# ENSIAS

## École Nationale Supérieure d'Informatique et d'Analyse des Systèmes

Projet Systèmes distribués

25 Mai 2023

---

# Système Distribué pour la Gestion Intelligente des Données Médicales de la Maladie : Hypertension de Type 2

---

Réalisé par:

Soussou Youness

Touahri Imane

El harnaf Nihad

El khatabbi Oussama

Jury: Mr GUERMAH Hatim

- Année universitaire : 2023/2024 –

Table des Matières

## INTRODUCTION GENERALE

La gestion et la surveillance des patients atteints d'hypertension de type 2 représentent un défi majeur dans le domaine médical. L'hypertension de type 2, caractérisée par une pression artérielle élevée, nécessite une surveillance continue pour prévenir les complications graves telles que les accidents vasculaires cérébraux, les crises cardiaques et les maladies rénales. Avec l'augmentation de l'incidence de cette maladie, il devient essentiel de mettre en place des systèmes efficaces pour surveiller et gérer les données médicales des patients. Ce projet vise à développer un système distribué pour la gestion intelligente des données médicales des patients souffrant d'hypertension de type 2.

Le système utilise une infrastructure de capteurs médicaux pour surveiller en temps réel les signes vitaux des patients, tels que la pression artérielle, le rythme cardiaque, et la température corporelle. L'objectif principal de ce système est de fournir un outil convivial pour les médecins et le personnel médical, leur permettant d'accéder facilement aux données des patients, de visualiser les tendances de santé, et de recevoir des alertes en cas d'anomalies. En outre, le système générera des recommandations basées sur les données médicales pour aider les professionnels de la santé à prendre des décisions éclairées concernant les traitements à administrer.

Pour atteindre ces objectifs, le projet suivra plusieurs étapes clés, notamment l'identification des paramètres médicaux à surveiller, la mise en place des services de surveillance, le stockage et l'analyse des données, ainsi que le développement d'un système d'alerte et de recommandations. En intégrant des technologies avancées telles que JAX-RPC, JAX-WS, JAX-RS, Java RMI, Sockets TCP, Kafka et Graphite, le système distribuera efficacement la gestion des données médicales, assurant ainsi une surveillance continue et proactive des patients atteints d'hypertension de type 2. Ce projet est une étape importante vers l'amélioration de la gestion des maladies chroniques grâce à l'utilisation de systèmes distribués intelligents, offrant ainsi une meilleure qualité de soins aux patients.

# CHAPITRE 1 : PRESENTATION DU PROJET

# INTRODUCTION

Le but de ce chapitre est de présenter le contexte général du projet, qui est réparti sur trois sections. La première concerne la présentation du projet dans son contexte plus spécifique. La seconde section présente les objectifs du projet, en cite la problématique et les objectifs. Tandis que la troisième section traite la Méthodologie du projet, en présentant le choix et l'utilisation des technologies puis la description des étapes du projet.

## 1.1. Contexte général du projet

L'hypertension de type 2, nécessitant une surveillance continue, est mal gérée par les méthodes traditionnelles, entraînant des retards dans la détection des anomalies et une qualité de soins sous-optimale. Le défi réside dans la gestion en temps réel des données provenant de capteurs médicaux répartis. Pour résoudre ce problème, nous proposons un système distribué utilisant des capteurs pour surveiller la pression artérielle, le rythme cardiaque et la température corporelle. Les données seront transmises en temps réel à un tableau de bord centralisé pour les professionnels de la santé. Ce tableau de bord permettra de visualiser les tendances de santé, de générer des alertes en cas d'anomalies et de fournir des recommandations. En utilisant des technologies comme Java RMI, Sockets TCP et Apache Kafka, ce système améliorera la gestion des données médicales, la qualité des soins et permettra une intervention rapide et proactive.

## 1.2. Les objectifs du projet

Les objectifs du projet sont :

-La configuration des alertes ; lorsque certaines conditions sont remplies, comme lorsqu'un capteur détecte une anomalie ou lorsqu'un certain seuil est atteint.

