

Université Cadi Ayyad  
École Nationale des Sciences Appliquées – Marrakech  
Génie Cyber Défense & Systèmes de Télécommunications Embarqués

## RAPPORT DE PROJET JAVA

---

# Développement d'une plateforme de Mini-SOC basé sur JAVA SpringBoot

---

### Réalisé par :

- BOURHAT Aymane
- NAKAB Mohamed
- KAOUAK ALLAH Ziad
- OUBDA Dimitri-Gaetan
- SANOGO Cheikna

### Encadré par :

- M.Nejeoui Abderrazzak

Année universitaire 2024/2025

# Tables des matières

Tables des matières .....	2
Tables des figures.....	3
<b>1. Introduction.....</b>	<b>4</b>
<b>1.1 Objectif du projet .....</b>	<b>4</b>
<b>1.2 Contexte de mise en œuvre .....</b>	<b>4</b>
<b>1.3 Qu'est-ce qu'un SOC (Security Operations Center) ?.....</b>	<b>5</b>
<b>1.4 Technologies utilisées .....</b>	<b>5</b>
<b>2. Architecture générale du projet .....</b>	<b>7</b>
<b>2.1 Organisation du projet.....</b>	<b>7</b>
<b>2.2 Description détaillée des packages .....</b>	<b>8</b>
<b>2.3 Classe principale.....</b>	<b>10</b>
<b>2.4 Architecture technique (Vue d'ensemble) .....</b>	<b>10</b>
<b>3. Prérequis techniques .....</b>	<b>12</b>
<b>3.1 Environnement de développement.....</b>	<b>12</b>
<b>3.2 Dépendances principales .....</b>	<b>12</b>
<b>3.3 Configuration de la base de données .....</b>	<b>14</b>
<b>3.4 Configuration de l'application.....</b>	<b>15</b>
<b>3.5 Lancement de l'application .....</b>	<b>16</b>
<b>3.6 Outils complémentaires recommandés.....</b>	<b>16</b>
<b>4. Résultats et validation.....</b>	<b>17</b>
<b>4. 1 Présentation de la page de login.....</b>	<b>17</b>
<b>4. 2 Présentation de tableau de bord .....</b>	<b>17</b>
<b>4. 3 Exemple de simulation de logs .....</b>	<b>18</b>
<b>4. 3 Analyse des logs et génération des alertes.....</b>	<b>18</b>
<b>5. Conclusion .....</b>	<b>19</b>
<b>5.1 Résumé des points forts.....</b>	<b>19</b>
<b>5.2 Perspectives d'évolution .....</b>	<b>19</b>
<b>5.3 Remerciements .....</b>	<b>20</b>

## Tables des figures

Figure 1 : Logo JAVA.....	5
Figure 2 : Logo SpringBoot.....	6
Figure 3 : Logo de PostgreSQL .....	6
Figure 4 : Logo de Postman .....	6
Figure 5 : Logo Wireshark.....	6
Figure 6 : Logo du Projet Lombok.....	7
Figure 7 : Architecture fonctionnelle du système de collecte, de traitement et d’affichage des logs dans le mini-SOC.....	11
Figure 8 : Page de connexion .....	17
Figure 9 : Page principale .....	17
Figure 10 : Page pour consulter les logs .....	18
Figure 11 : Page pour l'analyse des logs .....	18

# 1. Introduction

## 1.1 Objectif du projet

Ce projet, intitulé miniSOC, a été réalisé dans le cadre du module Java EE (JEE) de notre formation en cybersécurité. Il s'inscrit dans une démarche pédagogique visant à concilier développement logiciel et sécurité informatique, à travers la conception d'une application modulaire simulant le fonctionnement d'un Security Operations Center (SOC).

L'objectif principal est de mettre en œuvre les fondamentaux de la supervision de sécurité en environnement réseau. Pour cela, miniSOC intègre plusieurs fonctionnalités essentielles :

- La génération de logs simulant des événements réseau réalistes (ex. : connexions SSH, accès HTTP).
- La collecte, le parsing et le stockage de ces logs dans une base de données relationnelle.
- La détection d'anomalies à l'aide de règles simples, simulant une logique de corrélation.
- La visualisation web des logs et alertes via un tableau de bord sécurisé.

Ce projet nous a permis d'appliquer nos compétences en développement **Spring Boot (Java EE)**, tout en explorant les enjeux de **la surveillance et de la réaction face aux menaces** dans un contexte de cybersécurité.

