# Remerciement

Au début, nous remercions Dieu tout-puissant pour la bénédiction de la santé et du succès tout au long de la période de nos études loin de notre famille et de nos amis, et nous le remercions de nous avoir donné suffisamment de force pour terminer le travail malgré les difficultés que nous avons rencontrées pour terminer nos études et aussi notre projet.

Nous tenons à exprimer notre profonde gratitude à nos familles et amis pour leur soutien constant, leur patience et leur encouragement tout au long de cette aventure.

Nous remercions également chaleureusement l’ensemble des enseignants et encadrants de l’IT Learning Academy ainsi que de l’Université de Pau et des Pays de l’Adour, pour la qualité de leur accompagnement, leur disponibilité et leurs précieux conseils qui ont grandement contribué à la réalisation de ce projet.

Nous percevons cette opportunité comme une étape importante dans notre développement de carrière. Nous nous efforcerons d'utiliser les compétences et les connaissances acquises de la meilleure façon possible et nous continuerons à travailler afin d'atteindre les objectifs de carrière souhaités ainsi que d'atteindre les objectifs de la science et de l'apprentissage.

# Avant-propos

Ce projet de fin d’études marque l’aboutissement de notre parcours de master et représente une étape significative dans notre formation en informatique. Il a été l’occasion de mettre en pratique les compétences acquises durant ces années d’apprentissage, à travers la conception et le développement d’une application de supervision.

Ce travail nous a permis de nous confronter à des problématiques concrètes et de développer des solutions techniques en lien avec les exigences du terrain. Il nous a également sensibilisés à l’importance de la rigueur, de la collaboration et de la gestion de projet dans un contexte professionnel.

Nous espérons que ce projet apportera une réelle valeur ajoutée à ceux qui l’utiliseront et qu’il contribuera, à son échelle, à répondre aux besoins en matière de suivi, de contrôle et de pilotage des systèmes.

# Résumé

Ce projet de fin d’études s’inscrit dans le cadre de notre formation en informatique et porte sur le développement d’une application de supervision. L’objectif principal est de concevoir un outil permettant de surveiller en temps réel l’état et le fonctionnement de différents systèmes ou services, afin de détecter rapidement les anomalies et faciliter la prise de décision.

Nous avons opté pour une architecture moderne, combinant performance, modularité et accessibilité, avec une interface utilisateur intuitive et des fonctionnalités adaptées aux besoins des utilisateurs. Le projet a été mené en respectant les bonnes pratiques de développement logiciel, en intégrant des outils de monitoring, d’alerte et de visualisation des données.

Ce travail nous a permis d’approfondir nos compétences techniques tout en nous confrontant aux défis de la gestion de projet, de la sécurité des données et de l’ergonomie.

# Abstract

This final-year project is part of our computer science training and focuses on the development of a supervision application. The main objective is to design a tool that enables real-time monitoring of the status and operation of various systems or services, in order to quickly detect anomalies and support decision-making.

We adopted a modern architecture that ensures performance, modularity, and accessibility, featuring an intuitive user interface and functionalities tailored to user needs. The project was carried out in compliance with best software development practices, integrating tools for monitoring, alerting, and data visualization.

This work allowed us to deepen our technical skills while tackling challenges related to project management, data security, and user experience.

# Liste des figures

[Figure 1: SNMP architecture 10](#_Toc201856717)

[Figure 2: logo PRTG 12](#_Toc201856718)

[Figure 3: logo Spring boot et React 12](#_Toc201856719)

[Figure 4 : logo NMAP 16](#_Toc201856720)

[Figure 5 : logo Docker 17](#_Toc201856721)

[Figure 6 : MVC 17](#_Toc201856722)

[Figure 7: Diagremme cas d'utilisation 19](#_Toc201856723)

[Figure 8 : Diagramme de classes 20](#_Toc201856724)

[Figure 9 : Diagramme de séquence - Ajout d’un équipement 21](#_Toc201856725)

[Figure 10 : Diagramme de séquence - Récupération de la liste des équipements supervisés 21](#_Toc201856726)

[Figure 11 : Diagramme de séquence - Récupération des détails complets d’un équipement 22](#_Toc201856727)

[Figure 12 : Diagramme de séquence - Navigation MIB 22](#_Toc201856728)

[Figure 13 : Simple Diagram Sequence Discovery Topologies 24](#_Toc201856729)

[Figure 14 : JPA & Hibernate 27](#_Toc201856730)

[Figure 15: SNMP4J 28](#_Toc201856731)

# Liste des tableaux

[Tableau 1 : Fonctions principales de l'application de supervision 13](#_Toc201856710)

[Tableau 2 : Objectifs non fonctionnels de l'application 15](#_Toc201856711)

[Tableau 3: Rôle des technologies complémentaires à SNMP 16](#_Toc201856712)

[Tableau 4 : Cas d’utilisation principaux 18](#_Toc201856713)

[Tableau 5 : Description des classes principales 19](#_Toc201856714)

[Tableau 6 : Entités liées à un équipement supervisé 27](#_Toc201856715)

[Tableau 7: Services responsables du polling SNMP et de la collecte des données réseau 28](#_Toc201856716)

Contents

[Remerciement 1](#_Toc201867919)

[Avant-propos 2](#_Toc201867920)

[Résumé 3](#_Toc201867921)

[Abstract 4](#_Toc201867922)

[Liste des figures 5](#_Toc201867923)

[Liste des tableaux 6](#_Toc201867924)

[Liste des abréviations 10](#_Toc201867925)

[Introduction générale 11](#_Toc201867926)

[Chapitre 1 : Présentation du projet 13](#_Toc201867927)

[1.1 Contexte général 13](#_Toc201867928)

[1.2 Problématique 13](#_Toc201867929)

[1.3 Objectifs du projet 13](#_Toc201867930)

[1.4 Cahier des charges fonctionnel 14](#_Toc201867931)

[1.5 Portée du projet 14](#_Toc201867932)

[1.5.1 Ce qui est inclus 14](#_Toc201867933)

[1.5.2 Ce qui est exclu 14](#_Toc201867934)

[1.6 Environnement de développement 15](#_Toc201867935)

[1.6.1 Frontend 15](#_Toc201867936)

[1.6.2 Backend 15](#_Toc201867937)

[1.6.3 Outils utilisés 15](#_Toc201867938)

[1.6.4 Plateformes de déploiement 15](#_Toc201867939)

[Chapitre 2 : Analyse et Conception 16](#_Toc201867940)

[2.1 Analyse des besoins 16](#_Toc201867941)

[2.1.1 Objectifs fonctionnels 16](#_Toc201867942)

[2.1.2 Objectifs non fonctionnels 16](#_Toc201867943)

[2.1.3 Contraintes techniques et environnementales 16](#_Toc201867944)

[2.2 Étude de l’existant 16](#_Toc201867945)

[2.3 Choix technologiques 17](#_Toc201867946)

[2.3.1 Justification de l'utilisation de SNMP 17](#_Toc201867947)

[2.3.2 Rôle de ICMP, LLDP et Nmap dans le système 17](#_Toc201867948)

[2.3.3 Choix de Spring Boot et React pour l’architecture logicielle 17](#_Toc201867949)

[2.4 Architecture du système 18](#_Toc201867950)

[2.4.1 Architecture générale (schéma) 18](#_Toc201867951)

[2.4.2 Architecture logicielle (backend, frontend, API, communication) 18](#_Toc201867952)

[2.4.3 Architecture réseau (agents SNMP, hôtes, routeurs) 19](#_Toc201867953)

[2.5 Modélisation UML 19](#_Toc201867954)

[2.5.1 Diagramme de cas d'utilisation 19](#_Toc201867955)

[2.5.2 Diagramme de classes 20](#_Toc201867956)

[2.5.3 Diagrammes de séquence 21](#_Toc201867957)

[2.5.4 Diagramme d’activités - processus de supervision 26](#_Toc201867958)

[2.6 Base de données 27](#_Toc201867959)

[2.6.1 Modèle conceptuel (MCD) 27](#_Toc201867960)

[2.6.2 Modèle logique 27](#_Toc201867961)

[2.6.3 Choix du type de base 27](#_Toc201867962)

[2.7 Sécurité et fiabilité 28](#_Toc201867963)

[2.7.1 Gestion des erreurs 28](#_Toc201867964)

[2.7.2 Sécurité des accès 28](#_Toc201867965)

[2.7.3 Résilience et tolérance aux pannes 28](#_Toc201867966)

[Chapitre 3 : Réalisation 29](#_Toc201867967)

[3.1 Introduction 29](#_Toc201867968)

[3.2 Architecture technique 29](#_Toc201867969)

[3.2.1 Architecture logicielle (Spring Boot, React, API REST, WebSockets) 29](#_Toc201867970)

[3.2.2 Architecture réseau simulée ou réelle (topologie utilisée pour les tests) 29](#_Toc201867971)

[3.3 Mise en place du backend 29](#_Toc201867972)

[3.3.1 Configuration du projet Spring Boot 29](#_Toc201867973)

[3.3.2 Gestion des dispositifs (Device Management) 29](#_Toc201867974)

[3.3.3 SNMP Polling et mise à jour des équipements 31](#_Toc201867975)

[3.3.4 Système d’alertes 32](#_Toc201867976)

[3.3.5 Découverte avec ICMP, LLDP et Nmap 34](#_Toc201867977)

[3.3.6 Navigateur MIB (MIB Browser) 35](#_Toc201867978)

[3.4 Développement du frontend 36](#_Toc201867979)

[3.5 Gestion de la persistance 36](#_Toc201867980)

[3.5.1 Choix de la base de données 36](#_Toc201867981)

[3.5.2 Modélisation des entités 36](#_Toc201867982)

[3.5.3 Sauvegarde des données collectées (historique, statuts) 36](#_Toc201867983)

[3.6 Sécurité et robustesse 37](#_Toc201867984)

[3.6.1 Authentification et sécurisation de l’accès à l’application (JWT, CORS) 37](#_Toc201867985)

[3.6.2 Gestion des erreurs et des cas d’échec (timeout SNMP, hôtes inaccessibles) 37](#_Toc201867986)

[3.6.3 Résilience des modules (tâches planifiées, relances automatiques) 37](#_Toc201867987)

[3.7 Tests et validation 37](#_Toc201867988)

[3.7.1 Tests unitaires et d’intégration (backend et frontend) 37](#_Toc201867989)

[3.7.2 Tests de performance (temps de réponse, charge) 37](#_Toc201867990)

[3.7.3 Scénarios de test fonctionnels 38](#_Toc201867991)

[3.8 Difficultés rencontrées et solutions apportées 38](#_Toc201867992)

[3.9 Interfaces principales (captures d’écran) 39](#_Toc201867993)

[3.10 Résumé 39](#_Toc201867994)

[Conclusion 40](#_Toc201867995)

[Références 41](#_Toc201867996)

# Liste des abréviations

|  |  |
| --- | --- |
| Abréviation | Signification |
| SNMP | *Simple Network Management Protocol* – Protocole de gestion de réseau |
| MIB | *Management Information Base* – Base d’informations de gestion |
| OID | *Object Identifier* – Identifiant unique d’un objet MIB |
| NMAP | *Network Mapper* – Outil de scan et de cartographie réseau |
| LLDP | *Link Layer Discovery Protocol* – Protocole de découverte de couche liaison |
| ICMP | *Internet Control Message Protocol* – Protocole de messages de contrôle Internet |
| UDP | *User Datagram Protocol* – Protocole de datagramme utilisateur |
| TCP | *Transmission Control Protocol* – Protocole de contrôle de transmission |
| IP | *Internet Protocol* – Protocole Internet |
| JSON | *JavaScript Object Notation* – Format léger d’échange de données |
| HTTP | *HyperText Transfer Protocol* – Protocole de transfert hypertexte |
| REST API | *Representational State Transfer API* – Interface de programmation basée sur REST |
| WebSockets | Protocole de communication bidirectionnelle en temps réel |
| MVC | *Model-View-Controller* – Modèle architectural de séparation des couches |
| API | *Application Programming Interface* – Interface de programmation applicative |
| CRUD | *Create, Read, Update, Delete* – Opérations de base sur les données |
| JWT | *JSON Web Token* – Jeton d’authentification sécurisé |
| CORS | *Cross-Origin Resource Sharing* – Partage des ressources entre origines |
| DOM | *Document Object Model* – Modèle objet du document HTML |
| SPA | *Single Page Application* – Application monopage |
| JSX | *JavaScript XML* – Syntaxe utilisée avec React |
| React | *React.js* – Bibliothèque JavaScript pour interfaces utilisateur |
| Spring Boot | Framework Java pour le développement rapide d’applications |
| JPA | *Java Persistence API* – Interface de persistance des données |
| Hibernate | Outil ORM (Object Relational Mapping) pour Java |
| SQL | *Structured Query Language* – Langage de requêtes pour bases de données |
| DTO | *Data Transfer Object* – Objet de transfert de données |
| IDE | *Integrated Development Environment* – Environnement de développement intégré |
| VM | *Virtual Machine* – Machine virtuelle |
| DNS | *Domain Name System* – Système de noms de domaine |
| CSS | *Cascading Style Sheets* – Feuilles de style en cascade |
| HTML | *HyperText Markup Language* – Langage de balisage de documents web |
| TCP/IP | Ensemble de protocoles fondamentaux d’Internet |
| SSL/TLS | *Secure Sockets Layer / Transport Layer Security* – Protocoles de sécurisation des échanges |

# Introduction générale

Dans un environnement où les infrastructures réseau sont de plus en plus complexes et critiques, la **supervision des équipements** est un impératif pour garantir la disponibilité, la performance et la sécurité des systèmes informatiques. Une solution de supervision est essentielle pour l'observation, l'analyse et l'intervention en temps réel sur l'état du réseau et de ses composants.