- Gestion Efficace des Données Médicales : Intégrer des technologies avancées comme Java RMI, Sockets TCP et Apache Kafka pour assurer la collecte, le stockage et l'analyse en temps réel des données médicales, améliorant ainsi la gestion et la qualité des soins des patients hypertendus.

## 1.4 Conclusion

Ce chapitre a été une brève présentation du contexte générale du projet ainsi ses objectifs et la méthodologie qu'on a suivie pour les réaliser.

# CHAPITRE 2 : REALISATION DU PROJET

## 2.1 Introduction

Le but de ce chapitre est de présenter l'architecture du projet et la hiérarchie utilisée pour le développement du projet. Ainsi que les différentes étapes du projet, les résultats du projet et leurs interprétations.

## 2.2 Les détails de la maladie Hypertension de type 2

Pour une maladie plus simple comme l'hypertension artérielle (HTA), voici une liste étendue de paramètres numériques à surveiller :

-La pression artérielle systolique normale se situe entre 90 et 120 mmHg, la préhypertension entre 120 et 139 mmHg, l'hypertension de stade 1 entre 140 et 159 mmHg, et l'hypertension de stade 2 à partir de 160 mmHg.

- Pour la pression artérielle diastolique, les valeurs normales sont entre 60 et 80 mmHg, la préhypertension entre 80 et 89 mmHg, l'hypertension de stade 1 entre 90 et 99 mmHg, et l'hypertension de stade 2 à partir de 100 mmHg.

-La fréquence cardiaque au repos normale est de 60 à 100 bpm, et pendant l'exercice modéré, elle devrait être de 50 à 70% de la fréquence cardiaque maximale, estimée à 220 moins l'âge en années.

- L'activité physique recommandée est de 5 000 à 10 000 pas par jour.

- Les lipides sanguins doivent être surveillés avec un cholestérol total idéal inférieur à 200 mg/dL,

-La saturation en oxygène normale se situe entre 95 et 100%, avec une hypoxie en dessous de 90%. La température corporelle normale est de 36.5 à 37.5°C.

## 2.3 Architecture du Projet

Notre architecture repose sur un modèle de producteur-consommateur. Les services web agissent comme des producteurs, collectant et envoyant les données des capteurs dans le système. Ces données sont alors consommées par deux entités distinctes. La première, le système d'alerte, surveille les données en temps réel pour déceler toute anomalie ou tout problème potentiel. La deuxième, le système de recommandation, analyse les données pour fournir des conseils pratiques sur la gestion des cultures et du bétail. Cette architecture nous permet d'assurer une surveillance efficace et précise de l'hypertension .