## 1.2 Contexte de mise en œuvre

À l'heure où les menaces numériques deviennent de plus en plus fréquentes et sophistiquées, la mise en place d'une surveillance continue est indispensable pour toute organisation. Les Security Operations Centers (SOCs) jouent un rôle clé dans cette stratégie en assurant la détection et la réponse aux incidents de sécurité.

Toutefois, la conception d'un SOC complet est complexe, coûteuse et difficilement réalisable dans un cadre académique. D'où l'intérêt de miniSOC, qui propose une alternative simplifiée et pédagogique, tout en respectant les logiques de modularité, d'automatisation et de centralisation propres à un SOC.

Ce projet est ainsi une expérimentation pratique permettant :

- D'approfondir notre compréhension du fonctionnement des systèmes de supervision de sécurité.
- De manipuler des concepts concrets tels que les logs, alertes, détections et visualisations de menaces.
- De mobiliser nos acquis en cybersécurité et en développement Java EE dans un contexte réaliste mais maîtrisé.

En résumé, miniSOC constitue un projet formateur à la croisée du développement backend, de la gestion des données, et des pratiques de sécurité opérationnelle.

### 1.3 Qu'est-ce qu'un SOC (Security Operations Center) ?

Un Security Operations Center (SOC) est une structure organisationnelle centrale chargée de la surveillance, de la détection et de la réponse aux incidents de sécurité. Il joue un rôle crucial dans la défense des systèmes d'information d'une entreprise ou d'une organisation.

Ses principales fonctions sont :

- **Collecter et centraliser les logs** provenant de multiples sources
- **Détecter les activités suspectes ou malveillantes**
- **Déclencher des alertes** et orchestrer les réponses appropriées
- **Documenter et analyser les incidents** pour renforcer la posture de sécurité

Dans un environnement professionnel, un SOC s'appuie souvent sur des outils comme les **SIEM** pour automatiser l'analyse, la corrélation et l'investigation.

### 1.4 Technologies utilisées

Pour la réalisation de ce projet, plusieurs technologies ont été mobilisées, en cohérence avec les objectifs pédagogiques du module JEE et les besoins en cybersécurité :

- **Java 21** : langage principal de développement.



*Figure 1 : Logo JAVA*

- **Spring Boot** : framework pour développer des microservices modulaires rapidement et efficacement.
- **Spring Security** : pour la gestion de l'authentification et de l'autorisation.
- **Spring Data JPA** : pour la communication avec la base de données en mode ORM.



*Figure 2 : Logo SpringBoot*

- **PostgreSQL** : système de gestion de base de données relationnelle utilisé pour le stockage des logs.
- **pgAdmin / DBeaver** : outils de gestion de la base PostgreSQL.



*Figure 3 : Logo de PostgreSQL*

- **Postman / Insomnia** : outils de test des API REST développées.



*Figure 4 : Logo de Postman*

- **Wireshark / tcpdump (optionnels)** : pour analyser le trafic réseau simulé.



*Figure 5 : Logo Wireshark*

- **Lombok** : pour simplifier le code Java en réduisant le boilerplate



*Figure 6 : Logo du Projet Lombok*

- **Thymeleaf** : moteur de templates utilisé pour générer dynamiquement les pages HTML.

Ce stack technique nous a permis de mettre en place une architecture cohérente et fonctionnelle, tout en respectant les bonnes pratiques du développement JEE orienté cybersécurité.

## 2. Architecture générale du projet

Le projet suit une architecture modulaire et orientée services, reposant sur le framework Spring Boot. Il est organisé en packages cohérents selon les responsabilités fonctionnelles, ce qui facilite la maintenance, les tests et l'extensibilité.

### 2.1 Organisation du projet

La structure principale du projet est la suivante :

```
src/  
├── main/  
├── java/  
├── ma.ensa.minisoc/  
│   ├── alerts/  
│   ├── auth/  
│   ├── dashboard/  
│   ├── logs/  
│   ├── common/  
└── MinisocApplication.java
```

Chaque sous-package représente un **module fonctionnel** du mini-SOC :

- **alerts** : détection des comportements suspects (ex. brute force) et génération des alertes.
- **auth** : gestion de l'authentification et des utilisateurs avec sécurité JWT.
- **dashboard** : gestion des statistiques et affichage dans une interface.
- **logs** : collecte, simulation et stockage des logs.
- **common** : utilitaires partagés et gestion centralisée des exceptions.

