Recherche de solutions IR

langages de développement: pour le backend C++ (performance plus élevée que le python, consomme moins que le language de python,**Contrôle bas niveau** → Utile pour gérer la mémoire et l’optimisation fin)

Pour le frontend on a choisit le Vue.js au lieu du react car un framework plus simple, plus intuitif et avec une meilleure prise en main

Base de données pour l'historique des consommations :

MySQL:

**Rapide et efficace** pour les lectures et écritures fréquentes.  
✅ **Facile à configurer et à gérer**, avec un large écosystème.  
✅ **Bonne scalabilité en lecture** (réplication facile).  
❌ **Moins flexible pour des données complexes** → Pas idéal si les données ont une structure changeante.  
❌ **Moins performant pour les requêtes analytiques avancées** par rapport à PostgreSQL.

📌 **Cas d’usage recommandé** :

* Données bien structurées avec relations fixes (historique simple avec ID utilisateur, date, consommation).
* Applications web classiques où la rapidité en lecture est importante.

interface web: HTML(base), css(style)

API REST pour permettre des communications entre la plateforme et les capteurs.

Frameworks de communication réseau (LoRa car c plus etendue et c parfait pour le campings.

cloud computing

2 serveur en redondance

**architecture projet:**

## **Vue d’ensemble de l’architecture**

Le système repose sur trois grandes parties :

**Acquisition des données** : Capteurs connectés qui collectent les données de consommation d’eau et d’électricité.

**Transmission et stockage** : Envoi des données via un réseau (LoRa) vers un serveur central.

**Traitement et visualisation** : Interface web permettant aux gestionnaires de surveiller et analyser les consommations.

Architecture technique détaillée

### **1. Acquisition des données (IoT)**

**Capteurs** :

Capteurs d’eau (débitmètres)

Capteurs d’électricité (compteurs Linky, télé-information)

**Communication** :

Transmission des données via **LoRa**

Passerelle (Gateway) pour récupérer les données et les envoyer au serveur

### **Backend (API et Base de données)**

**Serveur Backend (REST API en C++)**

* **Frameworks possibles** : (C++), Express.js (Node.js)
* Rôles :
  + Recevoir et stocker les données envoyées par les capteurs
  + Gérer les utilisateurs et les droits d’accès
  + Envoyer des alertes en cas d’anomalies  
    ✅ **Base de données** (stockage de l’historique des consommations)
* **MySQL** → Si données bien structurées

✅ **Gestion des notifications**

* Alertes par **email, SMS ou notification push** via Firebase

### **Frontend (Interface utilisateur web et mobile)**

✅ **Technologies** :

* **Vue.js** → Interface web

✅ **Fonctionnalités principales** :

* Tableau de bord avec graphiques des consommations
* Configuration des seuils d’alerte
* Consultation des historiques et rapports

**Schema architecture :**

Capteurs IoT (eau & électricité)

│

Passerelle LoRa

│

Serveur backend (C++ Node.js)

│

Base de données (MySQL)

│

API REST

│

Interface Web ( Vue.js)

| **Composant** | **Technologie recommandée** |
| --- | --- |

| Capteurs IoT | LoRa |
| --- | --- |

| Backend API | C++ Node.js (Express.js) |
| --- | --- |

| Base de données | MySQL |
| --- | --- |

| Frontend Web | Vue.js |
| --- | --- |

|  |  |
| --- | --- |

| Notifications | Firebase, Twilio (SMS) |
| --- | --- |

Conclusion

### **Conclusion**

Cette architecture permet :  
✅ Une collecte fiable des données via des **capteurs IoT**.  
✅ Un **serveur backend** robuste pour traiter et stocker les consommations.  
✅ Une **interface intuitive** permettant aux gestionnaires de surveiller l’historique et recevoir des alertes.