Harvest Hub

Rapport d'Analyse des Technologies pour un Système de Sondes Connectées

Introduction

Un système de sondes connectées repose sur des technologies permettant de collecter, traiter et transmettre des données issues de capteurs. Ce rapport analyse les principaux langages de programmation pouvant être utilisés pour implémenter un tel système, en mettant en avant leur domaine d'application, leurs avantages et leurs inconvénients.

Partie 1 : Technologies pour le Développement Web et Mobile

1. Technologies pour l'Application Web

- **React.js** : Utilisé pour créer une interface utilisateur dynamique et interactive.
    - Exemples d'applications : Facebook, Instagram.
    - Priorisé pour : Sa réactivité et la réutilisation des composants.
    - Avantages : Performance élevée, réutilisation des composants, large écosystème.
    - Inconvénients : Complexité de la gestion de l'état, courbe d'apprentissage pour les débutants.

- **Angular** : Un framework complet pour les applications web.
    - Exemples d'applications : Google, Gmail.
    - Priorisé pour : Son architecture robuste et évolutive.
    - Avantages : Architecture robuste, bonne gestion des performances.
    - Inconvénients : Plus lourd que React, syntaxe plus complexe.

- **Vue.js** : Framework progressif pour les applications web.
    - Exemples d'applications : Alibaba, Xiaomi.
    - Priorisé pour : Sa simplicité et légèreté.
    - Avantages : Facilité d'apprentissage, léger et performant.
    - Inconvénients : Moins utilisé pour les grandes applications complexes.

2. Technologies pour l'Application Mobile

- **Flutter (Dart)** : Développement d'applications mobiles multi-plateformes.
    - Exemples d'applications : Google Ads, BMW.
    - Priorisé pour : Son interface graphique performante et unifiée.

    - Avantages : UI performante, compilation native rapide.
    - Inconvénients : Taille des applications plus grande.

- **React Native** : Développement hybride performant.
    - Exemples d'applications : Facebook, Airbnb.
    - Priorisé pour : Son développement rapide avec un code unique.
    - Avantages : Code unique pour iOS et Android, large communauté.
    - Inconvénients : Performances inférieures aux applications natives.

- **Swift (iOS) et Kotlin (Android)** : Développement natif.
    - Exemples d'applications : Uber (Kotlin), LinkedIn (Swift).
    - Priorisé pour : L’optimisation et l’intégration native avec les OS mobiles.
    - Avantages : Performance optimisée, accès direct aux fonctionnalités natives.
    - Inconvénients : Temps de développement plus long.

Partie 2 : Technologies pour les Sondes et le Hub

1. Langages de Programmation (API + firmeware)

C/C++

- Exemples d'applications : Arduino, ESP32.
- Priorisé pour : Ses performances et son faible coût énergétique.
- Domaine d'application : Logiciels embarqués et systèmes à faible consommation d'énergie.
- Avantages : Haute performance, accès direct au matériel.
- Inconvénients : Complexité du développement et gestion manuelle de la mémoire.

Python

- Exemples d'applications : Raspberry Pi, Machine Learning (TensorFlow).
- Priorisé pour : Son accessibilité et son large écosystème.
- Domaine d'application : Prototypage rapide, traitement des données.
- Avantages : Lisibilité du code, bibliothèques riches.
- Inconvénients : Moins performant que C/C++ pour le temps réel.

Java

- Exemples d'applications : Android, Serveurs IoT.
- Priorisé pour : Sa portabilité et robustesse.
- Domaine d'application : Cloud, middleware.
- Avantages : Sécurité, large écosystème.
- Inconvénients : Consommation mémoire élevée.

JavaScript (Node.js)

- Exemples d'applications : PayPal, Netflix.
- Priorisé pour : Son exécution asynchrone et sa rapidité en backend.
- Domaine d'application : Interfaces web, serveurs backend.
- Avantages : Performances concurrentielles, large choix de bibliothèques.

- Inconvénients : Moins adapté aux systèmes embarqués.

Rust

- Exemples d'applications : Firefox, AWS.
- Priorisé pour : Sa gestion sécurisée de la mémoire et performance.
- Domaine d'application : Applications embarquées critiques.
- Avantages : Sécurité mémoire, performance proche de C/C++.
- Inconvénients : Courbe d’apprentissage élevée.

Go (Golang)

- Exemples d'applications : Docker, Kubernetes.
- Priorisé pour : Sa simplicité et performance réseau.
- Domaine d'application : Architectures distribuées, cloud.
- Avantages : Rapidité de compilation, gestion efficace des communications réseau.
- Inconvénients : Moins adapté aux systèmes embarqués.