Ce projet vise le développement d'une application de supervision réseau intégrant plusieurs technologies complémentaires. Le cœur de ce système repose sur le **SNMP (Simple Network Management Protocol)**, un protocole standard largement plébiscité pour la collecte d'informations auprès des équipements réseau. L'application est développée avec **Spring Boot**, un Framework moderne reconnu pour le déploiement d'applications web performantes et modulaires, tandis que son interface utilisateur est construite avec **React**, assurant une expérience interactive et réactive.

Afin de renforcer les capacités de détection et de diagnostic du système, d'autres technologies sont intégrées :

* **ICMP (Internet Control Message Protocol)** est utilisé pour vérifier la connectivité des hôtes, de manière similaire à la commande ping.
* **LLDP (Link Layer Discovery Protocol)** permet une découverte dynamique de la topologie réseau.
* **Nmap**, un outil de scan réseau puissant, est employé pour identifier les hôtes actifs, les ports ouverts et les services en fonctionnement.

**SNMP (Simple Network Management Protocol)** est un protocole de couche application qui facilite l'échange d'informations de gestion entre les périphériques réseau. Il fonctionne selon un modèle client-serveur, où un gestionnaire SNMP (notre application de supervision) interroge des agents SNMP résidant sur les équipements réseau (routeurs, commutateurs, serveurs, imprimantes, etc.).

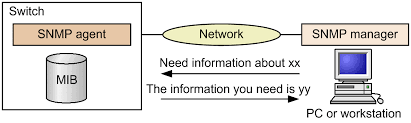


Figure 1: SNMP architecture

Ces agents maintiennent une base d'informations appelée **MIB (Management Information Base)**, qui est une collection structurée d'objets de gestion. Chaque objet MIB représente une donnée spécifique sur l'état ou les performances du périphérique (par exemple, l'utilisation du CPU, le trafic d'interface, l'état d'un port). Le gestionnaire SNMP peut effectuer des requêtes (Get, GetNext, GetBulk) pour récupérer ces informations ou envoyer des commandes (Set) pour modifier la configuration d'un équipement. Les agents peuvent également envoyer des notifications non sollicitées (Traps ou Inform) au gestionnaire pour signaler des événements critiques, tels qu'une panne de lien ou une saturation de ressource.

L'efficacité de la supervision réseau est maximisée lorsque SNMP est combiné avec d'autres protocoles et outils, chacun apportant une couche d'information complémentaire :

* **Intégration avec ICMP :** Avant d'initier des requêtes SNMP coûteuses en ressources, une vérification de la joignabilité d'un équipement via ICMP (ping) peut être effectuée. Cela permet de confirmer la présence d'un hôte sur le réseau et d'éviter des tentatives de connexion SNMP inutiles si l'équipement est hors ligne. ICMP fournit une première couche de détection de connectivité, tandis que SNMP fournit des informations détaillées sur l'état interne de l'équipement une fois la connectivité établie.
* **Intégration avec LLDP :** LLDP est un protocole de couche liaison qui permet aux périphériques réseau de s'annoncer aux périphériques voisins et de découvrir des informations sur eux. En combinant les données LLDP (par exemple, les identifiants de port local et distant, les capacités du périphérique voisin) avec les informations SNMP (comme l'état des interfaces, les adresses IP), il est possible de construire dynamiquement une topologie réseau précise. Cette cartographie aide à visualiser les interconnexions physiques, à identifier les goulots d'étranglement ou les points de défaillance potentiels, et à améliorer la compréhension globale de l'infrastructure.
* **Intégration avec Nmap :** Nmap est un puissant outil d'audit de sécurité et de découverte de réseau. Il peut être utilisé en amont pour identifier les hôtes actifs sur un segment de réseau, détecter les ports ouverts et déterminer les services en cours d'exécution. Cette information est cruciale pour identifier les équipements potentiellement gérables via SNMP (ceux avec le port UDP 161 ouvert) ou pour enrichir le contexte des données SNMP collectées. Par exemple, après avoir découvert un serveur via Nmap, des requêtes SNMP peuvent être ciblées pour obtenir des métriques spécifiques de ce serveur, offrant ainsi une vision plus holistique de sa santé et de son fonctionnement.

# Chapitre 1 : Présentation du projet

## 1.1 Contexte général

Dans un monde de plus en plus connecté, les infrastructures informatiques jouent un rôle central dans le bon fonctionnement des entreprises, des institutions et des services numériques. Face à cette dépendance croissante, la supervision des équipements réseau devient une nécessité stratégique pour garantir la disponibilité, la performance et la sécurité des systèmes. Le domaine de la supervision réseau englobe l’ensemble des outils, protocoles et pratiques permettant de surveiller en temps réel l’état des équipements connectés, de détecter les anomalies et de réagir rapidement en cas de défaillance.

Aujourd’hui, les réseaux sont de plus en plus hétérogènes, dynamiques et distribués. Cette complexité nécessite des solutions de supervision capables non seulement de collecter des informations techniques (état des interfaces, bande passante, charge CPU...), mais aussi de fournir une visualisation claire et exploitable de l’ensemble de l’infrastructure.

## 1.2 Problématique

Bien que plusieurs outils de supervision soient déjà disponibles sur le marché, comme Nagios, Zabbix ou PRTG, ils présentent souvent des limites lorsqu’il s’agit de les adapter à des environnements spécifiques ou de les intégrer à des infrastructures modernes. Certains sont complexes à configurer, d'autres peu flexibles ou payants dans leurs versions avancées.



Figure 2: logo PRTG

Face à ces contraintes, il devient pertinent de concevoir une solution de supervision personnalisée, centrée sur les besoins spécifiques de l’environnement cible. L’objectif est de proposer un système léger, évolutif, et capable de combiner plusieurs sources d’information réseau, tout en offrant une interface utilisateur intuitive.

## 1.3 Objectifs du projet

Le présent projet a pour but de concevoir et développer une application de supervision réseau intelligente et modulable, s’appuyant sur une architecture moderne combinant **Spring Boot** pour le backend et **React** pour le frontend.

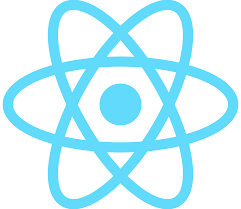


Figure 3: logo Spring boot et React

L’objectif principal est de fournir une solution capable de surveiller efficacement une infrastructure réseau, en intégrant plusieurs technologies complémentaires. L’application devra permettre la détection des hôtes actifs via **Nmap** et **ICMP**, la découverte de la topologie réseau grâce à **LLDP**, ainsi que la récupération d’informations détaillées sur les équipements à l’aide du protocole **SNMP**. L’interface utilisateur, développée avec React, devra offrir une visualisation dynamique et conviviale de l’état du réseau supervisé, tout en facilitant l’interaction avec les données collectées.

## 1.4 Cahier des charges fonctionnel

Le cahier des charges fonctionnel définit les fonctionnalités principales que doit offrir l’application de supervision. L’objectif est de proposer un outil efficace, accessible et interactif pour surveiller en temps réel l’état d’un réseau informatique.

Les fonctionnalités principales attendues sont les suivantes :

Tableau 1 : Fonctions principales de l'application de supervision

|  |  |
| --- | --- |
| Fonction | Description |
| Supervision des équipements | L’application doit permettre de suivre l’état des hôtes et des équipements réseau, en identifiant leur disponibilité (en ligne / hors ligne), en recueillant des informations techniques (trafic, statut des interfaces, etc.), et en générant des alertes en cas d’anomalie. |
| Découverte automatique du réseau | Grâce à des outils comme Nmap et LLDP, le système doit être capable de détecter automatiquement les hôtes présents sur le réseau, sans nécessiter de configuration manuelle. |
| Visualisation de la topologie | Une interface graphique doit permettre de représenter la topologie du réseau de manière dynamique, en indiquant les relations physiques ou logiques entre les équipements. |
| Historique des événements et données collectées | Les données issues des interrogations SNMP, des pings ICMP, ou des scans réseau doivent être historisées dans une base de données, afin de permettre une analyse a posteriori de l’activité et des incidents. |

## 1.5 Portée du projet

Cette section précise les limites du projet, en distinguant ce qui est couvert dans le périmètre de réalisation, et ce qui est volontairement exclu ou considéré comme hors champ.

### 1.5.1 Ce qui est inclus

Le projet couvre les éléments suivants :

* **Supervision multi-protocoles** : interrogation SNMP, vérification de connectivité via ICMP, détection des équipements par scan Nmap, et exploitation du protocole LLDP pour la découverte de la topologie.
* **Interface de supervision** : un tableau de bord web interactif, développé avec React, permettant d’afficher les équipements, leur statut en temps réel, ainsi que la cartographie du réseau.

### 1.5.2 Ce qui est exclu

Certaines fonctionnalités avancées ne seront pas implémentées dans le cadre de ce projet :

* **Gestion fine des droits utilisateurs** : le système ne comportera pas de module complexe de gestion des rôles, permissions ou hiérarchies d’accès.
* **Mécanismes avancés de prédiction d’incidents** : bien que le système collecte des données utiles à l’analyse, il n’intègre pas d’intelligence artificielle ni d’algorithme prédictif pour anticiper les pannes ou les dégradations de performance.

## 1.6 Environnement de développement

Le développement de l’application de supervision s’appuie sur un environnement technique moderne et cohérent, combinant des outils robustes, des langages largement adoptés et des technologies facilitant le déploiement et la maintenabilité du système.

### 1.6.1 Frontend

L’interface utilisateur est développée en **React** avec **TypeScript**, afin de garantir une meilleure sécurité de typage et une maintenabilité accrue. Le projet est initialisé via **Vite**, un outil de build rapide et léger. Pour le design de l’interface, la bibliothèque **shadcn/ui**, associée à **Tailwind CSS**, est utilisée pour accélérer la création de composants modernes et responsives. Les **icônes** sont intégrées via **Lucide Icons**, assurant une cohérence visuelle et une personnalisation facile.

Pour la gestion des formulaires et de la validation, **React Hook Form** est combiné avec **Zod**. La communication avec l’API backend repose sur **React Query**, qui facilite le traitement des appels asynchrones, la mise en cache des données et la gestion des états de chargement.

### 1.6.2 Backend

Le backend est construit en **Java 17** à l’aide du Framework **Spring Boot**, permettant une architecture modulaire et l’exposition d’**API REST**. La couche de persistance repose sur **Spring Data JPA** avec **Hibernate**, connectée à une base de données **MySQL**.

La communication avec les équipements réseau s’effectue via :

* **SNMP4J** pour la gestion des requêtes SNMP (Get, Walk, Trap, etc.) ;
* **mibble** pour le parsing des fichiers MIB, facilitant l'interprétation des OID ;
* **nmap4j**, une bibliothèque Java permettant d’exécuter des scans Nmap et de parser les résultats XML.

Pour sécuriser l’accès aux ressources, **Spring Security** et **JWT (JSON Web Token)** peuvent être intégrés de manière optionnelle, assurant un mécanisme d’authentification et d’autorisation fiable. La documentation des API est générée automatiquement grâce à **Springdoc OpenAPI**, exposée via **Swagger UI**.

Les tests unitaires sont rédigés en **JUnit 5** avec l’aide de **Mockito** pour simuler les dépendances.