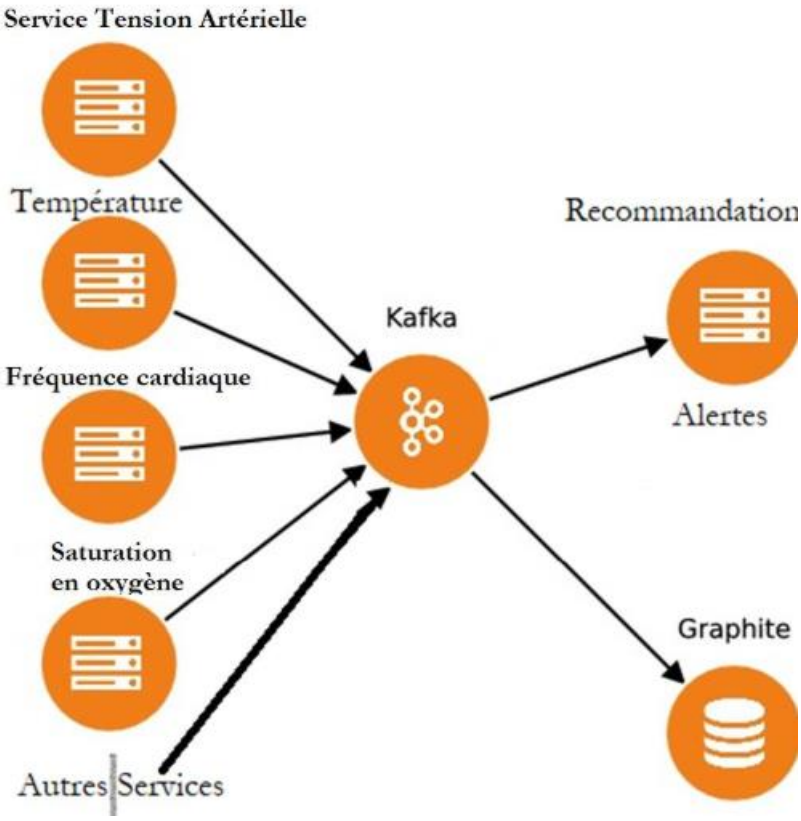


*Figure 2.1: Architecture du système distribué pour une Gestion Intelligente des Données Médicales*

## 2.4 Résultats

### 2.4.1 Les dépendances du projet

Les captures suivantes montrent les dépendances utilisées dans ce projet.

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
<version>3.2.5</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework.kafka/spring-kafka -->
<dependency>
<groupId>org.springframework.kafka</groupId>
<artifactId>spring-kafka</artifactId>
<version>3.1.4</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework.kafka/spring-kafka-test -->
<dependency>
<groupId>org.springframework.kafka</groupId>
<artifactId>spring-kafka-test</artifactId>
<version>3.1.4</version>
<scope>test</scope>
</dependency>
</dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

*Figure 2.2: Les dépendances du projet*

```
    <dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients -->
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>3.7.0</version>
</dependency>

<dependency>
<groupId>com.sun.xml.ws</groupId>
<artifactId>jaxws-rt</artifactId>
<version>2.3.2</version>
</dependency>
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-simple</artifactId>
<version>1.7.25</version>
</dependency>
<!--  https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-test  -->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<version>3.2.5</version>
<scope>test</scope>
</dependency>
<!--  https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web  -->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
<version>3.2.5</version>
</dependency>
```

*Figure 2.3: Les dépendances de kafka et jax-ws*

### 2.4.2 Lancement des Serveurs utilisées

-Zookeper

```
[2024-05-19 11:20:33,708] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsP
rovider@3d680b5a (org.apache.zookeeper.server.ServerMetrics)
[2024-05-19 11:20:33,707] INFO ACL digest algorithm is: SHA1 (org.apache.zookeeper.server.auth.DigestAuthenticationProvi
der)
[2024-05-19 11:20:33,708] INFO zookeeper.DigestAuthenticationProvider.enabled = true (org.apache.zookeeper.server.auth.D
igestAuthenticationProvider)
[2024-05-19 11:20:33,713] INFO zookeeper.snapshot.trust.empty : false (org.apache.zookeeper.server.persistence.FileTxnSn
apLog)
[2024-05-19 11:20:33,729] INFO  (org.apache.zookeeper.server.ZooKeeperServer)
[2024-05-19 11:20:33,730] INFO   ______                  _                                          (org.apache.zookeep
er.server.ZooKeeperServer)
[2024-05-19 11:20:33,730] INFO  |___  /                 | |                                         (org.apache.zookeep
er.server.ZooKeeperServer)
[2024-05-19 11:20:33,731] INFO     / /    ___     ___   | | __   ___    ___   _ __     ___    _ __    (org.apache.zookeep
er.server.ZooKeeperServer)
[2024-05-19 11:20:33,732] INFO    / /    / _ \   / _ \  | |/ /  / _ \  / _ \ | '_ \   / _ \  | '__|  (org.apache.zookeeper
.server.ZooKeeperServer)
[2024-05-19 11:20:33,733] INFO   / /__  | (_) | |  (_) | |   <  |  __/ |  __/ | |_) | |  __/  | |     (org.apache.zookeepe
r.server.ZooKeeperServer)
[2024-05-19 11:20:33,733] INFO  /_____|  \___/   \___/  |_|\_\  \___|  \___| | .__/   \___|  |_| (org.apache.zookeeper.se
rver.ZooKeeperServer)
[2024-05-19 11:20:33,734] INFO                                               | |                    (org.apache.zookee
per.server.ZooKeeperServer)
[2024-05-19 11:20:33,734] INFO                                               |_|                    (org.apache.zookee
per.server.ZooKeeperServer)
[2024-05-19 11:20:33,736] INFO  (org.apache.zookeeper.server.ZooKeeperServer)
[2024-05-19 11:20:38,284] INFO Server environment:zookeeper.version=3.8.3-6ad6d364c7c0bcf0de452d54ebefa3058098ab56, buil
t on 2023-10-05 10:34 UTC (org.apache.zookeeper.server.ZooKeeperServer)
[2024-05-19 11:20:38,285] INFO Server environment:host.name=DESKTOP-4OS1L54 (org.apache.zookeeper.server.ZooKeeperServer
)
```

