# Remerciement

# Avant-propos

# Résumé

# Abstract

# Liste des figures

[Figure 1: SNMP architecture 10](#_Toc201696332)

[Figure 2: logo PRTG 12](#_Toc201696333)

[Figure 3: logo Spring boot et React 12](#_Toc201696334)

[Figure 4: MVC 17](#_Toc201696335)

# Liste des tableaux

Contents

[Remerciement 1](#_Toc201704189)

[Avant-propos 2](#_Toc201704190)

[Résumé 3](#_Toc201704191)

[Abstract 4](#_Toc201704192)

[Liste des figures 5](#_Toc201704193)

[Liste des tableaux 6](#_Toc201704194)

[Liste des abréviations 9](#_Toc201704195)

[Introduction générale 10](#_Toc201704196)

[Chapitre 1 : Présentation du projet 12](#_Toc201704197)

[1.1 Contexte général 12](#_Toc201704198)

[1.2 Problématique 12](#_Toc201704199)

[1.3 Objectifs du projet 12](#_Toc201704200)

[1.4 Cahier des charges fonctionnel 13](#_Toc201704201)

[1.5 Portée du projet 13](#_Toc201704202)

[1.5.1 Ce qui est inclus 13](#_Toc201704203)

[1.5.2 Ce qui est exclu 13](#_Toc201704204)

[1.6 Environnement de développement 14](#_Toc201704205)

[1.6.1 Frontend 14](#_Toc201704206)

[1.6.2 Backend 14](#_Toc201704207)

[1.6.3 Outils utilisés 14](#_Toc201704208)

[1.6.4 Plateformes de déploiement 14](#_Toc201704209)

[Chapitre 2 : Analyse et Conception 15](#_Toc201704210)

[2.1 Analyse des besoins 15](#_Toc201704211)

[2.1.1 Objectifs fonctionnels 15](#_Toc201704212)

[2.1.2 Objectifs non fonctionnels 15](#_Toc201704213)

[2.1.3 Contraintes techniques et environnementales 15](#_Toc201704214)

[2.2 Étude de l’existant 15](#_Toc201704215)

[2.3 Choix technologiques 15](#_Toc201704216)

[2.3.1 Justification de l'utilisation de SNMP 15](#_Toc201704217)

[2.3.2 Rôle de ICMP, LLDP et Nmap dans le système 16](#_Toc201704218)

[2.3.3 Choix de Spring Boot et React pour l’architecture logicielle 16](#_Toc201704219)

[2.4 Architecture du système 16](#_Toc201704220)

[2.4.1 Architecture générale (schéma) 16](#_Toc201704221)

[2.4.2 Architecture logicielle (backend, frontend, API, communication) 17](#_Toc201704222)

[2.4.3 Architecture réseau (agents SNMP, hôtes, routeurs) 17](#_Toc201704223)

[2.5 Modélisation UML 17](#_Toc201704224)

[2.5.1 Diagramme de cas d'utilisation 17](#_Toc201704225)

[2.5.2 Diagramme de classes 18](#_Toc201704226)

[2.5.3 Diagramme de séquence (ex. : découverte d’un équipement) 18](#_Toc201704227)

[2.5.4 Diagramme d’activités (ex. : processus de supervision) 18](#_Toc201704228)

[2.6 Base de données 18](#_Toc201704229)

[2.6.1 Modèle conceptuel (MCD) 18](#_Toc201704230)

[2.6.2 Modèle logique 18](#_Toc201704231)

[2.6.3 Choix du type de base (ex. : relationnelle, NoSQL) 18](#_Toc201704232)

[2.7 Sécurité et fiabilité 18](#_Toc201704233)

[2.7.1 Gestion des erreurs (hôte injoignable, timeout SNMP) 18](#_Toc201704234)

[2.7.2 Sécurité des accès (authentification, REST API sécurisée) 18](#_Toc201704235)