### 1.6.3 Outils utilisés

Plusieurs outils viennent enrichir le développement, les tests et la supervision :

* **Postman** : pour le test des points d’API REST ;
* **Wireshark** : pour l’analyse réseau et la capture de trames SNMP ou LLDP ;
* **Nmap** : pour l’analyse et la découverte du réseau ;
* **Docker** : pour la conteneurisation de l’ensemble des composants et la gestion simplifiée des environnements.

### 1.6.4 Plateformes de déploiement

L’application est pensée pour être **déployée localement** lors du développement, puis facilement transférable sur des serveurs ou des machines virtuelles via **Docker**. Chaque composant (backend, frontend, base de données) est conteneurisé, permettant un déploiement unifié, reproductible et indépendant du système hôte. Cette approche facilite également l’évolution vers un déploiement en environnement cloud (Docker Swarm, Kubernetes) si nécessaire.

# Chapitre 2 : Analyse et Conception

## 2.1 Analyse des besoins

L’analyse des besoins vise à identifier les exigences auxquelles l’application de supervision doit répondre, en tenant compte à la fois des fonctionnalités attendues, des performances souhaitées, et des contraintes imposées par l’environnement technique.

### 2.1.1 Objectifs fonctionnels

Le système de supervision doit permettre la surveillance continue d’une infrastructure réseau. Il doit :

Identifier les hôtes actifs présents dans un réseau donné ;

* Vérifier l’accessibilité des équipements à l’aide de requêtes ICMP (ping) ;
* Collecter des informations détaillées via SNMP sur les équipements compatibles ;
* Découvrir automatiquement la topologie réseau grâce à LLDP ;
* Visualiser les données collectées de manière claire et interactive via une interface web.
* L’utilisateur doit pouvoir consulter l’état des équipements, recevoir des alertes en cas de panne ou de changement, et visualiser la structure du réseau de manière dynamique.

### 2.1.2 Objectifs non fonctionnels

Outre les fonctionnalités, plusieurs exigences de qualité sont essentielles :

Tableau 2 : Objectifs non fonctionnels de l'application

|  |  |
| --- | --- |
| Objectif | Description |
| Réactivité | L’interface utilisateur doit être fluide et réactive, même sur de grands réseaux supervisés. |
| Fiabilité | Le système doit pouvoir fonctionner de manière autonome et stable sur une longue période sans intervention manuelle. |
| Extensibilité | La solution doit permettre l’ajout futur de nouveaux protocoles, modules ou fonctionnalités sans refonte majeure. |
| Sécurité | Les échanges entre les composants (notamment via l’API) doivent être sécurisés : authentification, gestion CORS, et jetons JWT. |
| Portabilité | Grâce à l’utilisation de Docker, l’application doit être facilement déployable sur différents environnements sans configuration complexe. |

### 2.1.3 Contraintes techniques et environnementales

Le projet est soumis à plusieurs contraintes techniques :

* Le système doit fonctionner dans des réseaux hétérogènes (Windows, Linux, équipements variés) ;
* La supervision repose sur des protocoles bas-niveau comme SNMP et ICMP, qui peuvent être filtrés par des pare-feux ou des règles réseau ;
* Le développement doit s’appuyer sur un stack technologique défini : **Spring Boot** (Java) pour le backend et **React.js** pour le frontend ;
* L’application doit être conteneurisée à l’aide de **Docker**, ce qui impose certaines pratiques de modularité et de configuration réseau.

## 2.2 Étude de l’existant

* 2.2.1 Outils et solutions de supervision existants (ex. : Zabbix, Nagios)
* 2.2.2 Limites ou motivations pour une solution personnalisée

## 2.3 Choix technologiques

### 2.3.1 Justification de l'utilisation de SNMP

Le protocole **SNMP (Simple Network Management Protocol)** est un standard incontournable pour la supervision et la gestion des équipements réseau. Son adoption dans ce projet se justifie par sa légèreté, sa compatibilité étendue avec une grande variété d’équipements (switches, routeurs, serveurs, imprimantes, etc.) et sa capacité à interroger et modifier l’état d’un périphérique à distance. SNMP fonctionne selon un modèle client-serveur, où l’application agit comme un **gestionnaire SNMP** qui communique avec des **agents SNMP** résidant sur les équipements supervisés.

Grâce à l’utilisation de **MIB (Management Information Base)** et d’**OID (Object Identifiers)**, il est possible d’accéder à des informations très détaillées telles que le trafic réseau par interface, l’état des ports, l’usage du processeur, ou encore la température d’un équipement. Cette richesse fonctionnelle, combinée à une implémentation simple via UDP, en fait un choix parfaitement adapté aux exigences du système de supervision développé.

### 2.3.2 Rôle de ICMP, LLDP et Nmap dans le système

Pour enrichir la capacité de détection et de diagnostic du système, plusieurs technologies complémentaires à SNMP sont intégrées.

Tableau 3: Rôle des technologies complémentaires à SNMP

|  |  |
| --- | --- |
| Technologie | Rôle dans le système de supervision |
| ICMP (Internet Control Message Protocol) | Utilisé pour vérifier la disponibilité d’un hôte via des requêtes *echo request/reply* (ping). Il permet d’éviter les requêtes SNMP inutiles vers des hôtes hors ligne, ce qui réduit la charge réseau et accélère la supervision. |
| LLDP (Link Layer Discovery Protocol) | Permet de découvrir dynamiquement la **topologie physique** du réseau. Les informations LLDP récupérées via SNMP ou directement à partir des trames permettent de visualiser les connexions entre dispositifs et d’identifier les points de défaillance potentiels. |
| Nmap | Joue un rôle clé dans la **découverte initiale** du réseau. Il identifie les hôtes actifs, les ports ouverts et les services en cours d'exécution. Ces informations sont croisées avec les données SNMP pour détecter les équipements gérables et enrichir la supervision. |



Figure 4 : logo NMAP

### 2.3.3 Choix de Spring Boot et React pour l’architecture logicielle

Le choix de **Spring Boot** pour le backend repose sur sa capacité à simplifier le développement d'applications Java robustes, modulaires et facilement extensibles. Ce framework fournit une configuration automatique, un système intégré de gestion des dépendances, ainsi qu’un support natif pour les API REST, ce qui en fait un excellent candidat pour la couche serveur d’une application de supervision.

Pour le frontend, l'utilisation de **React.js** s’est imposée naturellement en raison de sa performance, sa réactivité et sa grande flexibilité. Grâce à sa structure basée sur les composants, React permet de créer des interfaces utilisateur dynamiques et interactives, parfaitement adaptées à l’affichage temps réel des données de supervision.

Ensemble, **Spring Boot** et **React** forment une architecture moderne **full stack**, capable de gérer des communications asynchrones via **REST APIs** et éventuellement **WebSockets** pour les mises à jour temps réel. Cette combinaison assure une séparation claire des responsabilités entre le traitement des données (backend) et la présentation (frontend), facilitant la maintenance, l’évolutivité, et l’expérience utilisateur.

## 2.4 Architecture du système

### 2.4.1 Architecture générale (schéma)

L’architecture générale du système repose sur le modèle **MVC (Model - View - Controller)**, combiné à une architecture **client-serveur découplée**. L’interface utilisateur (View), développée en **React.js**, interagit avec le contrôleur et la logique métier hébergée dans un backend **Spring Boot** via une **API REST** sécurisée.

Le backend assure le traitement des requêtes, la gestion des communications avec les équipements réseau (via SNMP, ICMP, Nmap, etc.), la logique de supervision, et la persistance des données. Le frontend, de son côté, interroge l’API pour afficher dynamiquement les statuts des équipements, les alertes, et la topologie réseau.

L’ensemble de l’application (backend, frontend et éventuellement base de données) est encapsulé dans des **conteneurs Docker**, facilitant ainsi le déploiement, la portabilité et la montée en charge. Cette architecture permet d'exécuter facilement la solution sur n’importe quel environnement compatible avec Docker, que ce soit en local, sur serveur dédié, ou en cloud.



Figure 5 : logo Docker

### 2.4.2 Architecture logicielle (backend, frontend, API, communication)

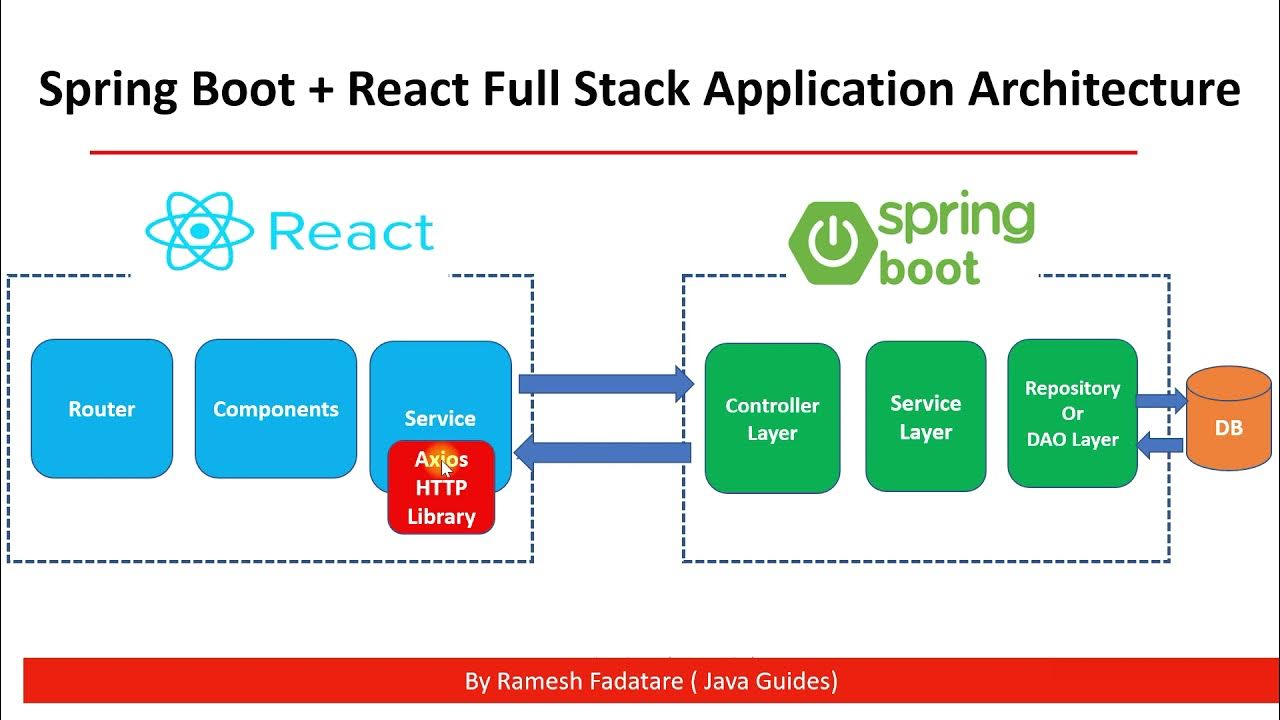


Figure 6 : MVC

### 2.4.3 Architecture réseau (agents SNMP, hôtes, routeurs)

L’architecture réseau adoptée dans ce projet repose sur un principe fondamental : le **serveur de supervision doit être situé au sein du réseau qu’il surveille**, afin de garantir un accès direct aux équipements et une communication fiable avec les agents SNMP. Cette disposition permet de minimiser les problèmes liés à la latence, à la sécurité, ou à des restrictions de routage.

Dans le cas où l'on souhaite superviser un réseau distant, deux solutions sont envisageables :

* Déployer une instance locale du serveur de supervision dans le réseau cible (via un VPN ou un relais)
* Ou bien configurer un **routeur central** capable de relayer les requêtes SNMP et ICMP entre le réseau de supervision et le réseau distant

Le serveur agit comme un gestionnaire central, interrogeant les agents SNMP installés sur les hôtes (routeurs, switches, serveurs) pour collecter les métriques, détecter les anomalies et visualiser la topologie réseau.

## 2.5 Modélisation UML

### 2.5.1 Diagramme de cas d'utilisation

Le diagramme ci-dessous présente les **principaux cas d’utilisation** du système de supervision réseau. Il distingue les rôles des utilisateurs (acteur) et les interactions possibles avec l’application selon leurs privilèges.

#### Acteurs

* **Opérateur Réseaux** : utilisateur classique ayant accès aux fonctionnalités de supervision, de découverte et de consultation. Il ne gère pas les utilisateurs.
* **Administrateur** : utilisateur ayant les mêmes droits que l’opérateur, avec des privilèges supplémentaires tels que la gestion des utilisateurs.