2. Protocoles de Communication

Wi-Fi

- Utilisé dans : Les hubs et sondes connectés aux réseaux domestiques.
- Priorisé pour : Sa portée et sa compatibilité avec les infrastructures existantes.
- Avantages : Débit élevé, large adoption.
- Inconvénients : Consommation énergétique importante.

Bluetooth Low Energy (BLE)

- Utilisé dans : Les capteurs nécessitant une faible consommation d'énergie.
- Priorisé pour : Son efficacité énergétique et sa compatibilité mobile.
- Avantages : Faible consommation, bon pour les connexions intermittentes.
- Inconvénients : Portée limitée.

ESP-NOW

- Utilisé dans : La communication entre capteurs sans infrastructure réseau.

- Priorisé pour : Sa faible latence et l'absence de connexion Wi-Fi nécessaire.
- Avantages : Très faible consommation, rapide.
- Inconvénients : Portée réduite et faible bande passante.

LoRaWAN

- Utilisé dans : Les systèmes nécessitant une communication à longue portée.
- Priorisé pour : Son efficacité énergétique et sa couverture étendue.
- Avantages : Longue portée, très basse consommation.
- Inconvénients : Débit faible.

MQTT

- Utilisé dans : La transmission des données des capteurs vers le cloud.
- Priorisé pour : Son modèle léger et efficace pour l’IoT.
- Avantages : Asynchrone, faible bande passante requise.
- Inconvénients : Sécurisation nécessaire.

Partie 3 : Choix d'implémentation pour Harvest Hub

1. Choix de la base de données

Pour le stockage et la gestion des données issues des sondes, plusieurs bases de données ont été envisagées :

- **TimescaleDB** : Une extension de PostgreSQL optimisée pour les séries temporelles, idéale pour l'analyse des données issues des capteurs.
- **InfluxDB** : Une solution spécialisée dans les bases de données temporelles, offrant de hautes performances pour les requêtes en temps réel.
- **MySQL** : Une base de données relationnelle classique, pouvant être utilisée pour les besoins généraux.

Choix final : TimescaleDB

TimescaleDB a été retenu pour sa scalabilité, sa capacité à gérer des flux de données en continu et sa compatibilité avec PostgreSQL.

2. Choix de l'API backend

Le développement de l'API backend repose sur **Go (Golang)** pour plusieurs raisons :

- Haute performance et gestion efficace de la concurrence.
- Faible empreinte mémoire et rapidité d'exécution.
- Bonne prise en charge des applications distribuées et de l'IoT.

Choix final : Go (Golang)

3. Choix des technologies pour l'application Web

Pour l'interface utilisateur web, **React.js** a été choisi en raison de :

- Sa flexibilité et sa grande communauté.
- Son écosystème riche et performant.
- La réutilisation des composants facilitant le développement.

Choix final : React.js

4. Choix des technologies pour l'application mobile

Le développement mobile s'appuiera sur **React Native** pour garantir :

- Un code unique déployable sur iOS et Android.
- Une rapidité de développement accrue.
- Une communauté active et de nombreuses bibliothèques prêtes à l'emploi.

Choix final : React Native

5. Choix des technologies pour les sondes et le hub

5.1. Firmware des sondes et du hub

Le firmware des sondes et du hub sera développé en **Rust**, car :

- Il offre une excellente gestion de la mémoire et de la sécurité.
- Il est performant, proche de C/C++, tout en étant plus sûr.
- Il est adapté aux systèmes embarqués critiques.

Choix final : Rust

5.2. Protocoles de communication entre les sondes

Deux protocoles ont été retenus pour les sondes :

- **ESP-NOW** : Faible latence, faible consommation, absence de dépendance à une infrastructure réseau.
- **Bluetooth Low Energy (BLE)** : Compatible avec les appareils mobiles, consommation d'énergie réduite.

Choix final : ESP-NOW et BLE

5.3. Communication du hub avec le cloud

Pour assurer la connectivité du hub avec le cloud, les technologies suivantes ont été retenues :

- **Wi-Fi** : Pour une connexion rapide lorsque disponible.
- **Cellulaire (4G/5G)** : Pour garantir une connexion en toutes circonstances.

Choix final : Wi-Fi et cellulaire

Conclusion

Le choix des technologies pour Harvest Hub repose sur des critères de performance, de scalabilité et d'efficacité énergétique. L'utilisation de **Rust** pour le firmware, **Go** pour l'API, et **React.js/React Native** pour les interfaces garantit un système performant et modulaire, adapté aux exigences des sondes connectées.