## 2.2 Description détaillée des packages

### ► **ma.ensa.minisoc.alerts**

- **controller/AlertController.java** : expose des endpoints REST pour consulter les alertes.
- **model/Alert.java** : entité représentant une alerte (timestamp, type, description...).
- **service/AlertService.java** : logique métier de détection et gestion des alertes.
- **service/BruteForceDetector.java** : classe appliquant une règle de détection des tentatives de connexion répétées (attaque par force brute).



### ► **ma.ensa.minisoc.auth**

- **controller/AuthController.java** : endpoints pour l'inscription, la connexion et la récupération de jetons JWT.
- **model/User.java, Role.java** : entités de sécurité pour gérer les utilisateurs et leurs rôles.
- **service/UserService.java, AuthService.java** : services métier de gestion des utilisateurs.
- **security/** :
  - **JwtAuthenticationFilter.java** : filtre HTTP pour la validation des requêtes authentifiées.
  - **JwtProvider.java** : génération et validation des tokens JWT.
  - **SecurityConfig.java** : configuration de Spring Security (accès, endpoints publics, etc.).

### ► **ma.ensa.minisoc.dashboard**

- **controller/DashboardController.java** : contrôle l'affichage général du tableau de bord.
- **controller/StatsRestController.java** : API REST pour récupérer des données statistiques.
- **service/DashboardService.java** : regroupe les informations (logs, alertes...) pour les visualiser de manière centralisée.

### ► **ma.ensa.minisoc.logs**

- **controller/LogController.java** : expose des endpoints pour consulter les logs collectés.
- **model/Log.java** : entité représentant une entrée de log (source IP, message, date, etc.).
- **service/** :
  - **LogService.java** : logique métier associée aux logs (stockage, recherche...).
  - **LogSimulationService.java** : permet de simuler des flux de logs.
  - **TcpServer.java** : serveur TCP qui reçoit les logs sur un port spécifique.
  - **TrafficSimulator.java** : génère du trafic simulé à destination du serveur TCP.

### ► **ma.ensa.minisoc.common**

- **exception/GlobalExceptionHandler.java** : gestion centralisée des erreurs HTTP avec des réponses claires.

- **util/DateUtils.java** : méthodes utilitaires pour le traitement de dates (par ex. conversion, formatage).

## 2.3 Classe principale

- **MinisocApplication.java** : point d'entrée principal de l'application Spring Boot, annotée avec `@SpringBootApplication`.

## 2.4 Architecture technique (Vue d'ensemble)

L'architecture suit une logique de traitement **en pipeline** :

### 1. Réception des logs :

- Des logs sont envoyés via un port TCP.
- `TcpServer` les réceptionne et les transmet pour traitement.

### 2. Parsing et stockage :

- Chaque log est transformé (`LogSimulationService`, `Log`) et enregistré en base de données via `LogService`.

### 3. Détection d'incidents :

- Des règles (ex. brute force) sont appliquées à ces données via `AlertService` et `BruteForceDetector`.

### 4. Visualisation :

- Les résultats (logs, alertes) sont accessibles via des API REST (`LogController`, `AlertController`) et intégrés à une interface web (`dashboard`).

### 5. Sécurité :

- Toute la plateforme est protégée par un système JWT avec rôles utilisateur (`auth`).

