configuration, Compteurs, Alertes, Gestion des Incidents, etc.).
* Développer les définitions de workflow jBPM (modèles BPMN) et les intégrer pleinement avec le Service de Gestion des Incidents.
* Construire les points d’accès API pour tous les services selon les contrats définis (par exemple, OpenAPI).
* Développer l’Interface Utilisateur (UI) pour la surveillance et l’interaction.
* Mettre en œuvre une observabilité robuste (journalisation, métriques, traçage).
* Mettre en place une infrastructure de déploiement de qualité production (Kubernetes, pipelines CI/CD).
* Effectuer des tests d’intégration et de bout en bout approfondis.

**Fin du Rapport**