#### Cas d’utilisation principaux

Tableau 4 : Cas d’utilisation principaux

|  |  |
| --- | --- |
| Cas d’utilisation | Description |
| Se connecter | Authentification préalable à toute action, avec gestion des rôles (JWT). |
| Tableau de bord | Accès centralisé aux informations principales (état réseau, alertes, équipements). |
| Gérer les utilisateurs | Fonction réservée à l’administrateur pour créer, modifier ou supprimer des comptes. |
| Superviser les appareils | Permet de visualiser et interroger les équipements réseau via SNMP. |
| Découvrir les appareils | Lance la détection d’équipements par ICMP, Nmap ou LLDP. |
| Navigateur MIB | Permet de parcourir les objets SNMP définis dans les fichiers MIB importés. |
| Recevoir des alertes | Consultation en temps réel des alertes générées automatiquement. |
| Visualiser la topologie | Génère une représentation graphique des interconnexions réseau détectées. |

#### Relations notables dans le diagramme

* Tous les cas d’utilisation nécessitent au préalable une **authentification** (<<include>> vers "Se connecter").
* Le **tableau de bord** permet un accès indirect aux fonctions de supervision, de découverte et de topologie (<<extend>>).
* Le cas "Superviser les appareils" peut **étendre** l’utilisation du navigateur MIB, en cas de besoin d’interprétation des OID.

#### Diagramme UML

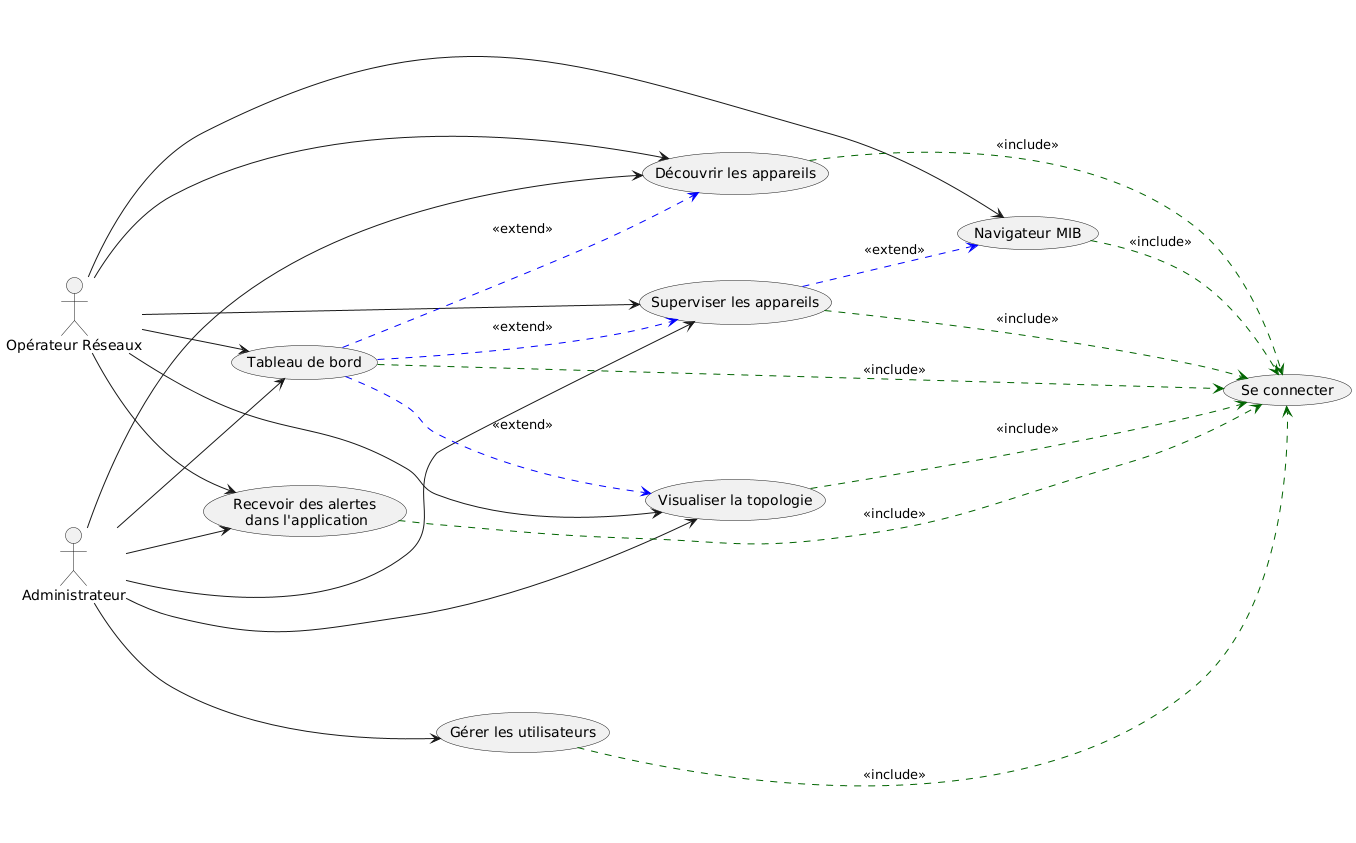


Figure 7: Diagremme cas d'utilisation

### 2.5.2 Diagramme de classes

Le diagramme suivant présente la **modélisation des entités principales** du système de supervision. Il illustre les relations entre les utilisateurs, les équipements, les profils de supervision, les alertes, les traps SNMP et les fichiers MIB.

Tableau 5 : Description des classes principales

|  |  |
| --- | --- |
| Classe | Rôle dans le système |
| User | Représente un utilisateur du système, qui peut gérer des dispositifs, recevoir des alertes, ou importer des fichiers MIB. |
| UserRole | Définit le rôle (ex. : ADMIN, USER) assigné à chaque utilisateur. |
| Device | Représente un équipement réseau supervisé, avec son état, sa configuration SNMP, ses interfaces et ses profils associés. |
| DeviceConfig | Contient les paramètres SNMP nécessaires pour interroger un équipement (adresse IP, sécurité, etc.). |
| Alert | Signale un événement critique ou une anomalie détectée sur un dispositif (niveau de sévérité, source, fréquence). |
| TrapEvent | Représente un trap SNMP reçu, déclenchant potentiellement une alerte. |
| MibFile | Fichier MIB importé par un utilisateur, contenant des objets SNMP spécifiques. |
| MibObject | Objet défini dans un fichier MIB, utilisé pour interpréter les OIDs SNMP. |
| DeviceInterface | Interface réseau d’un dispositif (nom, statut admin/oper, description). |
| SystemInfo | Informations générales du système SNMP (nom, description, uptime). |
| SystemUnit | Unités logiques internes d’un équipement, pour une vision détaillée de la structure. |
| IcmpProfile | Profil de supervision basé sur le protocole ICMP (ex. : ping). |
| IpProfile | Profil IP indiquant les statistiques entrantes/sortantes. |
| UdpProfile | Profil UDP affichant les datagrammes envoyés/reçus. |

#### Relations clés du diagramme

* Un User peut **gérer plusieurs Device**, recevoir des Alert, ou **analyser des TrapEvent**.
* Chaque Device est **lié à un DeviceConfig**, un ou plusieurs DeviceInterface, des profils (IcmpProfile, IpProfile, UdpProfile), et éventuellement à une structure interne (SystemUnit, SystemInfo).
* Les TrapEvent reçus depuis un Device peuvent **déclencher des Alert**.
* Les MibFile importés définissent des MibObject, explorables par les utilisateurs.
* Chaque Alert est liée à un Device, et chaque User a un UserRole.

#### Utilité

Ce diagramme traduit fidèlement la structure de la base de données et les **dépendances fonctionnelles entre les objets métier**. Il permet aussi de comprendre comment les modules SNMP, les alertes et les profils de supervision interagissent au sein du backend Spring Boot.

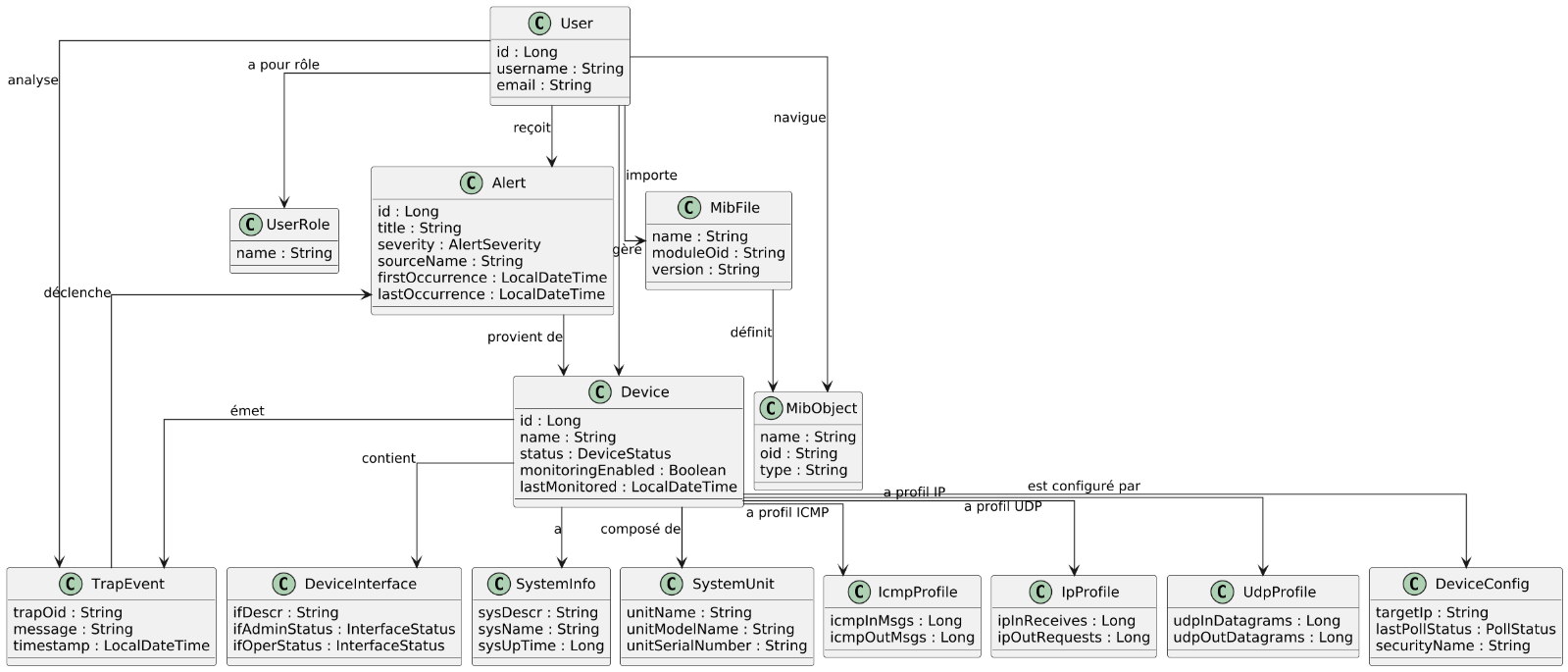


Figure 8 : Diagramme de classes

### 2.5.3 Diagrammes de séquence

Les diagrammes de séquence suivants illustrent les principales interactions entre l’utilisateur, l’interface web (frontend), l’API backend et les services associés. Chaque scénario décrit une fonctionnalité clé du système de supervision réseau, depuis la saisie utilisateur jusqu’à la manipulation des données côté serveur et base de données.

#### a) Ajout d’un équipement

Ce scénario couvre l’enregistrement d’un nouvel équipement réseau avec ses paramètres SNMP via l’interface web. Le backend sauvegarde les données dans la base de données, puis renvoie les informations de l’équipement avec son identifiant généré.