*Figure 2.4: Lancement Zookeper*

-Kafka

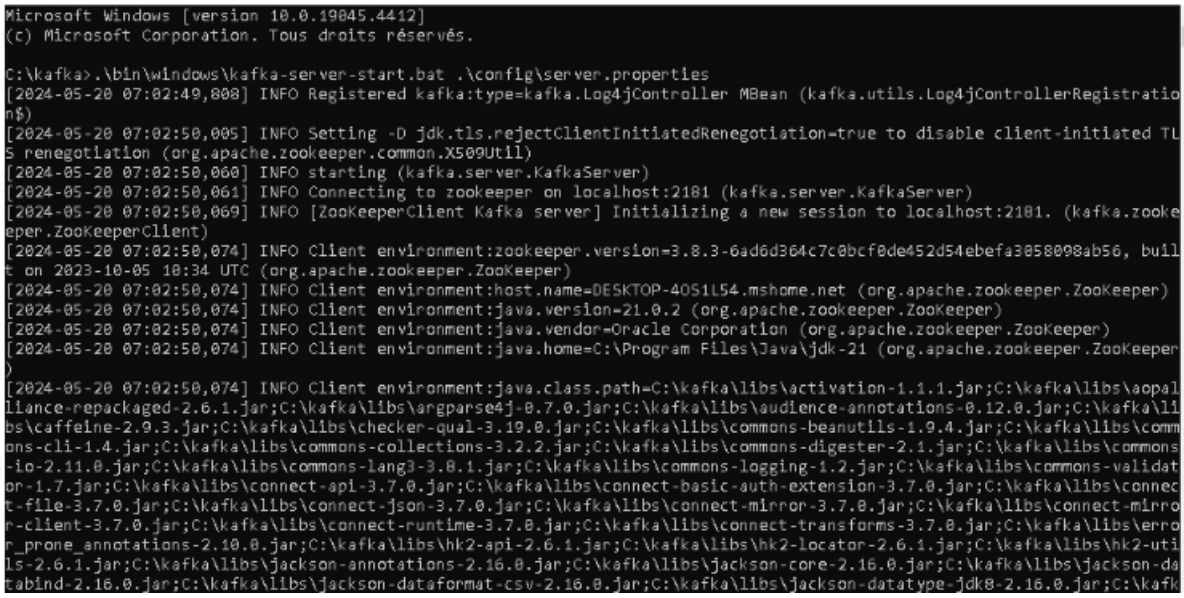


*Figure 2.5: Lancement de Kafka*

-Spring Boot

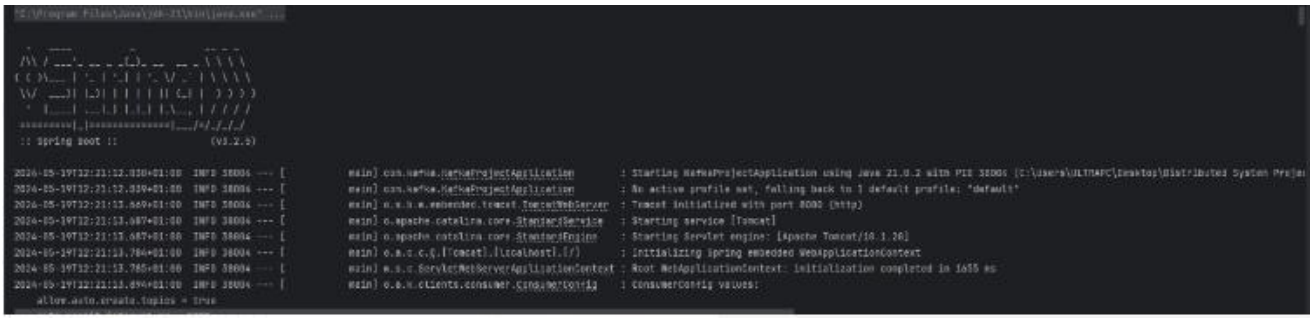

*Figure 2.6: Lancement de Spring Boot*

## 2.4.3 La hiérarchie du projet

Voici la hiérarchie utilisée pour le développement du projet:

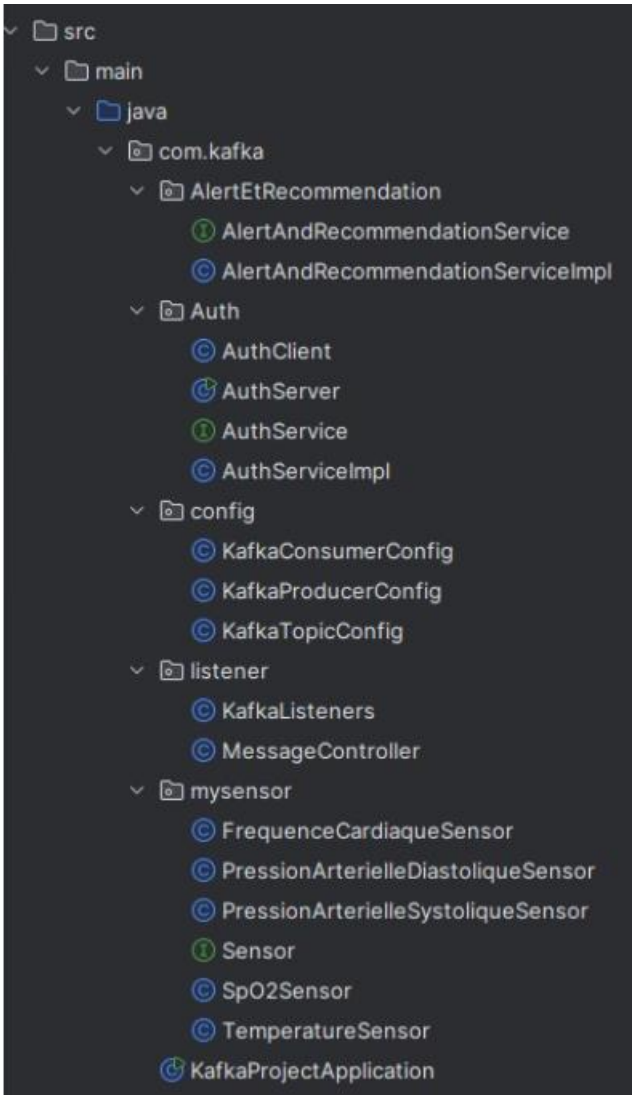


*Figure 2.7: Hiérarchie du projet*

### 2.4.4 l'approche utilisée pour les services web

Nous avons utilisée l'approche Bottom/Up étudié dans le cours

Approche Bottom /Up ou Code First

L'approche Bottom / Up consiste à démarrer le développement à partir d'une classe Java (POJO)

- Ajouter l'annotation @WebService
- Déployer l'application sur un serveur d'application
- Le document WSDL est généré automatiquement en respectant les valeurs par défauts

Les Etapes d'implémentation de l'approche :

1. Etape optionnel : Création de l'interface java du service
2. Etape obligatoire : Création de l'implémentation du service
3. Etape obligatoire: Publication du service web

*Figure 2.8: Approche Bottom Up*

### 2.4.5 Authentification RMI

```
public interface AuthService extends Remote {  5 usages  1 implementation
    boolean authenticate(String username, String password) throws RemoteException;  1 usage  1 implementation
}
```