## 2.5 Diagramme de l'architecture fonctionnelle du mini-SOC

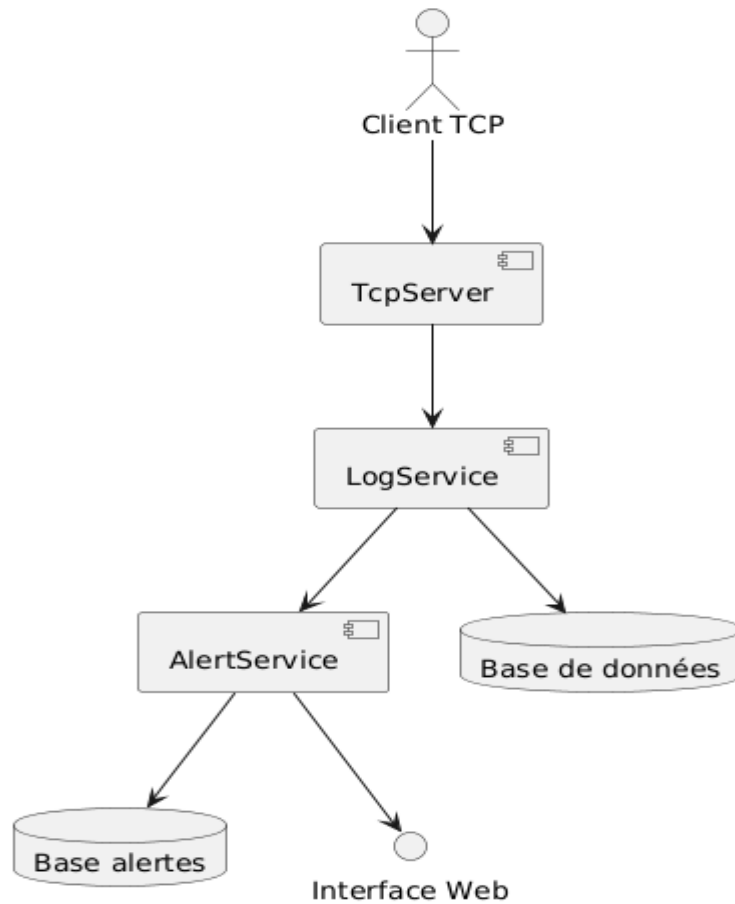


Figure 7 : Architecture fonctionnelle du système de collecte, de traitement et d'affichage des logs dans le mini-SOC

L'architecture de l'application suit un flux de traitement en pipeline où :

- 3- Les Clients TCP se connectent au TcpServer
- 2- Le TcpServer transmet les données au LogService
- 3- Le LogService stocke les logs dans la Base de données
- 4- Le LogService transmet également les alertes à l'AlertService
- 5- L'AlertService persiste les alertes dans la Base alertes
- 6- Enfin, l'AlertService alimente l'Interface Web en données d'alerte »

## 3. Prérequis techniques

Cette section présente les outils, langages, bibliothèques et configurations nécessaires pour exécuter le projet **Mini-SOC** en local ou dans un environnement de test/production.

### 3.1 Environnement de développement

- **Java** : version 21
- **Spring Boot** : version 3.4.5
- **Maven** : gestionnaire de dépendances
- **IDE** : Eclipse
- **Système d'exploitation** : Windows

### 3.2 Dépendances principales

Le projet utilise **Maven** comme gestionnaire de dépendances, avec un ensemble cohérent de bibliothèques Spring Boot et d'outils tiers pour la sécurité, la persistance des données, la validation et le traitement des logs. Voici les principales dépendances incluses dans le fichier `pom.xml` :

Tableau 1 : Tableau des dépendances utilisées

Dépendance	Description
<code>spring-boot-starter-web</code>	Fournit les composants nécessaires à la création d'API REST (contrôleurs, serveur intégré, etc.).
<code>spring-boot-starter-security</code>	Permet d'ajouter une couche de sécurité à l'application (authentification, autorisation).
<code>spring-boot-starter-thymeleaf</code>	Intègre le moteur de templates Thymeleaf pour le rendu côté serveur (tableaux de bord web, vues).

spring-boot-starter-data-jpa	Facilite l'accès aux bases de données relationnelles avec Spring Data et Hibernate.
spring-boot-starter-validation	Gère la validation des entrées via des annotations (ex : <code>@NotNull</code> , <code>@Email</code> , etc.).
spring-boot-starter-mail	Permet l'envoi de courriels (ex. : notifications d'alerte).
thymeleaf-extras-springsecurity6	Intègre la sécurité Spring dans les vues Thymeleaf (affichage conditionnel selon les rôles).
lombok	Réduit le code répétitif via des annotations ( <code>@Getter</code> , <code>@Setter</code> , <code>@Builder</code> , etc.).
jackson-databind	Utilisé pour la sérialisation/désérialisation JSON (conversion entre objets Java et JSON).
commons-io	Fournit des utilitaires pour manipuler les fichiers et les flux d'entrée/sortie.
mysql-connector-j	Pilote JDBC pour les bases de données MySQL (utile en phase de développement ou alternative à PostgreSQL).
postgresql	Pilote JDBC pour la base PostgreSQL (utilisé en production pour la persistance des logs).
spring-boot-devtools	Accélère le développement avec rechargement automatique et autres outils de confort.
spring-boot-starter-test & spring-security-test	Bibliothèques dédiées aux tests unitaires et d'intégration avec support des composants de sécurité.