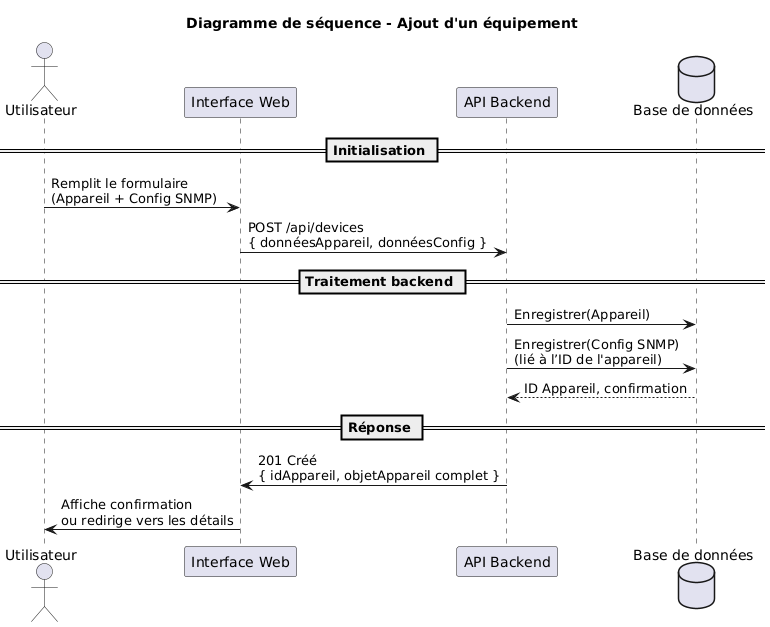


Figure 9 : Diagramme de séquence - Ajout d’un équipement

#### b) Récupération de la liste des équipements supervisés (vue allégée)

Ce scénario correspond à l’affichage **du tableau de bord** ou de la liste des équipements. Les données retournées sont légères (uptime, nom système, etc.) et sont issues directement de la **base de données**, sans requête SNMP en temps réel.

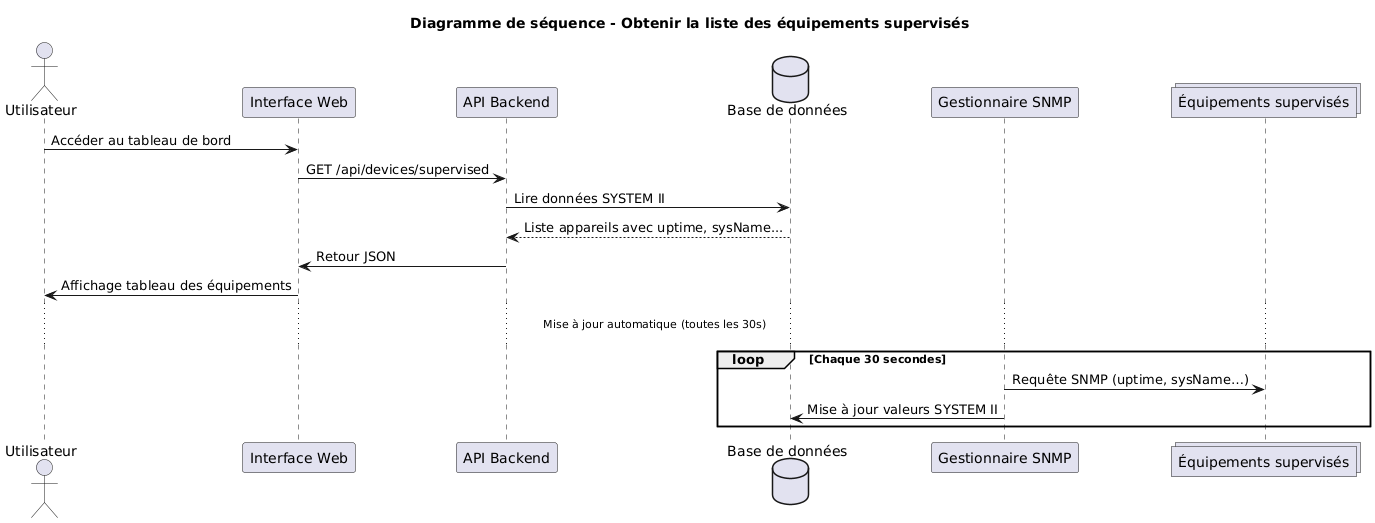


Figure 10 : Diagramme de séquence - Récupération de la liste des équipements supervisés

#### c) Récupération des détails complets d’un équipement

Ce scénario intervient quand l'utilisateur clique sur un équipement pour consulter ses détails (interfaces, profils, unités, etc.). Les données sont lues depuis la base, qui est elle-même mise à jour périodiquement en arrière-plan via SNMP.

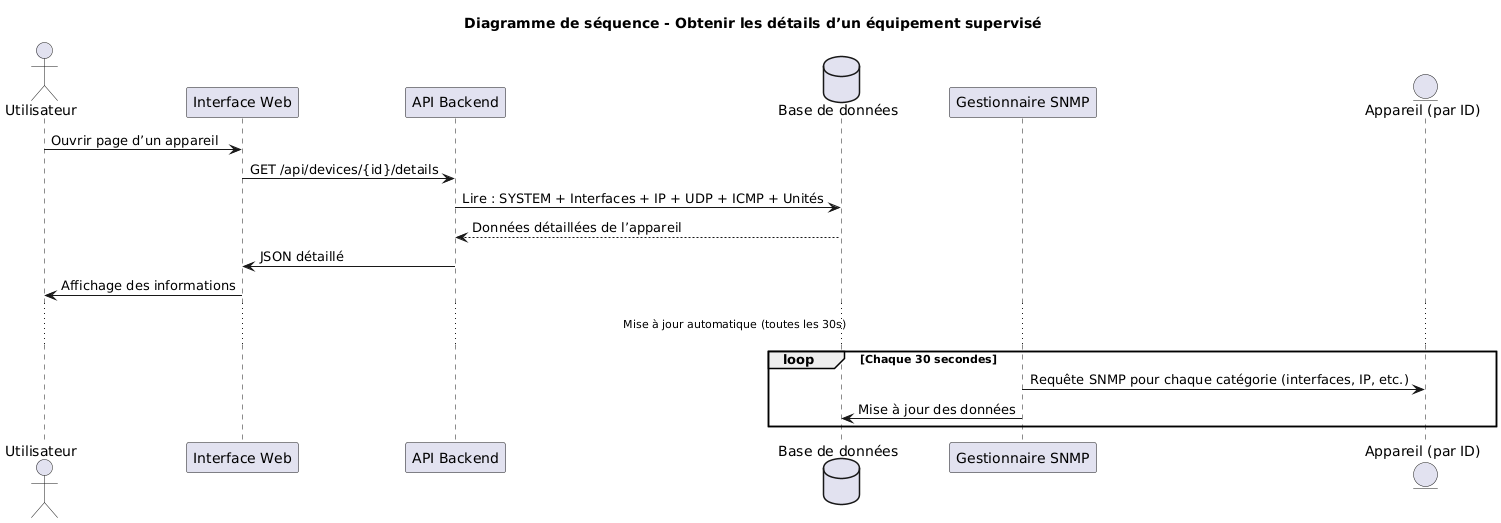


Figure 11 : Diagramme de séquence - Récupération des détails complets d’un équipement

#### e) Navigation MIB (MIB Browser)

Le navigateur MIB permet à un utilisateur d’effectuer une **requête ciblée sur un appareil** supervisé, en fournissant son identifiant et ses paramètres SNMP (ex : version, communauté, OID, etc.). Contrairement à la supervision périodique, cette opération est **à la demande**, lancée manuellement par l’utilisateur.

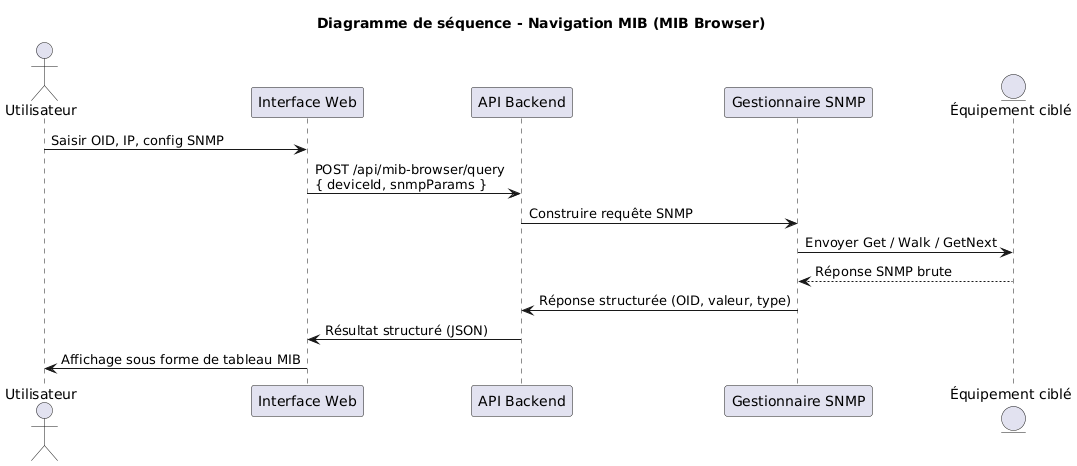


Figure 12 : Diagramme de séquence - Navigation MIB

#### f) Découverte de la topologie réseau

Le diagramme suivant illustre le **flux conceptuel** de la découverte de la topologie réseau, tel qu’implémenté dans l'application. Ce processus repose sur une coordination entre plusieurs composants backend (API, services, scanner, WebSocket) et une interaction continue avec le frontend.

|  |  |
| --- | --- |
| Phase | Description |
| 1. Requête de découverte | L'utilisateur démarre une requête POST /api/discovery/start en spécifiant une plage d'adresses IP (ex. : 192.168.1.0/24) et les méthodes souhaitées (SNMP, ICMP, scan de ports). L’API retourne un discovery ID et un endpoint WebSocket. |
| 2. Connexion temps réel | Le frontend établit une connexion WebSocket pour recevoir des mises à jour progressives (progress bar, messages d'état). |
| 3. Découverte en arrière-plan | Le DiscoveryService lance une tâche asynchrone qui exécute les phases suivantes : |
| → *Phase 1 - Scan réseau* | Le scanner envoie des pings et/ou des scans de port à chaque IP de la plage. Les hôtes actifs sont identifiés et ajoutés à la liste des appareils. |
| → *Phase 2 - Détails des équipements* | Pour chaque hôte détecté, une interrogation SNMP est effectuée pour obtenir ses caractéristiques : informations système, interfaces réseau, voisins LLDP. |
| → *Phase 3 - Construction de la topologie* | Les liens entre équipements sont analysés pour construire une carte logique ou physique du réseau. |
| 4. Finalisation et récupération | Une fois la découverte terminée, le client peut envoyer une requête GET /api/discovery/results pour récupérer les données topologiques à afficher dans l'interface. |

