**🏠 Cahier des charges – Projet IoT : Maison Intelligente Connectée et Distribuée**

**1️ Présentation du projet**

**Titre du projet :**  
**SmartHome IoT – Système de maison intelligente à architecture distribuée (ESP32, Raspberry Pi, Spring Boot Microservices, Cloud, Mobile)**

**Objectif général :**  
Mettre en place une solution complète de **maison intelligente connectée**, permettant la **surveillance**, le **contrôle** et l’**automatisation** des équipements domestiques (capteurs et actionneurs) via une architecture IoT distribuée et modulaire.

**Contexte :**  
Avec l’avancée des technologies IoT, il est possible de connecter des appareils domestiques pour les rendre intelligents, autonomes et accessibles à distance. Ce projet vise à combiner des microcontrôleurs (ESP32, Arduino) avec une passerelle locale (Raspberry Pi) et un backend basé sur **Spring Boot microservices**, pour assurer un système évolutif, robuste et sécurisé.

**2️ Objectifs spécifiques**

* Développer un **réseau de dispositifs IoT** (ESP32, Arduino, caméras, capteurs, etc.).
* Concevoir une **passerelle locale** (Raspberry Pi) pour la communication entre le réseau local et le cloud.
* Créer un **backend cloud** en **Spring Boot (architecture microservices)**.
* Gérer les données, les utilisateurs et les règles automatiques sur le **Cloud Firebase ou AWS**.
* Développer une **application mobile** pour le contrôle à distance, les notifications et le monitoring.
* Garantir la **sécurité, la fiabilité et la scalabilité** du système.

**3️ Architecture du système**

**🧠 Description générale :**

+---------------------------------------------------------------+

| Application Mobile |

| (Android / iOS - surveillance, contrôle, notifications) |

+------------------------------↑--------------------------------+

│ (API REST / HTTPS)

+-----+-------------------------+

| Cloud / Backend Spring Boot |

| (microservices par pièce) |

+-----+-------------------------+

│ (MQTT / REST / WebSocket)

+-----------+-----------+

| Passerelle Locale (Raspberry Pi) |

| (Python – Broker MQTT + API) |

+-----------+-----------+

│ (Wi-Fi)

+----------------------+----------------------+

| Réseau local de dispositifs IoT |

| (ESP32, Arduino, caméras, capteurs, relais) |

+----------------------------------------------+

**4️ Composants du système**

**🔹 1. Dispositifs IoT (ESP32 / Arduino / Capteurs / Actionneurs)**

* Capteurs : DHT22 (température/humidité), MQ-2 (gaz/fumée), PIR (mouvement), LDR (luminosité), capteurs de sol, capteurs de porte, etc.
* Actionneurs : relais (lampes, moteurs), buzzer, servo-moteur, ventilateur.
* Caméra (ESP32-CAM ou caméra IP) pour la surveillance.
* Communication : Wi-Fi / MQTT vers Raspberry Pi.
* Langages : MicroPython ou Arduino C++.

**🔹 2. Passerelle IoT – Raspberry Pi**

* Rôle de **pont** entre les ESP32 et le Cloud.
* Exécute un **broker MQTT (Mosquitto)** pour la communication locale.
* Script **Python** pour :
  + Recevoir les données des capteurs.
  + Publier les mesures vers le backend Spring Boot.
  + Transmettre les ordres du cloud vers les ESP32.
* Peut stocker temporairement les données (cache local).
* Connecté en permanence au réseau Wi-Fi et Internet.

**🔹 3. Backend – Spring Boot (microservices)**

* Hébergé sur le **Cloud (AWS EC2 / Firebase Hosting / Google Cloud)**.
* Contient plusieurs **microservices**, chacun responsable d’une **zone ou fonction** :
  + garage-service → Gestion du garage.
  + jardin-service → Gestion du jardin et arrosage automatique.
  + salon-service → Contrôle du salon (température, lumière, etc.).
  + cuisine-service → Détection de gaz et incendie.
  + securite-service → Caméras, alarmes et détection d’intrusion.
  + user-service → Authentification (JWT / Firebase Auth).
  + data-service → Stockage et consultation des mesures.
  + rule-engine-service → Automatisations intelligentes.
  + alert-service → Notifications et alertes.

**Communication entre microservices :**

* REST API / RabbitMQ / Kafka (asynchrone).
* API Gateway (Spring Cloud Gateway).
* Service Discovery (Eureka Server).