[2.7.3 Résilience et tolérance aux pannes 18](#_Toc201704236)

[Chapitre 3 : Réalisation 19](#_Toc201704237)

[3.1 Introduction 19](#_Toc201704238)

[3.2 Architecture technique 19](#_Toc201704239)

[3.2.1 Architecture logicielle (Spring Boot, React, API REST, WebSockets) 19](#_Toc201704240)

[3.2.2 Architecture réseau simulée ou réelle (topologie utilisée pour les tests) 19](#_Toc201704241)

[3.3 Mise en place du backend 19](#_Toc201704242)

[3.3.1 Configuration du projet Spring Boot 19](#_Toc201704243)

[3.3.2 Gestion des dispositifs (Device Management) 19](#_Toc201704244)

[3.3.3 SNMP Polling et mise à jour des équipements 21](#_Toc201704245)

[3.3.4 Système d’alertes 22](#_Toc201704246)

[3.3.5 Découverte avec ICMP, LLDP et Nmap 23](#_Toc201704247)

[3.3.6 Navigateur MIB (MIB Browser) 24](#_Toc201704248)

[3.4 Développement du frontend 25](#_Toc201704249)

[3.5 Gestion de la persistance 25](#_Toc201704250)

[3.6 Sécurité et robustesse 26](#_Toc201704251)

[3.7 Tests et validation 27](#_Toc201704252)

[3.8 Difficultés rencontrées et solutions apportées 27](#_Toc201704253)

[3.9 Interfaces principales (captures d’écran) 27](#_Toc201704254)

[3.10 Résumé 27](#_Toc201704255)

[Conclusion 28](#_Toc201704256)

# 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.).

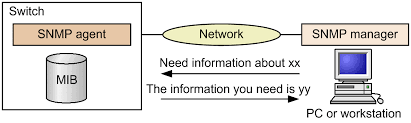


Figure 1: SNMP architecture

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.

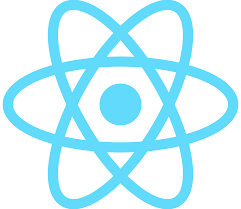
 

Figure 3: logo Spring boot et React

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 :

1. **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.
2. **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.
3. **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.
4. **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 :

* **Réactivité** : l’interface utilisateur doit être fluide et réactive, même sur de grands réseaux ;
* **Fiabilité** : le système doit pouvoir fonctionner de manière autonome et stable sur une longue durée ;
* **Extensibilité** : la solution doit permettre l’ajout futur de protocoles, de modules ou de fonctionnalités supplémentaires ;
* **Sécurité** : les échanges entre les composants, en particulier via l’API, doivent être sécurisés (ex. : authentification, CORS, tokens JWT) ;
* **Portabilité** : grâce à Docker, l’application doit pouvoir être déployée sur différentes machines 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.

**ICMP (Internet Control Message Protocol)** est utilisé pour vérifier la disponibilité d’un hôte via des requêtes de type echo request/reply (ping). Avant d’interroger un équipement via SNMP, une vérification de sa joignabilité permet d’éviter des requêtes inutiles sur des hôtes hors ligne, réduisant ainsi la charge réseau et accélérant les processus de supervision.

**LLDP (Link Layer Discovery Protocol)** permet de découvrir dynamiquement la topologie physique du réseau. En récupérant les informations LLDP des équipements (via SNMP ou directement depuis les trames), il devient possible de cartographier les connexions entre les dispositifs, ce qui est essentiel pour visualiser l'infrastructure et identifier les points de défaillance potentiels.

**Nmap** joue un rôle clé dans la phase de découverte initiale du réseau. Cet outil permet d’identifier les hôtes actifs, les ports ouverts, ainsi que les services en cours d’exécution. Ces informations sont ensuite croisées avec les données SNMP pour cibler les équipements gérables, ou pour enrichir le contexte des métriques collectées.