Ces dépendances sont compatibles avec Java 21 et Spring Boot 3.4.5, comme spécifié dans les propriétés du projet.

### 3.3 Configuration de la base de données

Le système utilise **PostgreSQL** comme base de données relationnelle pour stocker et interroger les logs et les alertes générés ainsi le stockage des identifiants des utilisateurs. La configuration de la connexion à la base de données est centralisée dans le fichier `application.properties`, situé dans le répertoire `src/main/resources`. Cette configuration permet à Spring Boot de se connecter automatiquement à la base et de gérer le cycle de vie des entités JPA.

Voici les principaux paramètres définis :

Tableau 2 : Tableau de configuration de base de données

Propriété	Description
<code>spring.application.name</code>	Nom de l'application Spring, ici défini comme <code>minisoc</code> .
<code>spring.datasource.url</code>	URL JDBC permettant la connexion à la base PostgreSQL : <code>jdbc:postgresql://localhost:5432/minisoc_logs</code> .
<code>spring.datasource.username</code>	Identifiant de l'utilisateur de la base de données : <code>soc_user</code> .
<code>spring.datasource.password</code>	Mot de passe associé à l'utilisateur de la base : <code>mini_soc</code> .
<code>spring.datasource.driver-class-name</code>	Spécifie le driver JDBC PostgreSQL.
<code>spring.jpa.hibernate.ddl-auto</code>	Définit la stratégie de création/mise à jour des tables : ici <code>update</code> , ce qui signifie que les schémas sont automatiquement synchronisés avec les entités JPA sans supprimer les données existantes.

<code>spring.jpa.show-sql</code>	Active l'affichage des requêtes SQL générées par Hibernate dans la console.
<code>spring.jpa.properties.hibernate.dialect</code>	Spécifie le dialecte SQL utilisé par Hibernate pour PostgreSQL.

Cette configuration permet un **démarrage rapide** en développement sans avoir à gérer manuellement les scripts SQL, tout en conservant la compatibilité avec les outils de production.

### 3.4 Configuration de l'application

La configuration de l'application Spring Boot repose sur le fichier `application.properties`, situé dans le répertoire `src/main/resources`. Ce fichier centralise les paramètres essentiels nécessaires au bon fonctionnement du système, notamment l'identité de l'application, les paramètres de connexion à la base de données PostgreSQL, ainsi que les options de persistance des données via JPA/Hibernate.