*Figure 2.8 : Création d'interface RMI*

```
public class AuthServer {
    public static void main(String[] args) {
        try {
            LocateRegistry.createRegistry( port: 1099);
            AuthService authService = new AuthServiceImpl();
            Naming.rebind( name: "rmi://localhost/AuthService", authService);
            System.out.println("AuthServer is running...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

*Figure 2.9 :Création du serveur RMI*

```
@Configuration  no usages
public class AuthClient {

    @Bean(name = "authServiceClient")  no usages
    public RmiProxyFactoryBean authService() {
        RmiProxyFactoryBean proxy = new RmiProxyFactoryBean();
        proxy.setServiceUrl("rmi://localhost:1099/AuthService");
        proxy.setServiceInterface(AuthService.class);
        return proxy;
    }
}
```

*Fifure2.10 :Création du client RMI*

```java
public class AuthServiceImpl extends UnicastRemoteObject implements AuthService {  1 usage

    protected AuthServiceImpl() throws RemoteException {  1 usage
        super();
    }

    @Override  1 usage
    public boolean authenticate(String username, String password) throws RemoteException {
        // Logique d'authentification simple
        return "user".equals(username) && "password".equals(password);
    }
}
```

*Figure 2.11 : Implémentation de service*

## 2.4.6 La classe Contrôleur

Ce classe définit un contrôleur REST qui expose trois points de terminaison pour récupérer des messages, des alertes et des recommandations.

```java
@RequestMapping("/kafka-api-v1")  no usages
@RestController
public class MessageController {

    @Autowired  1 usage
    KafkaListeners listener;

    @Autowired  2 usages
    AlertAndRecommendationServiceImpl alertAndRecommendationManager;

    @GetMapping("/messages")  no usages
    @ResponseBody
    public List<String> getMessages() { return listener.getMessages(); }

    @GetMapping("/alerts")  no usages
    @ResponseBody
    public List<String> getAlerts(){
        List<String> alerts=alertAndRecommendationManager.getAlerts();
        return alerts;
    }

    @GetMapping("/recommendations")  no usages
    @ResponseBody
    public List<String> getRecommendations(){
        List<String> recommendations=alertAndRecommendationManager.getRecommendations();
        return recommendations;
    }