**🔹 4. Cloud / Base de données**

* Stockage des données dans :
  + **Firebase Firestore** (realtime + cloud functions), ou
  + **AWS DynamoDB / MySQL Cloud**.
* Sauvegarde historique et synchronisation des mesures.
* Authentification via Firebase Auth.

**🔹 5. Application Mobile**

* Développée sous **Android (Java / Kotlin)** ou **React Native**.
* Fonctionnalités :
  + Affichage en temps réel des mesures.
  + Commande à distance (ON/OFF, alarme, portes, éclairage).
  + Notifications push (Firebase Cloud Messaging).
  + Interface moderne (dashboard, graphique, carte de la maison).

**5️ Fonctionnalités attendues**

**🔸 Surveillance**

* Consultation en temps réel de la température, humidité, luminosité, etc.
* Suivi vidéo (flux caméra).

**🔸 Contrôle**

* Activation/désactivation d’équipements via mobile.
* Contrôle automatique selon des règles définies.

**🔸 Automatisation**

* Exemples :
  + “Si humidité du sol < 30%, activer arrosage du jardin.”
  + “Si fumée détectée, déclencher alarme et notification.”
  + “Si mouvement détecté la nuit, allumer la lumière du garage.”

**🔸 Notifications**

* Alertes en cas d’anomalie : gaz, feu, intrusion, température critique.
* Historique des alertes.

**🔸 Gestion des utilisateurs**

* Rôles : administrateur / utilisateur.
* Sécurisation par JWT et Firebase Auth.

**6️ Exigences techniques**

| **Catégorie** | **Exigence** |
| --- | --- |
| **Matériel** | ESP32, Arduino UNO, Raspberry Pi 4, capteurs divers |
| **Langages** | MicroPython / C++ (IoT), Python (gateway), Java (Spring Boot), Java/Kotlin (mobile) |
| **Protocole** | MQTT (communication locale), HTTP/HTTPS (cloud) |
| **Cloud** | Firebase ou AWS (stockage, auth, notifications) |
| **Architecture logicielle** | Microservices Spring Boot + Gateway + Discovery |
| **Sécurité** | HTTPS, JWT, chiffrement MQTT |
| **Temps réel** | Publication immédiate via MQTT et WebSocket |
| **Base de données** | Firestore / MySQL / MongoDB |
| **Application mobile** | Interface ergonomique, notifications, contrôle temps réel |
| **Performance** | Latence < 2 s, taux de disponibilité > 95 % |

**7️ Contraintes**

* Connexion Internet stable requise.
* Raspberry Pi doit être alimenté et connecté en permanence.
* Gestion de la synchronisation si coupure réseau.
* Sécurité du réseau domestique et du cloud.
* Optimisation énergétique des ESP32.

**8️ Livrables**

* Schéma d’architecture IoT complet (matériel + logiciel).
* Code source :
  + ESP32 / Arduino (firmware)
  + Raspberry Pi (passerelle Python)
  + Spring Boot (microservices)
  + Application mobile
* Documentation technique détaillée.
* Rapport final (architecture, tests, résultats).
* Démonstration fonctionnelle.

**9️ Tests et validation**

| **Type de test** | **Description** | **Résultat attendu** |
| --- | --- | --- |
| Test MQTT | Communication ESP32 ↔ Raspberry Pi | 100 % messages reçus |
| Test Cloud | Synchronisation Raspberry ↔ Spring Boot | Données cohérentes |
| Test microservices | Communication inter-services (REST / RabbitMQ) | Réponses valides |
| Test mobile | Commande des équipements via app | Actionneur réactif |
| Test alerte | Détection fumée / intrusion | Notification instantanée |
| Test réseau | Reconnexion automatique | Stabilité confirmée |

**🔟 Évolutions possibles**

* Ajout d’un **dashboard web** (Angular / React).
* Intégration **IA / Machine Learning** (prédiction d’anomalies).
* Commande vocale (Google Assistant / Alexa).
* Analyse énergétique.
* Extension multi-maisons.

**🧾 Conclusion**

Ce projet propose une **architecture IoT distribuée et modulaire** reposant sur :

* des **devices intelligents (ESP32 / Arduino)**,
* une **passerelle locale Raspberry Pi**,
* un **backend cloud microservices Spring Boot**,
* et une **application mobile conviviale**.

Cette approche garantit un système **sécurisé, évolutif, et intelligent**, apte à être déployé dans des environnements domestiques ou industriels.