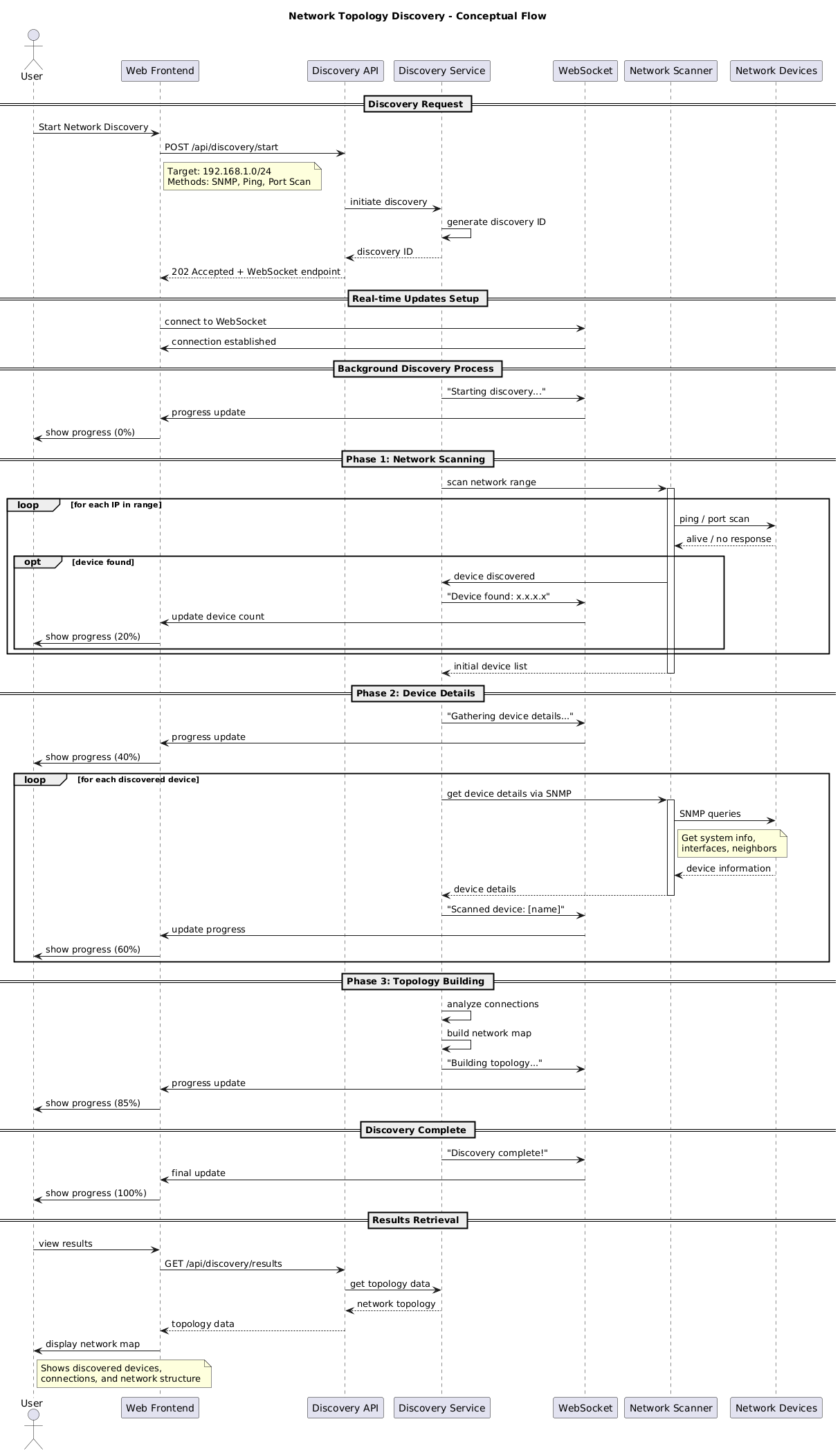


Figure 13 : Simple Diagram Sequence Discovery Topologies

### 2.5.4 Diagramme d’activités - processus de supervision

Le diagramme d’activités ci-dessous décrit le **fonctionnement global de la supervision automatique** des équipements réseau. Il s’agit d’un processus périodique, déclenché toutes les 30 secondes (ou selon la configuration), qui interroge les périphériques supervisés via SNMP et met à jour la base de données avec les informations collectées.

#### Étapes du processus :

1. Lancer le **planificateur** de tâches périodiques
2. Récupérer la **liste des équipements** ayant la supervision activée
3. Pour chaque équipement :
   * Vérifier s’il est **joignable** (ping ou timeout)
   * Envoyer des **requêtes SNMP** ciblées (SYSTEM, interfaces, IP, etc.)
   * Traiter les **réponses SNMP**
   * Enregistrer les données dans la **base de données**
   * Générer des alertes en cas d’anomalie détectée

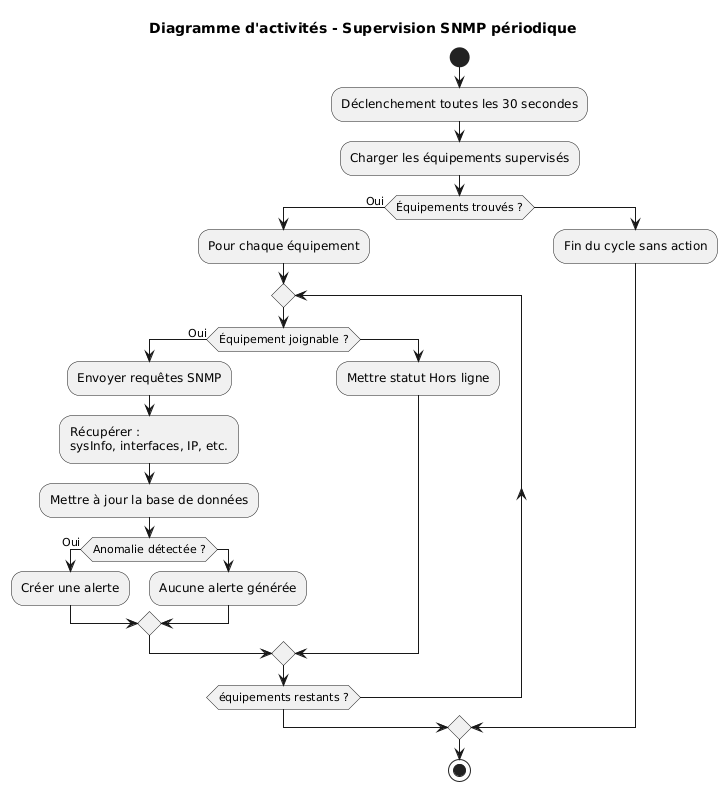


Figure 14 : Diagramme d’activités - processus de supervision

## 2.6 Base de données

L'application de supervision repose sur une base de données robuste et relationnelle afin de garantir la cohérence, la persistance et l’exploitabilité des données collectées à travers le système. Cette couche de persistance est essentielle pour le fonctionnement des modules de supervision, d’alerte, d’historique, de découverte, ainsi que pour l’interface web.

### 2.6.1 Modèle conceptuel (MCD)

### 2.6.2 Modèle logique

Le **Modèle Logique de Données** a été implémenté via **Spring Data JPA**, qui génère automatiquement les tables SQL à partir des entités Java annotées. Les relations sont exprimées en @OneToMany, @ManyToOne, @OneToOne, avec gestion des clés étrangères, contraintes d’intégrité, et indexations selon les besoins (ex : timestamps pour l’historique, deviceId pour les agrégats, etc.).

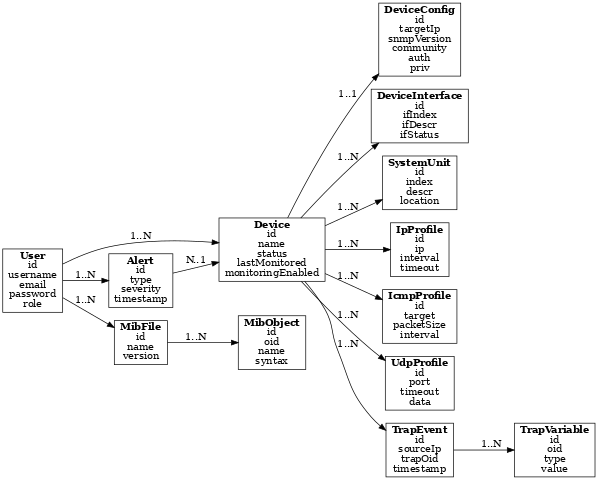


Figure 15 : MCD

### 2.6.3 Choix du type de base

Le choix de **PostgreSQL**, une **base de données relationnelle**, s’explique par plusieurs raisons :

Tableau 6 : Justification pour choisi PostgreSQL

|  |  |
| --- | --- |
| Critère | Justification |
| Compatibilité | Excellente intégration avec **Spring Boot** via Spring Data JPA |
| Fiabilité | PostgreSQL est reconnu pour sa stabilité et sa conformité ACID |
| Richesse SQL | Support avancé pour les jointures, index, agrégats temporels, etc. |
| Historique | Besoin de stocker des données temporelles (ex : supervision) |
| Outillage | Compatible avec des outils comme PgAdmin, DBeaver, JPA Inspector |
| Structuration | Les relations fortes entre entités (SNMP profiles, interfaces, traps, etc.) nécessitent un modèle structuré et cohérent |

* Une base NoSQL aurait été moins adaptée ici, car la structure des objets est complexe, fortement relationnelle, et nécessite des opérations transactionnelles.

## 2.7 Sécurité et fiabilité

L’architecture de l’application accorde une importance particulière à la sécurité des échanges et à la robustesse des traitements, en tenant compte des contraintes du protocole SNMP et de la nature critique des données manipulées. Cette section présente les mécanismes mis en place pour garantir la protection des accès, la gestion des erreurs, et la tolérance aux pannes.

### 2.7.1 Gestion des erreurs

Les communications réseau sont naturellement sujettes à des défaillances : équipements injoignables, timeout SNMP, ou erreurs d’authentification. Plusieurs mécanismes ont été mis en place pour limiter l’impact de ces problèmes :

* **Vérification ICMP préalable** : avant toute interrogation SNMP, un ping est envoyé à l’hôte cible. En cas d’absence de réponse, le système ignore l’équipement concerné pour ce cycle, évitant ainsi des tentatives de connexion bloquantes.
* **Gestion des timeouts SNMP** : chaque session SNMP est configurée avec un délai d’attente maximum (timeout), ainsi qu’un nombre limité de tentatives (retries), ce qui évite les blocages prolongés du système.
* **Logging centralisé** : toutes les erreurs de supervision sont enregistrées pour permettre un diagnostic ultérieur.
* **Statut de supervision** : les équipements sont automatiquement marqués comme "hors ligne" s’ils échouent plusieurs fois consécutivement à répondre.

### 2.7.2 Sécurité des accès

La plateforme repose sur une architecture sécurisée de type REST, renforcée par une couche d’authentification robuste :

* **Authentification par JWT (JSON Web Token)** : chaque utilisateur reçoit un jeton d’accès lors de la connexion. Ce jeton est vérifié à chaque requête API pour garantir l’identité et les droits de l’utilisateur.
* **Isolation des ressources** : les utilisateurs ne peuvent interagir qu’avec les équipements qu’ils possèdent. Toute tentative d’accès à des ressources externes est rejetée côté serveur.
* **CORS et HTTPS** : l’API REST est protégée par des règles de CORS et peut être déployée derrière un proxy HTTPS, assurant la confidentialité des échanges.
* **Sécurité Spring Boot** : les routes sont filtrées par rôles (ADMIN, USER), et seules les opérations autorisées sont exposées selon les droits.

### 2.7.3 Résilience et tolérance aux pannes

L’architecture du système garantit la continuité de la supervision même en cas d’absence d’activité utilisateur ou de panne partielle :

* **Supervision asynchrone** : les équipements sont interrogés de manière automatique toutes les 30 secondes via un mécanisme de polling interne, assurant une collecte régulière des données.
* **Système d’alerte basé sur SNMP traps** : en complément de la supervision active, le backend peut recevoir des notifications instantanées (traps SNMP) pour réagir à des événements critiques (ex. : lien coupé, surcharge).
* **Tâches planifiées résilientes** : les tâches de supervision sont isolées dans des services dédiés, avec gestion des exceptions et redémarrage automatique en cas d’échec partiel.

# Chapitre 3 : Réalisation

## 3.1 Introduction

* Brève présentation de l'approche de développement adoptée (itérative, modulaire…)

## 3.2 Architecture technique

### 3.2.1 Architecture logicielle (Spring Boot, React, API REST, WebSockets)



### 3.2.2 Architecture réseau simulée ou réelle (topologie utilisée pour les tests)

## 3.3 Mise en place du backend

### 3.3.1 Configuration du projet Spring Boot

Pour accélérer la mise en place du backend, le projet a été initialisé à partir d’un **boilerplate Spring Boot** complet, déjà configuré avec les composants essentiels à une architecture moderne, modulaire et sécurisée. Cette base préexistante permet de se concentrer rapidement sur l’implémentation des modules spécifiques liés à la supervision réseau.

Le boilerplate utilisé repose sur **Spring Boot 3.5.0** et intègre les éléments suivants :

* **Spring Data JPA** : pour la gestion de la persistance, avec une configuration orientée vers **PostgreSQL** ;
* **Spring Validation** : pour la validation des entrées au niveau des contrôleurs et des DTO ;
* **Spring Security** avec **JWT** : pour l’authentification et la sécurisation des endpoints via tokens ;
* **MapStruct** : pour faciliter la conversion entre entités et DTOs, améliorant la lisibilité et la maintenabilité du code ;
* **Lombok** : pour réduire le boilerplate Java en générant automatiquement les getters, setters, constructeurs, etc. ;
* **Swagger (Springdoc OpenAPI)** : pour exposer automatiquement la documentation de l’API REST via une interface Swagger UI accessible depuis /swagger-ui.html.

Ce projet de base fournit déjà une structure MVC bien organisée, avec une séparation claire des couches (Controller, Service, Repository, DTO), un système de gestion des exceptions globales, ainsi qu’un mécanisme de sécurité fonctionnel basé sur des rôles et des tokens JWT.

Une fois le projet cloné, les étapes suivantes ont été réalisées :

* Configuration des paramètres PostgreSQL dans application.yml (ou application.properties) ;
* Suppression ou adaptation des entités/domaines inutiles issus du boilerplate ;
* Intégration des modules spécifiques à la supervision (SNMP, Nmap, etc.) ;
* Ajout de nouveaux endpoints REST pour les opérations réseau (ping, scan, topologie, etc.).