Voici la configuration utilisée dans le projet miniSOC :

```
# Nom de l'application

spring.application.name=minisoc

# Configuration de la base de données PostgreSQL

spring.datasource.url=jdbc:postgresql://localhost:5432/minisoc_logs

spring.datasource.username=soc_user

spring.datasource.password=mini_soc

spring.datasource.driver-class-name=org.postgresql.Driver

# Paramètres JPA / Hibernate

spring.jpa.hibernate.ddl-auto=update    # Mise à jour automatique du schéma

spring.jpa.show-sql=true                # Affichage des requêtes SQL dans la console

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Cette configuration permet à l'application de se connecter à la base de données `minisoc_logs` en utilisant l'utilisateur `soc_user`. Grâce à l'option `ddl-auto=update`, les entités définies dans le code sont automatiquement synchronisées avec la base de données, ce qui facilite le développement sans avoir à gérer manuellement les scripts SQL.

### 3.5 Lancement de l'application

Une fois l'environnement correctement configuré et les dépendances Maven installées, l'application miniSOC peut être démarrée en suivant les étapes ci-dessous :

1. **Démarrer le serveur PostgreSQL**, puis s'assurer que la base de données `minisoc_logs` est bien créée et accessible avec l'utilisateur `soc_user`.
2. **Ouvrir le projet dans un IDE** compatible avec Spring Boot, tel qu'IntelliJ IDEA, Eclipse ou VS Code.
3. **Lancer la classe principale `MinisocApplication.java`** en tant qu'application Spring Boot. Cette classe se trouve généralement dans le package racine `ma.ensa.minisoc`.
4. Une fois l'application démarrée avec succès, elle sera accessible à l'adresse suivante : <http://localhost:8080>

### 3.6 Outils complémentaires recommandés

Pour faciliter le développement, le test et la supervision de l'application miniSOC, l'utilisation des outils suivants est fortement recommandée :

- **Postman** ou **Insomnia** : pour tester facilement les endpoints REST de l'application (authentification, récupération des logs, consultation des alertes, etc.).
- **DBeaver** ou **pgAdmin** : pour explorer visuellement la base de données PostgreSQL, exécuter des requêtes SQL, et surveiller les tables créées automatiquement par JPA.
- **Wireshark** ou **tcpdump** (*optionnel*) : pour capturer et analyser le trafic réseau simulé. Ces outils peuvent être utiles si vous souhaitez valider la cohérence des logs réseau générés.