### 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.

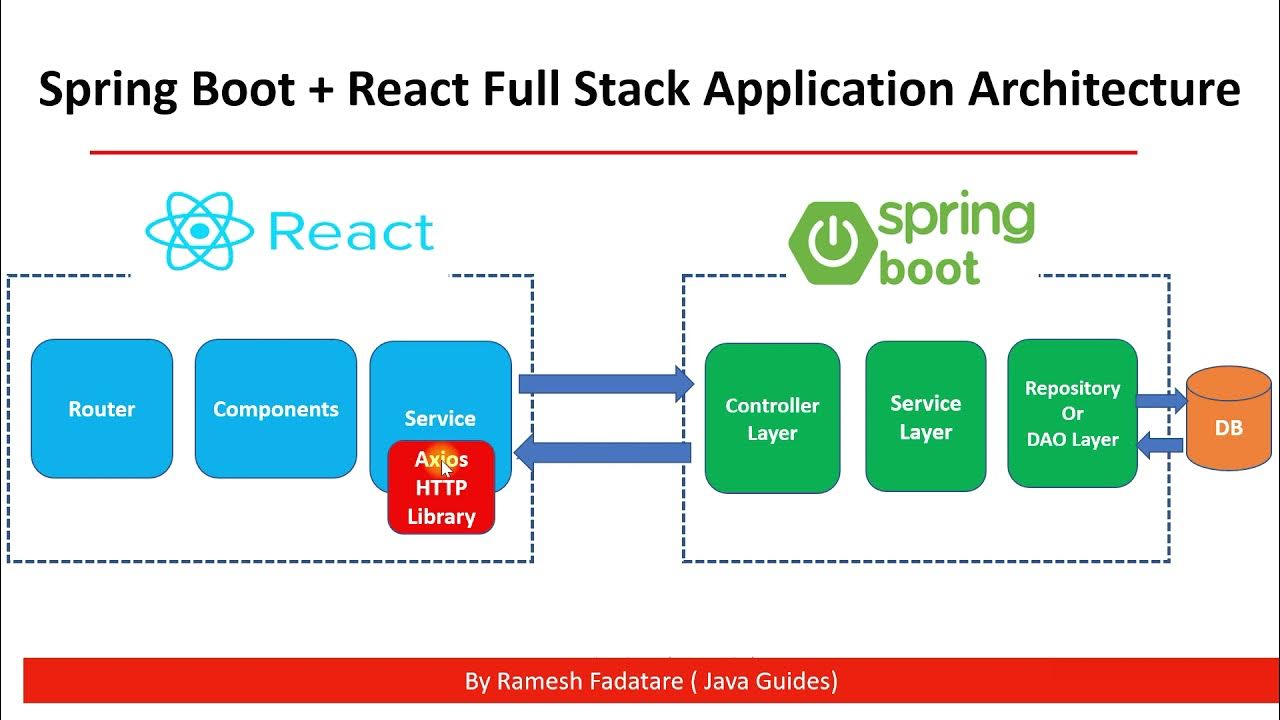


Figure 4: MVC

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

### 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

### 2.5.2 Diagramme de classes

### 2.5.3 Diagramme de séquence (ex. : topologie Discovery)

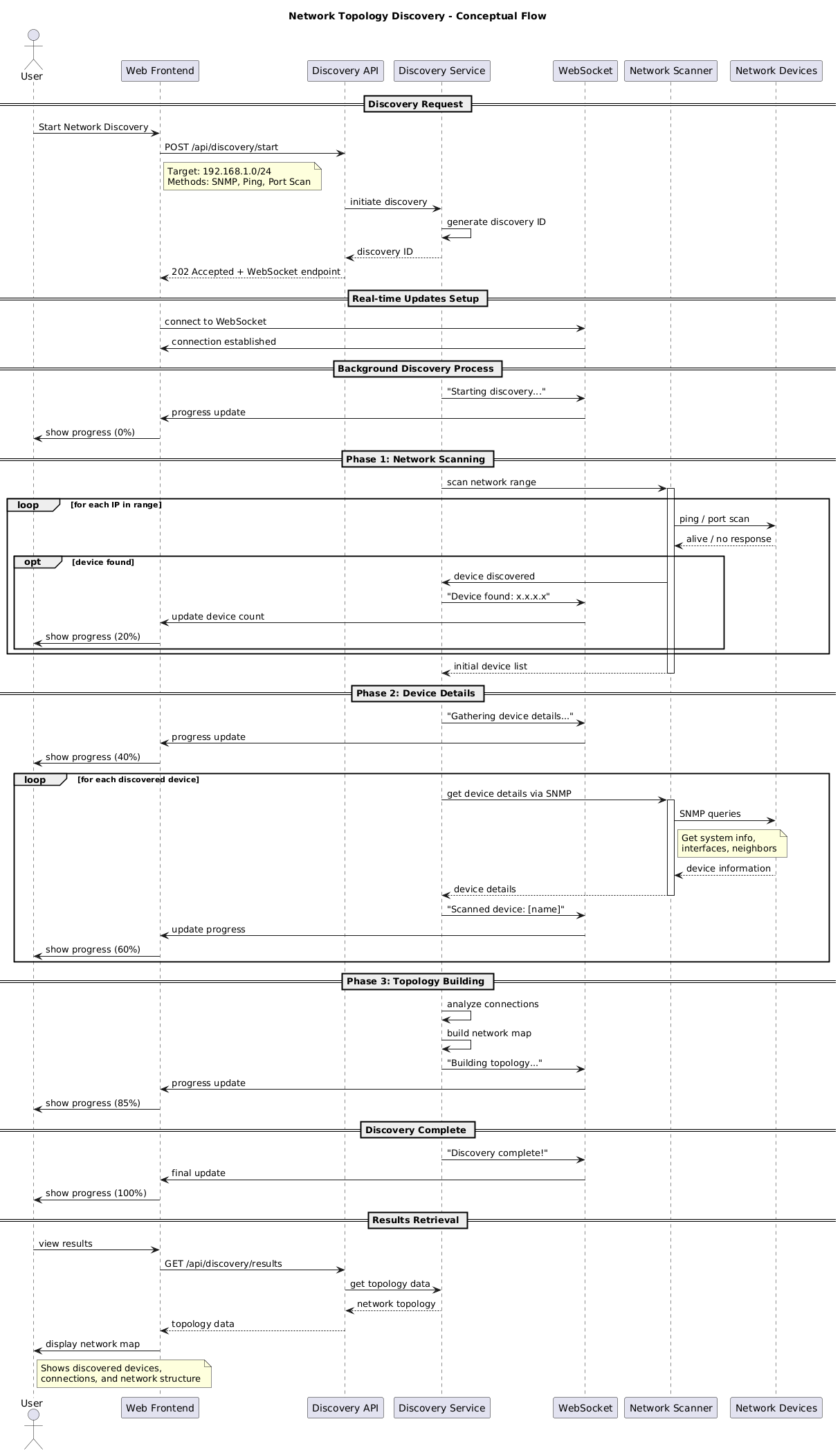


Figure 5 : Simple Diagram Sequence Discovery Topologies

### 2.5.4 Diagramme d’activités (ex. : processus de supervision)

## 2.6 Base de données

### 2.6.1 Modèle conceptuel (MCD)

### 2.6.2 Modèle logique

### 2.6.3 Choix du type de base (ex. : relationnelle, NoSQL)

## 2.7 Sécurité et fiabilité

### 2.7.1 Gestion des erreurs (hôte injoignable, timeout SNMP)

### 2.7.2 Sécurité des accès (authentification, REST API sécurisée)

### 2.7.3 Résilience et tolérance aux pannes

# 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.

#### 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 :

* **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, etc.), 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.

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

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

* **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

* Problèmes techniques, limitations protocolaires, obstacles liés à l’environnement

## 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