### 3.3.2 Gestion des dispositifs (Device Management)

La gestion des dispositifs réseau constitue l’un des axes fondamentaux du système de supervision. Elle regroupe toutes les opérations liées à l’ajout, la configuration, le suivi, l’analyse et la représentation des **équipements supervisés**. L’ensemble des entités, services et contrôleurs associés à la gestion des dispositifs est conçu de manière modulaire et extensible.



Figure 16 : JPA & Hibernate

#### a) Modélisation des entités

L'entité principale est Device, qui représente un équipement réseau supervisé (routeur, switch, serveur, etc.). Elle contient des attributs issus du protocole SNMP, tels que :

* systemObjectId, systemName, systemLocation (issus de sysObjectID, sysName, sysLocation) ;
* systemUptime, systemServices, etc.

Chaque Device est lié à un User, propriétaire ou responsable de sa supervision.

Autour de cette entité centrale gravitent plusieurs sous-entités :

Tableau 7 : Entités liées à un équipement supervisé

|  |  |
| --- | --- |
| Entité | Description |
| DeviceInterface | Représente les interfaces réseau (ifIndex, ifDescr, ifStatus...) et leurs statistiques (trafic entrant/sortant, erreurs, etc.). |
| SystemUnit | Stocke des informations système détaillées si l’équipement est composé de plusieurs sous-unités logiques. |
| DeviceConfig | Contient tous les paramètres SNMP pour interroger l’équipement (IP cible, version SNMP, communauté, ports, timeout…), y compris les options avancées pour SNMPv3. |
| IpProfile, IcmpProfile, UdpProfile | Profils de supervision spécialisés pour des tests actifs sur l’équipement via différents protocoles. |

Cette modélisation permet d’assurer une grande flexibilité, avec la possibilité de gérer des équipements aux structures variées.

#### b) Services métiers

La logique de gestion des équipements est centralisée dans le **DeviceService**, responsable :

* de l’ajout et la modification des équipements ;
* de l’association automatique d’un DeviceConfig ;
* de la synchronisation avec les données collectées via SNMP (mise à jour des interfaces, de l’état, etc.) ;
* de la désactivation ou suppression logique des dispositifs.

Des services secondaires permettent d’interagir avec les composants liés : DeviceInterfaceService, SystemUnitService, etc.

#### c) Contrôleurs REST

Les contrôleurs REST exposés dans la couche Controller Layer permettent aux clients (frontend React ou outils externes) d’interagir avec le backend :

* DeviceController : endpoints CRUD pour les dispositifs, filtre par utilisateur, statut, type, etc.
* DeviceInterfaceController, SystemUnitController, etc. : permettent de consulter les sous-composants liés à un équipement.

Chaque réponse est encapsulée dans des **DTOs** mappés automatiquement grâce à **MapStruct**, assurant une séparation claire entre les entités de persistance et les données exposées à l’utilisateur.

#### d) Statut et typologie des équipements

Deux énumérations importantes enrichissent le système :

* **DeviceStatus** : reflète l’état de fonctionnement (ACTIVE, INACTIVE, MAINTENANCE, ERROR) ;
* **DeviceType** : catégorise les équipements (switch, routeur, serveur, imprimante, etc.).

Ces statuts sont automatiquement mis à jour par le module de supervision (SNMP, ICMP, etc.) en fonction des résultats de communication.

### 3.3.3 SNMP Polling et mise à jour des équipements

La supervision des équipements via SNMP repose sur un processus de **polling périodique**, qui permet de récupérer à intervalles réguliers des données d’état, de performance et de configuration des dispositifs réseau. Ce mécanisme est central dans le fonctionnement du système, car il alimente en temps réel la base de données avec des informations actualisées, essentielles à l’analyse et à l’affichage de la topologie ou des alertes.



Figure 17: SNMP4J

#### a) Structure du module de polling SNMP

L’architecture logicielle du polling est modulaire et répartie en plusieurs services spécialisés :

Tableau 8: Services responsables du polling SNMP et de la collecte des données réseau

|  |  |
| --- | --- |
| Service | Rôle |
| SnmpPollingService | Coordonne les tâches de supervision en interrogeant successivement les équipements enregistrés avec SNMP activé. |
| SnmpClientService | Encapsule la logique bas-niveau de communication avec les agents SNMP (Get, GetNext, Walk...), en s’appuyant sur SNMP4J. |
| SystemInfoPollService | Récupère les informations générales d’un équipement (sysName, sysLocation, uptime...). |
| SystemUnitPollService | Extrait les unités logiques internes à un dispositif, si elles sont disponibles via MIB. |
| InterfacePollService | Collecte les données sur les interfaces réseau (état, bande passante, erreurs, etc.). |
| IpProfilePollService, IcmpProfilePollService, UdpProfilePollService | Exécutent des sondes spécifiques définies par l’utilisateur (selon les profils configurés). |

Chaque service est responsable d’une **partie fonctionnelle bien définie**, ce qui facilite l’extension du système à de nouveaux types de sondes ou de profils de surveillance.

#### b) Cycle de polling

Le cycle de polling suit les étapes suivantes :

1. Sélection des dispositifs actifs avec la supervision SNMP activée (monitoringEnabled = true) ;
2. Récupération de la configuration SNMP via DeviceConfig (IP, port, version, communauté ou credentials v3) ;
3. Appel de SnmpClientService pour envoyer les requêtes vers les OID nécessaires ;
4. Traitement des réponses, mise à jour des entités associées (Device, DeviceInterface, SystemUnit, etc.) ;
5. Gestion des erreurs : timeout, unreachable host, OID non disponible, etc.

Chaque résultat est persisté en base, et horodaté (updatedAt) pour garantir une traçabilité des données et permettre une visualisation temporelle dans l’interface utilisateur.

#### c) Mise à jour automatique des équipements

Les données collectées sont utilisées pour :

* **Mettre à jour dynamiquement les objets Device** : nom, localisation, statut global, uptime, etc. ;
* **Réinitialiser ou créer dynamiquement les interfaces (DeviceInterface)** si des changements sont détectés (ex. : ajout ou retrait d’une carte réseau) ;
* **Mettre à jour l’état (DeviceStatus)** selon la disponibilité ou l’état des services détectés ;
* **Déclencher des alertes** si des anomalies sont constatées (interface down, erreur SNMP, etc.).

Ce système de mise à jour continue garantit une vision fidèle de l’état du réseau, tout en facilitant la détection des pannes ou modifications structurelles.

#### d) Extensibilité et planification

Le polling peut être lancé :

* Manuellement à partir d’un endpoint REST ou via l’interface web ;
* De manière automatique via une **planification (scheduler)** intégrée, utilisant @Scheduled dans SnmpPollingService.

La modularité du système permet d’intégrer facilement d’autres types de polling ou de données SNMP issues de MIB spécifiques, en implémentant de nouveaux services spécialisés.

### 3.3.4 Système d’alertes

Le système d’alertes constitue un **composant central de la supervision** réseau. Il permet de signaler les événements critiques, les anomalies ou les changements d’état des dispositifs, afin de garantir une réactivité maximale de l’utilisateur face aux incidents.

#### a) Modélisation des alertes

Les alertes sont représentées dans le système par l’entité Alert, qui encapsule les données suivantes :

* **Type d’alerte** (AlertType) : par exemple DEVICE\_DOWN, HIGH\_CPU\_USAGE, TRAP\_RECEIVED, etc.
* **Gravité** (AlertSeverity) : cinq niveaux sont disponibles (INFO, LOW, MEDIUM, HIGH, CRITICAL).
* **Statut** (AlertStatus) : état de traitement de l’alerte (ACTIVE, ACKNOWLEDGED, RESOLVED, CLOSED).
* **Source** (SourceType, sourceId) : permet d’identifier l’origine de l’événement (équipement, interface, profil de supervision, etc.).
* **Utilisateurs liés** : émetteur, acquitteur, et responsable de la résolution de l’alerte (liés via l’entité User).

Chaque alerte est horodatée (createdAt, acknowledgedAt, resolvedAt) pour garantir la traçabilité et l’historique des incidents.

#### b) Déclenchement automatique des alertes

Les alertes sont générées automatiquement via plusieurs sources du système :

* **SNMP polling** : si un équipement ne répond plus, ou si une interface réseau passe en DOWN, une alerte est générée avec le type DEVICE\_DOWN ou INTERFACE\_DOWN.
* **ICMP/Ping** : en cas d’inaccessibilité par ping, une alerte DEVICE\_DOWN de gravité élevée peut être déclenchée.
* **Profils personnalisés (UDP, ICMP, IP)** : si les seuils définis sont dépassés (ex. : timeout, perte de paquets), une alerte personnalisée est créée.
* **SNMP Traps** : lorsqu’un trap SNMP est reçu et identifié comme critique, le système crée une alerte de type TRAP\_RECEIVED.

Le service AlertServiceImpl centralise cette logique métier. Il expose des méthodes pour créer une alerte, vérifier si une alerte équivalente est déjà active (afin d’éviter les doublons), et mettre à jour son état.

Le service DuplicatePreventionService est utilisé pour éviter la redondance d’alertes similaires générées à intervalles rapprochés, ce qui réduit le bruit dans le système de notifications.

#### c) Notification des alertes

Lorsqu’une alerte est créée, elle est immédiatement transmise au **système de notification**, représenté par AlertNotificationService. Ce service a pour rôle de :

* Propager l’alerte en temps réel via WebSockets ou autres canaux vers l’interface frontend ;
* Préparer les alertes à destination d’éventuelles intégrations futures (mail, SMS, Slack, etc.) ;
* Associer l’alerte à l’utilisateur concerné, si applicable.

L’implémentation AlertNotificationServiceImpl gère la sérialisation des objets Alert et leur diffusion aux clients abonnés.

#### d) Gestion des alertes par l’utilisateur

Depuis l’interface utilisateur, l’utilisateur peut :

* **Consulter la liste des alertes actives ou historiques** ;
* **Acquitter** une alerte (changement de statut en ACKNOWLEDGED) ;
* **Marquer comme résolue** une alerte après intervention (statut RESOLVED) ;
* **Fermer** une alerte définitivement si elle n’est plus pertinente (CLOSED).

Toutes les actions sont horodatées et historisées, avec le nom de l’utilisateur associé à chaque étape.

### 3.3.5 Découverte avec ICMP, LLDP et Nmap

Le système de découverte permet d’identifier automatiquement les équipements présents dans un réseau cible, avec un minimum d'intervention manuelle. Il s’agit d’un **processus central pour initialiser la supervision**, cartographier la topologie et détecter les nouveaux hôtes à surveiller.

Cette découverte repose sur la combinaison de trois mécanismes complémentaires :

* **ICMP (ping)** pour la détection de la connectivité,
* **Nmap** pour le scan des ports et des services actifs,
* **SNMP/LLDP** pour enrichir les données et reconstruire dynamiquement la topologie physique.

#### a) Architecture des services de découverte

Le processus de découverte est orchestré par un service principal, DiscoveryServiceImpl, qui coordonne plusieurs sous-services spécialisés :

* NmapDiscoveryService : utilise **nmap4j** pour lancer des scans Nmap sur une plage IP donnée, détecter les hôtes actifs et leurs ports ouverts ;
* SnmpDiscoveryService : tente une requête SNMP sur les hôtes identifiés pour vérifier leur compatibilité et extraire des informations supplémentaires ;
* ProgressCallback : interface utilisée pour notifier en temps réel l'avancement de la découverte (utile pour le frontend ou les logs).

Chacun de ces services travaille de façon indépendante mais peut être chaîné pour enrichir progressivement les résultats.

#### b) Étapes du processus de découverte

1. **Scan initial avec Nmap** : un scan de type -sP (ping scan) ou -sS (TCP SYN scan) est lancé pour identifier les hôtes en ligne dans une plage IP donnée.
2. **Détection SNMP** : pour chaque hôte détecté, une tentative d'interrogation SNMP est faite (GetRequest sur des OID standards) afin de vérifier la compatibilité et de récupérer des infos système (sysName, sysDescr, etc.).
3. **Exploration LLDP** : si SNMP est activé et les MIB LLDP disponibles, les informations de voisinage sont extraites (lldpRemSysName, lldpRemPortId, etc.) pour reconstruire la **topologie physique du réseau**.
4. **Création ou mise à jour des objets Device** : les hôtes validés sont enregistrés dans la base, avec toutes les informations disponibles (IP, type, SNMP supporté, services actifs...).

#### c) Détection de la topologie réseau (LLDP)

LLDP est exploité via le service SNMP. Lorsqu’un équipement compatible expose ses informations LLDP, le système peut déterminer :

* À quels autres équipements il est connecté (nom, port distant) ;
* Le port local utilisé pour chaque lien ;
* Des données descriptives sur le voisinage.