Ces outils ne sont pas strictement nécessaires pour faire fonctionner le miniSOC, mais ils permettent de mieux comprendre et diagnostiquer le comportement du système



## 4. Résultats et validation

### 4.1 Présentation de la page de login

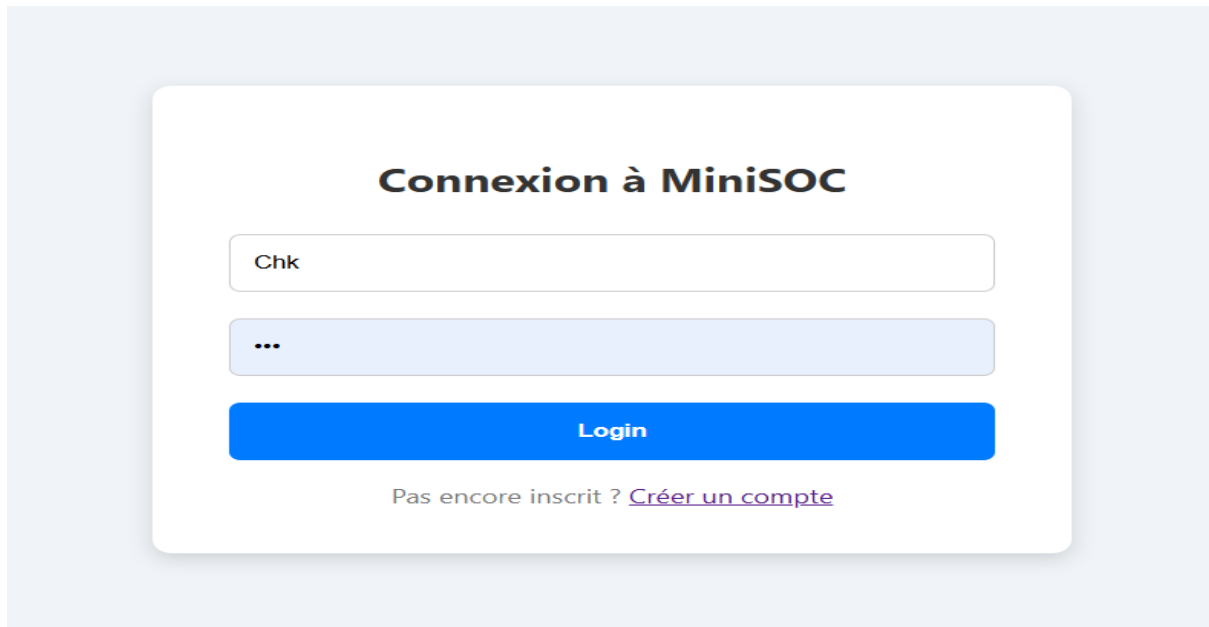


Figure 8 : Page de connexion

### 4.2 Présentation de tableau de bord

- Aperçu de l'interface web

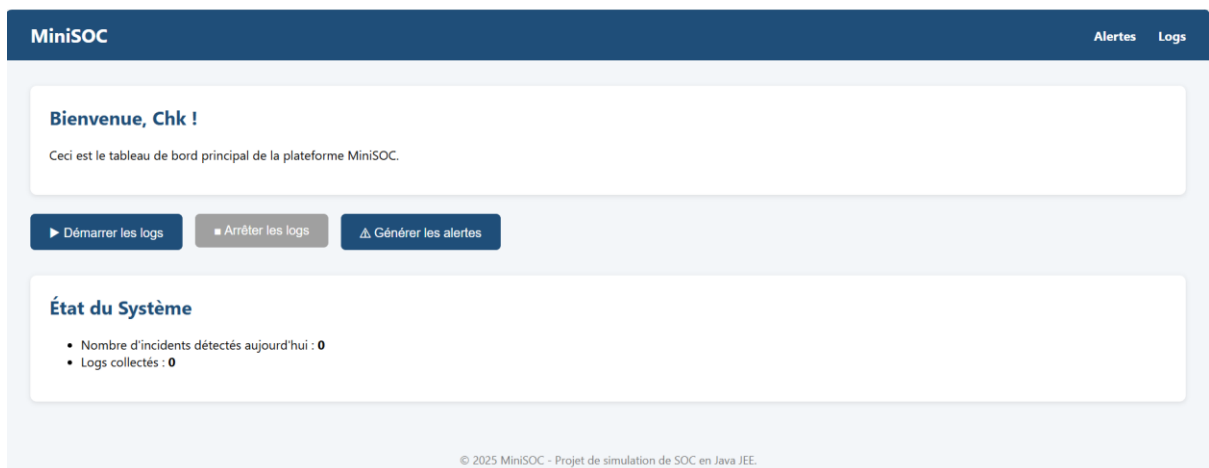
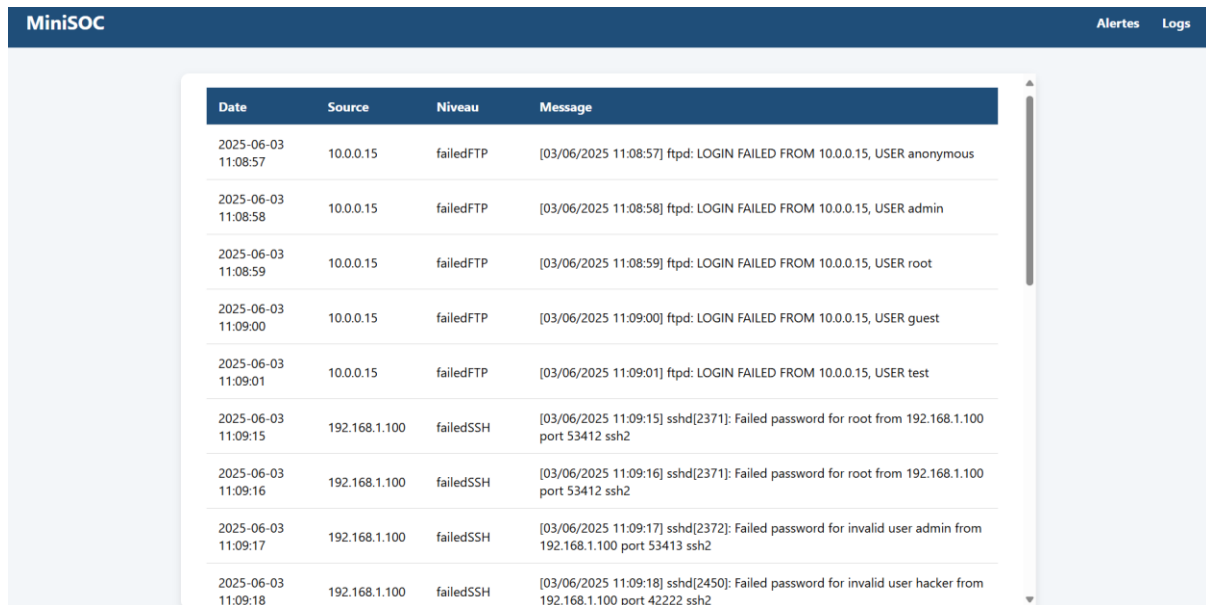


Figure 9 : Page principale

- Fonctionnalités principales visibles (logs, alertes, statistiques...)

## 4. 3 Exemple de simulation de logs



Date	Source	Niveau	Message
2025-06-03 11:08:57	10.0.0.15	failedFTP	[03/06/2025 11:08:57] ftpd: LOGIN FAILED FROM 10.0.0.15, USER anonymous
2025-06-03 11:08:58	10.0.0.15	failedFTP	[03/06/2025 11:08:58] ftpd: LOGIN FAILED FROM 10.0.0.15, USER admin
2025-06-03 11:08:59	10.0.0.15	failedFTP	[03/06/2025 11:08:59] ftpd: LOGIN FAILED FROM 10.0.0.15, USER root
2025-06-03 11:09:00	10.0.0.15	failedFTP	[03/06/2025 11:09:00] ftpd: LOGIN FAILED FROM 10.0.0.15, USER guest
2025-06-03 11:09:01	10.0.0.15	failedFTP	[03/06/2025 11:09:01] ftpd: LOGIN FAILED FROM 10.0.0.15, USER test
2025-06-03 11:09:15	192.168.1.100	failedSSH	[03/06/2025 11:09:15] sshd[2371]: Failed password for root from 192.168.1.100 port 53412 ssh2
2025-06-03 11:09:16	192.168.1.100	failedSSH	[03/06/2025 11:09:16] sshd[2371]: Failed password for root from 192.168.1.100 port 53412 ssh2
2025-06-03 11:09:17	192.168.1.100	failedSSH	[03/06/2025 11:09:17] sshd[2372]: Failed password for invalid user admin from 192.168.1.100 port 53413 ssh2
2025-06-03 11:09:18	192.168.1.100	failedSSH	[03/06/2025 11:09:18] sshd[2450]: Failed password for invalid user hacker from 192.168.1.100 port 42222 ssh2

Figure 10 : Page pour consulter les logs

- Description des types de logs générés (SSH, FTP...)

## 4. 3 Analyse des logs et génération des alertes



Date	Catégorie	Source	Message	Gravité
2025-06-03 11:09:30	Brute Force	10.0.0.15	Tentative de brute force détectée depuis la source : 10.0.0.15	CRITICAL
2025-06-03 11:09:30	Brute Force	192.168.1.100	Tentative de brute force détectée depuis la source : 192.168.1.100	CRITICAL

Figure 11 : Page pour l'analyse des logs

## 5. Conclusion

### 5.1 Résumé des points forts

Ce projet de mini Security Operations Center (miniSOC) a permis de mettre en pratique les fondamentaux de la cybersécurité défensive dans un cadre pédagogique et modulaire. Construit en Java avec le framework Spring Boot, il offre une solution simple mais fonctionnelle pour :