}
```

*Figure 2.12: La classe Contrôleur*

## 2.4.7 Visualisation des données

On peut voir les données détéctées par les capteurs dans la console ou bien via des interfaces web accessibles dans le port 8080. Ces données sont générées chaque cinq secondes.

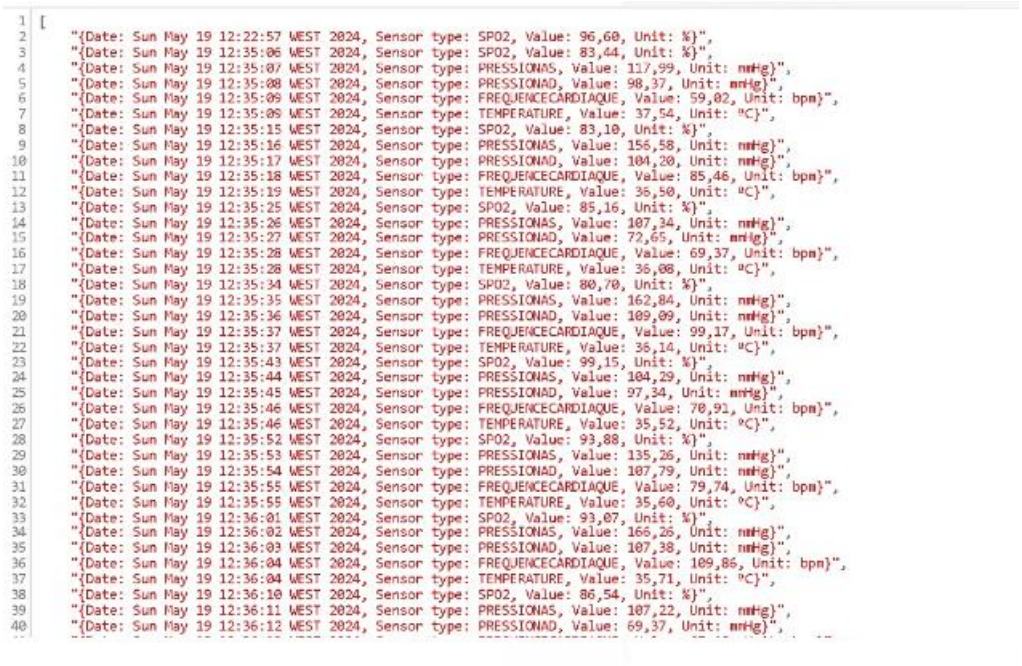


*Figure 2.13: Les messages via l'interface web*

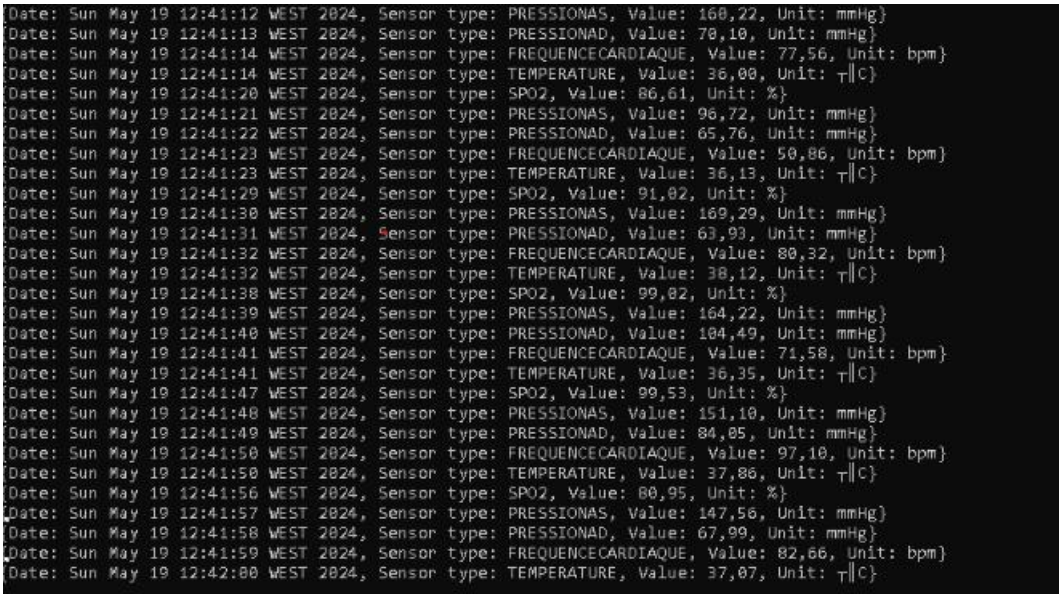


*Figure 2.14: Les messages dans la console*

### 2.4.8 Les alertes et les recommandations

- A partir de la console :


*Figure 2.15: Les alertes et recommandations dans la console*

## 2.5 Les outils utilisés

Dans cette section, nous allons présenter les outils que nous avons utilisés pour réaliser ce projet.

### 2.5.1 Java :

*Figure 2.16: Java logo*

Java est un langage de programmation polyvalent et populaire, largement utilisé dans le développement d'applications, de logiciels et de solutions informatiques. Il a été développé comme un langage portable; il peut fonctionner sur différents systèmes d'exploitation tels que Windows, macOS et Linux. Il est utilisé dans une multitude de domaines, notamment le développement d'applications mobiles, d'applications d'entreprise, de systèmes embarqués, de jeux vidéo, d'applications web et bien d'autres. Il dispose d'une vaste bibliothèque standard (API Java) avec de nombreuses fonctionnalités prêtes à l'emploi qui accélèrent le processus de développement. Ainsi il utilise une approche orientée objet pour facilité la réutilisation du code et permet de développer des applications modulaires et évolutives.

### 2.5.2 Apache Kafka

*Figure 2.17 : Apache kafka logo*

Apache Kafka est une plateforme de streaming de données open source, développée par la Fondation Apache. Elle fournit une architecture de messagerie distribuée pour la gestion de flux de données en temps réel. Kafka est conçu pour traiter de grandes quantités de données et permet une communication fiable, performante et évolutive entre les différents composants d'un système distribué.Il peut être utilisé pour divers cas d'utilisation tels que le traitement de flux, la gestion des logs, l'ingestion de données, les pipelines de données, les architectures de microservices, etc.

### 2.5.3 Zookeeper


*Figure 2.18: Zookeeper logo*

Zookeeper est un logiciel de haut niveau développé par Apache qui agit comme un service centralisé et est utilisé pour maintenir les données de nommage et de configuration et pour fournir une synchronisation flexible et robuste au sein des systèmes distribués. Zookeeper garde la trace de l'état des nœuds du cluster Kafka et il garde également la trace des sujets Kafka, des partitions, etc. Zookeeper lui-même permet à plusieurs clients d'effectuer des lectures et des écritures simultanées et agit comme un service de configuration partagé au sein du système. Le protocole Zookeeper atomic broadcast (ZAB) est le cerveau de tout le système, permettant à Zookeeper d'agir comme un système de diffusion atomique et d'émettre des mises à jour ordonnées.

### 2.5.6 Spring boot


*Figure 2.19: Spring boot logo*

