Cahier de Charge : Application Avancée pour la Détection Automatique des Instruments de Mesure de Bus

# 1. Introduction

Ce projet vise à développer une application avancée permettant la détection automatique et la gestion des instruments de mesure de bus dans divers environnements. L'application sera conçue pour garantir une communication efficace, une interface utilisateur intuitive et des fonctionnalités analytiques pour maximiser l'utilité des données recueillies.

# 2. Contexte

Avec la montée en puissance des instruments de mesure modernes, la nécessité d’un système capable de les détecter et de les gérer efficacement devient cruciale. Ce projet ambitionne d’automatiser cette détection tout en offrant des outils d’analyse et de visualisation des données en temps réel.

# 3. Objectifs du Projet

* Développer une application capable de détecter automatiquement les instruments de mesure de bus sur le réseau.
* Fournir une interface utilisateur intuitive permettant aux utilisateurs de visualiser, analyser et interagir avec les données des instruments.
* Assurer une communication sécurisée et fiable entre l'application et les instruments.

# 4. Fonctionnalités Principales

* **Détection Automatique des Instruments** :
  + Utiliser des protocoles de communication tels que mDNS et UPnP pour détecter les appareils sur le réseau.
* **Interface Utilisateur Interactive** :
  + Utiliser des Framework modernes (React, Angular, ou Vue.js) pour une expérience utilisateur fluide.
  + Dashboard en temps réel affichant les données des instruments détectés, avec possibilité de filtrer par type d'instrument.
* **Analytique et Visualisation** :
  + Intégrer des bibliothèques de visualisation comme Chart.js ou D3.js pour afficher les données sous forme de graphiques.
  + Fonctionnalité d’exportation de données (CSV, PDF) pour les rapports.
* **Notifications et Alertes** :
  + Système de notifications pour alerter les utilisateurs sur des événements critiques (ex : instrument hors ligne, anomalie détectée).

# 5. Architecture du Système

* **Architecture Générale** :

(Insérez ici votre propre diagramme d'architecture en utilisant les éléments décrits ci-dessous)

**Composants de l’Architecture :**

* + **Frontend** : Application web développée en React/Angular/Vue.js.
    - **Composants** : Interface utilisateur, gestion des instruments, authentification.
  + **Backend** : API RESTful développée en C# & .NET.
    - **Composants** : Serveur Web Socket, module de découverte des instruments, gestion des erreurs.
  + **Base de Données** : Base de données SQL pour stocker les informations des utilisateurs et des instruments.
    - **Tables** : Utilisateurs, Instruments, Logs.
  + **Cloud Infrastructure** : Utilisation de AWS ou Azure pour l'hébergement et la scalabilité.
    - **Composants** : Load Balancer, services de stockage, monitoring.

# 6. Technologies Utilisées

* **Backend** : C# & .NET
* **Frontend** : Choix entre React, Angular ou Vue.js
* **Protocoles de Communication** : WebSocket, HTTP/REST API, MQTT
* **Gestion des Données** : JSON pour l'échange de données, Protocol Buffers pour la sérialisation efficace.
* **Base de Données** : SQL (MySQL, PostgreSQL, etc.)
* **Cloud** : AWS ou Azure pour l’hébergement

# 7. Sécurité

* **HTTPS** : Utiliser HTTPS pour la communication sécurisée.
* **Authentification** : Implémenter OAuth ou JWT pour sécuriser l'accès.
* **Validation des Entrées** : Assurer la validation des données pour prévenir les attaques par injection.

# 8. Tests et Surveillance

* **Tests Automatisés** : Utiliser Jest ou Mocha pour les tests unitaires et d'intégration.
* **Surveillance** : Outils comme Prometheus ou Grafana pour le suivi des performances et des alertes en temps réel.

# 9. Gestion des Erreurs

* **Mécanisme de Gestion des Erreurs** :
  + Utilisation de blocs try-catch et d’un système de logs pour capturer et analyser les erreurs.
  + Définir des codes d'erreur avec messages explicites pour une meilleure traçabilité.

# 10. Déploiement

* **Hébergement** : Utiliser des services cloud (AWS, Azure) pour le déploiement.
* **Containerisation** : Utiliser Docker pour empaqueter l'application, facilitant la gestion des dépendances et le déploiement.

# 11. Calendrier de Projet

* **Semaine 1-2** : Analyse des besoins et conception de l’architecture
* **Semaine 3-6** : Développement du Backend
* **Semaine 6-9** : Développement du Frontend
* **Semaine 10-11** : Intégration et tests
* **Semaine 12-13** : Documentation et préparation de la présentation
* **Semaine 14-15** : Phase de feedback et ajustements finaux

# 12. Conclusion

Ce projet vise à fournir une solution complète pour la détection et la gestion des instruments de mesure de bus. En combinant une interface utilisateur intuitive avec une puissante logique backend, l'application répondra aux besoins croissants de gestion automatisée des instruments.