- **Générer et simuler des événements réseau** (connexions SSH, requêtes HTTP, etc.),
- **Parser et stocker les logs** dans une base de données PostgreSQL,
- **Détecter des comportements suspects ou anomalies**,
- **Visualiser les activités et alertes** à travers un tableau de bord web sécurisé.

En plus des aspects techniques (développement backend, structuration en packages, sécurisation par JWT et Spring Security), le projet met en œuvre plusieurs concepts clés en cybersécurité, notamment la surveillance, l'analyse de logs, et la détection d'anomalies.

### 5.2 Perspectives d'évolution

Ce projet, bien qu'élémentaire, constitue une base solide pour des évolutions futures. Plusieurs pistes d'amélioration peuvent être envisagées :

- **Déploiement d'agents de collecte distribués** : pour capter les logs depuis plusieurs machines du réseau local ou distant, les transmettre au serveur central via TCP ou HTTP.
- **Ajout d'un moteur de corrélation ou d'analyse avancée** (basé sur des règles, ou intégrant des techniques d'IA),
- **Intégration avec des sources de logs réelles** (journaux système, agents Syslog, Logstash, etc.),
- **Mécanismes d'alerte en temps réel** (notifications e-mail, webhook, Slack...),
- **Interface de gestion plus complète** (CRUD utilisateurs, filtrage par type de log, export CSV ou JSON),
- **Déploiement dans un environnement cloud, conteneurisé ou virtualisé** (Docker, Kubernetes).

Ces perspectives visent à faire évoluer le miniSOC vers une solution de supervision réseau utilisable dans un contexte plus réaliste ou professionnel.

## 5.3 Remerciements

Nous exprimons notre profonde gratitude Monsieur Nejeoui pour l'accompagnement pédagogique et les conseils techniques fournis tout au long du module. Nous tenons également à le remercier tout particulièrement pour la qualité de l'enseignement du cours de Java et des concepts Java EE, qui nous a permis de structurer et développer ce projet avec rigueur et professionnalisme.

Merci également à nos camarades de promotion pour leur collaboration et leur soutien lors des phases de tests et de mise en œuvre.

Ce projet a été réalisé dans le cadre du module JEE & cybersécurité de notre formation, avec pour objectif de simuler un centre d'opérations de sécurité (SOC) simplifié, afin d'acquérir des compétences concrètes en développement backend sécurisé et supervision des systèmes d'information.