Spring Boot est un framework Java puissant et polyvalent, largement utilisé pour le développement d'applications autonomes et prêtes à l'emploi. Grâce à sa configuration automatique intelligente, il permet de réduire considérablement le temps et les efforts nécessaires pour démarrer un projet. Spring Boot offre une grande flexibilité en matière de choix de base de données, d'intégration avec des technologies telles que Hibernate et Spring Data, et de création d'API RESTful. Il facilite également la gestion des dépendances grâce à Maven ou Gradle, tout en offrant un serveur Web intégré pour le déploiement facile des applications. Avec sa philosophie basée sur les conventions plutôt que sur la configuration,

Spring Boot permet aux développeurs de se concentrer sur le développement de fonctionnalités de haute qualité, tout en assurant la robustesse, la scalabilité et la performance des applications Java.

## 2.6 Conclusion

Dans ce chapitre, nous avons détaillé la réalisation du projet en commençant par la présentation de l'hypertension de type 2 et les paramètres essentiels à surveiller pour cette maladie. Nous avons ensuite décrit l'architecture du système distribué, basé sur un modèle de producteur-consommateur, qui permet une surveillance efficace et précise des données médicales en temps réel. Les différentes étapes du projet, les outils utilisés, ainsi que les résultats obtenus ont été présentés, illustrant la mise en œuvre concrète et l'efficacité de notre système. L'approche Bottom-Up a été utilisée pour développer les services web, assurant une gestion optimale des données et des alertes. Ce projet démontre comment l'intégration de technologies avancées comme Java, Apache Kafka, et Spring Boot peut améliorer significativement la gestion des maladies chroniques, offrant ainsi une meilleure qualité de soins et une intervention médicale plus rapide et proactive.

## CONCLISION GENERAL

Ce projet de système distribué pour la gestion intelligente des données médicales des patients atteints d'hypertension de type 2 a démontré l'efficacité et la nécessité d'intégrer des technologies avancées dans le domaine médical. En utilisant des capteurs pour surveiller en temps réel des paramètres critiques tels que la pression artérielle, la fréquence cardiaque et la température corporelle, nous avons pu créer un tableau de bord centralisé permettant aux professionnels de la santé de visualiser les données, détecter les anomalies et recevoir des recommandations basées sur des analyses approfondies.

L'architecture du système, reposant sur un modèle de producteur-consommateur avec des technologies telles que Java RMI, Sockets TCP et Apache Kafka, assure une collecte, un stockage et une analyse efficaces des données. Les résultats obtenus montrent que ce système peut améliorer considérablement la qualité des soins en permettant une intervention rapide et proactive des médecins, réduisant ainsi les risques associés à l'hypertension.

En conclusion, ce projet illustre comment les systèmes distribués peuvent révolutionner la gestion des maladies chroniques, en offrant une surveillance continue et une gestion intelligente des données médicales. Cela ouvre la voie à des innovations futures dans le domaine de la santé, où la technologie joue un rôle crucial dans l'amélioration des soins aux patients.