Les données LLDP permettent ainsi de générer dynamiquement une **carte topologique du réseau**, qui peut ensuite être visualisée dans le frontend (via un graphe, par exemple).

#### d) Résultats et enrichissement

Chaque découverte aboutit à :

* L’ajout de nouveaux équipements Device dans le système ;
* L’identification automatique du type d’équipement (DeviceType) selon les ports/services détectés ;
* L’association de configurations SNMP (DeviceConfig) pour les hôtes compatibles ;
* L’enrichissement de la base de données en préparation de la phase de polling.

Cette phase est également essentielle pour la **mise à jour du réseau supervisé**, en détectant les hôtes ajoutés, déplacés ou supprimés.

### 3.3.6 Navigateur MIB (MIB Browser)

Un composant important de la plateforme de supervision est le **navigateur MIB**, qui permet d’analyser, importer et parcourir les fichiers MIB afin d’extraire des informations compréhensibles à partir des OID SNMP. Ce module apporte une **couche d’intelligence lisible** par l’utilisateur en transformant les identifiants numériques en noms et descriptions interprétables.

#### a) Objectifs du module

Le navigateur MIB permet de :

* Charger et parser dynamiquement des fichiers MIB au format texte ;
* Extraire la liste des objets SNMP (OIDs), leurs noms symboliques, types, accès, statuts et descriptions ;
* Permettre leur utilisation dans l’interface utilisateur (ex. : explorer les OID disponibles sur un équipement) ;
* Associer ces objets à des entités supervisées (interfaces, métriques, traps…).

#### b) Structure du module MIB

Le traitement est géré par le service MibServiceImpl, qui utilise une bibliothèque tierce (probablement **mibble**) pour analyser le contenu des fichiers MIB et les transformer en objets exploitables dans le backend.

Deux entités principales structurent le modèle :

* **MibFile** : représente un fichier MIB importé (nom, version, description, chemin, contenu brut, état actif) ;
* **MibObject** : représente un objet SNMP défini dans ce fichier, avec son OID, nom, syntaxe, accès, statut, et description.

Chaque fichier MIB peut contenir des centaines d’objets OID. Ces objets sont extraits et persistés dans la base de données, permettant une recherche rapide ou un affichage hiérarchique.

#### c) Fonctionnement

1. **Import d’un fichier MIB** : l’utilisateur peut téléverser un fichier .mib via l’interface frontend ou une API.
2. **Parsing** : le fichier est traité via MibServiceImpl, qui utilise la bibliothèque de parsing pour extraire les définitions d’objets.
3. **Sauvegarde** : chaque MibObject extrait est associé au MibFile parent et sauvegardé dans la base.
4. **Consultation** : l’utilisateur peut explorer les objets MIB dans une vue arborescente, effectuer des recherches ou utiliser les OIDs lors de la configuration d’interrogations SNMP.

#### d) Utilité dans la supervision

Le navigateur MIB est indispensable pour :

* **Traduire les OIDs** bruts reçus dans les réponses SNMP ou les traps en noms humains compréhensibles ;
* **Explorer dynamiquement les équipements** selon leurs MIB spécifiques ;
* **Personnaliser les requêtes SNMP** en ciblant des objets d’intérêt précis définis dans les MIB propriétaires ou standards.

## 3.4 Développement du frontend

* 3.4.1 Mise en place du projet React
* 3.4.2 Connexion avec l’API backend (fetch, axios)
* 3.4.3 Affichage des équipements et de leurs statuts
* 3.4.4 Visualisation de la topologie réseau (ex. : avec D3.js ou autre lib graphique)
* 3.4.5 Gestion en temps réel (ex. : WebSockets ou polling pour updates dynamiques)

## 3.5 Gestion de la persistance

### 3.5.1 Choix de la base de données

Le système utilise **PostgreSQL** comme base de données relationnelle. Ce choix est motivé par :

* Sa **stabilité et fiabilité** dans les applications critiques,
* Sa **compatibilité complète avec Spring Data JPA**,
* Son support avancé des types temporels (utile pour tracer les événements SNMP, les alertes, ou les mises à jour),
* Ses performances adaptées à la gestion de grandes quantités de données historisées.

### 3.5.2 Modélisation des entités

La couche de persistance est structurée autour de **plusieurs entités métier**, représentées en base via **JPA (Java Persistence API)**. Les entités principales sont :

* Device, DeviceConfig, DeviceInterface, SystemUnit pour représenter les équipements et leur configuration ;
* Alert, TrapEvent, MibObject, MibFile pour les événements SNMP ;
* IpProfile, IcmpProfile, UdpProfile pour les profils de supervision ;
* User, UserRole pour la gestion des comptes.

Des relations fortes (@OneToMany, @ManyToOne, @OneToOne) lient les entités, offrant une structure cohérente et navigable, facilitant les requêtes complexes.

### 3.5.3 Sauvegarde des données collectées (historique, statuts)

Chaque service de collecte (polling, découverte) met à jour les entités en base :

* L’état des équipements (DeviceStatus) est actualisé en temps réel ;
* Les informations reçues (trafic, uptime, erreurs) sont historisées avec createdAt, updatedAt ;
* Les alertes, traps et erreurs sont conservés pour **auditer les événements passés** ;
* La base peut ainsi servir à **générer des graphiques**, **filtrer les incidents**, ou **rejouer un état réseau**.

## 3.6 Sécurité et robustesse

### 3.6.1 Authentification et sécurisation de l’accès à l’application (JWT, CORS)

La sécurité de l’application est assurée via :

* **Spring Security**, configuré pour protéger tous les endpoints sensibles ;
* **JWT (JSON Web Tokens)** pour l’authentification stateless entre le frontend React et le backend ;
* Gestion des rôles ADMIN et USER pour restreindre certaines actions ;
* **CORS (Cross-Origin Resource Sharing)** est configuré pour autoriser uniquement les appels depuis le frontend autorisé, renforçant la sécurité côté navigateur.

### 3.6.2 Gestion des erreurs et des cas d’échec (timeout SNMP, hôtes inaccessibles)

Des mécanismes de tolérance sont en place pour éviter les blocages :

* Timeout SNMP configurable via DeviceConfig, avec retries automatiques ;
* Vérification ICMP préalable pour éviter des tentatives inutiles sur des hôtes hors ligne ;
* Gestion des exceptions globales via un @ControllerAdvice, renvoyant des messages clairs côté frontend ;
* Journalisation des erreurs pour chaque phase de collecte ou découverte.

### 3.6.3 Résilience des modules (tâches planifiées, relances automatiques)

Certains modules (polling SNMP, ping, découverte) sont exécutés via des **tâches planifiées** (@Scheduled) permettant :

* Une **relance automatique** à intervalles réguliers ;
* Un traitement en **asynchrone** sans bloquer l’application ;
* Une **extensibilité facile** en ajustant la fréquence de supervision selon le type d’équipement.

## 3.7 Tests et validation

### 3.7.1 Tests unitaires et d’intégration (backend et frontend)

Le backend est testé avec :

* **JUnit 5** pour les tests unitaires des services et contrôleurs,
* **Mockito** pour le mock des dépendances,
* **Spring Boot Test** pour les tests d’intégration (chargement de contexte, base H2...).

Le frontend React est testé à l’aide de :

* **Jest** et **React Testing Library** pour vérifier les composants critiques (dashboard, alertes, formulaires...).

### 3.7.2 Tests de performance (temps de réponse, charge)

Des tests simples ont été réalisés pour évaluer :

* Le **temps de réponse** des requêtes REST critiques (notamment /devices, /alerts) ;
* La **capacité du backend à supporter un nombre élevé d’équipements** lors du polling (grâce à des appels parallélisés et optimisés).

### 3.7.3 Scénarios de test fonctionnels

Plusieurs **scénarios réels** ont été simulés :

* Détection d’un nouvel équipement via Nmap ;
* Rupture de connectivité simulée via ICMP/SNMP → génération d’alerte DEVICE\_DOWN ;
* Réception d’un Trap SNMP et création d’un TrapEvent lié à une alerte ;
* Exploration interactive des objets MIB depuis l’interface.

Ces scénarios valident la cohérence fonctionnelle du système et sa capacité à réagir aux situations réseau courantes.

Top of Form

Bottom of Form

## 3.8 Difficultés rencontrées et solutions apportées

Le développement de l'application de supervision réseau a été confronté à plusieurs obstacles techniques, notamment liés à la nature des protocoles réseau utilisés, aux performances du système et à la gestion des ressources. Cette section détaille les principaux problèmes rencontrés ainsi que les solutions mises en place.

#### Problème 1 : Découverte lente du réseau

La phase de découverte automatique, notamment sur de grandes plages d’adresses IP, s’est révélée lente lorsqu’elle était effectuée de manière séquentielle. L’exécution série des pings, scans de ports (Nmap) et requêtes SNMP entraînait une latence importante.

**Solution proposée :**

* Mise en œuvre de la parallélisation via des pools de threads, permettant l’exécution simultanée de plusieurs tâches de découverte.
* Utilisation préalable de requêtes ICMP (ping) afin de filtrer les hôtes inaccessibles avant d’engager des ressources plus coûteuses comme SNMP ou Nmap.

#### Problème 2 : Supervision SNMP lente ou inefficace

L’interrogation régulière des équipements via SNMP, notamment pour collecter plusieurs catégories de données (interfaces, système, profils IP/UDP/ICMP), entraînait un ralentissement du backend lorsque les connexions SNMP étaient mal gérées.

**Solution proposée :**

* Intégration de la bibliothèque Apache Commons Pool pour la gestion centralisée des sessions SNMP, avec réutilisation contrôlée des connexions.
* Découplage entre la collecte des données et l’interface utilisateur : les données sont collectées en arrière-plan à intervalles réguliers (polling toutes les 30 secondes) et stockées dans la base de données, consultables à tout moment par l’API sans solliciter directement le réseau.

#### Problème 3 : Blocages liés aux timeouts SNMP

Certains équipements inaccessibles ou mal configurés causaient des blocages lors de requêtes SNMP, en raison de l’absence de réponse ou d’un mauvais paramétrage. Ces timeouts bloquaient les threads de supervision, impactant la stabilité globale du service.

**Solution proposée :**

* Vérification systématique de la disponibilité des hôtes via ICMP avant toute tentative de communication SNMP.
* Mise en place de timeout SNMP stricts et d’une gestion des erreurs robuste pour isoler les équipements problématiques sans impacter l’ensemble du processus.

## 3.9 Interfaces principales (captures d’écran)

* Présentation commentée des vues principales de l’application

## 3.10 Résumé

* Synthèse des réalisations et transition vers la conclusion

# Conclusion

Ce projet de fin d’études nous a permis de concevoir et développer une application de supervision réseau complète, intégrant plusieurs technologies telles que SNMP, ICMP, LLDP et Nmap. En combinant ces outils avec une architecture moderne basée sur Spring Boot et React, nous avons pu mettre en place une solution efficace, évolutive et adaptée aux besoins réels d’un environnement réseau.

Ce travail nous a permis de consolider nos compétences techniques en développement full stack, gestion de protocoles réseau, sécurité, conteneurisation avec Docker, et modélisation de données. Il nous a également confrontés aux défis concrets de la supervision, tels que la gestion des erreurs, l’optimisation des performances ou encore la visualisation en temps réel.

Pour aller plus loin, plusieurs axes d’amélioration peuvent être envisagés :

* **Ajout de technologies de supervision avancées** telles que NetFlow, sFlow ou TR-069, pour enrichir la collecte de données sur le trafic et la configuration des équipements.
* **Intégration d’intelligence artificielle** pour l’analyse comportementale des équipements, la détection d’anomalies, ou la prédiction de pannes (AIOps).
* **Système de notifications amélioré**, avec envoi d’alertes par e-mail, SMS ou messagerie instantanée (Slack, Telegram...).
* **Dashboard personnalisable**, avec des widgets dynamiques et des filtres intelligents.
* **Supervision multi-site ou multi-tenant**, avec isolation des environnements et gestion centralisée.

Ainsi, cette première version pose les fondations d’une plateforme de supervision puissante et ouverte, susceptible d’évoluer vers une solution professionnelle intégrée.

# Références

<https://www.geeksforgeeks.org/icmp-in-computer-network>

<https://standards.ieee.org/ieee/802.1AB/6047/>

<https://mibs.snmplabs.com/>

<https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol>

<http://oid-info.com/>

<https://mibbrowser.online/mibdb_search.php>

<https://nmap.org/>

<https://www.baeldung.com/spring-boot>